Requirements Traceability in Model-Driven Development: Applying Model and Transformation Conformance

João Paulo A. Almeida[1,3], Maria-Eugenia Iacob[2], Pascal van Eck[3]

[1]*Computer Science Department, Federal University of Espírito Santo, Brazil*

[2]*Department of Information Systems & Change Management,*

University of Twente, The Netherlands

[3]*Centre for Telematics and Information Technology, University of Twente, The Netherlands*

jpalmeida@ieee.org, m.e.iacob@utwente.nl, p.vaneck@utwente.nl

Abstract

The variety of design artifacts (models) produced in a model-driven design process results in an intricate relationship between requirements and the various models. This paper proposes a methodological framework that simplifies management of this relationship, which helps in assessing the quality of models, realizations and transformation specifications. Our framework is a basis for understanding requirements traceability in model-driven development, as well as for the design of tools that support requirements traceability in model-driven development processes. We propose a notion of conformance between application models which reduces the effort needed for assessment activities. We discuss how this notion of conformance can be integrated with model transformations.

Keywords: requirements traceability, assessment, conformance, model transformation, model-driven design

1. Introduction

Model-driven design holds the promise of improving application development significantly by capturing design steps in explicit model transformations (Object Management Group, 2003a). The design of an application in model-driven design can be seen as the process of building a realization of the application specification that satisfies all application requirements stated in the specification by applying appropriate transformations.

At several stages in the application lifecycle, application maintainers need to know which application models and/or components satisfy requirements that have been explicitly stated. This relation between requirements and elements of the solution (e.g., application models and components) is called requirements traceability. Requirements traceability is for instance used during acceptance testing, when application users (or procurers) are interested in assessing the extent to which an application adheres to its requirements.

We observe that, in a model-driven design process, the great variety of modelling artifacts pose challenges to requirements traceability and assessment. Not only application realizations have to be assessed for requirements satisfaction, but also application models, metamodels and model transformation specifications since these may also be considered products of the model-driven design process.

The main contribution of this paper is to provide a methodological framework which allows designers to relate requirements to the various products of the model-driven design process. This framework is a basis for tracing requirements and assessing the quality of model transformation specifications, models and realizations. Furthermore, we propose a requirements traceability metamodel that serves two purposes: it models the main elements of the above-mentioned framework, and serves as basis for the design of requirements tracing tool support.

Since the model-driven design process may consist of different levels of abstraction (and platform-independence, see Almeida, van Sinderen, Ferreira Pires & Quartel, 2003), requirements are traced throughout these levels. We propose a notion of conformance between models which simplifies requirements tracing. The idea is that transformations which are assumed to produce conformant results can be reused, deeming some assessment activities redundant.

This paper is structured as follows. Section 2 provides some background in the area of requirements engineering. Section 3 defines the basic notions of model-driven design required in this paper. It defines the notion of satisfaction of requirements in terms of the relation between requirements, the various application models and realizations of an application. Section 4 defines and justifies the notion of conformance between models proposed here. Section 5 extends the view of the model-driven design process defined in section 3 by introducing model transformation chains. This allows us to discuss how conformant transformations can simplify assessment activities. Section 6 presents our requirements traceability metamodel, and defines the conditions that must apply to the results of the requirements traceability process. We define these conditions such that this methodological framework can be used with different model-driven development processes and practices. Section 7 illustrates our approach with an example. Section 8 discusses related work, and finally, section 9 presents our conclusions and outlines topics for further research.

2. Requirements Engineering

The term Requirements Engineering (RE) refers to the phase in application development in which requirements of different stakeholders are gathered and processed, in general resulting in a requirements specification or software specification. Requirements can be formulated as either

properties of the problem that the stakeholders want to solve using the application under development or desired properties of that application. This phase is called requirements engineering to indicate that more is needed than only requirements elicitation: requirements have to be processed to resolve conflicts, prioritized, and captured in a consistent requirements specification.

We assume in this paper that a requirement specification is verifiable (Firesmith, 2003; IEEE, 1998) i.e., given a realization, it is possible to conduct assessment activities to determine whether the requirements can be considered satisfied. We use the term "assessment activity" for the act of checking whether a requirement is satisfied. Examples of assessment activities are acceptance testing by end users, model checking or formal correctness proofs.

We conceptualize requirements as implicitly defining a set of application realizations that satisfy them. Figure 1 shows the relation between requirements and the space of possible realizations. An arbitrary grouping of the requirement specification into sets $RS_A \subset RS_B \subset RS_C$ is considered. The realization sets IS_1, IS_2, and IS_3 represent realizations that satisfy RS_1, RS_2, and RS_3 respectively. The realizations IS_C that satisfy the total set of requirements RS_C (the union of RS_1, RS_2, and RS_3) lie in the intersection between IS_1, IS_2, IS_3. Note that this is a conceptual notion, independent of whether requirements are formalized.

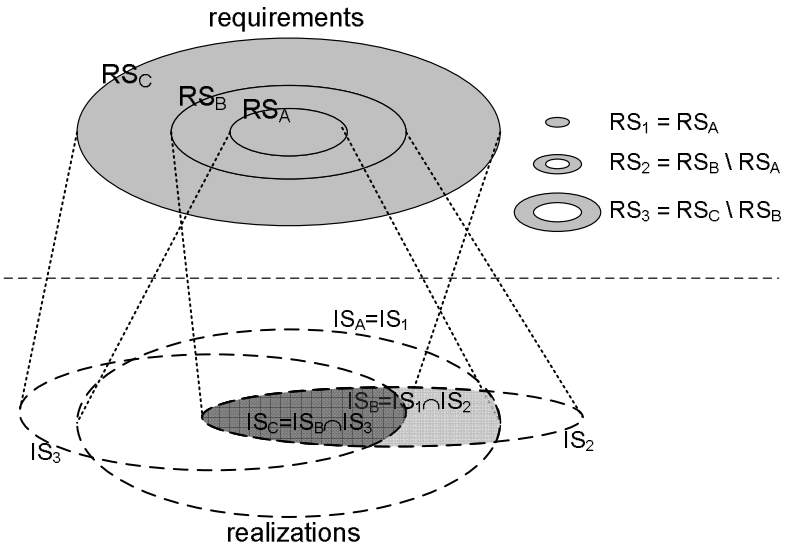

Figure 1 Requirements and realizations

In this paper we address *traceability of requirements.* Several definitions of traceability are presented by Gotel (1995). The one most suitable for the purpose of this paper is: “the means whereby software producers can ‘prove’ to their client that: the requirements have been understood; the product will fully comply with the requirements; and the product does not exhibit any unnecessary feature or functionality” (Wright, 1991, as quoted by Gotel,1995). Our notion of assessment activities exactly operationalises the notion of ‘prove’ in this definition. In terms of the IEEE Recommended Practice for Software Requirements Specifications (IEEE, 1998), we are interested in forward traceability, in which artifacts (in our case: models) constrained by the requirements specification need to be traced back to the requirements specification. In order to trace requirements throughout the design process, we partition the set of requirements into subsets as illustrated in Figure 1. The partitioning strategy is discussed in the remainder of this paper.

3. Requirements and Artifacts in Model-Driven Design

Before we explain how requirements are related to the several different artifacts in model-driven design, we need to guarantee some common understanding of the model-driven design process and of these artifacts.

3.1. Artifacts in Model-Driven Design

Model-driven design is based on capturing different aspects of a (distributed) application into symbolic artifacts known as *models*. Models are manipulated throughout the design process resulting ultimately in one or more realizations of the application. The manipulation of application models in a model-driven design process often entails model transformation activities (Schot, 1992) which may be determined or constrained by (*model*) *transformation specifications*. These specifications or their implementations may be executed automatically, with the purpose of improving the overall efficiency of the design process. In this paper, we consider that transformations are used to relate source and target models at different levels of abstraction. The notions of source and target models are thus relative to a design step. Models are expressed in suitable modelling languages, with their abstract syntax described in *metamodels*.

Model transformation specifications and metamodels are defined in an application-independent phase of the model-driven design process (known as the preparation phase in (Almeida, 2006; Gavras, Belaunde, Ferreira Pires & Almeida, 2004)). They are used by designers to build specific applications. In this context, model transformation specifications capture reusable design knowledge, and metamodels capture reusable concepts and patterns for application modelling.

Figure 2 shows an example of model-driven design trajectory, depicting schematically the dependencies between the various artifacts. Three levels of models are shown. In the lowest level of

models, two alternative application models are produced (M3 and M3'), which are defined in terms of different metamodels. Figure 2 includes model libraries of reusable models.

