Requirements Engineering in Model Transformation Development: A Technique Suitability Framework for Model Transformation Applications

Sobhan Yassipour Tehrani, Kevin Lano
Department of Informatics, King's College London, London WC2R 2LS, U.K.
E-mail: {sobhan.yassipour_tehrani, kevin.lano}@kcl.ac.uk

***Abstract*—Model transformations (MTs) are central to model-driven engineering (MDE). They can be used for a range of purposes, including to improve the quality of models, to refactor models, to migrate or translate models from one representation to another, and to generate code or other artifacts from models. At present, the development of model transformation is mainly focused on the specification and implementation phases, whereas there is a lack of support in other phases including: requirements, analysis, design and testing. Furthermore, there is a lack of cohesive support for transformations including: notations, methods and tools within all phases during the development process, which makes the maintenance and understandability of the transformation code problematic. One of the main hindrances for not using a systematic Requirements Engineering (RE) process, the initial phase of the software development life-cycle where software's specifications are declared, before starting the development, could be the false assumption that it is a waste of time/cost and would delay implementation. The goal of this paper is to evaluate model transformation technology from a requirements engineering process point of view. Moreover, we identify criteria for selecting appropriate requirements engineering techniques, and we propose a framework for this selection process.**

***Index Terms*—model transformations; requirements engineering; requirements engineering framework; model transformation case study; RE technique framework**

I. Introduction

Requirements engineering (RE) has been a relatively neglected aspect of model transformation (MT) development because the emphasis in transformation development has been upon specifications and implementations. The failure to explicitly identify requirements may result in developed transformations, which do not satisfy the needs of the users of the transformation. Problems may arise because implicitly-assumed requirements have not been explicitly stated; for instance, that a migration or refactoring transformation should preserve the semantics of its source model in the target model, or that a transformation is only required to operate on a restricted range of input models. Without thorough requirements elicitation, important requirements may be omitted from consideration, resulting in a developed transformation, which fails to achieve its intended purpose.

In [1] we reviewed the current practice of RE for MT and identified a framework for an improved RE process. In this paper we extend [1] with more details of the framework, and we give extracts from a large-scale application of the framework to a C code generator.

We use the 4-phase RE process model proposed by Sommerville [2] and adapt it according to our specific needs. This process model is widely accepted by researchers and professional experts. The model defines the following as the most important phases of RE, which should be applied: domain analysis and requirements elicitation, evaluation and negotiation, specification and documentation, validation and verification.

In Section II we describe related work. Section III gives a background on requirements engineering for model transformations as well as transformation semantics and its nature. We also identify how formalised requirements can be validated and can be used to guide the selection of design patterns for the development of the transformation. In Section IV we examine RE techniques and identify how these can be applied to MT development. In Section V we present a framework for an RE process and RE technique selection for MT. In Section VI we give a case study to evaluate the application of our framework.

II. Related Work

The increasing complexity and size of today's software systems has resulted in raising the complexity and size of model transformations. Although there have been different transformation tools and languages, most of them are focused on the specification and implementation phases. According to [3], most of the transformation languages proposed by Model Driven Engineering (MDE), a software development methodology, are only focused towards the implementation phase and are not integrated in a unified engineering process. It could be said that at the moment, the transformation process is performed in an ad-hoc manner; defining the problem and then directly beginning the implementation process. At present, the development of model transformation is mainly focused on the specification and implementation phases, whereas there is a lack of support in other phases including: requirements, analysis, design and testing. Furthermore, there is a lack of cohesive support for transformations including: notations, methods and tools within all phases during the development process, which makes the maintenance and understandability of the transformation code problematic [3].

As Selic [4] argues, "we are far from making the writing of model transformations an established and repeatable technical task". The software engineering of model transformations has only recently been considered in a systematic way, and most of this work [5][6] is focussed upon design and verification rather than upon requirements engineering. The work on requirements engineering in *trans*ML is focused upon functional

requirements, and the use of abstract syntax rules to express them. Here, we consider a full range of functional and non-functional requirements and we use concrete syntax rules for the initial expression of functional requirements.

In order to trace the requirements into subsequent steps, *trans*ML defines a modelling language, which represents the requirements in the form of Systems Modeling Language (SysML) [7] diagrams. This allows the transformer(s) to link requirements of a model transformation to its corresponding analysis and design models, code and other artifacts. Having a connection amongst different artifacts in the model transformation development process enables the transformer(s) to check the correctness and completeness of all requirements [8].

We have carried out a survey and interview study of RE for MT in industrial cases [9]. This study showed that RE techniques are not used in a systematic way in current industrial MT practice. In this paper, we describe a requirements engineering process for transformations based on adaptions of the standard RE process model, and upon adaptions of RE techniques for transformations.

III. Requirements for Model Transformations

Requirements for a software product are generally divided into two main categories: functional requirements, which identify what functional capabilities the system should provide, and non-functional requirements, which identify quality characteristics expected from the developed system and restrictions upon the development process itself.

The functional requirements of a model transformation $\tau: S \rightarrow T$, which maps models of a source language S to a target language T are defined in terms of the effect of τ on model m of S, and the relationship of the resulting model n of T to m. It is a characteristic of model transformations that such functional requirements are usually decomposed into a set of mapping requirements for different cases of structures and elements within S. In addition, assumptions about the input model should be identified as part of the functional requirements.

It can be observed in many published examples of model transformations that the initial descriptions of their intended functional behaviour is in terms of a concrete syntax for the source and target languages, which they operate upon. For instance in [10], the three key effects of the transformation are expressed in terms of rewritings of Unified Modeling Language (UML) class diagrams. In [11], the transformation effects are expressed by parallel rewritings of Petri Nets and statecharts. In general, specification of the intended functionality of the transformation in terms of concrete syntax rules is more natural and comprehensible for the stakeholders than is specification in terms of abstract syntax. However, this form of description has the disadvantage that it may be imprecise; there may be significant details of models, which have no representation in the concrete syntax, or there may be ambiguities in the concrete syntax representation. Therefore, conversion of the concrete syntax rules into precise abstract syntax rules is a necessary step as part of the formalisation of the requirements.

Requirements may be functional or non-functional (e.g., concerned with the size of generated models, transformation efficiency or confluence). Another distinction, which is useful for transformations is between local and global requirements:

- Local requirements are concerned with localised parts of one or more models. Mapping requirements define when and how a part of one model should be mapped onto a part of another. Rewriting requirements dictate when and how a part of a model should be refactored/transformed in-place.
- Global requirements identify properties of an entire model. For example that some global measure of complexity or redundancy is decreased by a refactoring transformation. Invariants, assumptions and postconditions of a transformation usually apply at the entire model level.

Figure 1 shows a taxonomy of functional requirements for model transformations based on our experience of transformation requirements.

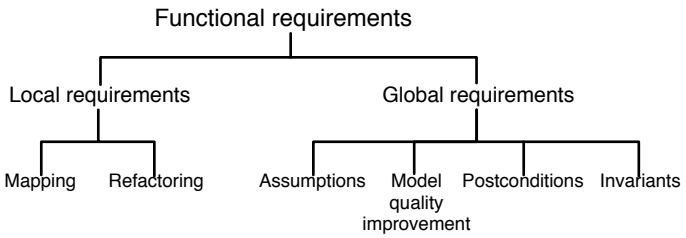

Figure 1. A taxonomy of functional requirements

We have also created a taxonomy of the non-functional requirements that one has to consider during the RE process. Figure 2 shows a general decomposition of non-functional requirements for model transformations. The quality of service categories correspond closely to the software quality characteristics identified by the IEC 25010 software quality standard [12].

Non-functional requirements for model transformations could be further detailed. For instance, regarding the performance requirements, boundaries (upper/lower) could be set on execution time, memory usage for models of a given size, and the maximum capability of the transformation (the largest model it can process within a given time). Restrictions can also be placed upon the rate of growth of execution time with input model size (for example, that this should be linear). Taxonomizing the requirements according to their type not only would make it clearer to understand what the requirements refer to, but also by having this type of distinction among them will allow for a more semantic characterization of requirements.

