Journal of Object Technology
Published by AITO — Association Internationale pour les Technologies Objets, © JOT 2011
Online at http://www.jot.fm.

A Satisficing Bi-Directional Model Transformation Engine using Mixed Integer Linear Programming

Glenn Callow[a] Roy Kalawsky[b]

a. g.m.callow@lboro.ac.uk, Loughborough University

b. r.s.kalawsky@lboro.ac.uk, Loughborough University

Abstract The use of model transformation in software engineering has increased significantly during the past decade, with the ability to rapidly transform models and ensure consistency between those models being a key property of Model Driven Architecture. However, these approaches can be applied to a wide variety of different model types and some of these models and associated transformations require different semantics than those popularised by current model transformation tools. Specifically, current relational model transformation languages typically prioritise matching relation patterns in the source model over creating a target model that is compliant with its meta-model. In this paper we describe a relational model transformation engine implemented as a series of Mixed Integer Linear Programs (MILP). This engine has a key novel feature; it prioritises target model compliance with its meta-model by considering multiple interpretations of applying the transformation specification in order to ensure a correct target model is generated. In this paper the MILP transformation engine and the representations it uses are described, followed by the results of applying it to examples of varying complexity.

Keywords Model Transformation; Model Driven Architecture; Mixed Integer Linear Programming ; Domain Specific Models

1 Introduction

Model transformation languages, approaches and engines have been a research focus within elements of the software engineering community for some years. These offer a number of potential benefits, including increasing the speed at which new models and views can be generated (as large parts of the process are automated) and the utility of models (because the same source model can be reused multiple times as the source of a number of different transformations). Indeed, model transformation is one of the key elements of the OMG's Model Driven Architecture (MDA) [MM03]. MDA looks to specify models at varying levels of abstraction, supporting the reuse of models across a variety of different platforms by separating the platform independent elements from platform specific elements, and automatically transforming the relevant Platform Independent Models (PIM) into Platform Specific Model (PSM) variants. This has led to a variety of different languages and approaches being defined with differing strengths and advantages of which there are many good summaries, such as [CH06] [GK10] [JK06]; it is not appropriate to revisit all of these languages here. However, one common element of these languages is that they are largely targeted at the software engineering community, and specifically either the generation of code from models or the transformation of general models into specific models (the PIM to PSM transformation). Model transformation is not unique to software engineering, and so determining how applicable these model transformation languages are to other model types is an important consideration. If we consider more general system models[1], equivalents to relation based model transformations can be seen in system verification approaches, such as Quality Function Deployment (QFD) models [Hau88] or MODAF Function to Operational Activity mappings [Bai05]. For system models it is not unusual for changes to be made to multiple domains at the same time. This means that bi-directional transformation specifications [CFH$^+$09] [HSST11] are of particular relevance, because the transformation specification between domains A and B does not have to be modified irrespective of whether domain A or B is the target domain. Incremental model updates in any of the domains are easily propagated to other domains as required.

Earlier work has considered whether model transformation languages offer a benefit when used with these types of models [CKWO11]. This work concluded that relational model transformations do potentially offer a benefit, but that limitations in current model transformation engines prevented these benefits from being fully realised. In particular, existing relational transformation languages prioritise source model matches over target model correctness, which can result in target models being generated that do not conform to their meta-model. In this paper, we describe an alternative approach to implementing a relational model transformation engine which addresses these problems. Section 2 presents an in-depth discussion on these existing model transformation languages and some of the identified problems. In section 3 the set-based model, meta-model and model transformation representations that are used in the Mixed Integer Linear Programming (MILP) based transformation engine are described, whilst section 4 describes the MILP problems in detail. The implementation of the MILP system is discussed in section 5, and results when using the system are presented in section 6. Finally, in section 7, conclusions are drawn and future work identified.

2 Background

Model transformation languages can largely be categorised into two types; Imperative/Operational and Relational/Declarative. In imperative approaches, such as QVT-Operational [OMG08a] or the OMG's Model To Text transformation language

[1]By 'system model' we mean models that describe hardware, software, system functionality, interfaces, components and requirements. The authors have a particular focus on models that support system verification, i.e. assessing whether the system, as specified, is capable of achieving the requirements associated with it.

[OMG08b], a transformation specification is typically specified as a set of modules or rules which match a source module construct (e.g. a class instance), and a list of commands which, when executed, initialise the appropriate target model constructs. These languages work well where the target model will be automatically generated, and where the emphasis is to *prioritise the source model* when generating the target model. That is, when presented with a source model and a transformation specification, an imperative transformation matches all the rules it can against the source model and executes the associated commands. This is irrespective of whether this creates a target model that is or is not compliant with its associated meta-model.

Declarative or Relational model transformations promote a different approach. Instead of rules containing a source model construct and lists of commands for creating target models, they contain both source and target model constructs or patterns, represented as a set of consistency relations. These are examined by a model transformation engine to determine if they hold given the supplied models. If a consistency relation does not hold, the transformation engine determines what modifications are required to allow the consistency relation to hold. Examples of languages that are declarative or relation based include QVT-Relation [OMG08a] and Triple Graph Grammars [Sch95] [KW07].

The differences between the two approaches can be briefly summarised as follows: a) Imperative languages allow a modeller to specify *how* a target model should be created whilst relational approaches allow a modeller to specify *what* should exist in a target model; a relational model transformation engine will automatically determine how the model is constructed or modified based on the models and transformation specification. b) Whilst both imperative and relational model transformation languages support *model creation*, this change in approach allows relational model transformation engines to easily support additional capabilities, such as *model synchronisation* where updates on pre-existing source and target models are automatically propagated between them and *consistency checking* where a target model is checked to see if it is a valid transformation of a given source model. c) Relation based approaches are often bi/multi-directional. The same specification can be used to execute a transformation in different directions between the specified domains. The same is not typically true of imperative approaches.

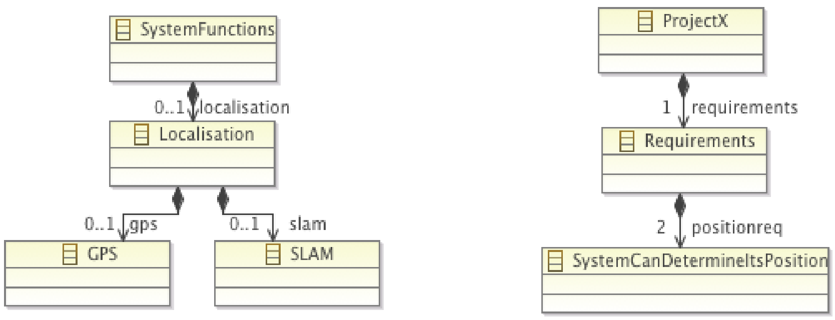
A pure relational approach to model transformations is potentially very powerful, but a key to the success of the approach is in the rigour with which the transformations can be specified, and the execution semantics of an associated model transformation engine. The more difficult it is for a modeller to interpret how a transformation engine will modify a target model to enforce consistency relations, the more challenging it is to write the transformation specification. If different engines interpret the same specification in different ways, this further increases the difficulty of writing a general transformation specification.

Whilst existing relational transformation engines consider what can be created in a target model due to relation matches in the source model, they rarely consider what is *not allowed* to exist due to restrictions placed on the target model by its meta-model. In this paper, this style of relational transformation shall be said to *prioritise the source model*. For example, the enforcement semantics for QVT-Relation (Appendix B in [OMG08a]) state that for each relation that correctly matches a source pattern in a source model, an equivalent update will be made to the target model, either by updating an existing object or creating a new object, as required. This behaviour is only disregarded when there is a clash with other transformation rules, meaning than an engine is allowed to create new instances even if those instances would violate constraints in the target meta-model. Considering an alternative relational approach, Triple Graph Grammars typically mandate *bind-exactly-once* semantics [GK10]. These semantics state that all source model elements must be bound during the model transformation; a transformation that does not bind all source model elements is marked as incomplete. This is a valid assumption for some, but not all, circumstances. There could be situations where it is only intended that a subset of the source model will be considered in the transformation. Triple Graph Grammars address this by requiring the creation of an alternative view of the source model which comprises only the sub-set to be included in the transformation.

The semantics of relational model transformation execution are also key to their utility. In Greenyer et al.'s comparison of QVT-Relation and Triple Graph Grammars [GK10] they investigated the different languages semantics, and in particular the associated determinism. How deterministic a relational model transformation is depends on how easy or difficult it is to specify ambiguous or conflicting relations, and how consistent the semantics are in the underlying model transformation engine in resolving these ambiguities or conflicts. For instance, Triple Graph Grammars allow non-deterministic transformations to be specified in the language but the semantics in how to address conflicts are not precisely defined. The general advice is simply to avoid non-deterministic transformation specifications where possible. This requires good design principles to be applied during the development of model transformations, and the thorough testing of transformation specifications. This is a nascent area, but there is work that can be drawn on, such as Heidenreich et al.'s approach to safe transformation composition [HKA11], Cabot et al's approach to both verifying and validating graph transformations by transforming them into OCL [CCG10], the *trans*ML developed by Guerra et al. [GdLK$^+$11] [GdLKP10] [GdLKP10] which is a modelling language designed specifically to support the development and testing of model transformation languages, and the work by Sen et al. on approaches to testing model transformations [SBM09] [SBM08].

Considering QVT-Relation, the problem of consistent semantics is exacerbated further. Specifically, the semantics of source model pattern matching and *check-then-enforce* object creation are not precisely specified and, as a consequence, different tools implement these semantics differently. Consider the meta-models in Figure 1. These meta-models describe two very simple but prescriptive domains. The first domain describes possible system functions. A model in this domain could have a *Localisation* function implemented by one (or both) of two underlying technologies; a GPS based sensor or a Simultaneous Localisation And Mapping (SLAM) software component. The second meta-model describes a requirements domain; it describes capabilities required for a particular system or task. The meta-model in this example is very precise; there is only one valid model that will satisfy the meta-model. A system must have two means of determining its position in order to satisfy the localisation requirement.

A model transformation will be used to generate a requirements model (which describes what the system can achieve) from the available system functions. If the generated requirements model satisfies the meta-model, then the system has the necessary functions to achieve the requirements. Instances of these meta-models and associated models for real systems would obviously be much more complex than portrayed in this example. However, the problems elaborated in the rest of this section would also manifest themselves in more complex models.



(a) System Functions Meta-Model (b) System Requirements Meta-Model

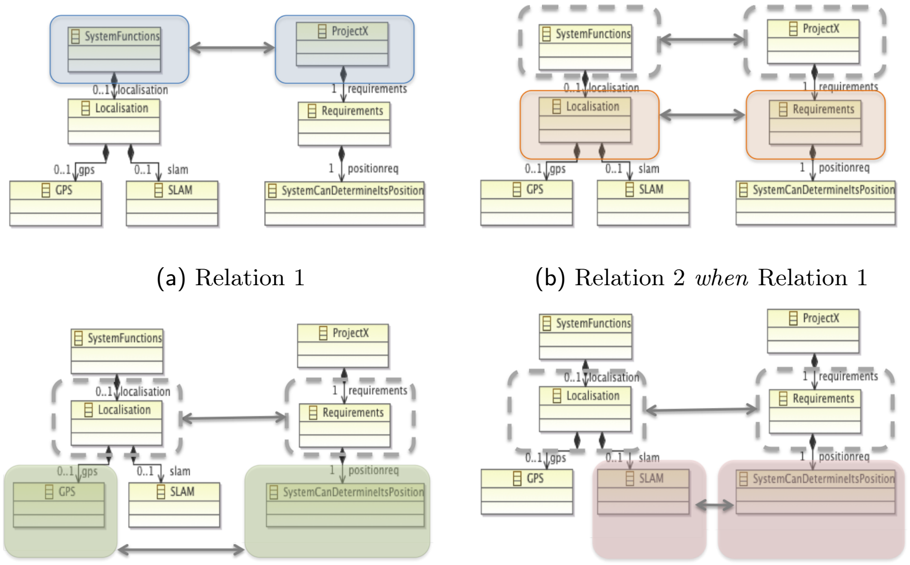
Figure 1 – Simple system's meta-models representing functionality (left) and requirements (right) domains

The transformation specification is shown graphically in Figure 2 and the actual QVT specification used is presented in Appendix A. Four relations are specified. Relation 1 acts similarly to the *axiom* in Triple Graph Grammars; it provides a root for the other relations. Relation 3 realises a *System Can Determine Its Position* requirement if GPS functionality is present in the system. Relation 4 realises a *System Can Determine Its Position* requirement if SLAM functionality is present in the system.

Consider the source model in Figure 3. All of the specified relations are *top* relations. The QVT-Relation specification states that *"The execution of a transformation requires that all its top-level relations hold"*. Therefore in our example, valid instances which can be bound to the target domain pattern must be created for all the relations; there is a corresponding source domain pattern for each of them. To determine what target model instances should be created, the enforcement semantics for QVT-Relation need to be considered. The QVT-Relation specification states that *"if there does not exist a valid binding of the remaining unbound variables of domain k that satisfies domain k's pattern and where condition, then create objects (or select and modify if they already exist) and assign properties as specified in domain k pattern."*. Whether an existing object should be selected, or whether a new object should be created (*check-then-enforce* semantics) is somewhat ambiguous. In the enforcement semantics the specification states *"Whether an object is selected from the model or created afresh depends on whether the model already contains an object that matches the key property values, if any, specified in the object template"*. Therefore, with no key statements specified in the transformation specification, a valid interpretation is that one instance of *ProjectX* class will be created from the first relation, one instance of *Requirements* will be created from the second relation and two instances of the *System Can Determine Its Position* class will be created, one each from the third and fourth relations. This model, if generated, would be compliant with the meta-model.