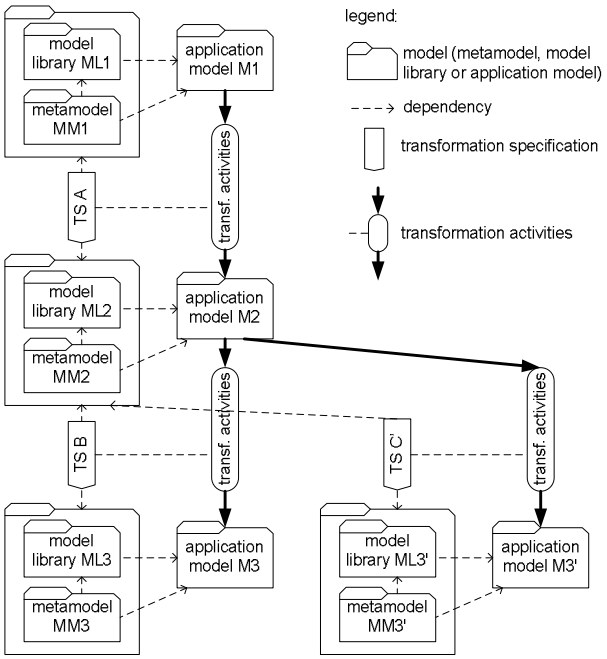

Figure 2 Artifacts in a model-driven design trajectory

3.2. Requirements and Application Models

The multitude of artifacts in model-driven design serves the ultimate purpose of producing application realizations that satisfy a particular set of requirements. Usually, there are (virtually infinitely) many application realizations that satisfy a set of requirements. The design task consists of obtaining a particular application realization that satisfies requirements while respecting implementation constraints and general design principles. Figure 3 illustrates the relation between requirements and application models at different levels of abstraction.

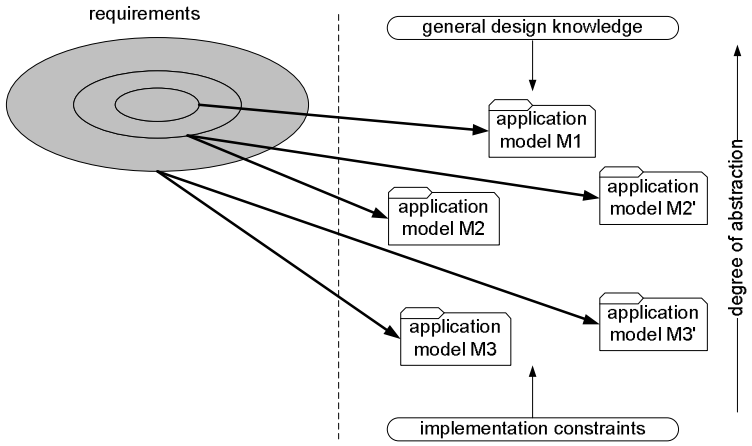

Figure 3 Requirements and application models

We assume that application models capture design decisions, defining characteristics of a potential application realization. Furthermore, we require that models have a well-defined semantics. More precisely, we say that a model has a well-defined semantics, if, and only if, given a realization and a model, it is possible to determine whether the realization exhibits the characteristics as defined in the model. The means by which this semantics is defined (e.g., mapping to a formal domain, natural language, or basic set of design concepts) is not prescribed by this definition.

We can conceptualize models as implicitly defining a set of realizations that realize them. Figure 4 (adapted from (Almeida, Dijkman, Ferreira Pires, Quartel & van Sinderen, 2006; Schot, 1992)) depicts the relation between models and the space of realizations. In this figure, an oval represents the sets of acceptable realizations for a particular model. Different design decisions may lead to alternative realizations, and this is shown by different sets of realizations (shaded) for alternative models (M2 and M2', M3 and M3').

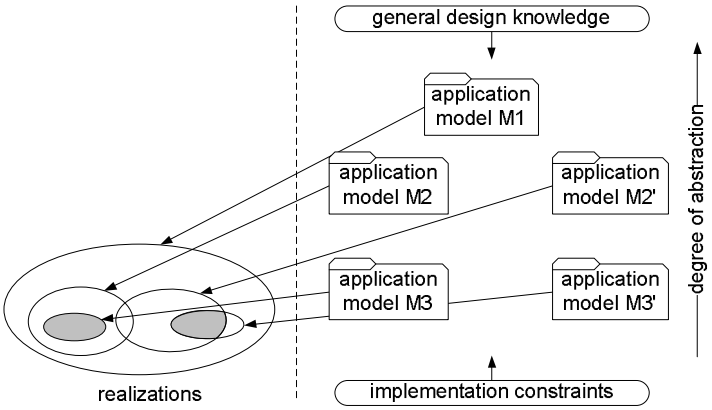

Figure 4 Models and the space of realizations

Design decisions should eventually lead to a design that defines all relevant characteristics of a realization of the system (Almeida, Dijkman, Ferreira Pires, Quartel & van Sinderen, 2006), satisfying all stated requirements and implementation constraints. It is not our intention to debate the distinction between realizations and models. For our purposes, a model that satisfies all requirements can be considered a realization. For example, a workflow model executed in an engine can be considered a realization, with no further transformation.

Figure 5 shows requirements, models and realizations in one picture (combining Figures 3 and 4). It reveals the (indirect) relation between requirements and realization. As can be observed in this figure, the set of realizations for an application model M1 is contained in the set of realizations that satisfy RS1. The set of realizations for an application model M3 is contained in the set of realizations that satisfy RS3. (Application models M2' and M3' are omitted for conciseness.)

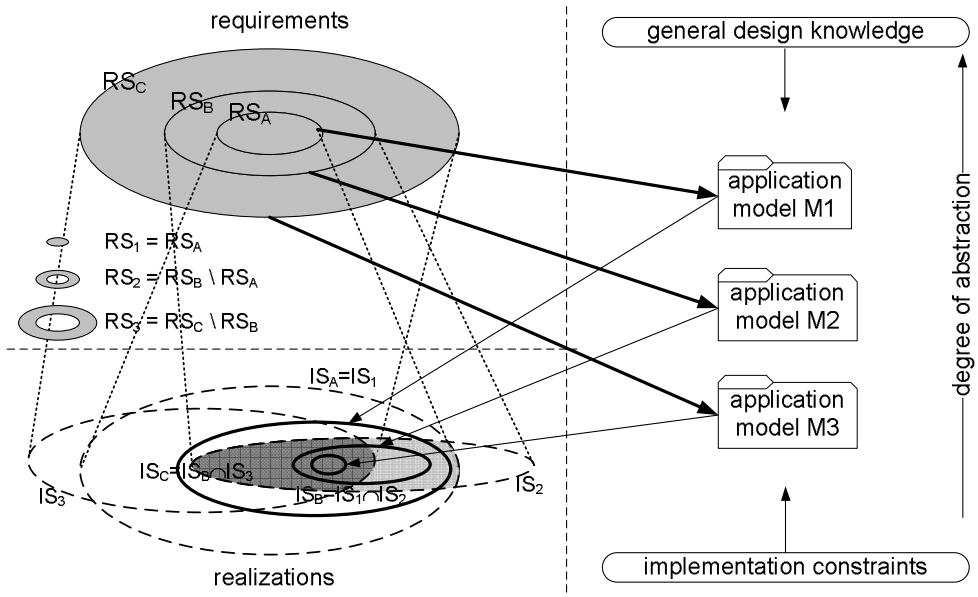

Figure 5 Requirements, models and realizations

At this point, we can formulate the notion of satisfaction of requirements by models. We say that a model M *satisfies a set of requirements* RS, if and only if, the set of acceptable realizations for M is contained in the set of realizations that satisfy RS. In order to support requirements traceability, it is the task of the designer to state which requirements are satisfied by which models, and to conduct assessment activities to support such claims of satisfaction. In the remainder of this paper, we work out which claims are required and discuss how they can be managed in a model-driven design process.

4. Preserving Satisfaction of Requirements through Conformance

The notion of conformance between models is central to our approach. We say that a model MT *conforms to another model* MS if, and only if, the set of acceptable realizations for MT is contained in the set of acceptable realizations for MS. Given this definition, we can observe that Figure 4 depicts both conformant and non-conformant pairs of models. For example, M3' does not conform to M2', while M2' conforms to M1.

If a model at a lower level of abstraction (M*i*+1) does not conform to a model at the previous level (M*i*), a designer is forced to consider both M*i*+1 and M*i* in a subsequent design step. This problem is exacerbated in the presence of multiple levels of abstraction that are not related by conformance. In the extreme case, a designer has to consider all models in a design step that produces the realization. This problem is addressed with conformant models. Conformant models can be regarded as replacing the models they conform to. For example, Figure 5 shows only conformant models M1, M2 and M3. Thus, in the design step from M2 to M3, M1 does not have to be considered. Further, M3 is sufficient to derive the realization.