Maturity and fault tolerance are a subset of reliability requirements for a transformation. Depending on its history and to the extent to which a transformation has been used, maturity requirements could be measured. Fault tolerance requirements

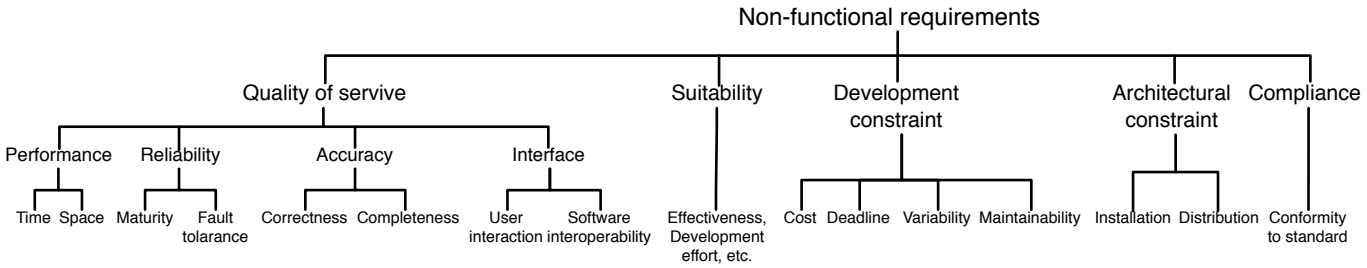

Figure 2. A taxonomy of non-functional requirements for MT

can be quantified in terms of the proportion of execution errors, which are successfully caught by an exception handling mechanism, and in terms of the ability of the transformation to detect and reject invalid input models.

As depicted in the above figure, the accuracy characteristic includes two sub-characteristics: correctness and completeness. Correctness requirements can be further divided into the following forms [13]:

- *Syntactic correctness*: a transformation τ is syntactically correct when a valid input model m from source language S is transformed to target language T, then (if τ terminates) it produces a valid result, in terms of conformation to the T's language constraints.
- *Termination*: a transformation τ will always terminate if applied to a valid S model.
- *Confluence*: all result models produced by transformation τ from a single source model are isomorphic.
- *Model-level semantic preservation*: a transformation τ is preserved model-level semantically, if m and n have equivalent semantics under semantics-assigning maps Sem_S on models of S and Sem_T on models of T.
- *Invariance*: some properties *Inv* should be preserved as true during the entire execution of transformation τ [13].

An additional accuracy property that can be considered is the existence of invertibility in a transformation $\sigma : T \rightarrow S$, which inverts the effect of τ. Given a model n derived from m by τ, σ applied to n produces a model $m\prime$ of S isomorphic to m. A related property is change propagation, which means that small changes to a source model can be propagated to the target model without re-executing the transformation on the entire source model. A further property of verifiability is important for transformations, which is part of a business-critical or safety-critical process. This property identifies how effectively a transformation can be verified. Size, complexity, abstraction level and modularity are contributory factors to this property. The traceability property is the requirement that an explicit trace between mapped target model elements and their corresponding source model elements should be maintained by the transformation, and be available at its termination. Under interface are requirements categories of User interaction (subdivided into usability and convenience) and software interoperability. Usability requirements can be decomposed into aspects, such as understandability, learnability and attractiveness [14]. Software interoperability can be decomposed into interoperability capabilities of the system with each intended environment and software system, with which it is expected to operate.

Based on [14], we define suitability as the capability of a transformation approach to provide an appropriate means to express the functionality of a transformation problem at an appropriate level of abstraction, and to solve the transformation problem effectively and with acceptable use of resources (developer time, computational resources, etc.). In [10] we identified the following subcharacteristics for the suitability quality characteristic of model transformation specifications: abstraction level, size, complexity, effectiveness and development effort.

Requirements of single transformations can be documented using the SysML notation adopted in [3], but with a wider range of requirement types represented. Use case diagrams can be used to describe the requirements of a system of transformations. Each use case represents an individual transformation, which may be available as a service for external users, or which may be used internally within the system as a subtransformation of other transformations.

We have investigated a specific functional requirements taxonomy according to the characteristic of model transformations (Table I). All types of functional requirements for model transformations including: mapping, assumptions and post-conditions requirements could be formalized as predicates or diagrams at the concrete and abstract syntax levels. Concrete syntax is often used at the early stages (RE stages) in the development cycle in order to validate the requirements by stakeholders since the concrete syntax level is more convenient, whereas abstract syntax rule, is often used in the implementation phase for developers. However, there should be a direct correspondence between the concrete syntax elements in the informal/semi-formal expression of the requirements, and the abstract syntax elements in the formalised versions.

IV. Application of RE in MT

In model transformation, requirements and specifications are very similar and sometimes are considered as the same element. Requirements determine what is needed and what needs to be achieved while taking into account the different

TABLE I. TRANSFORMATION REQUIREMENTS CATALOGUE

	Refactoring	**Refinement**	**Migration**
Local Functional	Rewrites/ Refactorings	Mappings	Mappings
Local Non-functional	Completeness (all cases considered)	Completeness (all source entities, features considered)	Completeness (all source entities, features considered)
Global Functional	Improvement in quality measure(s), Invariance of language constraints, Assumptions, Postconditions	Invariance, Assumptions, Postconditions	Invariance, Assumptions, Postconditions
Global Non-functional	Termination, Efficiency, Modularity, Model-level semantic preservation, Confluence, Fault tolerance, Security	Termination, Efficiency, Modularity, Traceability, Confluence, Fault tolerance, Security	Termination, Efficiency, Modularity, Traceability, Confluence, Fault tolerance

stakeholders, whereas specifications define precisely what is to be developed.

Requirements engineering for model transformations involves specialised techniques and approaches because transformations (i) have highly complex behaviour, involving non-deterministic application of rules and inspection/ construction of complex model data, (ii) are often high-integrity and business-critical systems with strong requirements for reliability and correctness.

Transformations do not usually involve much user interaction, but may have security requirements if they process secure data. Correctness requirements, which are specific to transformations, due to their characteristic execution as a series of rewrite rule applications, with the order of these applications not algorithmically determined, are: (i) confluence (that the output models produced by the transformation are equivalent, regardless of the rule application orders), (ii) termination (regardless of the execution order), (iii) to achieve specified properties of the target model, regardless of the execution order, which is referred to as semantic correctness [9].

The source and target languages of a transformation may be precisely specified by metamodels, whereas the requirements for its processing may initially be quite unclear. For a migration transformation, analysis will be needed to identify how elements of the source language should be mapped to elements of the target. There may not be a clear relationship between parts of these languages, there may be ambiguities and choices in mapping, and there may be necessary assumptions on the input models for a given mapping strategy to be well-defined. The requirements engineer should identify how each entity type and feature of the source language should be migrated.

For refactorings, the additional complications arising from update-in-place processing need to be considered and the application of one rule to a model may enable further rule applications, which were not originally enabled. The requirements engineer should identify all the distinct situations, which need to be processed by the transformation such as arrangements of model elements and their inter-relationships and significant feature values.

A. Application of RE Techniques for MT

A large number of requirements elicitation techniques have been devised. Through the analysis of surveys and case studies, we have identified the following adaption of RE techniques for MT.

The following techniques are the most suitable RE techniques to use during the requirements elicitation stage, which have been adapted according to the nature of model transformation technology.

Structured interviews: in this technique the requirements engineer asks stakeholders specific prepared questions about the domain and the system. The requirements engineer needs to define appropriate questions, which help to identify issues of scope and product (output model) requirements, similar to that of unstructured interviews. This technique is relevant to all forms of transformation problems. We have defined a catalogue of MT requirements for refactorings, refinements and migrations, as an aid for structured interviews, and as a checklist to ensure that all forms of requirements appropriate for the transformation are considered.

Rapid prototyping: in this technique a stakeholder is asked to comment on a prototype solution. This technique is relevant for all forms of transformation, where the transformation can be effectively prototyped. Rules could be expressed in a concrete grammar form and reviewed by stakeholders, along with visualisations of input and output models. This approach fits well with an Agile development process for transformations.

Scenario analysis: in this approach the requirements engineer formulates detailed scenarios/use cases of the system for discussion with the stakeholders. This is highly relevant for MT requirements elicitation. Scenarios can be defined for different required cases of transformation processing. The scenarios can be used as the basis of requirements formalisation. This technique is proposed for transformations in [3]. A risk with scenario analysis is that this may fail to be complete and may not cover all cases of expected transformation processing. It is more suited to the identification of local rather than global requirements.