Evaluating this example with the two primary tools that currently implement QVT-Relation, MediniQVT[2] and ModelMorf[3], different results are achieved. These

[2]Available at http://projects.ikv.de/qvt - last checked 30/3/12
[3]Available at http://www.tcs-trddc.com/trddc_website/ModelMorf/ModelMorf.htm - last checked 30/3/12



(a) Relation 1

(b) Relation 2 *when* Relation 1

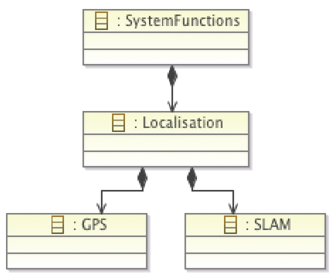
(c) Relation 3 *when* Relation 2

(d) Relation 4 *when* Relation 2

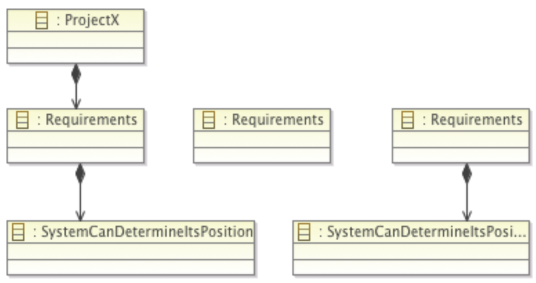
Figure 2 – Graphical representation of four consistency relations between meta-models. Coloured elements show non-dependent elements of a consistency relation. Elements bordered with a dashed line are dependent on another consistency relation through the *when* clause. The full QVT-Relation transformation specification can be found in Appendix A.

results are shown in Figure 3. MediniQVT produces a very unexpected model, with one *ProjectX* instance, three *Requirement* instances and two *System Can Determine Its Position* instances. The *Requirement* instances are not correctly associated, and therefore the model is invalid given the meta-model. This appears to be due to a bug in MediniQVT's interpretation of the *when* clause for the specified relations. ModelMorf, which is generally considered to be a more faithful implementation of the QVT-Relation specification [Ste11], fails to generate a target model with this transformation specification. This is because ModelMorf requires that a single relation be able to fully realise a compliant target model instance; e.g. a single relation that creates a *Requirement* instance must also create two *System Can Determine Its Position* instances. The fact that the required number of instances would be created by a combination of relations is ignored. This behaviour is not explicitly required by the QVT-Relation, nor is it explicitly excluded; instead it is simply how the developers of this tool have chosen to interpret the specification. One potential reaction to these results would be to modify the transformation specification to work with the semantics defined by the tool and allow the correct target model to be generated. However, in the ideal case this modification shouldn't be required; there is a valid interpretation of this specification so a transformation engine should be able to accommodate this. Requiring a transformation specification to be tailored to a particular engine implementation is a barrier to reuse, and increases the difficulty in writing a correct specification.

Improved approaches to engineering transformation specifications can go some way to mitigating these problems. For example, the correct use of *trans*ML [GdLK+11] during the design of a transformation specification would likely result in a transfor-



(a) Source Model For Transformation


: ProjectX
: Requirements
: Requirements
: Requirements
: SystemCanDetermineItsPosition
: SystemCanDetermineItsPosi...

(b) Target Model generated by MediniQVT

Figure 3 – Source model and target model generated by MediniQVT. The generated target model does not conform to the corresponding meta-model.

mation with fewer amibiguities (if that was desired), and one that can be tested in a principled manner using, perhaps, a set of synthesised test models using the approaches developed by Sen et al. [SBM09]. Whilst these approaches are undeniably important, they don't directly address the issues we have highlighted; namely that relational model transformations can have ambiguous interpretations, and that this ambiguity can actually be desirable by allowing an engine to choose the *best* matches based on the supplied models. Because of this, applying the work of Cabot et al. in verifying and validating graph transformations by generating an OCL based representation [CCG10] is more problematic because it makes implicit assumptions on the semantics of how a graph transformation will be applied. Whilst highly relevant and aiming for many of the same goals (e.g. ensuring that a generated target model is compliant with its meta-model, and is what the developer intended), their approach aims to determine whether a transformation specification is conflicting and ambiguous given these specific semantics.

An approach that does address ambiguity in transformations is the Janus Transformation Language (JTL) by Cicchetti et al [CDREP10]. JTL is specifically targeted at non-bijective transformations and has a novel capability amongst the transformation approaches considered in this section whereby it will guarantee that a generated target model is compliant with its target meta-model. The JTL has been modelled on QVT-Relations, and the execution engine utilises Answer Set Programming (a search technique designed for difficult search problems) to generate a correct transformation, given the matched relations in the source model and the imposed constraints of the target meta-model. A JTL transformation specification is itself transformed into an appropriate ASP representation, that can then be used as input to an ASP solver. JTL has many desirable properties that are required for the transformation of system models, the guarantee of compliance with the target meta-model being the most significant. However, there are several areas where we believe the JTL approach to relational model transformations can be built upon. In particular:

- The semantics of source model matching and handling relation dependencies are a key source of ambiguity in matching source relation patterns, and hence relational model transformations. Adding flexibility here may allow target models to be generated in circumstances where other approaches would reject them, as there are more potential source model matches available.

- Specifically allowing the model transformation engine the flexibility to instantiate only those relations that allow a target meta-model to be generated, even when there is a valid source model match for a relation. This would allow the engine to match only those relations that it needs to generate a valid target model.

The last point also opens up an interesting direction of work. If a model transformation engine is allowed to not utilise certain relations, then there is the potential for a transformation engine to only ever generate the simplest compliant target model, using only the minimum number of relations it needs to. Given this is probably not what a modeller intends, an additional mechanism to guide the transformation engine to a preferred target model out of all the possible compliant models is desirable.

3 Model, Meta-Model and Transformation Representations

We perceive two key problems preventing the practical use of current relational model transformation approaches. Firstly, Relational/Declarative approaches to model transformations should focus on the *what* should be generated, rather than *how*. Current languages do not readily support this. Knowledge of the detailed semantics about how a model transformation engine will apply the transformation is required to write a good specification and, in the instance of QVT-Relation, knowledge of the precise semantics of a particular implementation of a model transformation engine is required to write a transformation that will create a valid target model. The same models and specifications often cannot be used interchangeably. Secondly, as meta-models become more complex and precise, enforcing a single set of semantics becomes an increasing burden. There may be a natural semantic interpretation of a relational model transformation specification that would result in a valid target model but, because of the enforced semantics of the particular language or tool being used, a model is generated that is not compliant with the target meta-model.

In order to address this, we have developed a relational model transformation engine based on an underlying Mixed Integer Linear Programming (MILP) solver. Our approach has some similarities to work on UML model verification (i.e. determining whether a model is compliant with an associated meta-model) [CCR08] and partial model completion (i.e. determining the changes that need to be made to a model in order to make it compliant with an associated meta-model) [SBV10]. However, the work by Cabot et al. and Sen et al. have typically utilised SAT-solvers or derivatives thereof.

There are two novel aspects of our approach. Firstly, by applying a MILP solver we are able to take advantage of the fact that there are a large number of linear relationships present between elements specified in the models, meta-models and transformations. For model transformations, this means that many of the unknown variables (such as which relations have valid matches, which classes require instantiation and what relationships need to exists between those classes) are linearly related[4]. Using a MILP solver to find acceptable values for these distinct variables allows these relationships to be exploited to find a solution more efficiently. Secondly, a transformation engine based on a MILP solver can then exploit these relationships and improved efficiency to ensure that automatically generated target models are always

[4]the details of which depend on the details, such as allowable cardinalities, of those relationships.

compliant with their associated meta-models, if a solution exists, even from ambiguous model transformation specifications; it *prioritises target models*.

The transformation language the MILP transformation engine uses is heavily influenced by QVT-Relation and Triple Graph Grammars. The input to our Mixed Integer Linear Programming (MILP) solver shall be a set-based representation of models, meta-models and model transformation specifications, written using the Gnu MathProg Language (GMPL) [Mak10]. The set-based model and meta-model representations can be generated directly from ECORE-derived models in the Eclipse toolset. This is discussed further in section 5. A transformation specification must currently be written directly in GMPL, although this representation has been written with features of Triple Graph Grammars and QVT-Relation in mind. The remainder of this section shall focus on the set-based representations used for the solver.

3.1 Meta-Model Representation

Each meta-model consists of a set C of classes for a specific source or target domain, where each of these classes contain references or attributes to other classes or data-types. Each class $c \in C$ is further represented as a set P of 7-tuples where each tuple describes a particular reference or attribute as follows:

$$P_c = \{(n_1, c_1, y_1, l_1, h_1, u_1, o_1), ...\} \tag{1}$$

n describes the name of the reference or attribute, d is the type of attribute or reference endpoint, y defines whether the tuple refers to a containment reference, association references or attribute. l and h set the minimum and maximum cardinality for references and u is a boolean which sets the *isUnique* parameter for association references. Finally, o refers to an opposite association role name i.e. there exists an association reference between the two classes in the opposite direction with name o. If o is not *null* for an association reference then there will be an equivalent tuple in the P set for the target class. For example:

$$\begin{aligned} P_{c_1} =& \{(\text{class2end}, c_2, \text{assoc}, 1, 1, \text{true}, \text{class1end})\} \\ P_{c_2} =& \{(\text{class1end}, c_1, \text{assoc}, 1, 1, \text{true}, \text{class2end})\} \end{aligned} \tag{2}$$

Therefore class *ProjectX* as shown in Figure 1 is defined as follows:

$$P_{\text{ProjectX}} = \{(\text{requirements}, \text{Requirements}, \text{con}, 1, 1, \text{true}, \text{null})\} \tag{3}$$

3.2 Model Representation

Models are similarly represented as a series of sets. First, a set of class instances I is defined which contains (i, c) instance/class pairs. i refers to a Unique Identifier (UID) associated with each instance in the model. References and Attributes are represented slightly differently from the meta-model representation. Three distinct sets are used to capture these elements. I_{co} represents containment references as a set of 5-tuples (i_1, c_1, i_2, c_2, r) where c_1 is the class that contains the containment reference (as specified in the meta-model), r is the reference name, i_1 is the class instance that contains the containment reference, and i_2 is an instance of class c_2 which is the containment reference endpoint type. Association references are also described using a similarly defined set I_{as} of 5-tuples. Attributes are represented as a set I_{att} of 4-tuples (i, c, a, v) where i is the instance of c that contains attribute a, which has value v.

3.3 Transformation Representation

It is our intention to mirror the features of existing relational model transformation representations where possible. In particular, the transformation representation for the MILP solver has been designed to closely follow the properties of QVT-Relation. Let us consider the QVT-Relation specification shown graphically in Figure 2. This specification contains four relations, which relate two domains. All of these relations are *top* relations and contain patterns for each of the domains that they reference. These patterns should be subsets of the respective domain's meta-model, and each pattern starts from a particular class which acts as the root of that pattern. These domains can then follow containment references from the root class (through the nesting of classes within braces), follow associations through their roles or reference named attributes. Therefore, a natural way to represent relations is using a variant of the meta-model and model representations, with extensions to accommodate the variables that shall be bound to class instances for each pattern match. A transformation specification is made up of a number of relations. A set R_{rels} is used to capture the number of relations in a given specification:

$$R_{\text{rels}} = \{1, 2, ..., n_{\text{max}}\} \tag{4}$$

where n_{max} is the number of relations in the transformation specification. For the remainder of this paper, the following notation shall be used. Each set associated with the transformation specification shall have a super-scripted element. A single value super-script represents either the domain identifier or, if a scalar value, the relation identifier $n \in R_{\text{rels}}$ within the transformation specification. A 2-tuple superscript (n, s) describes both the relation and domain identifier. A 3-tuple superscript (n, j, s) adds a relation instance identifier j of relation n. So:

- R^1_{top} refers to set R_{top} for relation 1 in the transformation specification.
- $R^{(2,\text{train})}_{\text{cre}}$ refers to set R_{cre} which contains elements of a pattern associated with domain *train* of relation 2.
- $Q^{(2,1,\text{uml})}_{\text{inst}}$ refers to set Q_{inst} which containes elements associated with domain *uml* for relation instance 1 of relation 2.

Subscripts will be used to identify sets capturing particular aspects of the transformation (classes, containment references, etc) as per the model representation described previously. For each relation n, which relates domains s and t the following sets are defined. Firstly, whether the relation is a *top* relation is defined with the following set containing a single boolean.

$$R^n_{\text{top}} = \{[true|false]\} \tag{5}$$

Each relation has a number of variables v associated with it. These will be bound to class instances in the subsequent transformation. The variables are relation and domain specific.

$$\begin{aligned} R^{(n,s)}_{\text{bind}} &= \{v^s_1, v^s_2, ..., v^s_p\} \\ R^{(n,t)}_{\text{bind}} &= \{v^t_1, v^t_2, ..., v^t_q\} \end{aligned} \tag{6}$$

Then, the root class for the pattern needs to be defined for each domain referenced by the relation.

$$\begin{aligned} R^{(n,s)}_{\text{root}} &\subseteq \{c^s | c^s \in C^s\}, & \left|R^{(n,s)}_{\text{root}}\right| &= 1 \\ R^{(n,t)}_{\text{root}} &\subseteq \{c^t | c^t \in C^t\}, & \left|R^{(n,t)}_{\text{root}}\right| &= 1 \end{aligned} \tag{7}$$

The next two sets define the classes and associated patterns for each relation. This is one area where the GMPL representation deviates from the QVT-Relation specification and instead looks to Triple Graph Grammars. For each relation, it is explicitly specified whether an instantiation of that relation should be permitted to create an instance of that class, or whether that relation instance will be dependent on instances created by another relation which are bound through the *when* clause. The *when* clause is used in QVT-Relation to describe constraints or relations which must hold in order for this relation to be evaluated. It acts as a guard, but also has the effect of binding variables to values which have been bound by other relations. QVT-Relation therefore implicitly determines whether a relation is permitted to create relation instances, or whether it relies on other relations being instantiated to create those instances. Triple Graph Grammars are more explicit in their rule definition about these dependencies. Making these dependencies explicit in the transformation representation allows some of transformation semantics, such as how patterns should be matched or created, to be elaborated more formally. Therefore, for each relation n which relates domains s and t the following sets are used for classes which should be *instantiated* by relations that hold.