Intuitively, a model creates a sort of a "mould" such that all subsequent models should fit into it ("conform"). The same is not necessarily true with sets of requirements, which can be regarded as defining constraints that have to be considered in conjunction.

By populating a hierarchy of models with models that conform to models at a higher level of models, designers can simplify requirements traceability activities. This is possible because requirements satisfied by a model are also satisfied by all models that conform to it. The evidence showing that a model satisfies certain requirements can be reused for models that conform to it.

In a design step that produces a conformant target model, the designer only has to provide evidence for supplementary requirements that are satisfied in the target model but not in the source model. In Figure 5, this means that assessment of M2 only requires evidence for the satisfaction of RS2 instead of both RS1 and RS2. Further, modification of models at a lower abstraction level does not affect models at a higher abstraction level if the modified model remains conformant.

We can now observe that the partitioning of requirements in different sets as depicted in Figure 5 arises from the way in which the various sets of requirements are addressed throughout the model-driven design trajectory.

5. Requirements Traceability with Transformation

This section extends the view of the model-driven design process as described in section 3 with model transformation chains.

5.1. Automated Transformation Chains

We start by considering fully automated transformation chains. Fully automated transformation chains consist of a predefined series of transformation specifications that can be applied to relate different subsequent levels of models. All transformation activities are automated using the various transformation specifications. An application model that is used as input for the transformation chain is sufficient to obtain a realization of the application.

In the case of automated transformation chains, application requirements only influence the application model. This is shown in Figure 6. Note that there are no relations between model transformation specifications (TSA and TSB) and application requirements. The reason for this is that model transformation specifications capture application-independent design operations that can be reused in the development of several applications.

A useful analogy for automated transformation chains is the programming language compiler: source code can be regarded as the application model, and assembly code can be considered the realization on a target hardware platform (with intermediate representations often used for optimization purposes). The specification of the compiler (i.e., the model transformation specification) is independent of the applications compiled by the compiler.

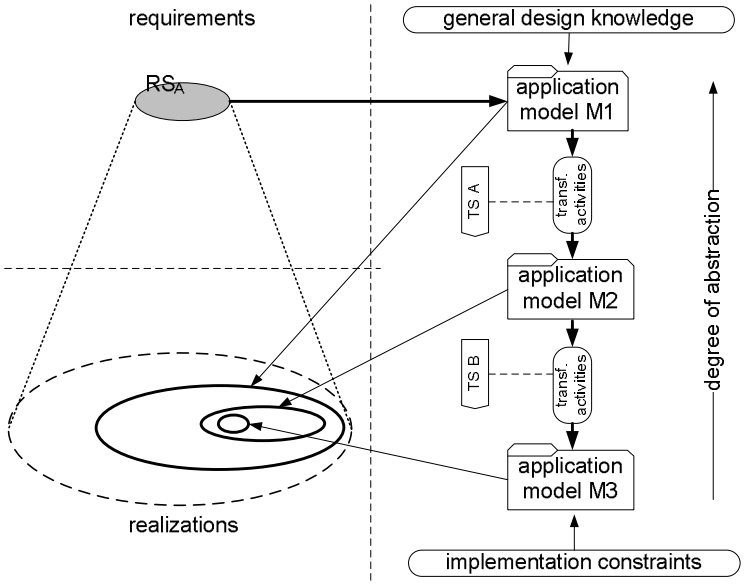

Figure 6 Application-specific requirements only affect the highest level of models

In the case of automated transformation chains, assessment activities can be summed up in (i) assessing whether the application model satisfies application requirements, and (ii) whether M*i*+1 conforms to M*i* for every transformation step (a special kind of assessment we call *conformance assessment*). In case models at intermediate levels are not considered reusable products of the design process, it suffices to assess whether the last model conforms to the first model.

When a transformation chain is assumed to produce conformant results, the only required activity left is assessing whether the application model satisfies application requirements. Other assessment activities are deemed redundant by the assumption of conformance. In the analogy of a programming language compiler, only source code is assessed if the compiler can be trusted.

To capture this reorganization of assessment activities in terms of the quality of a transformation specification, we define the notion of a *conformant transformation specification*. We say that a transformation specification is *conformant*, if, and only if, for every source and target models related by the specification, the target model conforms to the source model.

5.2. Partially Automated Transformation Chains

As discussed in the last section, the traceability of requirements can be largely simplified for the case of fully automated transformation chains with conformant transformation specifications. However, full automation of transformations is not always feasible or desirable. For example, it may be impossible to derive relevant design decisions from an high-level application model, or it may be inefficient to specify automated transformations having a limited reuse potential (see Almeida (2006) for an analysis on the costs/benefits of automated transformation). We distinguish the following approaches to decrease the level of automation without manual modification of target models:

(i) *transformation parameterization*, in which case the designer selects values for transformation parameters, i.e., arguments. Transformation parameters capture variation in the way source and target models are related; and,

(ii) *selection of transformations*, in which case a designer configures a transformation chain from a number of alternative predefined transformations. In order to simplify our discussion, we regard selection of alternative transformations as a special case of transformation parameterization, where a transformation specification includes the relations specified by all alternative transformations, and arguments are used to select an alternative.

In this case, application requirements influence transformation arguments (see Figure 7).

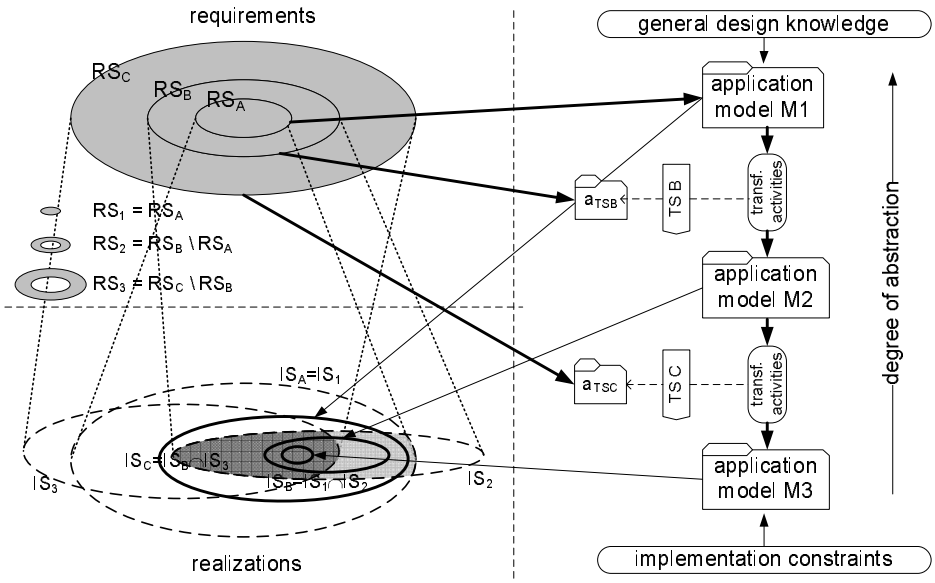

Figure 7 Application-specific requirements affect transformation arguments

The definition of a conformant transformation specification can be easily adjusted to incorporate transformation arguments. A transformation specification is said to be *conformant*, if, and only if, for every source and target models related by the specification *under every admissible set of transformation arguments*, the target model conforms to the source model.

For a transformation chain with parameterized conformant transformation specifications, one should assess whether the application model at level-1 satisfies application requirements RS1, and whether design decisions implied by transformation arguments satisfy different partitions of requirements (RSi).

5.3. Manual Modification

If necessary, the level of automation may be further lowered by allowing designers to manually modify target models. We assume in this case that modification is not unconstrained: the relations between source and target models as defined in a transformation specification should be respected (although tool support may allow these relations to be temporarily violated, as long as they are

eventually re-established). Figure 8 shows the relation between requirements and the various levels of models for this case.