Regarding the requirements evaluation and negotiation stage, prototyping techniques are useful for evaluating requirements, and for identifying deficiencies and areas where the intended behaviour is not yet understood. A goal-oriented analysis technique such as Knowledge Acquisition in automated specification (KAOS) or SysML can be used to decompose requirements into sub-goals. A formal modelling notation such as Object Constraint Language (OCL) or state machines/state charts can be used to expose the implications of requirements. For transformations, state machines may be useful to identify implicit orderings or conflicts of rules, which arise because the effect of one rule may enable or disable the occurrence of another. Requirements have to be prioritized according to

TABLE II. REQUIREMENTS PRIORITY FOR DIFFERENT TRANSFORMATIONS

Category	Primary requirement	Secondary requirement
Refactoring	Model quality improvement Model-level semantic preservation Syntactic correctness Termination	Invariance Confluence
Migration	Syntactic correctness Model-level semantic preservation Termination	Invertibility Confluence Traceability
Refinement	Syntactic correctness Model-level semantic preservation Confluence Termination	Traceability

their importance and the type of transformation. For instance, in a refinement transformation, the semantics of the source and target model have to be equivalent as the primary requirement and to have a traceability feature as a secondary requirement. Also, there should be no conflict among the requirements. For instance, there is often a conflict between the time, quality and budget of a project. The quality of the target model should be satisfactory with respect to the performance (time, cost and space) of the transformation. Several RE techniques exist, which could be applicable to MT during the requirements specification phase in which business goals are represented in terms of functional and non-functional requirements. In Table II, requirements have been categorised according to the type of the transformation.

Techniques for requirements specification and documentation stage include: UML and OCL, structured natural language, and formal modelling languages. At the initial stages of requirements elicitation and analysis, the intended effect of a transformation is often expressed by sketches or diagrams using the concrete grammar of the source and target languages concerned (if such grammars exist), or by node and line graphs if there is no concrete grammar. A benefit of concrete grammar rules is that they are directly understandable by stakeholders with knowledge of the source and target language notations. They are also independent of specific MT languages or technologies. Concrete grammar diagrams can be made more precise during requirements formalisation, or refined into abstract grammar rules. An informal mapping/refactoring requirement of the form of

> "*For each instance e of entity type E, that satisfies condition Cond, establish Pred* "

can be formalised as a use case postcondition such as:

> E::
> $\text{Cond}' \Rightarrow \text{Pred}'$

where *Cond*′ formalises Cond, and *Pred*′ formalises Pred.

For requirements verification and validation stage, the formalised rules can be checked for internal correctness properties such as definedness and determinacy, which should hold for meaningful rules. A prototype implementation can be generated, and its behaviour on a range of input models covering all of the scenarios considered during requirements elicitation can be checked. When a precise expression of the functional and non-functional requirements has been defined, it can be validated with the stakeholders to confirm that it does indeed accurately express the stakeholders intentions and needs for the system. The formalised requirements of a transformation τ: $S \rightarrow T$ can also be verified to check that they are consistent; the functional requirements must be mutually consistent. The assumptions and invariant of τ, and the language constraints of S must be jointly consistent. The invariant and postconditions of τ, and the language constraints of T must be jointly consistent. Each mapping rule Left-Hand Side (LHS) must be consistent with the invariant, as must each mapping rule Right-Hand Side (RHS).

These consistency properties can be checked using tools such as Z3 or Alloy, given suitable encodings [15], [16]. Model-level semantics preservation requirements can in some cases be characterised by additional invariant properties, which the transformation should maintain. For each functional and non-functional requirement, justification should be given as to why the formalised specification satisfies these requirements. For example, to justify termination, some variant quantity Q: Integer could be identified, which is always non-negative and which is strictly decreased by each application of a mapping rule [13]. Formalised requirements in temporal logic could then be checked for particular implementations using model-checking techniques, as in [17].

V. RE Technique Framework for MT

There are several methods and techniques proposed by the requirements engineering community, however selecting an appropriate set of requirements engineering techniques for a project is a challenging issue. Most of these methods and techniques were designed for a specific purpose and none cover the entire RE process. Researchers have classified RE techniques and categorised them according to their characteristics. For instance, Hickey *et al.* [18] proposed a selection model of elicitation techniques, Maiden *et al.* [19] came up with a framework for requirements acquisition methods and techniques. However, lack of support for selecting the most appropriate set of techniques for a software project has made requirements engineering one of the most complex parts of software engineering process. At the moment, RE techniques are selected mainly based on personal preference rather than characteristics and specifications of a project. In the following sections, we analyse the attributes of requirements engineering techniques and organizations in which the project is delivered and the actual type of project, in order to select a suitable set of RE techniques for specific projects.

A. RE Attribute Analysis

In general, a project is assigned to an organization in order to be developed. Usually the software developing organization is selected according to the type of project. Classification of RE techniques have a direct relation with the type of the

proposed project, the organization and the internal attributes of a specific technique. In this section, we analyse the attributes of techniques, the project and the organization in order to identify the most well-suited set of techniques for a particular type of project.

1) Technique Attribute: As mentioned earlier, multiple techniques exist in requirements engineering. Each technique has some attributes that could be used in choosing RE techniques. Identifying the technique attributes could be very useful as they allow us to compare different techniques. We have identified 33 attributes from which 23 were defined by [20]. These attributes are categorized according to the RE phase (*Kotonya* and *Sommerville* model) that they belong to. These attributes are selected based on characteristics of RE techniques as well as other researchers' criteria and frameworks [19], [21], [2]. For instance, some RE techniques are well-suited for identifying non-functional requirements. Therefore if non-functional requirements in a particular project have high priority then the attribute of *ability to help identify non-functional requirements* is important and applying the appropriate RE technique to find non-functional requirements would be necessary (such as the NFR framework). In Table III we have adapted the attributes of [20] to make them specific for MT.

2) Transformation Project Attribute: A transformation project's attribute is also an important factor in order to select RE techniques. Each project has a set of attributes and the priority of its attributes may vary based on the characteristics of a project. For instance, the category of a project that it belongs to is an attribute, therefore RE techniques for a category of safety-critical system may vary from a non-critical system. In this research, we have identified nine attributes, which shall be analysed in more detail relevant to a project. In Table IV, we have explained the selected transformation attributes in more detail.

3) Organization Attribute: Every software developing organization applies the RE process in a different manner. This difference is caused by the behaviour of developers and stakeholders involved in the project. This behaviour is influenced by different factors of the organization such as: size, culture, policy and complexity. These factors have a direct effect on the way the RE process is performed. For instance, in a small organization, new technologies and expensive RE techniques may not be the first choice due to the high cost of it, whereas in a large and complex organization more flexible and disciplined techniques are required to do RE tasks. Although there is no limit to the attributes of an organization we have identified the following:

- Ability to support customer/client involvement
- Ability to classify requirements based on different stakeholders
- Ability to predict and manage sudden requirements modifications by stakeholders
- Ability to assure stakeholders about their confidentiality and privacy.

B. Technique Framework

Our overall procedure for selecting RE techniques for a MT project involves:

- The set all suitable RE techniques (e.g. interview, prototype) in each category (i.e. elicitation, negotiation, specification, verification) is identified.
- For each requirement identified within the project, each RE technique $\mathtt{t}$ is assigned a value $\mathtt{RA}(t)$ (for Requirement Attribute) representing the suitability of applying $\mathtt{t}$ to fulfil this requirement, based on the technique's attributes. Tables (III, V, VI, VII, VIII) give examples of these adapted attribute measures.
- For each requirement identified within the project, each RE technique $\mathtt{t}$ is assigned a value $\mathtt{PD}(\mathtt{t})$ (for Project Description) representing the suitability of applying $\mathtt{t}$ to fulfil this requirement, based on the project's descriptions.
- Evaluating the degree $\mathtt{E}$ (for Experience) of experience/-expertise regarding the RE technique $\mathtt{t}$ available in the development team. $\mathtt{E}(\mathtt{t})$ represents the level of experience and practical and theoretical knowledge of the developer regarding $\mathtt{t}$.
- Using $\mathtt{S}(\mathtt{t})$, the overall suitability score of a particular RE technique ($\mathtt{t} \in \mathtt{T}$) can be calculated, and hence it would be possible to define a ranking of techniques $\mathtt{t}$ based on their suitability scores $\mathtt{S}(\mathtt{t})$ for use in the project. $\mathtt{S}(\mathtt{t})$ is defined as:

$$\mathtt{S}(\mathtt{t}) = \mathtt{RA}(\mathtt{t}) \times \mathtt{PD}(\mathtt{t}) \times \mathtt{E}(\mathtt{t})$$

See the Appendix for more details.

In the following sections, we will discuss the techniques and their attributes in more detail, we will consider only those types of RE techniques that are most relevant to MT development (according to our survey of industrial cases, and from expert experience).