$$\begin{aligned} R^{(n,s)}_{\text{cre}} \subseteq &\quad \left\{(v^s, c^s) \in \left(C^s \times R^{(n,s)}_{\text{bind}}\right)\right\} \\ R^{(n,t)}_{\text{cre}} \subseteq &\quad \left\{(v^t, c^t) \in \left(C^t \times R^{(n,t))}_{\text{bind}}\right)\right\} \end{aligned} \tag{8}$$

Each of these sets consist of (v, c) pairs which describe the class c to match in the pattern. Each relation pattern that is matched to an element within the source model will have the corresponding instance bound to variable v. The sets for classes that make up the relation pattern, but which are dependent on those instances being realised through other relations are specified similarly. The set R_{all} will be used to describe all the classes associated with a particular relation domain pattern. The same variable cannot be used within a relation to refer to both a class instance that must be instantiated as part of realising this relation, and to a class instance that will be realised by another relation.

$$\begin{aligned} R^{(n,s)}_{\text{dep}} &\subseteq \left\{(v^s, c^s) \in \left(C^s \times R^{(n,s)}_{\text{bind}}\right)\right\} \\ R^{(n,t)}_{\text{dep}} &\subseteq \left\{(v^t, c^t) \in \left(C^t \times R^{(n,t)}_{\text{bind}}\right)\right\} \end{aligned} \tag{9}$$

$$\begin{aligned} R^{(n,s)}_{\text{all}} &= R^{(n,s)}_{\text{dep}} \cup R^{(n,s)}_{\text{cre}} \\ R^{(n,t)}_{\text{all}} &= R^{(n,t)}_{\text{dep}} \cup R^{(n,t)}_{\text{cre}} \\ \emptyset &= R^{(n,s)}_{\text{dep}} \cap R^{(n,s)}_{\text{cre}} \\ \emptyset &= R^{(n,t)}_{\text{dep}} \cap R^{(n,t)}_{\text{cre}} \end{aligned} \tag{10}$$

Whilst these sets capture the classes that are present within the pattern, they do not describe the relationships between those classes that must also be captured. For that, separate sets are defined for containment and association references. Containment references are described using the following:

$$R^{(n,s)}_{\text{co}} \subseteq \left\{(v^s_1, c^s_1, v^s_2, c^s_2, r) \mid (v^s_1, c^s_1) \in R^{(n,s)}_{\text{all}} \wedge (v^s_2, c^s_2) \in R^{(n,s)}_{\text{all}}\right\} \tag{11}$$

Each containment relationship is captured as a 5-tuple which describes the containing class c_1, an instance of which should be bound to variable v_1. c_1 contains class c_2, an instance of which is accessed through the containment endpoint (role) named r. This instance is bound to v_2, and both of these pairs should exist in either R_{cre} or R_{dep}. An equivalent set $R^{(n,t)}_{\text{co}}$ exists for the target domain t. Associations are described similarly.

$$R^{(n,s)}_{\text{as}} \subseteq \left\{(v^s_1, c^s_1, v^s_2, c^s_2, r) \mid (v^s_1, c^s_1) \in R^{(n,s)}_{\text{all}} \wedge (v^s_2, c^s_2) \in R^{(n,s)}_{\text{all}}\right\} \tag{12}$$

The *when* and *where* clauses are the main mechanism by which dependencies are specified between relations in QVT-Relation. Therefore, for all the variables specified in R_{dep}, there should be an appropriate reference to an alternative relation as these instances must be created by another relation in the specification. The MILP solver only considers the *when* clause at present, and it is represented as follows.

$$\begin{aligned} R^n_{\text{when}} \subseteq \{(n_d, v^s, v^t) \mid\ & n_d \in R_{\text{rels}} \wedge n_d \neq n \wedge \\ & (v^s, c^s) \in R^{(n,s)}_{\text{dep}} \wedge (v^t, c^t) \in R^{(n,t)}_{\text{dep}}\} \end{aligned} \tag{13}$$

Therefore relation n is dependent on relation n_d, and the appropriate variables will be bound to class instances within a dependent relation. If no matches for relation n_d can be identified, then relation n cannot be instantiated; not all of the variables within the pattern can be bound.

Implementing precise semantics for variable binding and relation dependencies are a significant underlying cause of the complexity of relational transformations. Under what circumstances can variables be bound to the same class instance? Can a dependent relation have multiple variables bound to the same class instance in a source model, and does this require exactly the same structure be replicated in the target model? For the purposes of this paper, it shall be assumed that a variable may only appear once in a *when* statement for a particular relation.

$$\begin{aligned} \forall (v^s, c) \in R_{\text{all}}^{(n,s)} \sum_{(n_d, v^s, v^t) \in R_{\text{when}}} &\leq 1, \\ \forall (v^t, c) \in R_{\text{all}}^{(n,t)} \sum_{(n_d, v^s, v^t) \in R_{\text{when}}} &\leq 1 \end{aligned} \tag{14}$$

In order to support attributes, an additional element of the QVT-Relation specification is replicated in this set-based specification; relation-specific variables. These are variables that are independent of domain, and can be bound to particular values in any domain referenced as part of the relation. This makes them a convenient mechanism for transforming attribute values within the model. For each relation, a set contains each relation specific variable x and its associated type y.

$$R_{\text{var}}^{n} = \{(x_1, y_1), ...\} \tag{15}$$

For each domain, a set of 5-tuples is used to describe attributes associated with a particular class in a domain, and how they are bound to the relation specific variables.

$$R_{\text{att}}^{(n,s)} = \{(v, c, a, x, y, z), ...\} \tag{16}$$

where v is the domain variable which an instance of class c will be bound to, a is the attribute name in the class specification, x is the relation-specific variable, y is the attribute type of a and x, and z is a GMPL specific string which references and modifies the value associated with a. The GMPL string z is used to modify the attribute value for a particular domain; i.e. by concatenating an additional string, or carrying out an arithmetic operation[5]. The combination of the attribute value and z will be be used to determine x for the source model. The combination of x and z will be used to determine the attribute value for the target model.

There are aspects of MOF-specified meta-models and QVT-Relation that are not captured in the above set representation. Currently, the inheritance/generalisation relationship in not explicitly captured, although some of the common model structures associated with inheritance, such as multi-parent composition relationships, are supported. The QVT-Relation *where* element is not currently supported, nor are arbitrary OCL statements in *when* clauses. The support of *where* clauses should be a relatively straightforward addition, whilst the support of arbitrary OCL statements requires a means to automatically transform OCL into the GMPL representation. The *key* statement in QVT-Relation, which provides additional information for when target model elements should be reused, is not currently implemented. Finally, the wider problem of automatically transforming QVT-Relation specifications into the set-based representation should also be addressed in future work.

[5]See sections 3.1.5 and 3.1.9 in the GMPL manual [Mak10] for a summary of the arithmetic operators that could be used in z

4 Transformations using Mixed Integer Linear Programming

In order to transform a supplied source model into a target model, the set based representations described in the previous section are used as the basis of a series of Mixed Integer Linear Programs. A linear program is a sub-set of programming problems where the problem itself is made up of one or more linear relationships between the elements of the problem [Gas85]. These relationships are of the form:

$$a_1x_1 + a_2x_2 + ...a_ix_i + ...a_nx_n = b \tag{17}$$

where a_i's and b are known coefficients, and the x_i's are the unknown variables. These are the decision variables whose values will be determined in solving the particular linear programming problem.

The MILP transformation approach separates the problem into four distinct stages. Stages 2, 3 and 4 each require one MILP to be solved.

1. *Source Model Analysis* - Analyse the source model, given its meta-model and the transformation, using a simple graph matching algorithm and determine the maximum number of possible matches for each relation.

2. *Bind variables to source model instances* - Execute a Mixed Integer Linear Program which, for each relation, identifies a potential binding for all the variables from the source model. Every candidate set of variable bindings for each relation is a *potential relation instance.*

3. *Create target model instances* - Execute a Mixed Integer Linear Program which, for each relation, instantiates target model instances for each *potential relation instance*, providing the instances do not clash with another *potential relation instance* and that resulting completed model does not violate any constraints associated with the *target meta-model.* Any *potential relation instances* that have all of their target model variables bound to target model instances are *completed relation instances.* Any *potential relation instances* that are not fully realised are disregarded.

4. *Create containment and association references, and assign values to attributes* - The containment and association references for all class instances are initialised, given the *completed relation instances*, and attribute values set accordingly.

4.1 Differences with existing relational model transformation approaches

In order to *prioritise target models* a number of principles associated with existing model transformation engines must be modified. Firstly, a model transformation should be more permissive in how source model relations are matched. Triple Graph Grammars require that the entire source model (or appropriate view of that model) must be matched for the transformation to succeed. Subsets of source models must be created manually if that is not the case, and these are presented to the model transformation engine instead. An alternate system that prioritises target models should instead *determine* which aspects of the source model should be matched and will participate in the transformation. Secondly, the QVT-Relation specification requires that all *top* relations hold. If a source domain match is found for a *top* relation, then the corresponding target pattern must be created. A transformation engine that prioritises target models must relax this restriction; a *top* relation will instantiate the necessary target model pattern if there is a source model match *and* the creation of the target model elements will not ultimately result in an invalid target model. Thirdly, the more combinations, choices and alternatives for source model matches that are identified, the more flexibility the transformation engine has in finding a combination that allows a valid target model to be constructed. Overlapping bindings, where the same elements are used in different combinations in multiple relation instances, are actively encouraged. The system should choose which (if any) of these relation instances will be instantiated based on their effect on the target model.

4.2 Detailed Stage Description

4.2.1 Stage 1 - Source Model Analysis

The first stage of the algorithm provides a simple sub-graph matching algorithm to identify the maximum possible number of relation instances for each relation described in the model transformation specification. The sole purpose of this step is to provide an upper bound for the subsequent stages of the model transformation. Three simple algorithms determine the number of times a) the class instances, b) the containment references and c) the association references could be matched for each relation given the source model. Dependencies between relations are not considered at this point; each relation is considered in isolation. The minimum value from these three algorithms is then used to set an upper bound for each relation. This is the maximum number of *potential relation instances* that could be identified, given the source model. No actual binding of variables to class instances takes place at this stage. The number of *potential relation instances* under consideration in subsequent stages is represented for each relation n as follows:

$$Q_{\text{rels}}^{(n,s)} = \{1, 2, 3, ..., j_{\max}^{n}\} \tag{18}$$

where $j_{\max}^{n}$ is the maximum number of *potential relation instances* identified for relation n in stage 1. The Q set notation forms part of a correspondence model, and is explained further in the subsequent section.

4.2.2 Stage 2 - Bind variables to source model instances

To implement this stage as a MILP problem, three elements are required. 1) Decision variables or arrays must be described, which the solver is able to manipulate. 2) Constraints which reference these decision variables or vectors must be described; these bound the overall problem. 3) An objective function must be captured which specifies a metric which guides the solver.

The decision variables for this problem are relatively straightforward. The first set of binary decision variables, α, are captured within an array. The array maps variables within relations for a particular domain to class instances within the source model. For every variable within each *potential relation instance* identified after stage 1, the corresponding array element should be set to 1 if the class instance is bound to the variable for that relation instance after solving and zero otherwise.

$$\alpha_{(v,i)}^{(n,j,s)} = \begin{cases} 1 & \text{if } i \text{ is bound to relation instance } j\text{, var } v \text{ for relation } n \\ 0 & \text{otherwise} \end{cases} \tag{19}$$

A second decision vector β, is also used. This vector is used to explicitly state whether a relation instance is still a *potential relation instance* after this second stage, or whether one of the earlier relation instances identified during sub-graph analysis is now being disregarded. This could be because there is an unresolvable clash between relation instances when binding variables, that some of the relation dependencies could not be satisfied for that relation instance, or simply that stage 1 over-estimated the number of *potential relation instances.*

$$\beta^{(n,j,s)} = \begin{cases} 1 & \text{All of relation } n\text{, instance } j\text{'s variables are be bound} \\ 0 & \text{At least one variable in relation } n\text{, instance } j\text{'s could not be bound} \end{cases} \tag{20}$$

A series of constraints are required which define the semantics of the source model matching. These can be grouped into four categories; 1) ensuring class instances are correctly bound to variables, 2) ensuring these bindings satisfy any containment relationships specified in the relation, 3) ensuring these bindings satisfy any association relationships specified in the relation and 4) ensuring these bindings satisfy any dependencies between relations. Considering each of these groups in order.