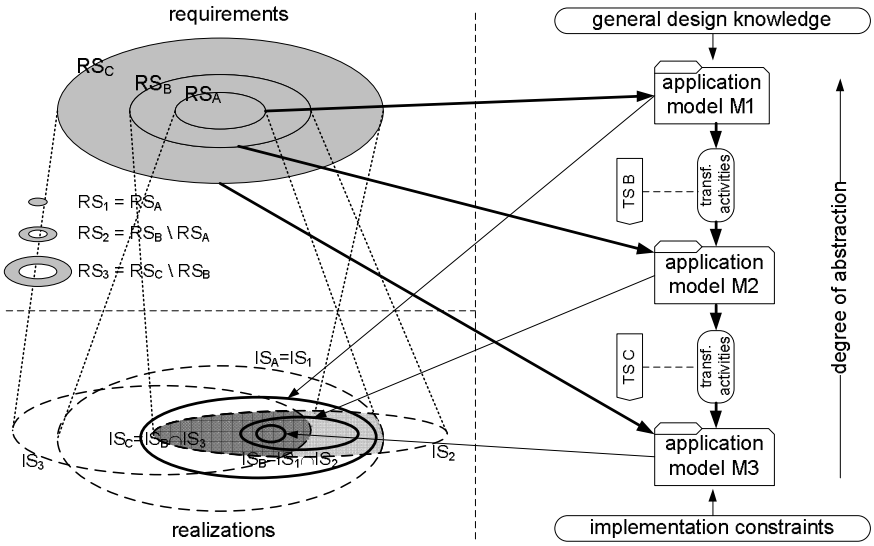

Figure 8 Application-specific requirements affect application models

Assessment activities in this case include: assessing whether models are conformant, and assessing whether the partitions of requirements (RSi) are satisfied progressively. If transformation specifications are conformant, this is simplified to assessing the satisfaction of the partitions of requirements (RSi) at the different levels of models.

6. A Requirements Traceability Metamodel for Model-Driven Development

In order to provide tool support for requirements traceability according to the framework proposed in this paper, we define a requirements traceability metamodel (Figure 9, using the Ecore meta-metamodel, as supported by the Eclipse Modelling Framework (EMF), see Budinsky et al, 2003).

This metamodel captures the main elements of the requirements traceability framework presented so far. We use a requirements traceability matrix to visualize the relation between requirements and the various artifacts. A traceability matrix "records the relationship between two or

more products of the development process" (IEEE, 2002). In our model, traceability is recorded in terms of model and transformation conformance, which makes it suitable for inclusion in tools for model-driven development.

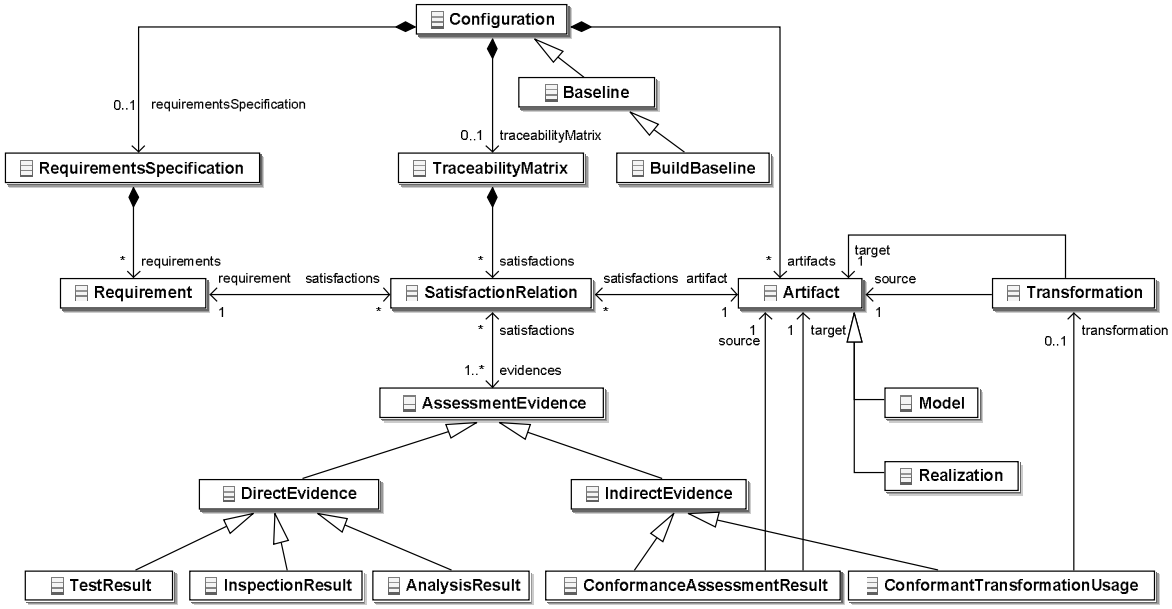

Figure 9 Requirements Traceability Metamodel

In the proposed metamodel, the traceability matrix consists of a satisfaction relation for each pair of related requirement and artifact. Each satisfaction relation must provide some assessment evidence. We distinguish assessment evidence into: (i) *direct evidence*, which is the result of assessment activities that directly verify the satisfaction of requirements into artifacts (e.g., testing, inspection and analysis); and (ii) *indirect evidence*, which is the result of model conformance assessment and conformant transformation usage.

We use a *configuration* to group a requirements traceability matrix, a requirements specification and a set of design artifacts (models at different levels and the realization). A configuration is a collection of items under *configuration management* (IEEE, 2002; Software Engineering Institute, 2000). (While configuration management usually considers versioning of configuration items, this is not discussed in this paper.)

A particular kind of configuration of special interest for us is a *baseline*, which is a formally approved configuration (IEEE, 2002). We define constraints for a baseline configuration such that requirements in the requirements specification are properly accounted for with suitable evidence captured in the traceability matrix (see constraints C1, C2, C3 in Table 1). These constraints are defined using OCL 2.0 (Object Management Group, 2003b). A particular kind of baseline of interest is a "build" baseline which contains a realization. In this baseline, there exists an artifact that satisfies all requirements (either directly or indirectly). This artifact is a realization (see constraint C4 in Table 1).

ID	Constraint
C1	"in a baseline, there must be satisfaction relations for all requirements" **context** Baseline **inv:** requirementsSpecification.requirements->forAll(r : Requirement | r.satisfactions->notEmpty())
C2	"in a baseline, no indirect evidence is based on itself (evidence graph is acyclic)" **context** Baseline **inv:** traceabilityMatrix.satisfactions.evidences->forAll(e : IndirectEvidence | not e.allSourceEvidence()->includes(e))
C3	"in a baseline, all indirect evidence must be ultimately based on direct evidence" **context** Baseline **inv:** requirementsSpecification.requirements.satisfactions.evidences->forAll (e : IndirectEvidence | e.allSourceEvidence()->exists(oclIsKindOf(DirectEvidence))
C4	"in a build baseline, there exists an artifact that satisfies all requirements (either directly or indirectly)" **context** BuildBaseline **inv:**artifacts->exists(satisfactions.requirement->asSet()=requirementsSpecification.requirements)
Helper	"expression that provides all evidence on which a given indirect evidence is based (transitive closure)" IndirectEvidence::allSourceEvidence(): Set(AssessmentEvidence) = source.getEvidence()->union(source.getIndirectEvidence()->collect(allSourceEvidence())->asSet()) "expressions that provide indirect and direct evidence for a given artifact" Artifact::getIndirectEvidence() : Set(IndirectEvidence) = self.getEvidence()->select(oclIsKindOf(IndirectEvidence)) Artifact::getEvidence() : Set(AssessmentEvidence) = self.satisfactions.evidences->asSet()->flatten()

Table 1 Constraints

The constraints capture the conditions that must apply to the result of the requirements traceability process, without unnecessarily constraining the requirements traceability process itself. This makes our approach suitable to different model-driven development processes and practices.

7. Example

To illustrate the application of our framework and to show how our approach reduces assessment activity effort, we present as an example the design of a telemonitoring system (Almeida, Iacob, Jonkers & Quartel, 2006). The goal of this system is to monitor a chronically ill patient continuously and warn the patient and care givers (e.g., at a hospital) of critical health conditions.

Table 2 presents (functional and non-functional) requirements for a specific telemonitoring system, which issues alarms for epileptic seizures.

ID	Description
AR1	Upon detection of an (eminent) epileptic seizure, the patient shall be alarmed.
AR2	Upon detection of an (eminent) epileptic seizure, aid persons in the surrounding of the patient may be alarmed.
AR3	Only aid persons with an available status are alarmed.
AR4	In case no aid persons can be alarmed an emergency health care team in the surrounding of the patient will be alarmed.
AR5	In case the epileptic seizure occurs at a speed higher than 8km/h, an emergency health care team will be alarmed (instead of aid persons) (rationale: this may involve high risk, e.g., if the patient is biking, jogging, driving).
AR6	Alarms to aid persons or health team inform them of the last known location of the patient.
AR7	Alarms should be realized through short messaging service or calling aid persons with voice messages (rationale: aid persons do not have to maintain and carry any complex devices in addition to their mobile phones).
AR8	Patient location and speed may be determined through GPS devices.
AR9	Patient and aid person location may be determined through Parlay-X.
AR10	Aid person availability status may be determined through Parlay-X presence.
AR11	In case patients/aid persons should carry mobile devices for monitoring, these should allow uninterrupted monitoring for 24 hours, without requiring battery recharges.
AR12	Costs of mobile communication should not exceed EUR 50,- per month per patient.