VI. Evaluation

In this section of the paper, we will evaluate the framework by applying it to a real industrial case study. This case study concerns the development of a code generator for the UML-Rigorous Systems Design Support (UML-RSDS) [22] dialect of UML. UML-RSDS is a model transformation tool, which is able to manufacture software systems in an automated manner which takes as input a text representation of a class diagram and use cases conforming to the UML-RSDS design metamodels, and produces as output text files in valid ANSI C, as defined in the current ANSI standard. Given a valid UML-RSDS model, the translator should produce a C application with the same semantics. The target code should be structured in the standard C style with header and code files and standard C libraries may be used. The produced code should be of comparable efficiency to hand-written code. The code generation process should not take longer than 1 minute for class diagrams with fewer than 100 classes.

The identified stakeholders included: (i) the UML-RSDS development team; (ii) users of UML-RSDS who require C

TABLE III. RE TECHNIQUE ATTRIBUTES AND CLASSIFICATIONS [20]

ID	Categories	Attributes of techniques
1		Ability to elicit MT requirements
2		Ability to facilitate communication
3		Ability to help understand social issues
4		Ability to help getting domain knowledge
5	Elicitation	Ability to help getting implicit knowledge
6		Ability to help identifying MT stakeholders
7		Ability to help identifying non-functional requirements
8		Ability to help identifying viewpoints
9		Ability to help model and understand requirements
10		Ability to analyse and model requirements with relevant MT notations
11		Ability to help analyse non-functional requirements
12	Evaluation & Negotiation	Ability to facilitate negotiation with customers
13		Ability to help prioritizing requirements according to stakeholders need
14		Ability to help prioritizing requirements according to the transformation
15		Ability to help identify accessibility of the transformation
16		Ability to help model interface requirements
17		Ability to help re-usability of MT requirements
18		Ability to represent MT requirements
19		Ability to help requirements verification
20		Completeness of the semantics of the notation
21	Specification & Documentation	Ability to help write precise requirements using MT notation
22		Ability to help write complete requirements
23		Ability to help with requirements management
24		Ability to help design highly modular systems
25		Implementability of the notation used
26		Ability to help identify ambiguous requirements
27	Validation & Verification	Ability to help identify inconsistency and conflict
28		Ability to help identify incomplete requirements
29		Ability to support MT language
30		Maturity of supporting tool
31	Other aspects	Learning curve (Introduction cost)
32		Application cost
33		Complexity of technique

code for embedded or limited resource systems; (iii) end-users of such systems. Direct access was only possible to stakeholders (i). Access to other stakeholders was substituted by research into the needs of such stakeholders. According to our RE technique framework, an initial phase of requirements elicitation for this system used document mining (research into the ANSI C language and existing UML to C translators) and a semi-structured interview with the principal stakeholder. This produced an initial set of requirements, with priorities.

The translator has the high-level functional (F) requirement:

F1: Translate UML-RSDS designs (class diagrams, OCL, activities and use cases) into ANSI C code.

The functional requirement was decomposed into five high-priority subgoals, each of which is the responsibility of a separate subtransformation including:

- F1.1: Translation of types
- F1.2: Translation of class diagrams
- F1.3: Translation of OCL expressions
- F1.4: Translation of activities
- F1.5: Translation of use cases

Each translation in this list is dependent upon all of the preceding translations. In addition, the translation of operations of classes depends upon the translation of expressions and activities. The development was therefore organised into five iterations, one for each translator part, and each iteration was given a maximum duration of one month. Other high-priority requirements identified for the translator were the following functional and non-functional (NF) system (product) requirements:

- NF1: Termination: given correct input
- F2: Syntactic correctness: given correct input, a valid C program will be produced
- F3: Model-level semantic preservation: the semantics of the source and target models are equivalent
- F4: Traceability: a record should be maintained of the correspondence between source and target elements

Medium-level priority requirements were:

- F5: Bidirectionality between source and target
- NF2: Efficiency: input models with 100 classes and 100 attributes should be processed within 30 seconds
- NF3: Modularity of the transformation

Low-priority requirements were:

- F6: Confluence
- NF4: Flexibility: ability to choose different C interpretations for UML elements

TABLE IV. TRANSFORMATION PROJECT ATTRIBUTE MEASURES

Transformation attributes	***Value***
Transformation size	*Very Big*: when the number of transformation rules are more than 300 *Big:* when the number of transformation rules are between 150 and 300 *Medium:* when the number of transformation rules are between 100 and 150 *Small:* when the number of transformation rules are between 50 and 100 *Very Small:* when the number of transformation rules are less than 50
Transformation complexity	*Very High:* transformation correctness, completeness and effectiveness are very complicated *High:* transformation correctness, completeness and effectiveness are complicated *Medium:* transformation correctness, completeness and effectiveness are medium level *Small:* transformation correctness, completeness and effectiveness are clear *Very small:* transformation correctness, completeness and effectiveness are easy to achieve
Transformation requirements volatility	*Very High:* transformation requirements keep changing throughout the entire development (more than 50% change of requirements) *High:* transformation requirements keep changing throughout the entire development (25%-50% change of requirements) *Medium:* Some of the requirements change during the development (10%-25% change of requirements) *Low:* A few requirements might change during the development (5%-10% change of requirements) *Very Low:* Change of requirements is unlikely to happen
Developer-customer relationship	*Very High:* there is a very good and constant interaction amongst the developers and the customer *High:* there is a good and constant interaction amongst the developers and the customer *Medium:* there are some contacts between the developers and customers when it is necessary *Low:* there are few meetings between the two parties only when it is essential *Very Low:* there is no contact between the customer and developers throughout the development
Transformation safety	*Very High:* there is a very high likelihood that the transformation will have safety consequences *High:* there is a high likelihood that the transformation will have safety consequences *Medium:* there is moderate likelihood that the transformation will have safety consequences *Low:* there is low possibility that the transformation could cause any danger *Very Low:* the transformation has no possibility of causing any danger
Transformation quality criteria	*Very High:* the transformation has a very high level of functionality, reliability and usability requirements *High:* the transformation has a high of functionality, reliability and usability requirements *Medium:* the transformation has a medium level of functionality, and usability requirements *Low:* there are low reliability, etc requirements *Very Low:* there are very low levels of reliability, etc requirements
Time constraint	*Very High:* the transformation has a very high level of efficiency, timing requirements *High:* the transformation has a high level of timing and efficiency requirements *Medium:* the transformation has a medium level of timing and efficiency requirements *Low:* there are low timing requirements *Very Low:* the transformation has no timing requirements
Cost constraint	*Very High:* the budget is very tight *High:* the budget is tight *Medium:* the transformation has a limited budget *Low:* the transformation has the budget to cover different aspects and unforeseen circumstances *Very Low:* the budgets are flexible
Understanding of domain	*Very High:* developers have a good background knowledge and previous experience regarding the domain *High:* there is a good amount of knowledge and experience regarding the domain *Medium:* there are some background knowledge and experience regarding the domain *Low:* the amount of experience and knowledge regarding the domain is low *Very Low:* there are no experience or knowledge about the domain

TABLE V. REQUIREMENTS ELICITATION TECHNIQUES EVALUATION

Attribute	*Interview*	*Question-naire*	*Document Mining*	*Brainstor-ming*	*Proto-types*	*Scenarios*	*Ethno Methodology*
Eliciting MT requirements	1	0.8	1	0.8	1	1	0.8
Facilitating communication	1	1	0	0.8	0.8	1	0.6
Understanding social issues	0.8	1	0.8	0.4	0.2	0.6	0.8
Getting domain knowledge	0.6	0.6	1	1	0.4	0.4	1
Getting implicit knowledge	0.2	0.2	0.2	0.2	0.2	0.2	1
identifying MT stakeholders	1	0.8	0.2	1	0	0.4	0.6
Identifying non-functional requirements	1	0.6	0.8	1	0.2	0.2	0.4
Identifying viewpoints	0.8	0.6	0.4	0.8	0	0.8	0.4

TABLE VI. REQUIREMENTS NEGOTIATION TECHNIQUES EVALUATION

Attribute	*Proto-types*	*Scen-arios*	*UML*	*Goal-oriented Analysis*	*Functional Decom-position*
Modelling MT requirements	0.8	1	1	0.8	0.6
Analysing requirements with relevant MT notations	0.6	1	0.8	0.8	0.8
Analysing non-functional requirements	0.2	0.2	0	0.6	0.2
Facilitate negotiation with stakeholders	0.8	0.6	0.8	0.8	0.4
Prioritizing requirements based on stakeholders	0.2	0.4	0	0.4	0.2
Identifying accessibility of the transformation	0.8	0.8	0.6	0.6	0.2
Modeling interface requirements	0.6	1	1	0.4	0.2
Re-usability of MT requirements	0	0	1	0.2	0

It was identified that a suitable overall architecture for the transformation was a sequential decomposition of a model-to-model transformation *design2C*, and a model-to-text transformation *genCtext*. Decomposing the code generator into two sub-transformations improves its modularity, and simplifies the constraints, which would otherwise need to combine language translation and text production. Figure 3 shows the resulting transformation architecture.

