Model-driven Round-trip Software Dependability Engineering

Michele Tucci
Department of Information Engineering, Computer Science and Mathematics
University of L’Aquila, Italy
michele.tucci@univaq.it

ABSTRACT
Supporting changes in software models is becoming increasingly important. Some of these changes are induced by non-functional analysis that is usually conducted on different models and tools. Therefore, it becomes crucial to develop methods that allow automated transformations between these two families of models throughout the development cycle. To this extent, in the last decade, a number of approaches have been introduced to generate non-functional analysis models from software models. However, when analysis models are modified to meet non-functional requirements, changes are not propagated to update the software model. Automating the identification and propagation of changes would better support a round-trip analysis process. In this PhD program, we aim at introducing automation in the model-driven assessment of dependability, and we propose to leverage bidirectional model transformations to: (i) generate dependability analysis models from software models, and (ii) automatically propagate changes, driven by dependability requirements satisfaction, from analysis models back to software models. In particular, we intend to extend JTL, that is a bidirectional model transformations framework designed for model synchronization and change propagation, to handle problems that may arise from the application of bidirectional transformations in the context of dependability assessment.

CCS CONCEPTS
• Software and its engineering → Extra-functional properties; Model-driven software engineering;

KEYWORDS
software refactoring, dependability, change propagation, model-driven engineering, bidirectional model transformations

ACKNOWLEDGMENTS
This research was supported by the ECSEL-JU through the Megamart2 project (grant agreement No. 737494)

1 PROBLEM
During the last two decades, a major challenge for researchers working on software modeling and evaluation has been the assessment of system Non Functional Properties (NFP), such as performance, schedulability, dependability or security. This is mostly due to the increasingly demanding user requirements, on one side, and to the pervasiveness of software over heterogeneous platforms on another side. Attributes like performance and reliability are very complex to analyze, because the designer is usually able to assess them only late in the life cycle, when most of the system has been designed and implemented. Nevertheless, if non-functional requirements are not met, then a variety of negative consequences (such as user dissatisfaction, lost income, etc.) can impact on significant parts of a project [33]. This has led to a widespread use of quantitative models for the assessment of non-functional properties from the early phases of software design [16]. To this extent, a considerable number of approaches, mostly based on Model-Driven Engineering (MDE) [31] techniques like model transformation, has been proposed in the last decade to automatically generate non-functional models from software models [11, 17].

The general process for a model-driven assessment of non-functional requirements on software is outlined in Figure 1. Starting from a software model $M_1$ (top-left corner) annotated with non-functional properties, an initial analysis model $AM_1$ is obtained in the first step. This first step is usually called the forward path from software models to analysis models. Model $AM_1$ is then solved in order to obtain analysis indices $IND_1$ (e.g., response time, probability of failure, etc.). These indices may not meet some non-functional requirements, thus leading the designer to perform changes ($\Delta_1$) to the initial analysis model ($AM_1$) with the aim of improving the model. The refactored analysis model $AM_2$ and analysis indices $IND_2$ are obtained in this way. This process is repeated until a satisfactory analysis model $AM_k$ is obtained when the corresponding analysis indices $IND_k$ meet the requirements. However, a satisfactory analysis model has undergone several refactoring steps, hence it no longer corresponds to the initial software model.

Hence, the next task should be to somehow replicate in the software domain the changes performed in the non-functional analysis domain. This phase, usually called the backward path, corresponds to the change propagation step in Figure 1. Such a step is conducted by interpreting the feedback gained during the analysis to produce a refactored software model ($M_k$). This informal process of deriving
changes is inherently difficult and usually based on the designer expertise and domain knowledge.

While a large variety of approaches have been proposed to implement the forward path, especially in the performance analysis domain [3, 24], there is still lack of automated solutions for the backward path. Introducing automation in this step goes beyond the mere advantage of achieving automated refactoring. In MDE, bidirectional model transformations are a key mechanism to propagate changes and maintain the consistency of related models.