Group 1 consists of three constraints. 1) If a relation is responsible for instantiating a class instance, then the source model semantic equivalent is that a class instance must only exist for one *potential relation instance* per relation n for classes specified in $R^{(n,s)}_{\text{cre}}$. Note - the class instance can be referenced in any number of dependent relation instances. This is shown in constraint 21. 2) For each *potential relation instance*, different variables cannot be bound to the same class instance (constraint 22). 3) For each *potential relation instance*, a variable must be bound to no more than one class instance (constraint 23). In this latter constraint, the comparison is made between α and β to ensure *potential relation instances* are either fully active or inactive in subsequent stages.

$$\forall n \in R,$$

$$\forall (v,c) \in R^{(n,s)}_{\text{cre}}, \forall (i,c) \in I^s \qquad \sum_{j \in Q^{(n,s)}_{\text{rels}}} \alpha^{(n,j,s)}_{(v,i)} \leq 1 \tag{21}$$

$$\forall j \in Q^{(n,s)}_{\text{rels}}, \forall (i,c) \in I^s \qquad \sum_{(v,c) \in R^{(n,s)}_{\text{all}}} \alpha^{(n,j,s)}_{(v,i)} \leq 1 \tag{22}$$

$$\forall j \in Q^{(n,s)}_{\text{rels}}, \forall (v,c) \in R^{(n,s)}_{\text{all}} \qquad \sum_{(i,c) \in I^s} \alpha^{(n,j,s)}_{(v,i)} = \beta^{(n,j,s)} \tag{23}$$

In group 2, the containment relationships are considered. This consists of two constraints. The first constraint (constaint 24) states that if a containment relationship is present in a relation, then a *potential relation instance* must contain class instances that are an appropriate container (i.e. they have an appropriately typed containment reference). The second constraint (constraint 25) focuses on the containment endpoint. A *potential relation instance* can only use a class instance as an endpoint if the containing instance is present as per the first constraint.

$$\forall n \in R, \forall j \in Q_{\text{rels}}^{(n,s)},$$
$$\forall (v_1, c_1, v_2, c_2, r) \in R_{\text{co}}^{(n,s)} \qquad \sum_{\substack{(i_1,c_1)\in I^s \mid \\ \exists (i_2,c_2)\in I^s \text{ s.t.} \\ (i_1,c_1,i_2,c_2)\in I_{\text{co}}^s}} \alpha_{(v_1,i_1)}^{(n,j,s)} = \beta^{(n,j,s)} \tag{24}$$

$$\forall (i_1, c_1) \in I^s \mid$$
$$\exists (i_2, c_2) \in I^s \text{ s.t. } (i_1, c_1, i_2, c_2) \in I_{\text{co}}^s \qquad \sum_{(i_1,c_1,i_2,c_2)\in I_{\text{co}}^s} \alpha_{(v_2,i_2)}^{(n,j,s)} = \alpha_{(v_1,i_1)}^{(n,j,s)} \tag{25}$$

Group 3 considers association references. These are similar to the containment constraints, with the only difference being the second constraint in this group. This comparison is different for associations than for containments due to the semantics of that relationship. For containments, if a variable is bound to a class instance in a relation as a containment target endpoint then only one class instance can be its container. There cannot be a situation where several class instances are the container for a single specific class instance as it would be an invalid model. However, for associations that is not the case; there could be several candidate association relationships where a class instance is bound to a variable as an association endpoint, with differing sources for that relationship. Some of these options may ultimately not be chosen when binding variables in this stage; the source variable will not be bound for that association reference.

$$\forall n \in R, \forall j \in Q_{\text{rels}}^{(n,s)},$$
$$\forall (v_1, c_1, v_2, c_2, r) \in R_{\text{as}}^{(n,s)} \qquad \sum_{\substack{(i_1,c_1)\in I^s \mid \\ \exists (i_2,c_2)\in I^s \text{ s.t.} \\ (i_1,c_1,i_2,c_2)\in I_{\text{as}}^s}} \alpha_{(v_1,i_1)}^{(n,j,s)} = \beta^{(n,j,s)} \tag{26}$$

$$\forall (i_2, c_2) \in I^s \qquad \sum_{(i_1,c_1,i_2,c_2)\in I_{\text{as}}^s} \alpha_{(v_1,i_1)}^{(n,j,s)} = \alpha_{(v_2,i_2)}^{(n,j,s)} \tag{27}$$

The final group consists of one constraint, and is concerned with the *when* dependencies. Specifically, if there is a variable[6] that is dependent on another relation then it must be ensured that a candidate dependent relation exists, and that one of the potential relation instances for that relation can be bound to an acceptable value for the dependent relation. If it can be, then both the parent and dependent relations must have the corresponding variables bound to the same class instance.

[6]Note, that the R_{when} set contains variables in both the source and target domains. The variable names in the tuples in R_{when} are compared with the variables in R_{all} for the chosen source domain to allow bi-directionality of the transformation specification.

$$\begin{array}{ll} \forall n \in R, \forall (v,c) \in R_{\text{all}}^{(n,s)}, & \\ \forall (n_p, v^s, v^t) \in R_{\text{when}}^{n}, \forall j \in Q_{\text{rels}}^{(n,s)}, & \\ \forall (i,c) \in I^s \mid (v = v^s) \vee (v = v^t) & \alpha_{(v,i)}^{(n,j,s)} \leq \sum_{\substack{(v_p,c) \in R_{\text{all}}^{(n_p,s)}, \\ j_p \in Q_{\text{rels}}^{(n_p,s)}}} \alpha_{(v_p,i)}^{(n_p,j_p,s)} \end{array} \tag{28}$$

To complete the MILP, an objective function is required. For the purposes of this paper, it shall be assumed the goal is to maximise the number of bound variables in *potential relation instances*. Other objective functions could be specified and these could give rise to interesting model transformations that are concerned with maximising or minimising some other property of the model.

$$max \left(\sum_{\substack{n \in R, j \in Q_{\text{rels}}^{(n,s)}, \\ (v,c) \in R_{\text{all}}^{(n,s)}, \\ (i,c) \in I^s}} \alpha_{(v,i)}^{(n,j,s)} \right) \tag{29}$$

This problem can now be passed through the solver, with the decision variables being set appropriately if a solution exists. When a solution is found, the decision variables are used to produce a series of new sets. These sets form the the source portion of a correspondence model Q, similar to that utilised by Triple Graph Grammars [Sch95] through its use of correspondence nodes. First, the number of *potential relation instances* still under consideration must be updated.

$$Q_{\text{rels}'}^{(n,s)} = \left\{ j \in Q_{\text{rels}}^{(n,s)} \mid \beta^{(n,j,s)} = 1 \right\} \tag{30}$$

This set defines which relation instances, up to the maximum number detected in the initial sub-graph matches, are still being considered as *potential relation instances*. Each one of these *potential relation instances* should have all of their variables bound, which is defined as a set of 3-tuples.

$$Q_{\text{inst}}^{(n,j,s)} = \left\{ (i,v,c) \in \left(I^s \times R_{\text{all}}^{(n,s)} \right) \mid \alpha_{(v,i)}^{(n,j,s)} = 1 \right\} \tag{31}$$

The correspondence model used also explicitly captures the relationship between dependent *potential relation instances*. Every dependent variable for a relation is mapped explicitly to a class instance. This same class instance must be bound to a variable within an instance of the parent relation. This is represented using the following set of 6-tuples:

$$Q_{\text{dep}}^{(n,j,s)} \subseteq \left\{ (v, n_p, v_p, j_p, i, c) \mid (v,i,c) \in Q_{\text{inst}}^{(n,j,s)} \wedge (v_p, i, c) \in Q_{\text{inst}}^{(n_p,j_p,s)} \right\} \tag{32}$$

For each relation instance j, each of its dependent variables v is mapped to the variable v_p in the instance j_p of relation n_p. These variables are bound to the instance

i of class c. Finally the relation variables, which are primarily used to capture and set attribute values, are set for each relation instance.

$$Q_{\text{vars}}^{(n,j,s)} = \{(x_i, y_i, m), ...\} \tag{33}$$

where x_i is the variable name, y_i is the variable type and m is the value assigned to that variable for instance j of relation n.

4.2.3 Stage 3 - Create Target Model Instances

At this point, a consolidated set of *potential relation instances* have been identified and the source components of a correspondence model have been constructed. The next step is to consider those *potential relation instances* and instantiate target model elements for non-conflicting relation instances that do not violate constraints associated with the target meta-model. For the purposes of this paper, only models that are fully compliant with the target meta-model shall be considered. If a model cannot be generated that complies with the meta-model, the transformation is considered invalid.

As with the previous problem, decision variables are required. Three related arrays of decision variables will be used in order to create the target model. The first decision variable, γ, specifies which of the *potential relation instances* will be fully realised as a *completed relation instance* in the target model. Each γ is described as follows:

$$\gamma^{(n,j,t)} = \begin{cases} 1 & j \in Q_{\text{rels}'}^{(n,s)} \wedge j \in Q_{\text{rels}}^{(n,t)} \\ 0 & j \in Q_{\text{rels}'}^{(n,s)} \wedge j \notin Q_{\text{rels}}^{(n,t)} \end{cases} \tag{34}$$

A second decision variable, ψ, is used to describe whether variable v bound to class c in each *potential relation instance* j of relation n should be instantiated.

$$\psi_{(v,c)}^{(n,j,t)} = \begin{cases} 1 & \text{an instance bound to } v \text{ in } n \text{ should be instantiated} \\ 0 & \text{an instance bound to } v \text{ in } n \text{ should not be instantiated} \end{cases} \tag{35}$$

Finally, a third decision variable τ_c^t describes how many class instances for each class c will be created for the model as a whole. The arrays of τ and ψ will be related through the constraints, with ψ being set appropriately to ensure the relations are properly satisfied, and τ being set appropriately to ensure the meta-model is satisfied.

Three groups of constraints will be considered. The first group is concerned with which class instances must be instantiated to satisfy a *completed relation instance*. The second group is concerned with which class instances must be instantiated to satisfy the target meta-model. Finally the third group relates the first two groups, and ensures both the relations and meta-model are satisfied.

The first group contains 4 constraints. The first constraint (constraint 36) is straightforward; for each *completed relation instance*, instantiate the required number of classes as specified in set $R_{\text{cre}}^{(n,t)}$. The second constraint (constraint 37) concerns relation dependencies. If a relation instance has been determined to be dependent on another relation instance in the previous stage, the dependent potential relation instance cannot be used if the parent relation is not used.

$$\forall n \in R, \forall j \in Q^{(n,s)}_{\mathrm{rels}'},$$

$$\forall (v,c) \in R^{(n,t)}_{\mathrm{cre}} \qquad \psi^{(n,j,t)}_{(v,c)} = \gamma^{(n,j,t)} \tag{36}$$

$$\forall (v,n_p,v_p,j_p,i,c) \in Q^{(n,j,s)}_{\mathrm{dep}} \qquad \gamma^{(n,j,t)} \le \gamma^{(n,j_p,t)} \tag{37}$$

The third constraint considers specifically the R_{cre} sets. Recall that the semantics for these sets is that relations in these sets are responsible for instantiating the referenced classes. The corresponding semantics when considering source model matching are that, for each class instance in the source model, only one relation instance can bind a specific class instance to a variable in those sets. However, multiple *potential relation instances* can have variables in a R_{cre} set bound to the same class instance. This allows the solver in this stage to choose the most appropriate *potential relation instance* to instantiate but, at this point, only one of those candidates can be completed. Therefore constraint 38 enforces this.

$$\forall (i,c) \in I^s \qquad \sum_{\substack{n \in R, \\ j \in Q^{(n,s)}_{\mathrm{rels}'}, \\ (v,c) \in R^{(n,s)}_{\mathrm{cre}} | \\ (i,v,c) \in Q^{(n,j,s)}_{\mathrm{inst}}}} \gamma^{(n,j,t)} \le 1 \tag{38}$$

The fourth constraint concerning relation dependencies is similar. In the previous stage, there may be mutually exclusive *potential relation instances* that have been identified. Different *potential relation instances* may have the same class instance as a containment endpoint with different containing classes. It would be invalid to instantiate all of them, so constraint 39 ensures that only one of these *potential relation instances* is completed.

$$\forall n_p \in R, \forall j_p \in Q^{(n_p,s)}_{\mathrm{rels}'}$$

$$\sum_{\substack{n \in R, j \in Q^{(n,s)}_{\mathrm{rels}'}, \\ (v_a,c_a,v_b,c_b,r) \in R^{(n,s)}_{\mathrm{co}}, \\ (v_b,n_p,v_p,j_p,i,c) \in Q^{(n,j,s)}_{\mathrm{dep}}}} \gamma^{(n,j,t)} \le \gamma^{(n_p,j_p,t)} \tag{39}$$

This set of constraints would, with a little modification, allow us to execute a model transformation that prioritises the source model as per a traditional model transformation engine. However, our goal is target model correctness and to ensure this the associated target meta-model must be considered. To achieve this, previous work on determining if a class diagram is satisfiable can be leveraged, such as that by Cadoli [CCGM07], Cabot [CCR08] and the completion of Domain Specific Models described by Sen [SBV10]. Therefore, the second group of constraints consists of four constraints which directly reference the meta-model.

This first constraint (40) states that for all classes that are instantiated in the target model and contain lower cardinality bounds for a containment reference, then there must be sufficient instances of the child class in the model (number of instances of the source class, multiplied by the lower cardinality bound). Similarly, if an upper cardinality bound exists (i.e. is not *) then there cannot be more child class instances than can be contained within the bounds (constraint 41).

$$\forall c \in C$$

$$\sum_{c_p \in C, (n,c,con,l,h,u,o) \in P_{c_p}} \tau^t_{c_p}.l \leq \tau^t_c \tag{40}$$

$$\sum_{c_p \in C, (n,c,con,l,h,u,o) \in P_{c_p} | h > 0} \tau^t_{c_p}.h \geq \tau^t_c \tag{41}$$

Association references are handled in a slightly different manner given the differing semantics. If the *isUnique* flag is set then there must be enough unique instances of the target class to satisfy the lower bounds of that association reference. If there is an *opposite* pairing for this reference then the number of classes that must be instantiated are linked; there must be sufficient instances to satisfy the relationship in both directions. However, if there is no *opposite* reference then *isUnique* does not prevent the same n target class instances being reused for multiple source instances; this would not be a violation of the target meta-model. In this case the number of instances required to satisfy an association endpoint is independent from the number of class instances that contain the source of the reference. If the *isUnique* flag is not set then the constraint is even weaker. If the lower bound of an association reference is greater then zero, then at least one instance of the target class must exist. That single instance can be used multiple times to satisfy the constraint.

$$\forall c_p \in C, \forall c \in C,$$

$$\forall (n, c, \text{assoc}, l, h, u, o) \in P_{c_p} \mid l > 0 \tag{42}$$

$$\tau^{(c,t)} \geq \begin{cases} l.\tau^{(c_p,t)} & \text{if } u \text{ is true, } o \text{ is not null} \\ l & \text{if } u \text{ is true, } o \text{ is null} \\ 1 & \text{if } u \text{ is false, } o \text{ is null} \end{cases}$$

Upper cardinality bounds for associations are not considered in these constraints. Having more candidate class instances than are required to satisfy all the association references is not a violation of the meta-model providing all of those class instances can be contained. Only the appropriate subset of class instances will participate in the association references.