After a further interview, the application of model-based testing and bx to achieve F3 was identified as an important area of work. Tests for the synthesised C code should, ideally, be automatically generated based on the source UML model. The bx property can be utilised for testing semantic equivalence by transforming UML to C, applying the reverse transformation, and comparing to identify if the two UML models are isomorphic.

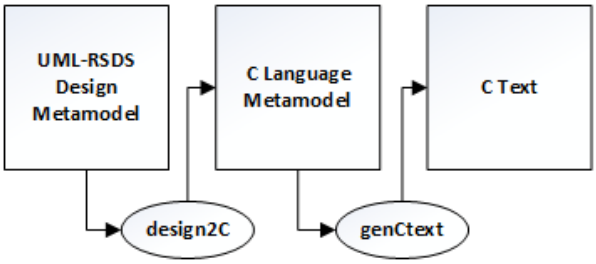

Figure 3. C code generator architecture

Evaluating four RE techniques according to the technique attribute measures RA(t) gives the following results for the UML to C case study for the elicitation stage (Table IX).

The framework interestingly reveals that there is no particular technique that helps strongly in prioritising requirements. A further technique appropriate for this project need should be selected. Evaluating a further three elicitation techniques gives Table X.

The overall ranking of techniques according to technique attributes is therefore: (i) *Questionnaire*; (ii) *Brainstorming*; (iii) *Mining*; (iv) *Interviews*.

Furthermore, we need to weight the techniques by a factor PD(t) representing the suitability of the technique in the context of the particular project environment. For example, techniques that depend on close customer collaboration are not favoured if the project customer relationship is low. In addition, a factor E(t) representing the experience in the technique in the development team or organisation is included. If, instead of rejecting a technique such as brainstorming because of lack of experience in it, in favour of introducing the technique, then the learning curve and cost of introduction need to be considered.. According to [23], this is relatively small for brainstorming. Table XI shows the overall evaluation for elicitation techniques for the case study.

Table XII shows the evaluation of techniques for the eval-

TABLE VII. REQUIREMENTS SPECIFICATION TECHNIQUES EVALUATION

Attribute	*SysML*	*KAOS*	*Structured language template*	*SADT*	*Evolutionary Prototypes*	*UML*
Representing MT requirements	0.8	0.8	0.8	0.6	0.8	1
Requirements verification	1	1	0	0.4	0.8	0.8
Semantics completeness	0.8	1	0.4	0.6	0.2	0.8
Representing requirements using MT notations	0.6	0.4	0.4	0.4	0.2	1
Writing complete requirements	0.8	0.8	0.6	0.6	0.4	0.8
Requirements management	0.8	0.4	0.6	1	0	0.8
Designing highly modular systems	0.8	0.6	0	0	0	0.8
Implementability of the notation(s)	1	1	0	0	0.8	0.8

TABLE VIII. REQUIREMENTS VALIDATION TECHNIQUES EVALUATION

Attribute	*Inspection*	*Desk-Checks*	*Rapid Prototyping*	*Checklist*
Identifying ambiguous requirements	0.4	0	0.4	0
Identifying inconsistency and conflict	0.4	1	0.8	1
Identifying incomplete requirements	0.8	0.8	0.8	0.8

TABLE IX. ELICITATION TECHNIQUE EVALUATION FOR UML TO C CASE (1)

Attribute	*Brainstorming*	*Interviews*	*Mining*	*Scenarios*
Elicit domain knowledge	1	0.6	1	0.4
Identify non-functional requirements	1	1	0.8	0.2
Requirements prioritisation	0	0	0	0.4
Totals RA(t):	2	1.6	1.8	1

TABLE X. ELICITATION TECHNIQUE EVALUATION FOR UML TO C CASE (2)

Attribute	*Questionnaire*	*Prototypes*	*Observation*
Elicit domain knowledge	0.6	0.4	1
Identify non-functional requirements	0.6	0.2	0.4
Requirements prioritisation	1	0.2	0
Totals RA(t):	2.2	0.8	1.4

TABLE XI. ELICITATION TECHNIQUE SELECTION FOR UML TO C CASE

Measure	*Brainstorming*	*Interviews*	*Mining*	*Scenarios*
RA(t)	2	1.6	1.8	1
PD(t)	X	0.66	0.68	0.66
E(t)	0	1	0.6	1
Totals S(t):	0	1.056	0.734	0.66

uation and negotiation stage.

Specific to MT is the ability to represent local and global functional requirements. The overall ranking of techniques is then: (i) UML; (ii) Prototypes; (iii) Scenarios. As with elicitation, factors PD(t) and E(t) need also to be considered to give an overall selection.

The most appropriate specification and documentation techniques are shown in Table XIII. The overall ranking for techniques is: (i) SysML; (ii) UML; (iii) Natural language. Figure 4 shows part of the requirements refinement and goal decomposition using SysML.

Validation and verification techniques are shown in Table XIV. The overall ranking for techniques is: (i) SysML; (ii) UML; (iii) Prototypes.

In the following subsections we present the application of the selected RE techniques on the case study.

A. F1.1: Type Translation

This iteration was divided into three phases: detailed requirements analysis; specification; testing. Detailed requirements elicitation used structured interviews to identify (i) the source language; (ii) the mapping requirements; (iii) the target language; (iv) other functional and non-functional requirements for this sub-transformation. Scenarios and test cases were prepared.

Using goal decomposition, the requirements were decomposed into specific mapping requirements, these are the local functional requirements F1.1.1 to F1.1.4 in Figure 4. Table XV shows the informal scenarios for these local mapping requirements, based on the concrete metaclasses of Type and the different cases of instances of these metaclasses. The schematic concrete grammar is shown for the C elements representing the UML concepts. As a result of requirements evaluation and negotiation with the principal stakeholder, using exploratory prototyping, it was determined that all these local requirements are of high priority except for the mapping F1.1.2 of enumerations (medium priority). The justification for this is that enumerations are not an essential UML language element. Bidirectionality was considered a high priority for this sub-transformation. It was identified that to meet this requirement, all source model Property elements must have a defined type, and specifically that elements representing many-valued association ends must have some CollectionType representing

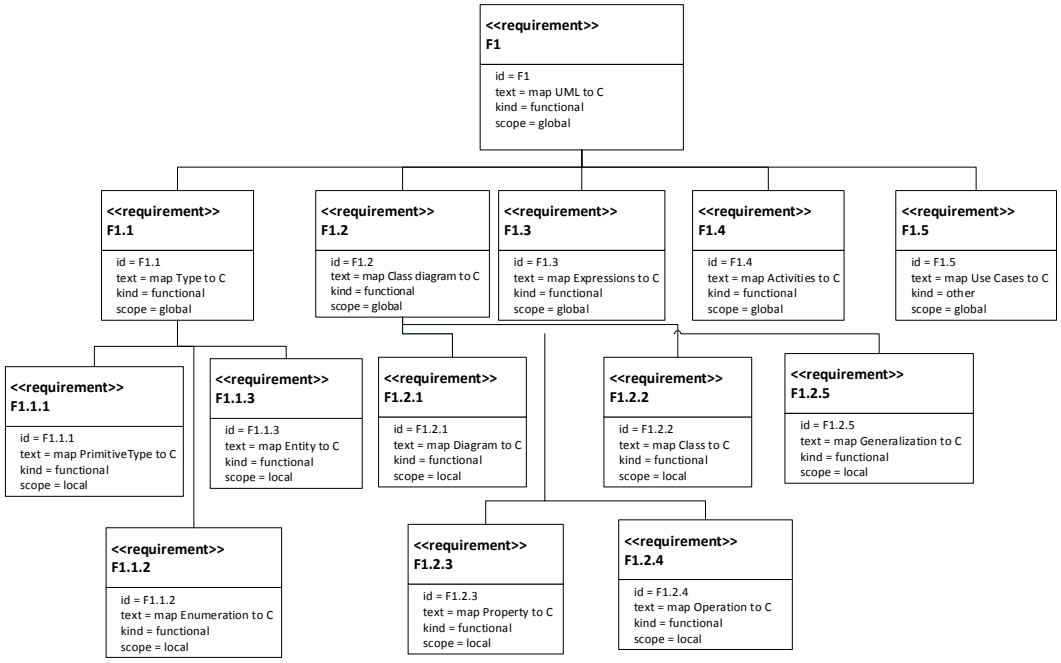

Figure 4. Functional requirements decomposition in SysML

TABLE XII. EVALUATION/NEGOTIATION TECHNIQUE EVALUATION FOR UML TO C CASE

Attribute	*Prototypes*	*State machines*	*UML*	*Scenarios*
Represent MT requirements	0.8	0.4	1	1
Identify incomplete requirements	0.8	0	0.4	0.2
Identify ambiguous requirements	0.4	0.6	0.6	0.4
Facilitate negotiation	0.8	0.4	0.8	0.4
Totals RA(t):	2.8	1.4	2.8	2