During my PhD program, we intend to study how to introduce such automation (as described above) with the aim to contribute to the model-driven dependability assessment [11] at the architectural level [29, 34]. Dependability is a term that encompasses several non-functional properties as defined in [6]. In our view, a key element for automating the backward path is the adoption of bidirectional model transformations. Hence, we rely on the Janus Transformation Language (JTL) [14], that is a bidirectional model transformations framework specifically designed to support model synchronization and change propagation. However, for this goal we need to extend JTL to tackle current challenges (such as tool support, framework expressiveness, non-determinism, consistency restoration) emerging from the application of bidirectional model transformations in such context. Furthermore, JTL can be exploited to explore design solutions emerging from the propagation of changes and that may not have been considered in the first place.

2 RELATED WORK

A considerable number of approaches have been proposed in the last few years to derive analysis models from annotated software models.

One of the first attempts at enriching a UML design to specify dependability aspects can be found in [12]. The authors extend UML with information needed to automatically generate Stochastic Petri Net models for dependability analysis. In the proposed approach, the authors derive high-level SPN models from UML structural diagrams and successively use UML behavioral specifications to refine the generated analysis models. The transformation relies upon an intermediate model, and no standard UML profiles are used since none were available at the time of publication. While this represents a seminal work for the design of forward transformations, we intend to rely only on standard UML profiles to exploit existing tools. A transformation between UML statechart diagrams and Stochastic Reward Nets (SRN) is proposed in [26]. The transformation is defined as a mapping over a set of SRN patterns and used to conduct a performance and dependability analysis under erroneous state and faulty behavior assumptions. The idea of matching patterns of the analysis model is particularly interesting to us and we expect to further explore it. In [28], another mapping is proposed between a probabilistic extension of statecharts and a Markov chain model for quantitative assessment of safety and reliability. We also intend to employ probabilistic extensions of UML diagrams as the starting point for the assessment of dependability. In [9], the authors present a transformation able to generate Generalized Stochastic Petri Nets (GSPN) from UML sequence, statechart and deployment diagrams. Software models are annotated using the former standard UML SPT profile. The authors show how to obtain an analysis model by gathering information from different source software model. However, no mention is made about how these models should be kept synchronized after changes.

In the context of model synchronization, the formal definition of a round-trip engineering process between models representing different views can be found in [25]. Despite the authors consider the general case of partial and non-injective bidirectional model transformations, modifications to target models are considered valid only when they can be translated into a corresponding exact change to the source model. In [4], the authors introduce an incremental model transformation to propagate changes from a UML software model annotated with MARTE to the corresponding Layered Queueing Network (LQN) performance model. Changes are detected with the Eclipse EMF Compare tool and the incremental transformation is implemented with the Epsilon Object Language (EOL). While the detection of changes is automated, the change propagation requires the additional effort of defining actions to be performed for each type of detected model difference.

The authors of [32] propose an approach to backward change propagation of view models, that are models derived from an underlying domain model by a unidirectional forward transformation. Changes are propagated back by a delta-based backward transformation using a logic solver (Alloy Analyser) to derive change candidates to be selected by the designer. When dealing with model uncertainty, intended as the set of possible admissible models, [30] describes how the problem of change propagation arises when changes in the level of uncertainty result in further model changes. The authors formally define the mechanism of change propagation and present two change propagation algorithms based on an SMT solver. Although both these works demonstrate the effectiveness of logic solvers on propagating changes, the problem of presenting the designer with multiple solutions remains open.

A study comparing two alternative methods to derive software changes from the results of a model-based performance analysis is presented in [5]. Refactoring can be performed either on the software side by detecting and solving performance antipatterns, or on the analysis model using bidirectional model transformations to induce software changes. Both represent valid alternative processes but, while several approaches have been proposed to perform the
refactoring on the software side, there is still lack of solutions for the propagation of changes performed on analysis models.

## 3 PROPOSED SOLUTION

In this section we present our proposed solution to the problem stated in Section 1. We briefly introduce the JTL framework as the main technological component at the basis of our solution, and then we show how we intend to exploit it in order to implement a round-trip process for the model-driven assessment of software dependability.