Finally, group 3 relates the two previous groups of constraints and consists of one constraint only. The total number of class instances created for a model, represented through the array of τ, should be equivalent to the number of instances required to be realised for all *complete relation instances*, represented through the array of ψ.

$$\forall c \in C$$

$$\sum_{\substack{n \in R, \\ (v,c) \in R^{(n,t)}_{\text{cre}}, \\ j \in Q^{(n,s)}_{\text{rels}'}}} \psi^{(n,j,t)}_{(v,c)} = \tau^{(c,t)} \tag{43}$$

As with the previous stage, an objective function is required. The examples in section 6 shall utilise an objective function with maximises the number of source model class instances that are included in the final correspondence model.

$$max \left(\sum_{\substack{(i,c)\in I^s, n\in R, j\in Q_{\text{rels}'}^{(n,s)}, \\ (v,c)\in R_{\text{cre}}^{(n,s)} | (i,v,c)\in Q_{\text{inst}}^{(n,j,s)}}} \gamma^{(n,j,t)} \right) \tag{44}$$

If a solution is found, the decision array represented by ψ can be used directly to create the target domain aspects of the correspondence model. ψ, when combined with the transformation specification, describe exactly which relation instances will be instantiated, which variables will require new class instances to be created, and which dependent relation instances are bound to which parent relation instances. This results in a new set $Q_{\text{inst}}^{(n,j,t)}$ for the target domain which represents the instances that will be created. Each created instance will have a Unique Identifier (UID) associated with it. To make the results of the transformation easier to trace, the UID is derived from the relation instance that instantiated it using a fixed format. So, $r1\_j1\_t1\_ClassA$ is the UID for an instance of ClassA that was created for, and bound to, variable $t1$ of the first instance ($j1$) of relation 1 ($r1$).

4.2.4 Stage 4 - Instantiate References

Given the existence of the updated correspondence model $Q_{\text{inst}}^{(n,j,t)}$, the fourth stage is relatively straightforward, with two possible approaches; a deterministic approach and a linear programming approach.

If the relations in the transformation specification fully describe all references within a meta-model, given that Q^t will bind all the variables in the model and that R^t describes how those variables participate in references, it is straightforward to instantiate I_{as}^t and I_{co}^t.

However, in many circumstances the transformation specification may not be complete with regards every possible reference between class instances. A common example is *opposite* references where the association is specified in only one direction explicitly in the transformation specification. This is a feature of the example used in section 6.2. Simply using the Q sets here would miss instantiating these references. To address this, the stage is treated as a MILP problem in a similar form to stage 3. Given the similarities to the previous stage, it shall only be briefly elaborated upon. Binary decision variables are introduced to determine whether a reference should be instantiated between two class instances. *Completed relation instances* must have the required references as specified in their transformation specification instantiated. However, the system is also allowed to instantiate additional references as required to conform with the meta-model, even if those references are not explicitly captured in the transformation specification.

5 Implementation

To execute the model transformation, the four stages of the approach (Source Model Analysis, Bind Variables, Create Target Model Instances and Create References, described in section 4) are represented using the Gnu MathProg Language (GMPL) [Mak10]. These programs are solved using the Gnu Linear Programming Kit (GLPK)[7].

[7]Available at http://www.gnu.org/s/glpk

GMPL and GLPK offer some rudimentary modularisation features, such as the separation of problems to be solved into *model* and *data* files. GMPL models are not equivalent to MOF-derived models. In our implementation, there is one GMPL *model* file (i.e. MILP problem) for each of the stages described in section 4. Each *model* file captures the constraints and objective functions described previously, and does not change irrespective of the inputs. After each stage of the method is completed an updated *data* file is written which contains the updated set representations as determined by the solver.

To assemble an initial GMPL *data* file from individual models, meta-models and transformation specification, a bash shell script is used which takes these specifications as inputs as well as a number of options, including which direction to conduct the transformation. The bash script then assembles the appropriate GMPL *data* file as an input to stage 1 of the transformation engine.

Whilst this solution provides a means to conduct the transformation, the representations used are not those commonly used for domain specific modelling. Direct integration with the Eclipse Modelling Framework (EMF) [SBPM09] is a desirable property. This has been achieved for meta-models specified using ECORE and associated models in EMF. The Acceleo implementation of the OMG Model To Text (M2T) transformation language [OMG08b] has been used to generate the GMPL meta-model and model representations from their Eclipse based equivalents.

As the solver is operating, it generates a text file which specifies which class instances are created, how they reference each other and what the attribute values should be set to, as determined by the appropriate stage of the solver. On completion, this file is read back into Eclipse using a custom Java plug-in which creates an EMF model based on the solver output. This system has been evaluated under both Scientific Linux 6 and Mac OS 10.7 operating systems.

6 Evaluation and Results

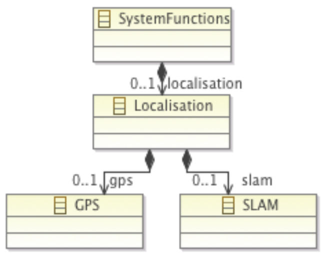
In this section results from an evaluation of the solver are described using two transformation examples. The first example uses the simple meta-models shown earlier in figure 1. This example illustrates the representations in more depth, and how the solver addresses this problematic transformation. The second example uses a more complex set of models and associated transformations. This is used to demonstrate the solver operating on more realistic models, and on transformations where there is significant ambiguity on how the relations should be applied. This second example is also used to demonstrate the system working on a wide variety of input models, and as a means to investigate the scalability of the system.

6.1 Simple Model Transformation

Consider the problematic source model, transformation and meta-models originally discussed in section 2. Two variants of the target meta-model shall be introduced. The only difference between the target meta-models is the lower and upper cardinality bounds on the *Requirement* to *System Can Determine Its Position* classes. The first meta-model sets the bounds to two (i.e. two system functions are required to independently achieve this requirement). The second meta-model has the bounds set to one (i.e. Exactly one system function is required to achieve this requirement). An example of the set based representations for the source meta-model is shown in Figure

4. All the models, meta-models and transformations are shown in Appendix A, along with the set-based representation.

The results of running the model transformation, with the same source model and transformation specification, but differing target meta-models is shown in Figure 5. In both cases the generated model complies with the target meta-model. However, in the second target model the system has only utilised relations 1, 2 and 4 in generating the model. Whilst relation 3 had a match in the source model (as evidenced by the first target model), realising that relation would have over-specified the target model with respect to the second target meta-model. It has therefore been disregarded for the second transformation, and utilised in the first.



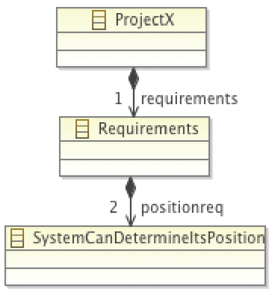
$$\begin{aligned}
C^s =& \{\text{SystemFunctions, Localisation,}\\
& \text{GPS, SLAM}\}\\
P^s_{\text{SystemFunctions}} =& \{(\text{SystemFunctions, Localisation,}\\
& \text{con, 0, 1, true, null})\}\\
P^s_{\text{Localisation}} =& \{(\text{gps, GPS, con, 0, 1, true, null}),\\
& (\text{slam, SLAM, con, 0, 1, true, null})\}\\
P^s_{\text{GPS}} =& \emptyset\\
P^s_{\text{SLAM}} =& \emptyset
\end{aligned}$$

Figure 4 – Second Source Meta-Model Set Representation

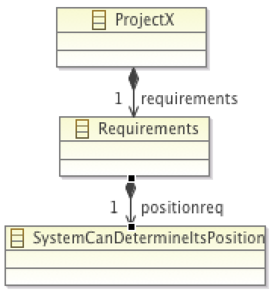
6.2 Train Sets to Petri Nets

A common example in some of the published work on Triple Graph Grammars is the transformation of Train Sets to Petri-Nets [GK10] [KW07]. Whilst this example appears relatively straightforward on first examination, it has several properties that can make transformation difficult. The example transformations have been adapted in the differing publications, such as the change in Petri-Net representation of the Join/Converging Switch between the earlier example described by Kindler and Wagner [KW07] and the later work by Greenyer et al. [GK10]. Some of these changes were likely motivated by the difficulty in implementing this transformation in QVT-Relation. An additional problem associated with this transformation is the ambiguity in the 'reverse' transformation from Petri-Nets to Train Sets; the same Petri-Net patterns participate in multiple relations. If the target meta-model is not considered when executing the transformation, it is relatively easy for invalid models to be generated by choosing the wrong relations to instantiate. Whilst this problem could be addressed by reworking the transformation to remove that ambiguity, this is not desirable. It can result in significant rework, and requires more monolithic relations; multiple relations that each cover an overlapping proportion of the model to explicitly cope with each ambiguous situation. Instead, the MILP transformation engine will use the target meta-model as an additional source of information to resolve these ambiguities and instantiate the correct relations.

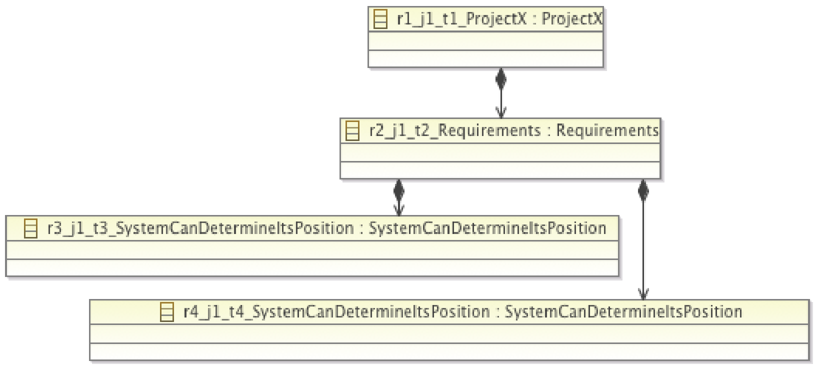
Given the lack of support for inheritance in our current solver implementation, expanded meta-models shall be used compared to those used by Greenyer et al. The abstract classes in the original examples are removed, and the concrete classes directly related to each other. In addition, explicit *Converging* and *Diverging Switch* types shall be introduced which are precisely specified in terms of the number of *InPort* and



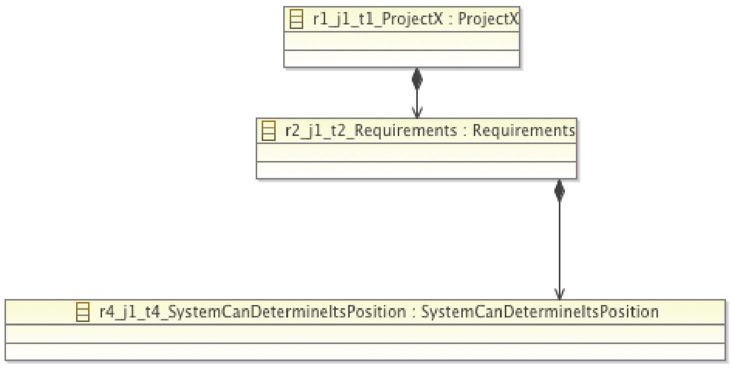
(a) Target Meta-Model A



(b) Target Meta-Model B

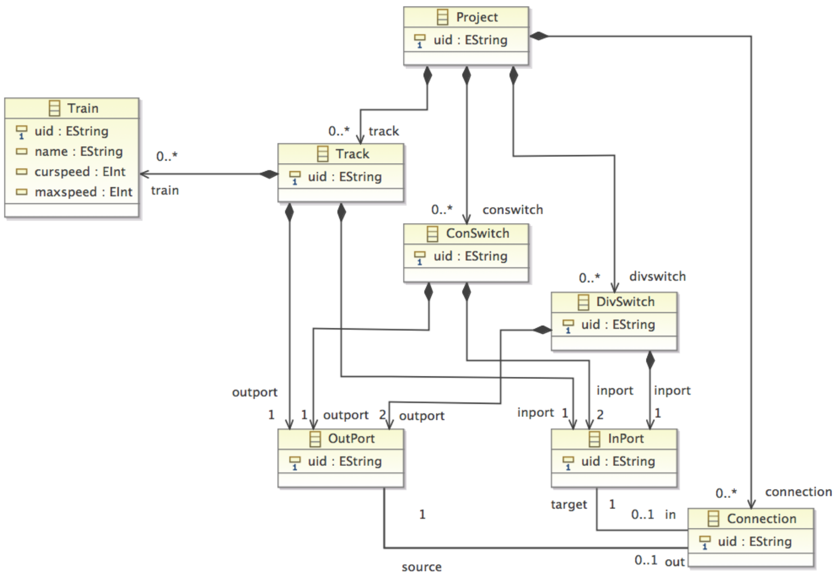

r1_j1_t1_ProjectX : ProjectX
r2_j1_t2_Requirements : Requirements
r3_j1_t3_SystemCanDetermineItsPosition : SystemCanDetermineItsPosition
r4_j1_t4_SystemCanDetermineItsPosition : SystemCanDetermineItsPosition

(c) Target model generated for Meta-Model A

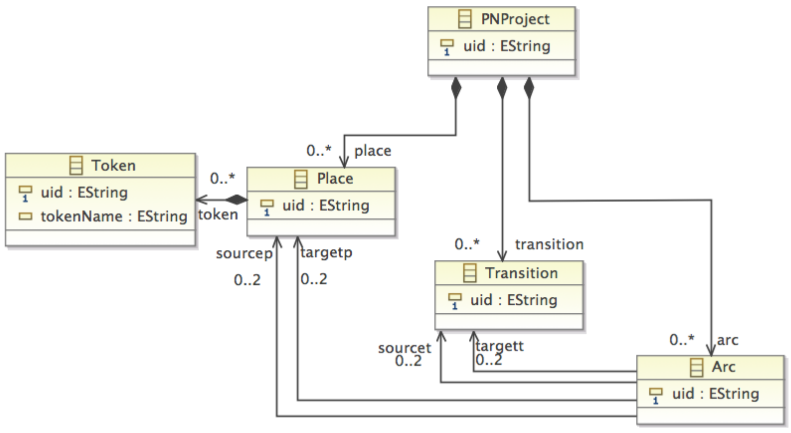

r1_j1_t1_ProjectX : ProjectX
r2_j1_t2_Requirements : Requirements
r4_j1_t4_SystemCanDetermineItsPosition : SystemCanDetermineItsPosition