TABLE XIII. SPECIFICATION/DOCUMENTATION TECHNIQUE EVALUATION FOR UML TO C CASE

Attribute	*Natural language*	*UML*	*SysML*
Write unambiguous and precise specication	0.6	0.8	1
Write complete requirements	0.6	0.8	0.8
Modularity	0	0.8	0.8
Totals RA(t):	1.2	2.4	2.6

TABLE XIV. VALIDATION AND VERIFICATION TECHNIQUE EVALUATION FOR UML TO C CASE

Attribute	*Prototypes*	*Scenarios*	*UML*	*SysML*
Implementability/ executability	0.8	0.4	0.8	1
Requirements verification	0.8	0.6	0.8	1
Notation	0.2	0.6	1	0.6
Totals RA(t):	1.8	1.6	2.6	2.6

TABLE XV. INFORMAL SCENARIOS FOR TYPES2C

Scenario	*UML element*	*C representation e'*
F1.1.1.1	*String type*	*char**
F1.1.1.2	*int, long, double types,*	*same-named C types*
F1.1.1.3	*boolean type*	*unsigned char*
F1.1.2	*Enumeration type*	*C enum*
F1.1.3	*Entity type E*	*struct E* type*
F1.1.4.1	*Set(E) type*	*struct E** (array of E, without duplicates)*
F1.1.4.2	*Sequence(E) type*	*struct E** (array of E, possibly with duplicates)*

their actual type. A limitation of the proposed mapping is that mapping collections of primitive values (integers, doubles, booleans) to C is not possible, because there is no means to identify the end of the collection in C (NULL is used as the terminator for collections of objects and collections of strings).

B. F1.2: Translation of Class Diagram

This iteration also used a three-phase approach, to define a subtransformation classdiagram2C. The class diagram elements *Property, Operation, Entity, Generalization* were identified as the input language. Exploratory prototyping was used for requirements elicitation and evaluation. During re-

TABLE XVI. INFORMAL SCENARIOS FOR THE MAPPING OF UML CLASS DIAGRAMS TO C

Scenario	*UML element* e	*C representation e'*
F1.2.1	Class diagram D	C program with D's name
F1.2.2	Class E	struct E {...}; Global variable struct E** e_instances; Global variable int e_size; struct E* createE() operation struct E** newEList() operation
F1.2.3.1	Property p : T (not principal identity attribute)	Member T' p; of the struct for p's owner, where T' represents T Operations T' getE_p(E' self) and setE_p(E' self, T' px)
F1.2.3.2	Principal identity attribute p : String of class E	Operation struct E* getEByPK(char* v) Key member char* p; of the struct for E
F1.2.4	Operation op(p : P) : T of E	C operation T' op(E' self, P' p) with scope = entity
F1.2.5	Inheritance of A by B	Member struct A* super; of struct B

quirements evaluation and negotiation it was agreed that the metafeatures *isStatic, isReadOnly, isDerived, isCached* would not be represented in C, nor would *addOnly, aggregation, constraint* or *linkedClass*. This means that aggregations, association classes and static/constant features are not specifically represented in C. Interfaces are not represented. Only single inheritance is represented.

The scenarios of the local mapping requirements for class diagram elements are shown in Table XVI.

C. F1.3: Translation of OCL Expressions

In this iteration, the detailed requirements for mapping OCL expressions to C are identified, then this subtransformation, expressions2C, is specified and tested. There are many cases to consider in the mapping requirements, so we divided these into four subgroups: (i) mapping of basic expressions; (ii) mapping of logical expressions; (iii) mapping of comparitor, numeric and string expressions; (iv) mapping of collection expressions. These were considered the natural groupings of operations and operators, and these follow in part the metaclass organisation of UML expressions.

1) Basic Expressions: The basic expressions of OCL generally map directly to corresponding C basic expressions. Table XVII shows the mapping for these. These mapping requirements are grouped together as requirement F1.3.1.

2) Logical Expressions: Table XVIII shows the mapping of logical expressions and operators to C. These mappings are grouped together as requirement F1.3.2.

3) Comparitor, Numeric and String Expressions: Table XIX lists the comparitor operators and their mappings to C. These mappings are grouped as requirement 1.3.3. Numeric operators for integers and real numbers are shown in Table XX. The types int, double and long are not guaranteed to have particular sizes in C. All operators take double values as arguments except mod and Integer.subrange, which have int parameters.

TABLE XVII. MAPPING SCENARIOS FOR BASIC EXPRESSIONS

OCL expression e	***C representation e'***
self	*self* as an operation parameter
Variable *v* or *v[ind]*	v *v[ind - 1]*
Data feature *f* with no objectRef	*self* $\rightarrow$ *f*
Data feature *f* of instance *ex*	*ex'* $\rightarrow$ *f*
Operation call *op(e1,...,en)* or *obj.op(e1,...,en)*	op(self, e1', ..., en') op(obj', e1', ..., en')
Attribute *f* of collection *exs*	get*AllE_f (exs')* (duplicate values preserved)
Single-valued role r : F of collection exs	*getAllE_r(exs')* defined by (*struct* F $**$) *collectE(exs', getE_r)*
col [ind] ordered collection col	(col')[ind-1]
E[v] *v* single-valued	getEByPK(v')
E[*vs*] *vs* collection-valued	getEByPKs(vs')
E.allInstances	*e_instances*
value of enumerated type, numeric or string value	*value*
boolean true, false	TRUE, FALSE

TABLE XVIII. MAPPING SCENARIOS FOR LOGICAL EXPRESSIONS

OCL expression e	***C representation e'***
A =>B	!A' || B'
A & B	A' && B'
A or B	A' || B'
not(A)	!A'
E->exists(P)	existsE(e_instances,fP) fP evaluates P
e->exists(P)	existsE(e',fP)
E->exists1(P)	exists1E(e_instances,fP) fP evaluates P
e->exists1(P)	exists1E(e',fP)
E->forAll(P)	forAllE(e instances,fP) fP evaluates P
e->forAll(P)	forAllE(e',fP)

Other math operators directly available in C are: log10, tanh, cosh, sinh, asin, acos, atan. These are double-valued functions of double-valued arguments. cbrt is missing and needs to be implemented as pow(x, 1.0/3).

4) Collection Expressions: Table XXII shows the values and operators that apply to sets and sequences, and their C translations. Some operators (unionAll, intersectAll, symmetricDifference, subcollections) were considered a low priority, because these are infrequently used, and were not translated. The requirements are grouped as F1.3.6. After evaluation and negotiation, it was decided that full implementation of *delete* should be deferred, because of the complex semantics of data deletion in C. In addition, prototyping revealed that compiler differences made the use of qsort impractical, and instead a custom sorting algorithm, *treesort*, was implemented. This has the signature (*void** treesort(void* col[], int (*comp)(void*, void*)*) and the translation of *x→sort()* is then: (rt) treesort((void$**$) x', comp) for the appropriate result type *rt* and comparitor function *comp*. Table XXI shows the translation of select and collect operators. These mappings are grouped as requirement F1.3.7.