Table 2 Requirements for telemonitoring system

In this example, the development of the system is guided by a model-driven design trajectory, in which three levels of models are defined (Almeida, Jonkers, Iacob & Quartel, 2005; Almeida, Iacob, Jonkers & Quartel, 2006): the service specification level (M1); the platform-independent service design level (M2); and the platform-specific service design level (M3). These three levels of models are depicted in Figure 10.

At the level of service specification a service can be described in terms of *events*, which represent contextual changes and occurrences of interest (e.g., an epileptic seizure), *queries* to providers of context information (e.g., a patient's location, speed and bio-signals), and *actions*, which represent actions to be performed in order to provide the service to the user (e.g., issuing an alarm). These elements are expressed in a domain-specific language (called ECA-DL (Almeida, Jonkers, Iacob & Quartel, 2005; Almeida, Iacob, Jonkers & Quartel, 2006)).

At the platform-independent service design level, behavioural aspects of service design are described with ISDL (*The Interaction Systems Design Language*; http://isdl.ctit.utwente.nl/) models and OCL (Object Management Group, 2003b) constraints. UML class diagrams (omitted here) are used to represent information models.

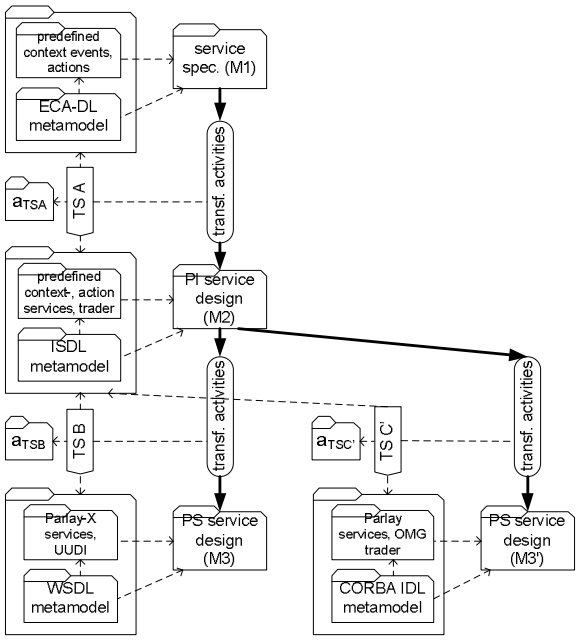

Figure 10 Design trajectory for context-aware mobile services (adapted from Almeida, Iacob, Jonkers & Quartel, 2006)

The transformation between the service specification and the service design level consists of refining events, queries and actions at the service specification level into sequences of interactions in the service design. At the service specification level, an action represents an activity performed

by the system as a whole (including any context sources and action services). However, at the service design level the same action has to be performed by cooperation of different services, in a service-oriented design which includes various context and action services. The transformation rules are defined extensively in (Almeida, Jonkers, Iacob & Quartel, 2005). TSA is parameterized so that the designer can define constraints on which services can be used to realize events and actions in the service specification (so it can be considered a partially automated transformation).

7.1. Models

We focus on the service specification and platform-independent service design levels in order to limit the size of this example. The Telemonitoring service specification is depicted in Figure 11 (this corresponds to M1 in Figure 10).

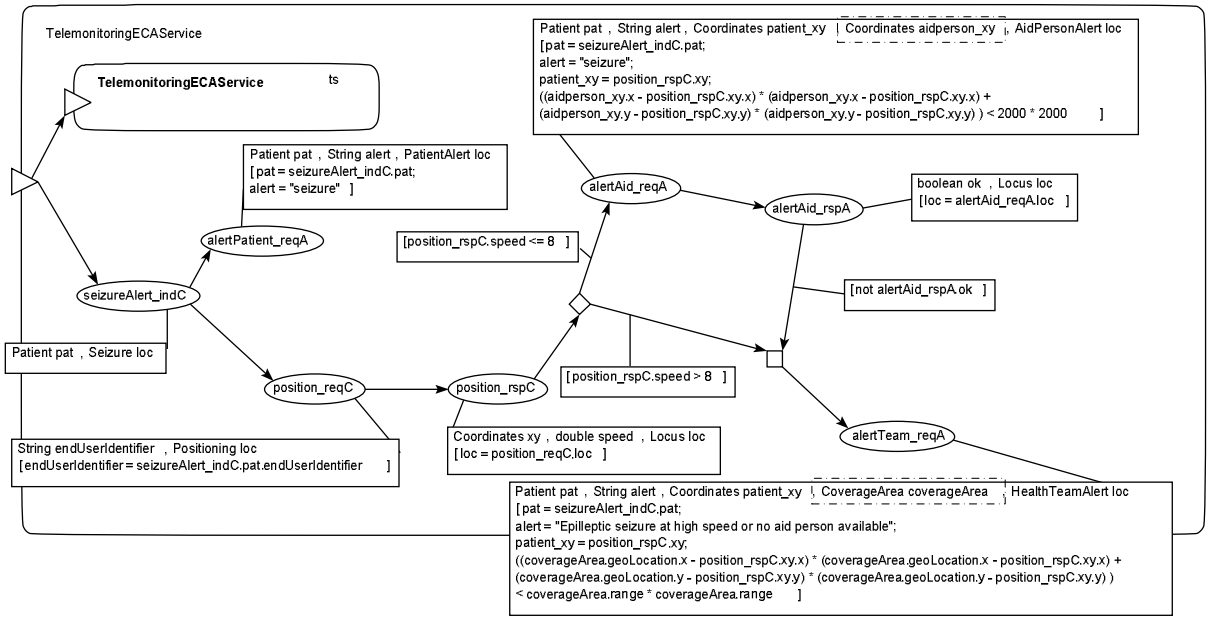

Figure 11 The Telemonitoring service specification (M1, see Almeida, Iacob, Jonkers & Quartel, 2006)

Ovals represent context events, queries and actions. The suffix _indC indicates a context event, the suffixes _reqC, _rspC indicate a request-response query to context sources and the suffixes _reqA, _rspA indicate request-response to action services. Arrows indicate enabling relations

between events, queries and actions; white diamonds represent choice (or-split) and white squares denote disjunction. Guards for enabling relations and constraints for information are depicted in boxes attached to context events, queries and actions.

The platform-independent service design is the result of the application of all transformation rules to the service specification. Figure 12 (this corresponds to a part of M2 in Figure 10) shows the generated coordination component. The dashed lines represent causality relations already present in the service specifications. Semi-ovals represent interactions in ISDL.

The generated coordination component interacts with a service trader to find context and action services. The service queries are generated from constraints at the service specification level, which are indicated in arguments a_{TSA} to the transformation (in this case, they are constraints on alertAid_reqA.aidperson_xy and alertTeam_reqA.coverageArea as marked with boxes in Figure 11).