### 3.1 Overview of the JTL framework

JTL [14, 20] is a constraint-based bidirectional model transformation framework specifically designed to support model synchronization and change propagation. JTL transformations allow a declarative specification of relationships between MOF models. The semantics is defined in terms of the Answer Set Programming (ASP) [23], a form of declarative programming oriented towards difficult (primarily NP-hard) search problems and based on the stable model (answer set) semantics of logic programming. Transformations are rule-based and the JTL syntax supports object pattern matching. A transformation is specified as a set of \( \text{relations} \) that must hold between matched elements in the source and target models. Each relation is composed by two \( \text{domains} \) and may include \( \text{when} \) and \( \text{where} \) predicates to specify pre- and post- conditions to be satisfied by elements of the candidate models. A bidirectional transformation is executed in a specific direction by selecting one of the candidate models as target. Using a deductive process, the JTL engine generates, in a single execution, all the possible models that can be derived from the application of the rules.

**Figure 2: Overview of the JTL framework**

An overview of the JTL framework architecture is reported in Figure 2. The engine executes bidirectional transformations written in the JTL syntax and later automatically transformed into ASP programs. During the execution, source and target metamodels of the defined transformation \( M_1, M_2 \) are automatically encoded in ASP and provided as input of the engine along with the source model \( M_1 \). The DLV [27] solver is employed in this phase to execute a transformation and obtain the resulting models (stable models) as a set of ASP facts. A trace model relating source and target model elements is also produced during the transformation \( (T_{M1M2}, T_{M2M1}) \). Trace models conforming to the trace metamodel \( TTM \) are automatically encoded in ASP as well, and can be optionally provided as additional input of consecutive executions. The framework employs an advanced traceability mechanism able to derive different types of trace links during the transformation execution: (i) usual trace links, mapping target elements to the source elements they are generated from; (ii) partial links, recording elements in the source model that are not matched by any relation; (iii) non-injective links, relating elements matched by a non-injective relation.

The traceability mechanism and the capability of leveraging a logic solver to automatically derive candidate target models are key features in the selection of JTL as the pivotal technology we intend to exploit in our solution. Another important aspect is interoperability with existing modeling technologies. This is the reason why JTL is implemented as a set of Eclipse plug-ins\(^2\) taking advantage of the widely used Eclipse Modelling Framework (EMF) [13].

### 3.2 Round-trip dependability engineering

The general process for the assessment of non-functional properties, represented in Figure 1, is instantiated in Figure 3 in the dependability context to illustrate our approach.

A software model is intended as a description of the system under design, defining structural and behavioral views. This model is provided by the software designer and it is expected to be annotated with dependability attributes, such as: type and number of physical hosts on which a software component is to be deployed, failure rates of components and communication channels, and probabilities of executing a specific behavior. The initial software model \( M \) will be the input of the forward execution of a bidirectional model transformation, together with the source metamodel (software domain) and the target metamodel (dependability domain). By executing the transformation, the JTL engine will produce as output a single dependability model \( DM \) corresponding to the initial software model. The dependability model could be any model having formal semantics to be used for the validation of dependability requirements, given that the appropriate JTL transformation has been defined. Models usually employed for this task are state-based models, like Markov chains and Stochastic Petri Nets, or path-based models like Fault Trees and Event Trees.

At this stage, a dependability expert will compute dependability metrics on the model obtained by the forward execution of the transformation. When the dependability analysis is completed and the dependability model is modified until requirements are met \( \Delta \), a refactored dependability model \( DM' \) is obtained. This model will be the input to the backward execution of the same bidirectional transformation used to generate the initial dependability model \( DM \). The JTL engine will generate as output a set of candidate refactored software models \( M_1', ... , M_n' \). Elements, attributes and references of \( DM \) that were not modified by \( \Delta \) are restored in each candidate solution as they were, and this is achieved by exploiting the traceability mechanism embedded in the JTL engine. Modifications matched by bijective relations are propagated in each target model. On the other hand, changes matching non-injective relations induce the generation of alternative models, each containing

\(^2\) JTL: http://jtl.di.univaq.it
We intend to validate our approach by applying it to a consider-
we intend to investigate is how to present a set of candidate so-
will produce only one candidate solution.

Since change propagation from analysis to software models may
consist in the generation of multiple design alternatives, an aspect
we intend to investigate is how to present a set of candidate so-
lutions to the software designer. A possible solution could be to
represent a set of models as one unified model explicitly depicting
alternative elements and features belonging to single candidate solu-
tions. Some work has been made in this direction [20] by adopting a
model for uncertainty to represent the output of a non-deterministic
transformation. Dealing with a single uncertainty model, instead of
a potentially large set of models, represents a clear advantage for
the designer. However, a major challenge is to study the process
of selecting or excluding suggested alternatives throughout the
software development lifecycle.