TABLE XIX. MAPPING SCENARIOS FOR COMPARITOR EXPRESSIONS

OCL expression e	*C representation e'*
x : E E entity type	isIn((void* x', (void **) e instances)
x : s s collection	isIn((*void**) x', (void **) s')
s->includes(x) s collection	Same as x : s
x /: E E entity type	!isIn((void*) x', (void **) e instances)
x /: s s collection	!isIn((void*) x', (void **) s')
s->excludes(x) s collection	Same as x /: s
x = y Numerics, Booleans	x'== y'
Strings	strcmp(x', y') == 0
Objects	x'== y'
Sets	equalsSet((void **) x', (void **) y')
Sequences	equalsSequence((void **) x', (void **) y')
x <y Numerics	x'<y'
Strings	strcmp(x', y') <0
Similarly for >, <=, >=, /=	>, <=, >=, ! =
s <: t s, t collections	containsAll ((void **) t', (void **) s')
t->includesAll(s)	Same as s <: t
t->excludesAll(s)	disjoint((void**) t', (void**) s')

TABLE XX. MAPPING SCENARIOS FOR NUMERIC EXPRESSIONS

OCL expression e	*Representation in C*
-x	-x'
x + y numbers	x' + y'
x - y	x' - y'
x* y	x' * y'
x / y	x' / y'
x mod y	x' % y'
x.sqr	(x' * x')
x.sqrt	sqrt(x')
x.floor	oclFloor(x') defined as: ((int) floor(x'))
x.round	oclRound(x')
x.ceil	oclCeil(x') defined as: ((int) ceil(x'))
x.abs	fabs(x')
x.exp	exp(x')
x.log	log(x')
x.pow(y)	pow(x',y')
x.sin, x.cos, x.tan	sin(x'), cos(x'), tan(x')
Integer.subrange(st,en)	intSubrange(st',en')

Unlike the types and class diagram mappings, a recursive descent style of specification is needed for the expressions mapping (and for activities). This is because the subordinate parts of an expression are themselves expressions. Thus it is not possible in general to map all the subordinate parts of an expression by prior rules: even for basic expressions, the parameters may be general expressions. In contrast, the element types of collection types cannot themselves be collection types or involve subparts that are collection types, so it is possible to map all element types before considering collection types. A recursive descent style of mapping specification uses operations of each source entity type to map instances of that type, invoking mapping operations recursively to map subparts of the instances.

TABLE XXI. SCENARIOS FOR THE MAPPING OF SELECTION AND COLLECTION EXPRESSIONS

UML expression e	C translation e'
s->select(P)	selectE(s', fP) fP evaluates P
s->select(x | P)	as above
s->reject(P)	rejectE(s', fP)
s->reject(x | P)	as above
s->collect(e)	(et'*) collectE(s', fe)
e of type et	fe evaluates e'
s->collect(x | e)	as above
s->selectMaximals(e)	-
s->selectMinimals(e)	-

D. Activities Translation

In this iteration, UML-RSDS activities are mapped to C statements by a subtransformation statements2C. UML-RSDS statements correspond closely to those of C. Table XXIII shows the main cases of the mapping of UML activities to C statements.

E. Use case Translation

In this iteration, the mapping usecases2C of use cases is specified and implemented. A large part of this iteration was also taken up with integration testing of the complete transformation.

F1.5.1: A use case uc is mapped to a C operation with *application* scope, and with parameters corresponding to those of uc. Its code is given by the C translation of the activity classifierBehaviour of uc.

F1.5.2: Included use cases are also mapped to operations, and invoked from the including use case.

F1.5.3: Operation activities are mapped to C code for the corresponding COperation.

F1.5.1 is formalised as:

```
UseCase::
COperation->exists( cop | cop.name = name &
cop.scope = "application" &
cop.isQuery = false &
cop.code = classifierBehaviour.mapStatement() &
cop.parameters = parameters.mapExpression() &
cop.returnType = CType[returnType.typeId] )
```

Similarly for the activities of UML operations.

This case study is the largest transformation, which has been developed using UML-RSDS, in terms of the number of rules (over 150 rules/operations in 5 subtransformations). By using a systematic requirements engineering and agile development approach, we were able to effectively modularise the transformation and to organise its structure and manage its

TABLE XXII. SCENARIOS FOR THE TRANSLATION OF COLLECTION OPERATORS

Expression e	*C translation e'*
Set{}	newEList()
Sequence{}	newEList()
Set{x1, x2, ... , xn}	insertE(... insertE(newEList(), x1'), ..., xn')
Sequence{x1, x2, ..., xn}	appendE(... appendE(newEList(), x1'), ..., xn')
s->size()	length((void∗∗) s')
s->including(x)	insertE(s',x') or appendE(s',x')
s->excluding(x)	removeE(s',x')
s - t	removeAllE(s',t')
s->prepend(x)	-
s->append(x)	appendE(s',x')
s->count(x)	count((void∗) x', (void∗∗) s')
s->indexOf(x)	indexOf((void*) x', (void**) s')
x∨y	unionE(x',y')
x∧y	intersectionE(x',y')
x⌢ y	concatenateE(x',y)
x->union(y)	unionE(x',y')
x->intersection(y)	intersectionE(x', y')
x->unionAll(e)	-
x->intersectAll(e)	-
x->symmetricDifference(y)	-
x->any()	x'[0]
x->subcollections()	-
x->reverse()	reverseE(x')
x->front()	subrangeE(x',1,length((void**) x')-1)
x->tail()	subrangeE(x',2,length((void**) x'))
x->first()	x'[0]
x->last()	x'[length((void∗∗) x')-1]
x->sort()	qsort((void∗∗) x', length((void∗∗) x'), sizeof(struct E∗), compareToE)
x->sortedBy(e)	qsort((void∗∗) x', length((void∗∗) x'), sizeof(struct E∗), compare) compare defines e-order
x->sum()	sumString(x'), sumint(x'), sumlong(x'), sumdouble(x')
x->prd()	prdint(x'), prdlong(x'), prddouble(x')
Integer.Sum(a,b,x,e)	sumInt(a',b',fe), sumDouble(a',b',fe) fe computes e'(x')
Integer.Prd(a,b,x,e)	prdInt(a',b',fe), prdDouble(a',b',fe)
x->max()	maxInt(x'), maxLong(x'), maxDouble(x'), maxString(x')
x->min()	minInt(x'), minLong(x'), minDouble(x'), minString(x')
x->asSet()	asSetE(x')
x->asSequence()	x'
s->isUnique(e)	isUniqueE(s',fe)
x->isDeleted()	killE(x')

TABLE XXIII. SCENARIOS FOR MAPPING OF STATEMENTS TO C

Requirement	*UML activity st*	*C statement st'*
F1.4.1	Creation statement x : T defaultT' is default value of T'	T' x = defaultT';
F1.4.2	Assign statement v := e	v' = e';
F1.4.3	Sequence statement st1 ; ... ; stn	st1' ... stn'
F1.4.4	Conditional statement if e then st1 else st2	if e' {st1'} else {st2'}
F1.4.5	Return statement return e	return e';
F1.4.6	Break statement break	break;
F1.4.7	Bounded loop for (x : e) do st on object collection e of entity element type E	int i = 0; for (; i <length((void**) e'); i++) { struct E* x = e'[i]; st' } New index variable i
F1.4.8	Unbounded loop while e do st	while (e') { st' }
F1.4.9	Operation call ex.op(pars)	op(ex',pars')

requirements. Despite the complexity of the transformation, it was possible to use patterns to enforce bx and other properties, and to effectively prove these properties.

VII. Conclusion and Future Work

We have identified ways in which requirements engineering can be applied systematically to model transformations. Comprehensive catalogues of functional and non-functional requirements categories for model transformations have been defined. We have examined a case study, which is typical of the current state of the art in transformation development, and identified how formal treatment of functional and non-functional requirements can benefit such developments. In this paper we have identified the need for a systematic requirements engineering process for model transformations. We have proposed such a process, and identified RE techniques that can be used in this process. Moreover, we have identified a requirements engineering process for model transformations, and requirements engineering techniques that can be used in this process. The process can be used to develop specifications in a range of declarative and hybrid MT languages. We have evaluated the process and techniques on a real industrial case study, UML to C translation, with positive results.

In future work, we will construct tool support for recording and tracing transformation requirements, which will help to ensure that developers systematically consider all necessary requirements and that these are all formalised, validated and verified correctly. We are currently carrying out research into improving the requirements engineering process in model transformation. We will investigate formal languages to express the requirements, as formalised rules can be checked for internal correctness properties, such as definedness and determinacy, which should hold for meaningful rules. Temporal logic can be used to define the specialised characteristics of particular transformation and to define transformation requirements in a formal but language-independent manner languages as model transformation systems necessarily involve a notion of time. Finally, we will be evaluating large case studies in order to compare results with and without RE process.