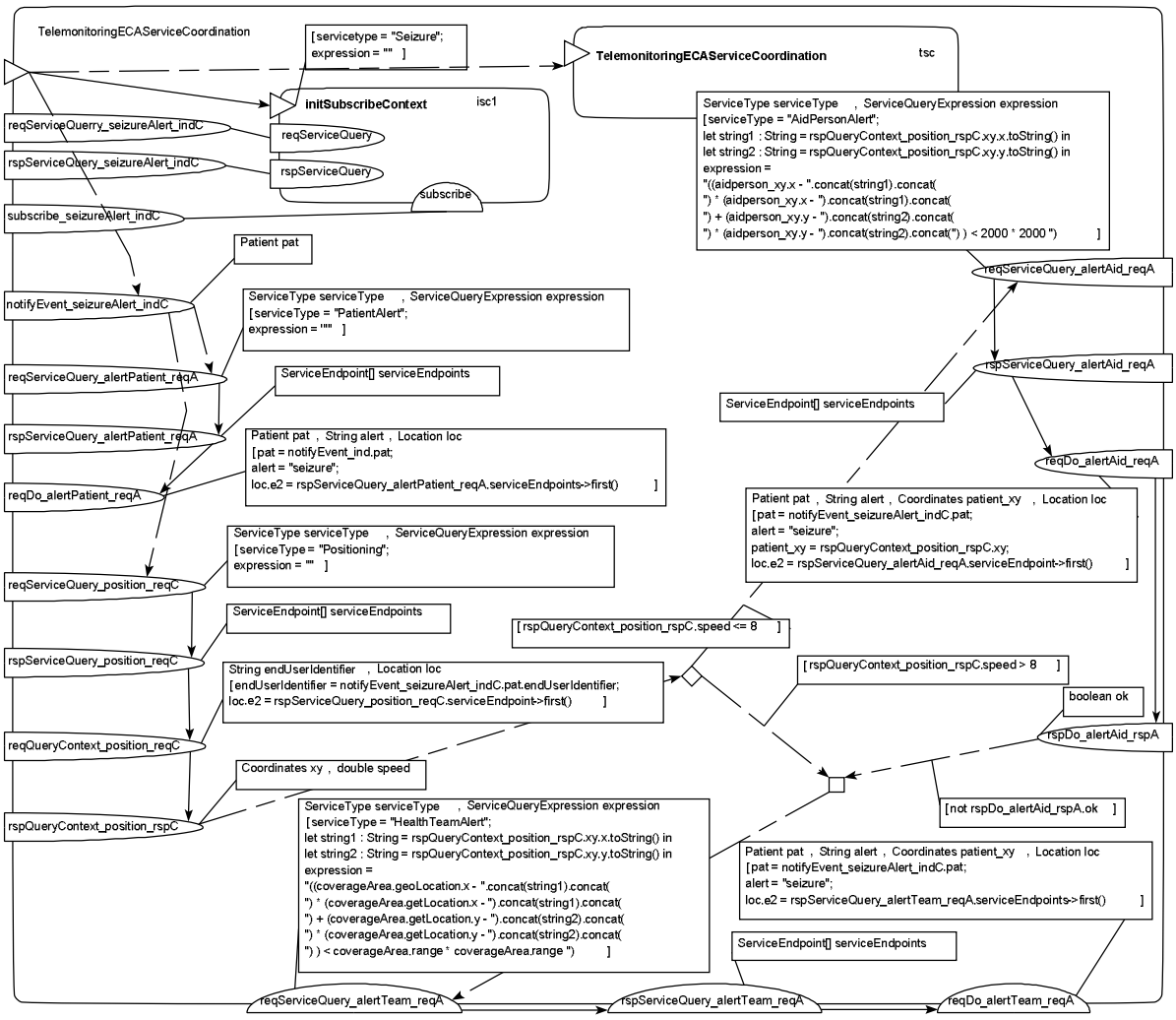

Figure 12 Generated coordination component for Telemonitoring service (M2)

7.2. Traceability

In this section, we present the resulting traceability information for our example. First, we show a requirements traceability matrix with only direct evidence for satisfaction of requirements (i.e., without using the notion of conformance proposed here), and then we present a requirements traceability matrix using the notion of conformant transformation. The objective is to illustrate the kinds of assessment techniques that may be employed in the framework, and to show how conformance simplifies assessment activities.

7.2.1. Traceability with Direct Evidence

The traceability matrix with direct evidence is depicted in Table 3, with a row for each requirement, and a column for each artifact. An instance of SatisfactionRelation (in our metamodel) is represented by a check mark in the matrix (✓). All marks must be justified by evidence resulting from assessment activities, for example:

- Marks in the column corresponding to M1 can be justified by directly inspecting M1 against the requirements specification; alternatively they can be justified by simulating M1 (e.g., with the Grizzle simulator, see http://isdl.ctit.utwente.nl/tools/grizzle).
- Marks in the column corresponding to arguments for TSA (a_{TSA}) can be justified by considering the characteristics of the action services implied by the particular choice of arguments.
- Marks in the column corresponding to M2 can be justified by simulation of the service design, by model checking behavioural constraints implied by AR1–AR10, etc.
- Marks in column M3 that correspond to requirements AR1–AR10 can be justified by executing test cases for AR1–AR10. Marks for the satisfaction of AR11 ("mobile devices for monitoring should allow uninterrupted monitoring for 24 hours, without requiring battery recharges.") in M3 may be justified by analysing M3 and the specifications of the target platform on which M3 relies (in this case the specifications of battery consumption for PDAs or mobile phones) and any characteristics of M3 that may influence battery consumption (e.g., communication and display usage). Assessment of satisfaction of AR12 ("mobile communication costs should not exceed EUR 50,- per month per patient") in M3 may be conducted by analysing the amount of traffic generated by M3 and its composition with the characteristics of the platform on which it relies (e.g., cost models and the traffic characteristics of communication protocols).

	M1	a_{TSA}	M2	M3
AR1	✓		✓	✓
AR2	✓		✓	✓
AR3	✓		✓	✓
AR4	✓		✓	✓
AR5	✓		✓	✓
AR6	✓		✓	✓
AR7		✓	✓	✓
AR8		✓	✓	✓
AR9		✓	✓	✓
AR10		✓	✓	✓
AR11				✓
AR12				✓

Table 3 Requirements traceability matrix with direct evidence

7.2.2. Traceability with Conformant Transformation Usage

As discussed in section 5.2, these assessment activities can be simplified by employing conformant transformations. We illustrate this with a transformation TSA, which has been designed such that it is conformant under the following assumptions (Almeida, Jonkers, Iacob & Quartel, 2005): (i) the service trader is always able to produce a service offer for a service query, (ii) context sources always reply to context query requests, and (iii) action services always reply to action invocation requests (in case action invocation request and action invocation response is used in a pattern). Assumption (i) can be guaranteed by availability of service offers in the service trader that correspond to actions and context queries and events in the service specification level (according to transformation arguments a_{TSA}). Assumptions (ii) and (iii) constrain the design of context sources and action services. These assumptions are necessary to integrate the interactions in the target design into actions and then apply the conformance assessment method described in (Quartel, Ferreira Pires & van Sinderen, 2002).

By employing conformant TSA and TSB, the resulting requirements traceability matrix is presented in Table 4. All marks in the M2 column and all marks for AR1–AR10 in the M3 column are implied (i.e., indirect evidence is provided), which is indicated by square brackets ([✓]). Assess-

ment activities to check them have become redundant, diminishing the assessment effort needed. In fact, M2 can even be considered a "black-box" by an application designer, without assessment activities required.

	M1	a_{TSA}	M2	M3
AR1	✓		[✓] TSA	[✓]TSB
AR2	✓		[✓] TSA	[✓]TSB
AR3	✓		[✓] TSA	[✓]TSB
AR4	✓		[✓] TSA	[✓]TSB
AR5	✓		[✓] TSA	[✓]TSB
AR6	✓		[✓] TSA	[✓]TSB
AR7		✓	[✓] a_{TSA}	[✓]TSB
AR8		✓	[✓] a_{TSA}	[✓]TSB
AR9		✓	[✓] a_{TSA}	[✓]TSB
AR10		✓	[✓] a_{TSA}	[✓]TSB
AR11				✓
AR12				✓

Table 4 Requirements traceability matrix with indirect evidence

The matrix also reveals a "natural" partitioning of requirements into sets, since certain sets of requirements are satisfied initially at a particular level of models. This is shown in Table 4 with thick borders delimiting three sets of requirements AR1-AR6, AR7-AR10 and AR11-AR12 which are satisfied at levels 1, 2 and 3 respectively.

All constraints defined in the metamodel are satisfied (see C1-C4 in section 6), such the configurations with the requirements traceability matrices shown in Tables 3 and 4 can be considered "build" baselines, i.e., an approved configuration which includes an approved realization.

8. Related Work

8.1. Requirements Engineering and Traceability

In the area of Requirements Engineering, the standard general introduction of the requirements traceability problem has been provided by Gotel and Finkelstein (1994). The Ph.D. thesis of

Gotel (1995, pages 71-72) provides extensive discussion of requirements traceability, including a number of definitions.

It has been recognized that requirements tracing is a laborious task and that any assistance in maintaining the interdependencies between requirements and other design artifacts is highly welcome. Egyed (2003) presents an approach in which dependencies are discovered automatically from data generated by executing a minimal set of scenarios. This approach requires that an executable version of the system is available to execute these scenarios. In our approach, however, traceability is not dependent on an executable system; therefore, traceability is already possible when the design process has not yet resulted in an executable prototype.

Ramesh and Jarke (2001) present a reference model for requirements traceability that they derived from an empirical study. Their reference model comprises a number of possible relations that can be traced between design artifacts and requirements. For different stakeholders (and different ambition levels with respect to traceability), a different subset of those relations can be chosen. In principle, our conformance-based approach is transparent with respect to the choice of this subset. An interesting question for future research is whether subsets can be identified that are particularly suitable for a model-driven design approach. In this sense, we mention here also the work of van den Berg, Tekinerdogan & Nguyen (2006) and van den Berg, Conejero & Hernández (2006) who use traceability matrices in their study of crosscutting concerns and impact analysis in model-driven design. Nevertheless, this approach does not address the issue of conformance and conformant transformations.