4 PLAN FOR EVALUATION AND VALIDATION

We intend to validate our approach by applying it to a consider-
able number of case studies. Specifically, we are interested in the
application to case studies by different sizes, which induce different
dependability models and on which we can test changes performed
on both the structure and the parameters of the models. We are
interested in determining in which practical cases such an approach
can effectively scale without loosing accuracy. It therefore becomes
essential to identify such case studies not only within the related
literature but also from industrial applications. On one side, we plan
to integrate the JTL engine in the MDEForge software-as-a-service
modeling platform [7] in order to base on the large number of mod-
els and metamodels contained within its repository. On the other
side, we want to take advantage of case studies proposed by com-
panies participating in the MegaM@Rt² european project ³, which
aims at creating a framework for continuous development and run-
time validation of complex systems. Finally, our research group has
been recently involved in a research project aimed at re-designing
the automated systems for controlling the transit of trains across
the Italian railway network (i.e., Rete Ferroviaria Italiana). We in-
tend to apply the proposed approach on the model-based design
of such system, especially considering its stringent dependability
requirements.

5 RESEARCH QUESTIONS AND EXPECTED
CONTRIBUTION

In this section, we discuss the main research questions that we
intend to address in the context of this PhD program.

RQ1: Can we identify changes in a software model that pro-
duce an improvement of dependability indices?
This is a central question in the research area of model-based
dependability assessment at design time. Our approach can help the
designer to identify software components that mostly affect the
system dependability. The identification of these critical compo-
nents can lead to a more targeted and efficient testing phase. In the
long run, this can contribute to a cost-effective construction of de-
dependable software components. Moreover, providing the designer
with the possibility of choosing among alternative architectural
changes, it can facilitate the exploration of dependability tradeoffs.

RQ2: How does an approach based on refactoring analysis
models differ significantly from an approach based on refac-
toring software models?
In order to support the designer in such complex task, a round-trip
process should be highly automated. In this respect, transformations
from software to dependability models have been fully automated
in the last decade even for high complexity cases. Moreover, bidirec-
tional model transformations are complex to build in this domain,
mostly due to mappings that collapse several elements of a software
model into a single element of a dependability model, and there-
fore being inherently non-injective. These are some of the reasons
why most approaches prefer to refactor the software model. Any-
way, refactoring actions performed on dependability models are
traditionally very well-supported and grounded in well-established
theory. In addition, refactoring the software model usually leads to
a prolonged iteration of the round-trip process. This is due to the
need of applying a forward transformation to a refactored software
model for obtaining a refactored dependability model to compute the
effect of the refactoring on dependability. The automation of the
backward path can actually support to refactor the same models
used for the assessment of dependability.

RQ3: Is our approach applicable to consistently propagate
to a variety of dependability case studies?
After modifying an analysis model obtained from a forward trans-
formation, reversing the mapping in order to obtain one or more
new refactored software models is a challenging task. In fact, the
automation introduced in this step should guarantee a consistent so-
olution. Although the adoption of automated mechanisms is crucial,
this automation makes sense only when the changes introduced
in the analysis model can be mapped to a (set of) desired soft-
ware model correctly translating the refactoring policy. Therefore,
backward execution becomes not only a means of model synchro-
nization, but above all a way of imposing a refactoring policy that
is consistent with the decisions made in the analysis domain. The

³https://megamart2-ecsel.eu
changes identified by the dependability analyst must be translated into modifications that produce the expected improvement and that are satisfactory for the software designer. The role of transformation designers is therefore fundamental, because they will impose a specific interpretation of the changes from which candidate solutions are generated. In this regard, the proposed approach investigates how a bidirectional transformation framework expressiveness can support the propagation of changes in the dependability context.