(d) Target model generated for Meta-Model B

Figure 5 – Target Models generated by the solver when targetting Meta-Model A (left) and Meta-Model B (right)



(a) Train MetaModel



(b) Petri-Net MetaModel

Figure 6 – Train and Petri-Net Meta-Models

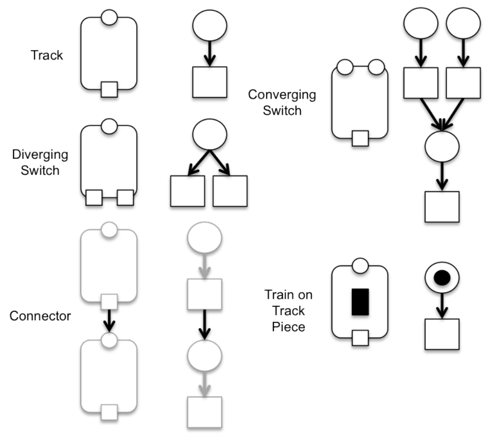


Figure 7 – Train Pieces and their equivalent Petri-Net representations

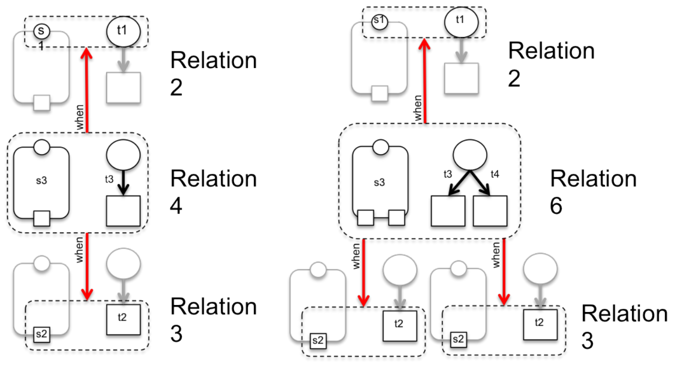


Figure 8 – Subset of Train to Petri-Net relations, showing *when* dependencies

OutPorts they contain. These meta-models are shown in Figure 6. To aid with the understanding of the example models, an alternative graphical notion shall be used, as shown in Figure 7. This shows the possible train model pieces, such as *Tracks* or *Diverging* or *Converging Switches* and their corresponding Petri-Net representations. For more information on this example, the reader is directed to the technical report by Kindler and Wagner [KW07]. The fully specified transformation uses eight relations, of which four are shown in Figure 8. Relation 4 describes the transformation of a *Track* piece containing one *InPort* and one *OutPort* into a Petri-Net *Place* and *Transition*, which are both associated with an *Arc* with the *Place* as the source. Note, in this specification the relations responsible for the creation of *InPorts* and *OutPorts* are relations 2 and 3 respectively. Relation 4 is dependent on these two relations. Similarly, Relation 5 describes the transformation of a *Diverging Switch* which contains one *InPort* and two *OutPorts*. This relation is dependent on one instance of relation 2, but two instances of relation 3 in order to realise the necessary *OutPorts*. Other relations not shown in this figure include the relations for transforming *Converging Switches*, *Connectors*, *Trains* and the containing *Project* into their corresponding Petri-Net representations.

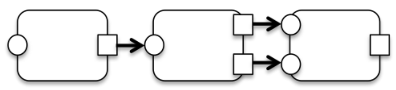


Figure 9 – Example Train Model

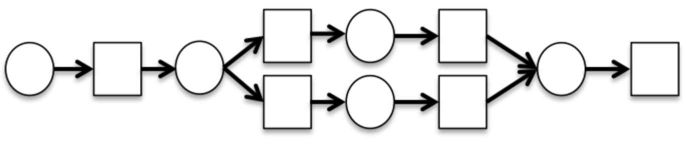


Figure 10 – Generated Petri-Net Model

Consider the example train model in Figure 9. If the transformation is executed in the direction of the Petri-Net meta-model using the MILP solver then the Petri-Net shown in Figure 10 is generated by the MILP transformation engine. This is a relatively straightforward transformation, largely due to to the precision of the Train Set meta-model. Each of the eight relations specified in the transformation specification unambiguously describe the entire meta-model. This is due to the strongly typed nature of the model which allows, for example, *Track* pieces to be distinct from *Switches* and the appropriate relations used. Examining the *potential relation instances* identified after the source model variables have been bound and the *completed relation instances* used to instantiate the target model shows that all of the *potential relation instances* are used. There is no ambiguity or conflict between those relation instances.

However, executing the transformation in the opposite direction is less clear, due to the ambiguous nature of the relation patterns for that domain. Specifically the Petri-Net representation for a *Track* piece, which comprises a *Place* and a *Transition* connected by an *Arc* is a sub-set of the petri-net representations for both *Diverging* and *Converging Switches.* Indeed, if the meta-models are ignored and relational patterns are simply matched there are multiple potential transformations of this model. For example, a potential transformation from a standard QVT-Relation engine is for this model to be transformed into six *Track* pieces; relation 4 can strictly be matched six times. The Train Set meta-model precludes a single *InPort* being contained within multiple *Track* pieces, but as was shown earlier, most current transformation languages do not consider the meta-models when generating target models.

The MILP solver described in this paper is able to regenerate the original Train Set model from this Petri-Net model. After binding variables to source model instances, there are a significant number of *potential relation instances* still under consideration. Whilst only one Petri-Net pattern for a *Converging Switch* has been identified (the most complex of the patterns), two Petri-Net patterns for a *Diverging Switch* have been identified with acceptable variable bindings, and five Petri-Nets patterns for *Track Pieces* have been identified. The solver is able to determine that instantiating one each of the *Track, Diverging Switch,* and *Converging Switch* relation instances maximises the objective function in equation 44, thus successfully recreating the original model.

This MILP solver does not guarantee the ability to reverse a previous transformation and regenerate the original source model. It depends on the information available to the system, including the detail within the transformation specification and the respective meta-models. In some cases, the ambiguity may be too great and an

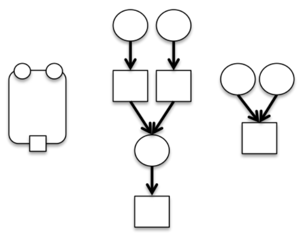


Figure 11 – Different Greenyer-derived (center) and Callow (right) petri-net representations of a track converging switch (left)

alternative model is generated. However, the system will guarantee that whatever model is generated, it will be generated from a valid combination of *potential relation instances* and the generated model will be compliant with its meta-model.

6.3 Correctness and Performance

To examine correctness and scalability of the system, a variety of 16 example test train models have been constructed. These models are summarised in Table 1 and range from single switch pieces through to a large loop that has multiple tracks, switches and trains. Two variants of the transformation specification have also been considered; the Greenyer-derived specification introduced previously and a Callow specification. The only difference between the two is the equivalent petri-net representation for a converging switch. This is summarised in Figure 11.

To test correctness of the transformation engine, two transformations were executed for each model. A petri-net model was automatically generated using the transformation engine from the selected train-model and then this automatically generated model was used as the input for the reverse transformation. All other aspects (meta-models, transformation specification) were kept constant for both transformations and this resulted in a newly generated train model. This model was then compared with the original model in terms of the class instances expected, attribute values and references. For all of the example models, and for both transformation specifications, the system successfully reconstructed all of the original source models except for the reverse transformation for model 16 using the Greenyer transformation. Even for this model, it is expected that the solver can generate the correct target model for this example. However, that example is the most complex that is considered in this paper and the system did not converge on a solution in a reasonable time. The test process (and computation times) for model 13 are shown in figure 12. Some of the more complex models (15 and 16) are shown in Appendix A.

To test performance of the system, the time taken to complete each of the stages in both the forward and reverse transformations in the correctness tests was measured. This was conducted for all input train models, using both the Greenyer-derived and Callow transformation specifications. These results are summarised in the charts in Figure 13. In the forward transformation, performance is relatively constant over stages 1, 2 and 3. It is only in the final two models where there is a measurable exponential growth in completion time and this is due to the increase in time it takes to complete stage 4, where the association and containment references are established. For models 1 to 14, there is no appreciable difference between the times to complete when using

Table 1 – Summary of example train models

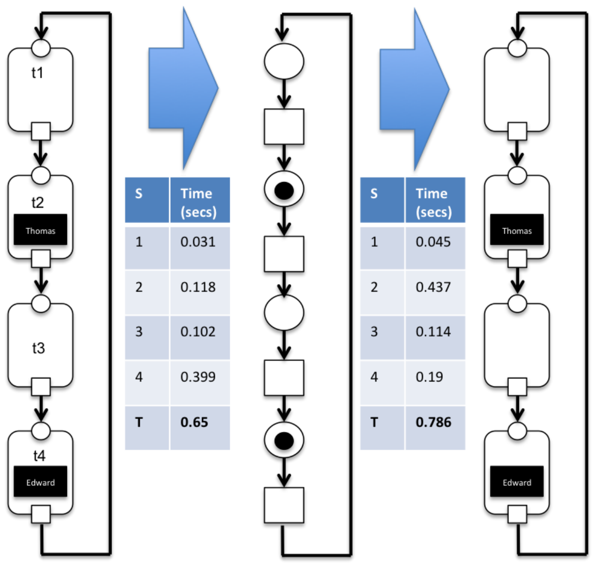
Model No.	Model Desc.	Class Inst.	Cont. Refs.	Assoc. Refs
1	2 Tracks, 1 Connection	8	7	4
2	3 Tracks, 2 Connections	12	11	8
3	4 Tracks, 3 Connections	16	15	12
4	2 Tracks, 1 DivSwitch, 2 Connections	13	12	8
5	4 Tracks, 4 Connections (Loop)	17	16	16
6	2 Tracks, 1 DivSwitch	11	10	0
7	2 Tracks, 2 DivSwitches, 3 Connections	18	17	12
8	2 Tracks, 3 DivSwitches, 5 Connections (Loop)	24	23	20
9	1 ConSwitch	5	4	0
10	1 Track, 1 ConSwitch, 1 Connection	9	8	4
11	4 Track, 3 Connections, 1 Train	17	16	12
12	4 Track, 3 Connections, 1 Train	17	16	12
13	4 Track, 4 Connections, 2 Trains (Loop)	19	18	16
14	1 Track, 2 ConSwitch, 2 Connections	13	12	8
15	4 Tracks, 2 DivSwitches, 2 ConSwitches, 10 Connections (Loop)	39	38	40
16	12 Tracks, 4 DivSwitches, 4 ConSwitches, 24 Connections, 3 Trains (Loop)	96	95	96

either the Greenyer-derived or Callow transformation specification. This includes model 8, the most complex model that does not contain any Converging Switches. Given that the only the difference between the specifications is the Converging Switch representation, this is not surprising. However, Models 15 and 16 contain two and four converging switches respectively and there is a significant increase in the completion time for these models. The engine takes longer to complete the fourth stage when using the Greenyer-derived specification compared to the Callow specification. This is because the Greenyer Converging Switch representation has double the number of petri-net elements as the Callow representation, and therefore more elements that need to be correctly associated. The petri-net meta-model is relatively unconstrained in that the same meta-model elements are reused in representing the train model components, and therefore there are many potential association references which would be permitted by the meta-model. These potential associations references are considered (thus increasing the search space) but disallowed in stage 4 due to not participating in the *potential relation instances*.

Table 2 – Table showing the number of potential relation instances identified by the transformation engine at the end of stage 1 in both the forward and reverse transformation for model 16, using the Callow transformation specification

Direction	R1	R2	R3	R4	R5	R6	R7	R8
Forward	1	24	24	12	24	4	4	3
Reverse	1	24	24	24	52	24	24	3

The reverse transformation has different properties. As with the forward transformation most of the models are solved in a reasonable time but the larger models exhibit an exponential increase, in relation to the increase in the size of the model, in their time to complete. However in the reverse transformation stage 2, where the binding of variables in the relations to the source model occurs, is where the majority



S	Time (secs)
1	0.031
2	0.118
3	0.102
4	0.399
T	**0.65**

S	Time (secs)
1	0.045
2	0.437
3	0.114
4	0.19
T	**0.786**

Figure 12 – Train Model 13, transformed to a Petri Model and then back to a Train Model using the Greenyer-derived transformation specification. Figure shows the total time taken to complete the transformation, broken down by stages.

of the time is spent. For the forward transformation stage 2 was not a significant proportion of the computation time. This is again due to the permissive nature of the petri-net meta-model, and is most clearly shown through the number of *potential relation instances* that are identified after the initial sub-graph matching algorithm in stage 1. The forward and reverse transformations for Model 16 are shown in Table 2. This shows that in the forward transformation, the sub-graph matching algorithm correctly identifies the exact number of *potential relation instances* for each relation. This is largely due to the strongly typed nature of the train meta-model. Each relation accommodates a specific type in that meta-model, and therefore it is a straightforward process to deterministically calculate the correct of number of matching relations. The combination of a strongly typed meta-model and the correctly identified number of *potential relation instances* ensures the optimisation problem in stage 2 is heavily constrained and can be solved quickly for large models. The same is not true for the reverse transformation. As many of the relations contain the same meta-model elements, there are many more *potential relation instances* under consideration after stage 1; over double the number for relation 4 and six times the number for the switch relations 6 and 7. The result of this is that stage 2 is a much less constrained optimisation problem than the forward transformation, and therefore can take significantly longer to solve. Enabling choice between *potential relation instances* in transformation specifications that reference permissive meta-models, and still generating correct target models, is the primary goal for this transformation engine and therefore this less constrained optimisation problem is simply a consequence of that research goal. There is also a large difference in completion times between the Greenyer-derived and Callow specifications in the reverse direction due to the differences between the Converging Switch representation; this difference results in a larger optimisation search space because of the larger Greenyer-derived representation of this switch. Stage 4, however, takes much less time to solve in the reverse transformation. This is because stage 4 in this instance works within the train meta-model domain and therefore is creating these references whilst considering a more precisely specified meta-model.