We do not account in this paper for explicit requirements on the design process itself, such as, e.g., cost, delivery schedules, validation and verification criteria (assessment criteria). This is in line with IEEE (1998), which states that “SRS should address the software product, not the

process of producing the software product." These should be captured in project requirements which "represent an understanding between the customer and the supplier about contractual matters pertaining to production of software and thus should not be included in the SRS." However, requirements on the model-driven design trajectory (so-called application-independent requirements) are addressed by Almeida, van Eck & Iacob (2006). According to Almeida, van Eck & Iacob (2006), these are to be maintained separately from application-specific requirements, and are relevant only to "suppliers" and their internal organization and are not visible to "customers."

8.2. Techniques for Conformance Assessment

So far we have argued that, in order to support requirements traceability and claims of satisfaction, it is the task of the designer to conduct assessment activities. The main question to be eventually answered by assessment activities is to what extent all the functional and non-functional requirements that have been derived from the original purpose of the application are met in the current application realisation. We have emphasized the assessment of conformance for transformation specifications, by demonstrating that assessment activities can be to a large extent diminished if only conformant transformations are assumed to be used during the design process. We have deliberately chosen to be neutral in our methodological framework with respect to specific tools or techniques that the designer may choose to trust to support the assessment process (as well as those he/she may chose to model the application). Instead, we have focused on how to manage the relations between models and requirements.

Examples of useful conformance assessment techniques are the "conformance rules" for "behaviour refinement" discussed by Quartel, Ferreira Pires & van Sinderen (2002) (and used in our

example), “refinement relations” discussed by Dijkman (2006) or “conformant transformations for interaction refinement” presented by Almeida, Dijkman, Ferreira Pires, Quartel & van Sinderen (2006).

In the area of formal methods, notions of transformation conformance have also been defined. Nevertheless, approaches based on formal methods rely on formal proofs as evidence for transformation conformance (see, e.g., “correct architectural refinement” in (Moriconi, Qian & Riemenschneider, 1995), and “correctness preserving transformations” in (Bolognesi, van de Lagemaat & Vissers, 1995; Gibson, Dowling & Malloy 2000)). We believe that formal proofs may not be required in many practical cases. Therefore, we have proposed definitions for conformance and requirements satisfaction that are independent of proofs of conformance or formalization of requirements.

We have made a clear distinction between the assessment of the conformance of transformation specifications and the assessment of conformance for transformation results, which we call model conformance. One of the reasons for making such a distinction has to do with the separation of roles in the design process, namely with the distinction between the transformation specifier and the transformation user (see Brottier, Fleurey, Steel, Baudry & Le Traon, 2006). Therefore, the techniques for assessing the conformance of models can be treated separately from those for transformation specifications, although one might claim that the two types of conformance are equivalent, namely, if a transformation specification is conformant then transformation results (i.e. target models) will also be conformant with the transformation inputs (i.e., source models), and the other way around: if for any possible source model, the target model obtained as result of the application of a transformation specification conforms with the source model, then the transformation specification is also conformant.

Transformation specification assessment has been recognised as an important issue by several authors that argue that transformation specifications and transformation results should undergo a rigorous validation and testing process (e.g., Judson, France & Carver, 2003 and Lin, Zhang & Gray, 2005). Accordingly, Fleurey, Steel & Baudry (2004) proposes a functional test adequacy criterion for the validation of model transformation programs. Küster (2004) goes a step further by identifying the most important properties that have to be checked through model transformation validation and testing approaches. These include the *syntactic correctness of a model transformation* (ensuring that the model transformation produces syntactically correct models, that conform to a specified target metamodel), *termination and confluence of a model transformation* (this would ensure that the model transformation always produces a unique result), *semantic equivalence or semantics preservation of a model transformation*, and *safety or liveness properties*. However, apart from identifying these properties, Küster's account only focuses on checking syntactic correctness, namely rule and non-terminals correctness, while our approach starts by assuming syntactic correctness and mostly focuses on the preservation of semantics through (partially automated) model transformation chains.

In the area of model conformance assessment techniques, important advances are currently made with respect to model-driven testing. This research (Dai, 2004; Hartman, Nagin & Olvovsky, 2004; Heckel & Lohmann, 2003; Pfaller, Fleischmann, Hartmann, Rappl, Rittmann & Wild, 2006; Zhu, Horgan, Cheung & Li 2006) is based on the distinction between platform-independent and platform-specific models and follows a corresponding strategy for model-driven testing with respect to the reuse of platform-independent test cases and the (automated) test generation. In this line of thinking, several authors propose approaches rooted in general model-based system testing theory and focusing on the reuse and generation of tests and oracles (e.g.,

Brottier, Fleurey, Steel, Baudry & Le Traon, 2006; Heckel & Lohmann, 2003), while others focus on specific methodologies, techniques and tool support for model-driven testing of UML models (e.g., Dai, 2004). An architecture for testing model transformations is proposed by Lin, Zhang & Gray (2004), which starts from the assumption that model transformation conformance assessment can be reduced to the verification of the conformance of the transformation results, i.e., to "the execution of a deterministic transformation specification with test data (i.e., input to test cases) and a comparison of the actual results (i.e., the target model) with the expected output (i.e., the expected model), which must satisfy the intent of the transformation". The authors propose an algorithmic approach and model transformation testing framework using model comparison (with the detection of a difference set) using graph representations of the compared models.

Finally, in the Reference Model of Open Distributed Processing (RM-ODP, see ISO/ITU-T, 1995), the term "conformance" is used as relation between a "specification" and an "implementation". In this paper, we have used the term as relation between two application models. Considering our stance on the distinction between an application model and an application realization (see section 3), our view on conformance does not conflict RM-ODP's approach to conformance. RM-ODP uses the term "conformance testing", and we use the more general term "conformance assessment" to include other forms of assessment activities.

9. Conclusions

We believe that a mature discipline for model-driven design must provide techniques to account for how requirements relate to the various artifacts produced during the design process. In this paper we have proposed a methodological framework that addresses this issue. Our framework

can be seen as basis for requirements traceability, but also serves to reveal the intricate relationship between requirements, application models and realizations, model transformation specifications and transformation arguments. The framework includes a metamodel that can be used as a basis for tool support for model-driven development.

In our view it is important for both application users and application designers to be able to produce evidence for satisfaction of requirements. This is realised through assessment activities. We have argued that some of these assessment activities can be deemed redundant under the assumption that conformant transformation specifications are used in the design process. Thus, we have concluded that conformance between models not only simplifies requirements tracing but also has the potential of reducing the amount of necessary assessment activities.

We acknowledge though that the quality of assessment depends ultimately on the quality of a requirements specification. Different characteristics of a "good" specification are defined by IEEE (1998) including correctness, lack of ambiguity, completeness, consistency, etc. Guidelines for obtaining these qualities are beyond the scope of this paper. Also it should be noted that the simplification of assessment activities results from the way in which requirements are partitioned and addressed at different levels of abstraction. Therefore, for sets of requirements that cannot be partitioned and that must be partially satisfied at multiple abstraction levels, simplification of assessment by conformance is limited. Finally, while we have discussed the potential benefits of conformant transformations, we would like to emphasize that evidence for transformation conformance may be costly to produce. One should therefore consider the pay-off in terms of assessment activities, depending on the reuse of transformation specifications.

In our future work, we intend to investigate both the specification of conformance relations and model transformations in the same transformation specification framework. More precisely, we

plan to focus on techniques and tools (based on our metamodel) for capturing, enforcing and assessing conformance between models; and assessing whether transformation specifications respect conformance. This may be feasible by taking (as suggested by Almeida, Dijkman, Ferreira Pires, Quartel & van Sinderen, 2006) a relational approach regarding model transformations and conformance (Akehurst, Kent & Patrascoiu, 2003; Object Management Group, 2005).

Future work could also investigate traceability of requirements in face of changes in requirements specifications, which may be trigged due to changing application requirements and due to improved understanding of requirements in an iterative design process.

Acknowledgements

This work is part of the Freeband A-MUSE project (http://a-muse.freeband.nl), which is sponsored by the Dutch government under contract BSIK 03025. Remco Dijkman, Luís Ferreira Pires, Henk Jonkers, Thijs Munsterman, Dick Quartel and Marten van Sinderen should be acknowledged for fruitful discussions on the topics addressed here. An earlier version of this work has been produced while the first and second authors were scientific researchers at the Telematica Instituut (Enschede, The Netherlands).

References

Akehurst, D., Kent, S., & Patrascoiu, O. (2003). A relational approach to defining and implementing transformations between metamodels. *Software and Systems Modeling*, vol. 2. no. 4, Springer-Verlag, 215–239.