RQ4: How the human role can change in the round-trip dependability engineering process?
So far, most of the activities involved in a round-trip process require experience and skills of software designers and dependability analysts. While a good level of automation is supported in the problem detection and solution step on dependability models, the same cannot be said for software models where automation has been introduced only recently, for example based on antipatterns detection [18]. Moreover, in both cases it may be still necessary to decide among alternative refactoring solutions, especially in large scale systems. Therefore, even if we are convinced that it is necessary to explore automated solutions to the problem, the human role remains fundamental. Any decision to automatically propagate changes to a unique design alternative may not consider central aspects that require human decision-making, such as long-term benefits, enforcing of company policies, and development team skill sets. Our approach shall introduce a fair level of automation in the process to provide enough guidance in the choice of refactoring actions but, at the same time, it shall leave enough freedom to the designer dedicated to the (above-mentioned) human-driven aspects. Therefore, this study can be a starting point to investigate how much human intervention is actually needed during such process and where it should be required.

6 CURRENT STATUS AND PLANNED TIMELINE
An early attempt at providing automation in the round-trip assessment of non-functional properties was presented in [19]. We used JTL to define a bidirectional model transformation between a software architecture designed using UML [2] annotated with the MARTE profile [1] and Queueing Networks (QN). The software architecture was defined by means of UML Use Case, Component and Sequence diagrams. The transformation was applied to a simple case study from which we were able to generate a QN. The obtained QN was successfully employed to conduct a performance analysis and the consequent refactoring to remove a performance bottleneck. Changes performed on the QN were back propagated inducing modifications in both the UML Component and Sequence diagrams.

Since this first attempt, the development of JTL was very active, bringing the initial prototype to a fully integrated framework based on the Eclipse Rich Client Platform (RCP) \(^\text{4}\). We completely reengineered the semantic anchoring between the JTL syntax and the ASP engine, providing full interoperability of the DLV solver system with EMF and introducing a more advanced traceability mechanism. We presented this work in a tool paper [21] demonstrating the JTL progress on the Collapse/Expand State Diagrams benchmark.

In [15] we introduced a more recent advancement in the context of round-trip assessment of availability at the architectural level. We defined a bidirectional model transformation between UML State Machines annotated with the MARTE-DAM [10] profile and Generalized Stochastic Petri Nets (GSPN). The transformation we presented is based on a unidirectional forward transformation between UML Statecharts and GSPN formally defined in [8]. In order to provide change propagation, we extended this approach by implementing and executing a bidirectional transformation in the JTL framework. We demonstrated the effectiveness of our approach on an Environmental Control System case study to which we applied well-known fault tolerance patterns with the objective of improving its software availability.

In [22] we proposed a traceability mechanism for bidirectional model transformations to achieve better model management in scenarios like round-trip engineering, synchronization, or consistency restoration. By considering bidirectional transformations to be, in general, partial and non-injective, we designed a trace metamodel able to represent such information. Trace models are automatically generated alongside target models during the transformation execution, so that the user can easily refine and reuse trace links in further stages. We realized this traceability mechanism within the JTL framework and we validated it on the Families to Persons case that was proposed at the Transformation Tool Contest (TTC) 2017 \(^\text{5}\). We showed how such traceability mechanism can support consistency restoration in various benchmark tests that other bidirectional transformation tools were not able pass.

From preliminary results, it seems that the proposed approach could actually be suitable to tackle the problem.

This PhD program has started in November 2017, hence it is in its first year. During the remainder of the first year, we plan to continue investigating open challenges resulting from the application of model-driven techniques in the field of dependability assessment.

In the second year, we will focus on issues emerging when restoring consistency among multiple models at different abstraction levels, with the objective of understanding limitations in the propagation of changes by means of bidirectional transformations. Specifically, we are interested in how the framework expressiveness and non-determinism play a role in the design of bidirectional transformations. We also plan to apply the proposed approach to other attributes in the dependability domain while addressing their interactions, relationships and tradeoffs. We will continue our work on the JTL framework by extending it and providing a version specifically targeted at dependability engineering.

The PhD program shall be completed in the third year by consolidating and validating the proposed approach against real world case studies, extending the round-trip process to other notations on both software and dependability side, as well as considering other relevant attributes, such as performance and energy. By the end of the third year we plan to finalize the tool support for the automated refactoring and exploration of solutions derived from the backward propagation of changes.

\(^\text{4}\)Eclipse RCP: https://wiki.eclipse.org/Rich_Client_Platform

\(^\text{5}\)http://www.transformation-tool-contest.eu/2017/