Let us consider further the relationship between stage 1 and stage 2. Stage 1, as described in section 4, provides a deterministic estimate of the number of *potential relation instances* which in turn establishes bounds for the optimisation problem in stage 2. Stage 1 only outputs a number of *potential relation instances*; it does not make any attempt to bind the variables in the relation to source model instances. If the algorithm used in stage 1 under-estimates the number of *potential relation instances*, then this can affect the target model being generated as it can overly constrain stage 2. This would prevent some viable relation instances from being realised, although the rest of the system will still assemble a target model that is compliant with its meta-model out of the relation instances it is allowed to use (if a compliant model is possible). If stage 1 over-estimates the number of *potential relation instances* then there is no effect on target model correctness. Any *potential relation instances* that cannot have their variables bound correctly to source model instances are ultimately disregarded. This prompts two questions; is stage 1 is required and why not assume arbitrarily large constant values for potential relation instances as an input to stage 2?

In table 3, a summary of stage 1 and 2 for three separately run reverse transformations of model 8 are shown. The difference between the three runs is how the number of potential relation instances is deterministically calculated in stage 1. Recall that stage 1 currently uses the minimum of three heuristics which determine 1) number of instance matches, 2) containment relationship matches and 3) association relationship matches in order to set bounds on *potential relation instances.* The difference between the three runs is simply a change in the logic in the containment relationship heuristic. Consider that a relation may specify multiple containment relationships. Run 1 (the default) returns the number of times the containment relationship with the fewest matches was matched. Run 3 returns the number of times the containment relationship with the most matches was matched. Run 2 adds a constant off-set of 1 to the containment values of Run 1. Whilst all three runs generated the same target model, run 3 took significantly longer to complete stage 2 because of the less constrained optimisation problem. This demonstrates that there is a significant performance benefit by accurately estimating the number of *potential relation instances* in the first stage.

Table 3 – Table showing the number of potential relation instances identified by the transformation engine at the end of stage 1 when transforming a petri-net representation of model 8 back to a train model using the Greenyer derived specification. Run 1 is tightly constrained in the number of potential relations identified. Run 3 is the most permissive.

Model Run	R1	R2	R3	R4	R5	R6	R7	R8	Stage 2 Completion Time (secs)
Run 1	1	5	8	6	13	6	6	0	6.846
Run 2	1	5	8	7	13	7	7	0	40.474
Run 3	1	5	8	13	13	13	13	0	953.887

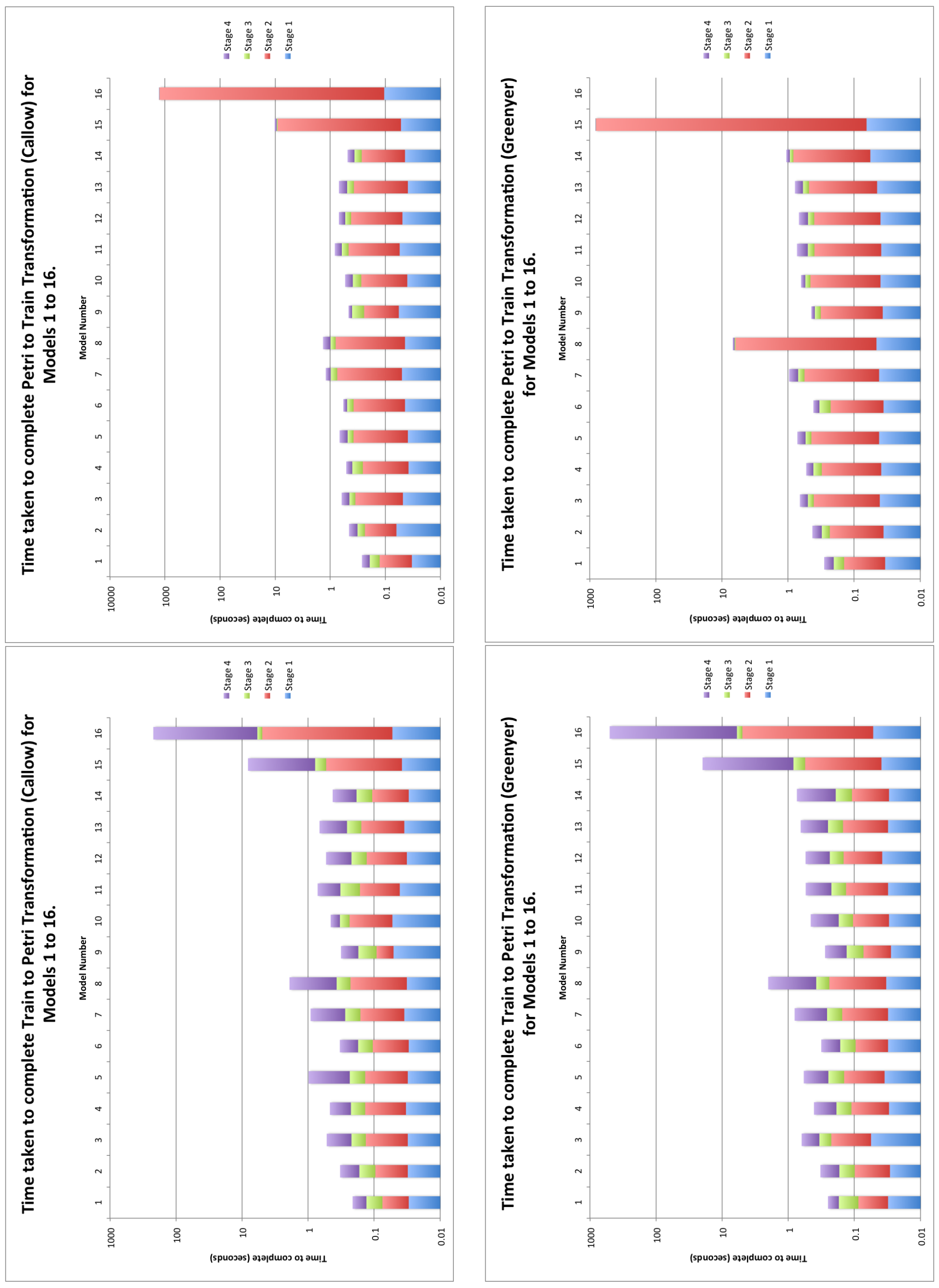


Figure 13 – Time to complete for forward and reverse transformation using the Greenyer-derived and Callow transformation specifications. Model 16 is not shown for the Greenyer-derived Petri to Train transformation due to the computation time required.

6.4 Scalability Discussion

These results demonstrate a wide range of performance results. For all of the small to medium sized models, the system is able to correctly generate a target model in both the forward and reverse directions in a practical time-scale. It is only with the largest models that we begin to see significant increases in completion times; in the forward direction in stage 4, and in the reverse direction in stage 2. Both of these stages are considering a lightly constrained meta-model when they show these extended times to completion.

Considering the forward transformation first, there appears to be some straight-forward improvements that could be made to improve the completion times. The current formulation for stage 4 considers the entire model when trying to determine the best association references to instantiate; the domain of the decision variables is all possible class instances that have the appropriate reference in the meta-model. This is then further constrained by which variables are bound to particular instances in the completed relation instances. A relatively straight-forward modification would be to modify the decision variables so that they are a) associated with each completed relation instance and b) only use the class instances that are bound to variables of that completed relation instance in that domain. This would have the effect of increasing the number of decision variables but drastically reducing the domain these decision variables operate over.

Improving the performance of stage 2 when used with very permissive meta-models is more difficult. This could potentially be considered a type of Bin Packing Problem with Conflicts (BPPC) where the system is looking to fit a number of different patterns most effectively over a given model to optimise a particular metric. For large models with permissive meta-models and where the associated transformation specification has relations which are sub-sets of other relations, there will be a large search space to examine and this is reflected in the results.

7 Conclusions and Future Research

This paper has described a novel model transformation engine, implemented as a series of Mixed Integer Linear Programs, which ensures that models created using the system are compliant with their meta-model. It achieves this by being less prescriptive with the source pattern matching semantics, allowing many more source pattern matches to be identified than can be realised and then choosing which of those relations should be instantiated to create an appropriate target model. The evaluations that have been conducted show the system can successfully create compliant target models, even when there is significant ambiguity in the transformation specification and source models provided.

7.1 Future Research

There are a number of areas for improving the current system. i) Integrating *model synchronisation* and *consistency checking* capabilities; the set-based representation used within this system has been developed with supporting these behaviours in mind. Consistency checking in particular is a useful capability for verifying system models, and this system potentially offers a significant benefit when applied to these models. It is straightforward to identify the relations that maximise coverage of source and/or target models. This could, for instance, be used to identify elements of system models that are not able to be verified (i.e. they don't currently participate in the transformation). ii) The use of an objective function to guide the different stages of the transformation also opens up other interesting possibilities. Whilst the objective function in this paper has been to either maximise the number of relations matched, or maximise the coverage of a particular model by the instantiated relations, other objective functions could be used. For example, the system could maximise or minimise a particular attribute value in the source or target models. When used for system verification, this could be used to identify the highest or lowest level of performance a system may achieve for a particular requirement. This would be a fruitful area for future research. iii) Additional meta-model constraints are often specified using an additional constraint language, such as OCL, when specifying meta-models. The current transformation engine can incorporate these additional constraints relatively easily within stage 3 of the approach, but the transformation from OCL to GMPL is currently a manual process. Automating this, by automatically transforming OCL into an equivalent GMPL form, would significantly increase the complexity of meta-models that can be considered when the engine assembles a target model. This would form a future research theme for this work.

Improving the performance of the system also has several promising research directions. iv) Performance of stage 2 is dependent on the number of potential relation instances identified in stage 1. At present, a relatively naive set of heuristics are used in stage 1. If a more accurate determination of the potential relation instances can be made, this could lead to an improved performance in stage 2. v) One of the benefits of specifying this transformation as a GMPL problem is that it opens up the possibility of easily using alternative solvers to address this problem. Whilst the results in this paper are generated from the GLPK solver, there are many other solvers that support AMPL[8] from which GMPL is derived. Some of these solvers may have improved performance over GLPK when applied to this problem. vi) Although BPPC is NP-hard, alternative optimisation approaches have been used previously in finding good approximations to the global optimum [MIM09] [Fal96]. The set-based representation, constraints and metrics described in section 4 could form the basis for implementing a heuristic optimisation solution as an alternative to a MILP that is better able to identify near-optimal solutions for stage 2 when dealing with very large search spaces.

7.2 Conclusions

One of the key problems that we believe has caused the limited uptake of relational model transformation languages is their restrictive or ambiguous semantics. However, the goal of relational transformation languages is to focus on *what* the transformation is expected to achieve rather than *how* a model transformation engine should execute a transformation. Unfortunately, relational transformation languages that have restrictive, deterministic semantics, such as Triple Graph Grammars, constrain the scope and style of relations that can be written. The fixed semantics restrict the relations that are, and are not allowed. QVT-Relation, which is ambiguous in its semantics, has additional problems; different tools interpret the semantics differently, and therefore have different behaviour. Moreover, it cannot be guaranteed that a particular transformation will generate a correct model, or that results will be consistent across different engines given the same inputs. A modeller must have knowledge of *how* a

[8]See http://www.ampl.com/solvers.html for a reasonably complete list. (Last checked 29/6/2012)

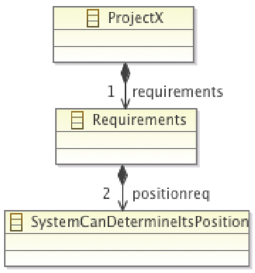
QVT-Relation transformation engine will interpret their transformation to write an effective specification.

The Mixed Integer Linear Program based model transformation approach described in this paper addresses this challenge by avoiding a prescriptive set of semantics. Instead, the generated target models are determined based on the target meta-model and the associated objective functions. This novel approach allows for a consistent interpretation of the transformation specification (in that a correct target model will be generated) independent from fixed semantics within the engine, and guarantees that a generated model will be correct with respect to its meta-model.

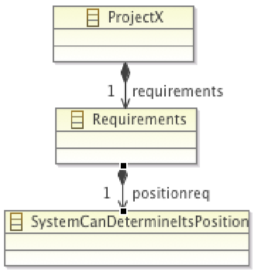
Consequently, there are significant potential benefits to modellers when using this transformation engine. Firstly, they have greater freedom in writing the model transformation specification. They do not need to be as cognisant of *how* a model transformation engine will interpret their specification, and can instead guide the transformation by the detail in the associated meta-model and user-specified metrics. Secondly, the modeller can be confident that a *correct* model will be generated, if one is possible; models that are non-compliant with their associated meta-model will not be generated. Our research clearly indicates the merits of our approach and signposts the important benefits for the evoling model based systems engineering community.

A Detailed Model Examples

In this Appendix, detailed information about some of the examples used in this paper are described. The full QVT-Relation specification described in section 2 is presented in Figure 17. The Target Meta-Models used for that example and the set-based equivalents (based on the descriptions in section 3) are shown in Figure 14. Similarly, the source model used for that transformation and the set-based equivalent is shown in Figure 15. Finally, the set based equivalent of the QVT-Relation transformation specification is shown in Figure 16.