Almeida, J. P. A. (2006). *Model-Driven Design of Distributed Applications*. CTIT Ph.D.-Thesis Series, No. 06-85, Telematica Instituut Fundamental Research Series, No. 018.

Almeida, J. P. A., Dijkman, R., Ferreira Pires, L., Quartel, D., & van Sinderen, M. (2006). Model Driven Design, Refinement and Transformation of Abstract Interactions. *Int'l J. Cooperative Information Systems (IJCIS)*, World Scientific, 599–632.

Almeida, J. P. A., Iacob, M. E., Jonkers, H., & Quartel, D. (2006). Model-Driven Development of Context-Aware Services. *Distributed Applications and Interoperable Systems (DAIS 2006), 6th IFIP Int'l Conference*, LNCS, vol. 4025, Springer, 213–227.

Almeida, J. P. A., Jonkers, H., Iacob, M. E., & Quartel, D. (2005). *Platform-Independent Modelling of Service Infrastructure Components: Towards the A-MUSE Abstract Platform*, Freeband A-MUSE/D1.6, Telematica Instituut, The Netherlands.

Almeida, J. P. A., van Eck, P., & Iacob, M. E. (2006). Requirements Traceability and Transformation Conformance in Model-Driven Development. *Proceedings of the 10th IEEE International Enterprise Distributed Object Computing Conference (EDOC'06)*, IEEE Computer Society Press, 355–366.

Almeida, J. P. A., van Sinderen, M., Ferreira Pires, L., & Quartel, D. (2003). A systematic approach to platform-independent design based on the service concept. *Proc. 7th IEEE Int'l Conf. on Enterprise Distributed Object Computing (EDOC 2003)*. IEEE CS Press, 112–134.

Bolognesi, T., van de Lagemaat, J., & Vissers, C., eds. (1995). *LOTOSphere: Software Development with LOTOS*, Kluwer Academic Publishers.

Brottier, E., Fleurey, F., Steel, J., Baudry, B., & Le Traon, Y. (2006). Metamodel-based Test Generation for Model Transformations: an Algorithm and a Tool. *Proceedings of the 17th International Symposium on Software Reliability Engineering (ISSRE 2006)*, 85–94.

Dai, Z. R. (2004). Model-Driven Testing with UML 2.0. *Proceedings of the Second European Workshop on Model Driven Architecture (MDA) with an Emphasis on Methodologies and Transformations (EWMDA-2)*, Technical Report No. 17-04, Computing Laboratory, University of Kent, Canterbury.

Dijkman, R. M. (2006). *Consistency in Multi-Viewpoint Architectural Design*, CTIT Ph.D.-Thesis Series, No. 06-80, Telematica Instituut Fundamental Research Series, No. 017.

Egyed, A. F. (2003). A Scenario-Driven Approach to Trace Dependency Analysis. *IEEE Transactions on Software Engineering*, vol. 29, no. 2, 116–132.

F. Budinsky et al. (2003). *Eclipse Modeling Framework*, Addison Wesley Professional.

Firesmith, D. (2003). Specifying Good Requirements. *Journal of Object Technology*, vol. 2, no. 4, 77–87.

Fleurey, F., Steel, J., & Baudry, B. (2004). MDE and validation: Testing model transformation. *Proc. of the Workshop on Specification Implementation and Validation Of Embedded Systems - Model Design and Validation (SIVOES-Modeva)*, 2004.

Gavras, A., Belaunde, M., Ferreira Pires, L., & Almeida, J. P. A. (2004). Towards an MDA-based Development Methodology for Distributed Applications. *Software Architecture: First European Workshop (EWSA'04)*, LNCS, vol. 3047, Springer, 230–240.

Gibson, J. P., Dowling, T. F., & Malloy, B. A. (2000). The Application of Correctness Preserving Transformations to Software Maintenance. *Proc. 16th IEEE International Conference on Software Maintenance (ICSM'00)*, IEEE CS Press, 108–119.

Gotel, O. (1995). *Contribution Structures for Requirements Traceability*. Ph.D. Thesis, London, England: Imperial College, Department of Computing.

Gotel, O. & Finkelstein, A. (1994). An Analysis of the Requirements Traceability Problem. *Proc. First Int'l Conf. Requirements Engineering*, 94–101.

Hartman, A., Nagin, K. & Olvovsky, S. (2004). Model Driven Testing and MDA, *Workshop on Model Driven Development (WMDD 2004)* at ECOOP 2004, Oslo, Norway (June 14–18, 2004), http://heim.ifi.uio.no/~janoa/wmdd2004/papers

Heckel, R. & Lohmann, M. (2003). Towards Model-Driven Testing. *Electronic Notes in Theoretical Computer Science*, 82, No. 6, 1–11.

IEEE (1998). *IEEE Recommended Practice for Software Requirements Specifications*, IEEE Std 830-1998.

IEEE (2002). *IEEE Standard Glossary of Software Engineering Terminology*, IEEE Std. 610.12-1990 (R2002).

ISO/ITU-T (1995), *Open Distributed Processing - Reference Model - Part 2: Foundations*, International Standard ISO/IEC 10746-2, ITU-T Recommendation X.902.

Judson, S. R., France, R., & Carver, D. L. (2003). Specifying Model Transformations at the Metamodel Level. *Proceedings of Workshop in Software Model Engineering associated to UML'03*, San Francisco, CA, USA.

Küster, J. M. (2004). Systematic Validation of Model Transformations. *3rd Workshop in Software Model Engineering (WiSME 2004)*, 7th International Conference on the UML (UML 2004), Lisbon, Portugal http://www.metamodel.com/wisme-2004/papers.html.

Lin, Y., Zhang, J., & Gray, J. (2004). Model Comparison: A Key Challenge for Transformation Testing and Version Control in Model Driven Software Development. *Proceedings OOPSLA Workshop on Best Practices for Model-Driven Software Development*, http://www.softmetaware.com/oopsla2004/mdsd-workshop.html

Lin, Y., Zhang, J., & Gray, J. (2005). A Testing Framework for Model Transformations. *Model-driven Software Development - Research and Practice in Software Engineering*, Springer, 219–236.

Moriconi, M., Qian, X., & Riemenschneider, R. A. (1995). Correct Architecture Refinement. *IEEE Transactions on Software Engineering*, 21(4), IEEE Computer Society Press, 356–3.

Object Management Group (2003a), *MDA-Guide, V1.0.1*, omg/03-06-01.

Object Management Group (2003b), *Unified Modelling Language: Object Constraint Language version 2.0*, ptc/03-10-04.

Object Management Group (2005), *MOF QVT Final Adopted Specification*, ptc/05-11-01.

Pfaller, C., Fleischmann, A., Hartmann, J., Rappl, M., Rittmann, S., & Wild, D. (2006). On the integration of design and test: a model-based approach for embedded systems. *Proceedings of the 2006 International Workshop on Automation of Software Test (AST '06)*, ACM Press, New York, NY, 15–21.

Quartel, D., Ferreira Pires, L., & van Sinderen, M. (2002). On Architectural Support for Behaviour Refinement in Distributed Systems Design. *Journal of Integrated Design and Process Science*, vol. 6, no. 1, Society for Design and Process Science, 1–30.

Ramesh, B. & Jarke, M. (2001). Toward reference models for requirements traceability. *IEEE Transactions on Software Engineering*, vol. 27, no. 1, 58–93.

Schot, J. (1992). *The role of Architectural Semantics in the formal approach of Distributed Systems design*, Ph.D. thesis, University of Twente, Enschede, The Netherlands.

Software Engineering Institute (SEI) (2000). *CMMI for Systems Engineering/Software Engineering, Version 1.02 (CMMI-SE/SW, V1.02)*, Tech. Report CMU/SEI-2000-TR-018.

van den Berg, K., Conejero, J. M., & Hernández, J. (2006). Analysis of Crosscutting across Software Development Phases based on Traceability. *Proceedings of the 2006 International Workshop on Early Aspects at ICSE*. New York: ACM Press, 43–50.

van den Berg, K., Tekinerdogan, B., & Nguyen, H. (2006). Analysis of Crosscutting in Model Transformations. *ECMDA Traceability Workshop*, SINTEF Report A219, 51–64.

Wright, S. (1991). Requirements Traceability - What? Why? and How?. *Tools and Techniques for Maintaining Traceability During Design*, IEE Colloquium, Computing and Control Division, Professional Group C1 (Software Engineering), U.K., Digest Number: 1991/180, 1/1–1/2.

Zhu, H., Horgan, J. R., Cheung, S. C., & Li, J. J. (2006). The first international workshop on automation of software test. *Proceeding of the 28th International Conference on Software Engineering (ICSE '06)*, ACM Press, New York, NY, 1028–1029.