Appendix

The procedure for selecting RE techniques for a MT project in more detail involves:

1) The set $\mathtt{T}$ of all suitable RE techniques (e.g. interview, prototype) in each category (i.e. elicitation, negotiation, specification, verification) is identified.
2) For each requirement identified within the project, each RE technique $\mathtt{t} \in \mathtt{T}$ is assigned a value $\mathtt{RA}(t)$ (for

Requirement Attribute) representing the suitability of applying t to fulfil this requirement, based on the requirement's attributes. The function $\mathtt{RA} : \mathtt{T} \longmapsto [0,1]$ is defined as:

$$\mathtt{RA}(t) = \frac{\sum_{a_x \in \mathtt{A}} \mathtt{I}(a_x) \times \mathtt{V}(a_x, \mathtt{t})}{|\mathtt{A}|}$$

where:

- The set of all technique attributes (e.g. facilitating communication, identifying MT stakeholders) is A (Table III). For instance, $\mathtt{A} = \{$Eliciting MT requirements, facilitating communication, ..., identifying incomplete requirements$\}$.
- $\mathtt{I}(a_x)$ which is in $[0,1]$, represents the importance of an attribute $a_x \in \mathtt{A}$ for a specific requirement of the project. A low $\mathtt{I}(a_x)$ value for an attribute $a_x \in \mathtt{A}$ means a_x is not important for the requirement of the MT project and a high value represents high importance. The assignment of $\mathtt{I}(a_x)$ to each $a_x \in \mathtt{A}$ is done by MT developers.
- $\mathtt{V}(a_x, t)$ is a function $\mathtt{V} : \mathtt{T} \times \mathtt{A} \longmapsto [0,1]$ which given a technique attribute and an RE technique, assigns a $[0,1]$ value. These values are based on the technique attribute measures of [23] as well as others that are identified in this research. Tables (V, VI, VII, VIII) give examples of these adapted attribute measures.

3) For each requirement identified within the project, each RE technique $\mathtt{t} \in \mathtt{T}$ is assigned a value $\mathtt{PD}(\mathtt{t})$ (for Project Description) representing the suitability of applying t to fulfil this requirement, based on the project's descriptions. The function $\mathtt{PD} : \mathtt{T} \longmapsto [0,1]$ is defined as:

$$\mathtt{PD}(\mathtt{t}) = \prod_{\mathtt{d}_x \in \mathtt{D}} \begin{cases} 1 - \mathtt{W}(\mathtt{d}_x) & \text{if } \mathtt{d}_x \in \mathtt{ID}_\mathtt{t} \\ \mathtt{W}(\mathtt{d}_x) & \text{otherwise} \end{cases}$$

where:

- The set of all project descriptors (e.g. size, complexity) is D. For instance, in this thesis, we are considering $\mathtt{D} = \{$size, complexity, volatility, relationship, safety, quality, time, cost, domain understanding$\}$.
- $\mathtt{W}(d_x)$ is a function $\mathtt{W} : \mathtt{D} \longmapsto [0,1]$ which represents the magnitude of a specific descriptor in the project. For example, for $\mathtt{d} =$ cost, a high value represents that the budget of the project is tight, while a low value indicates that the budget is flexible. Then for $\mathtt{d} =$ size, a high value means that the project involves a large number of transformation rules while a low value indicates a small number of rules involved.
- $\mathtt{ID}_\mathtt{t} \subseteq \mathtt{D}$ is a set containing all descriptors with inverse impact for a specific RE technique t. More specifically, for each $\mathtt{d} \in \mathtt{ID}_\mathtt{t}$, the higher the value of $\mathtt{W}(d_x)$ the more negative the impact of applying t in that project. An example of such a descriptor for technique "interview" is time, where the higher the value of W(time) in a specific project, the more negative the effectiveness of interviewing as a technique to fulfil a requirement in this project.

4) Evaluating the degree E (for Experience) of experience/-expertise regarding the RE technique t available in the development team. $\mathtt{E} : \mathtt{T} \longmapsto [0,1]$ is a function where $\mathtt{E}(\mathtt{t})$ represents the level of experience and practical and theoretical knowledge of the developer regarding t.
5) Using $\mathtt{S}(\mathtt{t})$, the overall suitability score of a particular RE technique ($\mathtt{t} \in \mathtt{T}$) can be calculated, and hence it would be possible to define a ranking of techniques t based on their suitability scores $\mathtt{S}(\mathtt{t})$ for use in the project. $\mathtt{S}(\mathtt{t})$ is defined in terms of the requirement attribute score $\mathtt{RA}(\mathtt{t})$, the project description score $\mathtt{PD}(\mathtt{t})$, and the experience score $\mathtt{E}(\mathtt{t})$ of RE technique t as follows:

$$\mathtt{S}(\mathtt{t}) = \mathtt{RA}(\mathtt{t}) \times \mathtt{PD}(\mathtt{t}) \times \mathtt{E}(\mathtt{t})$$

REFERENCES

[1] S. Yassipour Tehrani and K. Lano, "Model transformation applications from requirements engineering perspective," in The 10th International Conference on Software Engineering Advances, 2015.
[2] I. Sommerville and G. Kotonya, Requirements engineering: processes and techniques. John Wiley & Sons, Inc., 1998.
[3] E. Guerra, J. De Lara, D. S. Kolovos, R. F. Paige, and O. M. dos Santos, "transml: A family of languages to model model transformations," in Model Driven Engineering Languages and Systems. Springer, 2010, pp. 106–120.
[4] B. Selic, "What will it take? a view on adoption of model-based methods in practice," Software & Systems Modeling, vol. 11, no. 4, 2012, pp. 513–526.
[5] K. Lano and S. Kolahdouz-Rahimi, "Model-driven development of model transformations," in Theory and practice of model transformations. Springer, 2011, pp. 47–61.
[6] K. Lano and S. Rahimi, "Constraint-based specification of model transformations," Journal of Systems and Software, vol. 86, no. 2, 2013, pp. 412–436.
[7] S. Friedenthal, A. Moore, and R. Steiner, A practical guide to SysML: the systems modeling language. Morgan Kaufmann, 2014.
[8] T. Yue, L. C. Briand, and Y. Labiche, "A systematic review of transformation approaches between user requirements and analysis models," Requirements Engineering, vol. 16, no. 2, 2011, pp. 75–99.
[9] S. Y. Tehrani, S. Zschaler, and K. Lano, "Requirements engineering in model-transformation development: An interview-based study," in International Conference on Theory and Practice of Model Transformations. Springer, 2016, pp. 123–137.
[10] S. Kolahdouz-Rahimi, K. Lano, S. Pillay, J. Troya, and P. Van Gorp, "Evaluation of model transformation approaches for model refactoring," Science of Computer Programming, vol. 85, 2014, pp. 5–40.
[11] P. Van Gorp and L. M. Rose, "The petri-nets to statecharts transformation case," arXiv preprint arXiv:1312.0342, 2013.
[12] I. Iso, "Iec 25010: 2011,," Systems and Software EngineeringSystems and Software Quality Requirements and Evaluation (SQuaRE)System and Software Quality Models, 2011.
[13] K. Lano, S. Kolahdouz-Rahimi, and T. Clark, "Comparing verification techniques for model transformations," in Proceedings of the Workshop on Model-Driven Engineering, Verification and Validation. ACM, 2012, pp. 23–28.
[14] I. O. F. S. E. Commission et al., "Software engineering–product quality–part 1: Quality model," ISO/IEC, vol. 9126, 2001, p. 2001.
[15] K. Anastasakis, B. Bordbar, and J. M. Küster, "Analysis of model transformations via alloy," in Proceedings of the 4th MoDeVVa workshop Model-Driven Engineering, Verification and Validation, 2007, pp. 47–56.

[16] L. de Moura and N. Bjørner, "Z3–a tutorial," 2006.
[17] S. Yassipour Tehrani and K. Lano, "Temporal logic specification and analysis for model transformations," in Verification of Model Transformations, VOLT 2015, 2015.
[18] A. M. Hickey and A. M. Davis, "Requirements elicitation and elicitation technique selection: model for two knowledge-intensive software development processes," in System Sciences, 2003. Proceedings of the 36th Annual Hawaii International Conference on. IEEE, 2003, pp. 10–pp.
[19] N. Maiden and G. Rugg, "Acre: selecting methods for requirements acquisition," Software Engineering Journal, vol. 11, no. 3, 1996, pp. 183–192.
[20] L. Jiang, A. Eberlein, B. H. Far, and M. Mousavi, "A methodology for the selection of requirements engineering techniques," Software & Systems Modeling, vol. 7, no. 3, 2008, pp. 303–328.
[21] L. Macaulay, "Requirements for requirements engineering techniques," in Requirements Engineering, 1996., Proceedings of the Second International Conference on. IEEE, 1996, pp. 157–164.
[22] K. Lano, "The uml-rsds manual," 2014.
[23] L. Jiang, A framework for the requirements engineering process development. University of Calgary, 2005.