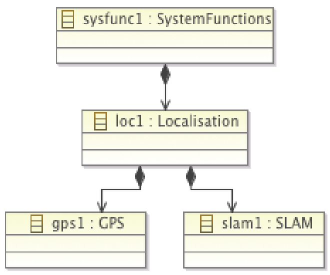


$$\begin{aligned}
C^t =& \{\text{ProjectX, Requirements,}\\
& \text{SystemCanDetermineItsPosition}\}\\
P^t_{\text{ProjectX}} =& \{(\text{requirements, Requirements, con, 1, 1,}\\
& \text{true, null})\}\\
P^t_{\text{Requirements}} =& \{(\text{positionreq,}\\
& \text{SystemCanDetermineItsPosition,}\\
& \text{con, 2, 2, true, null})\}\\
P^t_{\text{System...}} =& \emptyset
\end{aligned}$$



$$\begin{aligned}
C^t =& \{\text{ProjectX, Requirements,}\\
& \text{SystemCanDetermineItsPosition}\}\\
P^t_{\text{ProjectX}} =& \{(\text{requirements, Requirements, con, 1, 1,}\\
& \text{true, null})\}\\
P^t_{\text{Requirements}} =& \{(\text{positionreq,}\\
& \text{SystemCanDetermineItsPosition,}\\
& \text{con, 1, 1, true, null})\}\\
P^t_{\text{System...}} =& \emptyset
\end{aligned}$$

Figure 14 – Target Meta-Model Set Representations



$$
\begin{aligned}
I^s =&\{(\text{sysfunc1, SystemFunctions}),\\
&(\text{loc1, Localisation}),\\
&(\text{gps1, GPS}),\\
&(\text{slam1, SLAM})\}\\
I^s_{\text{co}} =&\{(\text{sysfunc1, SystemFunctions,}\\
&\text{loc1, Localisation}),\\
&(\text{loc1, Localisation,}\\
&\text{gps1, GPS}),\\
&(\text{loc1, Localisation,})\\
&\text{slam1, SLAM})\}
\end{aligned}
$$

Figure 15 – Simple Source Model Set Representation

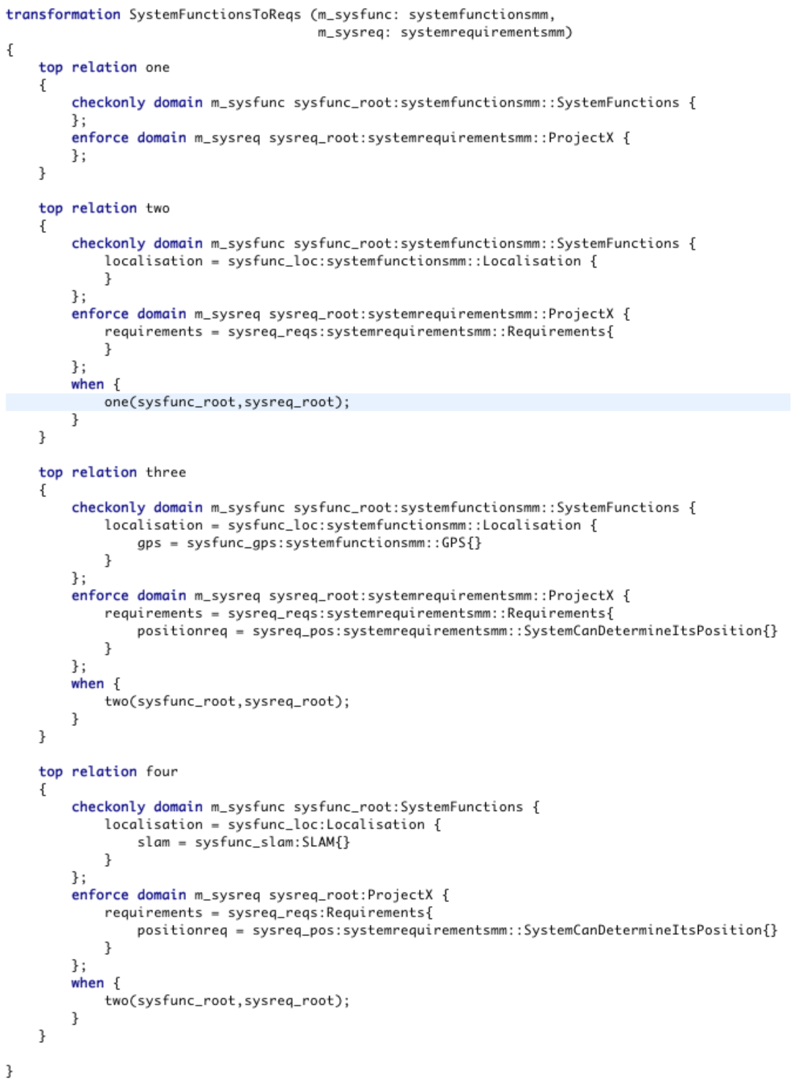
$$
\begin{aligned}
R^1_{\text{top}} =&\{\text{true}\}\\
R^{(1,s)}_{\text{root}} =&\{\text{SystemFunctions}\}\\
R^{(1,t)}_{\text{root}} =&\{\text{ProjectX}\}\\
R^{(1,s)}_{\text{cre}} =&\{(\text{s1, SystemFunctions})\}\\
R^{(1,t)}_{\text{cre}} =&\{(\text{t1, ProjectX})\}\\
R^{(1,s)}_{\text{dep}} =&\emptyset\\
R^{(1,t)}_{\text{dep}} =&\emptyset\\
R^{(1,s)}_{\text{co}} =&\emptyset\\
R^{(1,t)}_{\text{co}} =&\emptyset\\
R^1_{\text{when}} =&\emptyset
\end{aligned}
$$

$$
\begin{aligned}
R^2_{\text{top}} =&\{\text{true}\}\\
R^{(2,s)}_{\text{root}} =&\{\text{SystemFunctions}\}\\
R^{(2,t)}_{\text{root}} =&\{\text{ProjectX}\}\\
R^{(2,s)}_{\text{cre}} =&\{(\text{s2, Localisation})\}\\
R^{(2,t)}_{\text{cre}} =&\{(\text{t2, Requirements})\}\\
R^{(2,s)}_{\text{dep}} =&\{(\text{s1, SystemFunctions})\}\\
R^{(2,t)}_{\text{dep}} =&\{(\text{t1, ProjectX})\}\\
R^{(2,s)}_{\text{co}} =&\{(\text{s1, SystemFunctions, s2,}\\
&\text{Localisation,localisation})\}\\
R^{(2,t)}_{\text{co}} =&\{(\text{t1, ProjectX, t2, Requirements,}\\
&\text{requirements})\}\\
R^2_{\text{when}} =&\{(1, \text{s1, t1})\}
\end{aligned}
$$

$$
\begin{aligned}
R^3_{\text{top}} =&\{\text{true}\}\\
R^{(3,s)}_{\text{root}} =&\{\text{Localisation}\}\\
R^{(3,t)}_{\text{root}} =&\{\text{Requirements}\}\\
R^{(3,s)}_{\text{cre}} =&\{(\text{s3, GPS})\}\\
R^{(3,t)}_{\text{cre}} =&\{(\text{t3, SystemCanDetermineItsLocation})\}\\
R^{(3,s)}_{\text{dep}} =&\{(\text{s2, Localisation})\}\\
R^{(3,t)}_{\text{dep}} =&\{(\text{t2, Requirements})\}\\
R^{(3,s)}_{\text{co}} =&\{(\text{s2,Localisation,s3,GPS,gps})\}\\
R^{(3,t)}_{\text{co}} =&\{(\text{t2,Requirements,t3,}\}\\
&\{\text{SystemCanDetermineItsLocation,}\\
&\text{positionreq})\}\\
R^3_{\text{when}} =&\{(2, \text{s2, t2})\}
\end{aligned}
$$

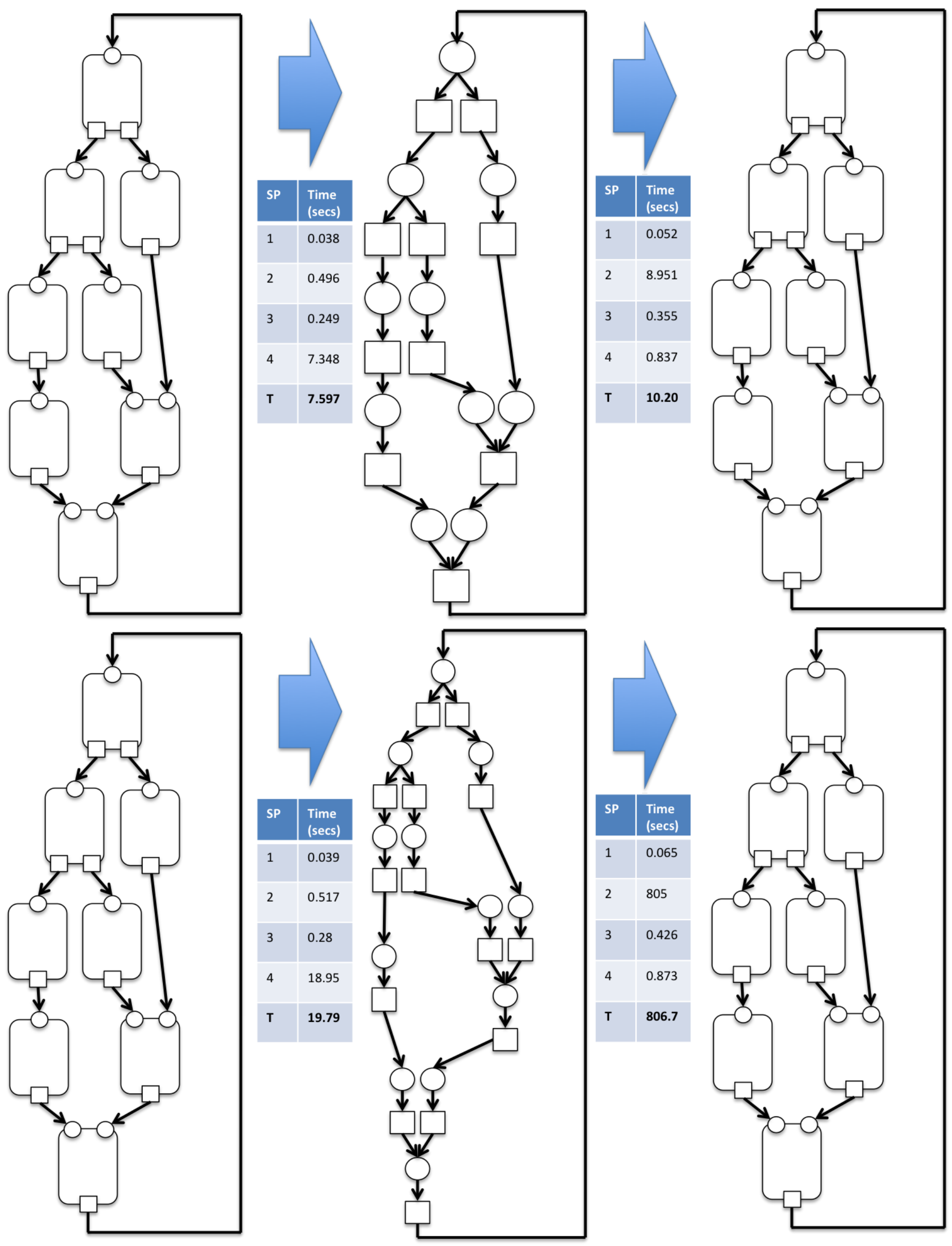
$$
\begin{aligned}
R^4_{\text{top}} =&\{\text{true}\}\\
R^{(4,s)}_{\text{root}} =&\{\text{Localisation}\}\\
R^{(4,t)}_{\text{root}} =&\{\text{Requirements}\}\\
R^{(4,s)}_{\text{cre}} =&\{(\text{s4, SLAM})\}\\
R^{(4,t)}_{\text{cre}} =&\{(\text{t3, SystemCanDetermineItsLocation})\}\\
R^{(4,s)}_{\text{dep}} =&\{(\text{s2, Localisation})\}\\
R^{(4,t)}_{\text{dep}} =&\{(\text{t2, Requirements})\}\\
R^{(4,s)}_{\text{co}} =&\{(\text{s2,Localisation,s4,SLAM,slam})\}\\
R^{(4,t)}_{\text{co}} =&\{(\text{t2,Requirements,t3,}\}\\
&\{\text{SystemCanDetermineItsLocation,}\\
&\text{positionreq})\}\\
R^4_{\text{when}} =&\{(2, \text{s2, t2})\}
\end{aligned}
$$

Figure 16 – Set Based Representation of Transformation shown in Figure 17



```
transformation SystemFunctionsToReqs (m_sysfunc: systemfunctionsmm,
                                      m_sysreq: systemrequirementsmm)
{
    top relation one
    {
        checkonly domain m_sysfunc sysfunc_root:systemfunctionsmm::SystemFunctions {
        };
        enforce domain m_sysreq sysreq_root:systemrequirementsmm::ProjectX {
        };
    }

    top relation two
    {
        checkonly domain m_sysfunc sysfunc_root:systemfunctionsmm::SystemFunctions {
            localisation = sysfunc_loc:systemfunctionsmm::Localisation {
            }
        };
        enforce domain m_sysreq sysreq_root:systemrequirementsmm::ProjectX {
            requirements = sysreq_reqs:systemrequirementsmm::Requirements{
            }
        };
        when {
            one(sysfunc_root,sysreq_root);
        }
    }

    top relation three
    {
        checkonly domain m_sysfunc sysfunc_root:systemfunctionsmm::SystemFunctions {
            localisation = sysfunc_loc:systemfunctionsmm::Localisation {
                gps = sysfunc_gps:systemfunctionsmm::GPS{}
            }
        };
        enforce domain m_sysreq sysreq_root:systemrequirementsmm::ProjectX {
            requirements = sysreq_reqs:systemrequirementsmm::Requirements{
                positionreq = sysreq_pos:systemrequirementsmm::SystemCanDetermineItsPosition{}
            }
        };
        when {
            two(sysfunc_root,sysreq_root);
        }
    }

    top relation four
    {
        checkonly domain m_sysfunc sysfunc_root:SystemFunctions {
            localisation = sysfunc_loc:Localisation {
                slam = sysfunc_slam:SLAM{}
            }
        };
        enforce domain m_sysreq sysreq_root:ProjectX {
            requirements = sysreq_reqs:Requirements{
                positionreq = sysreq_pos:systemrequirementsmm::SystemCanDetermineItsPosition{}
            }
        };
        when {
            two(sysfunc_root,sysreq_root);
        }
    }

}
```

Figure 17 – QVT-Relation specification used for example transformation in section 2



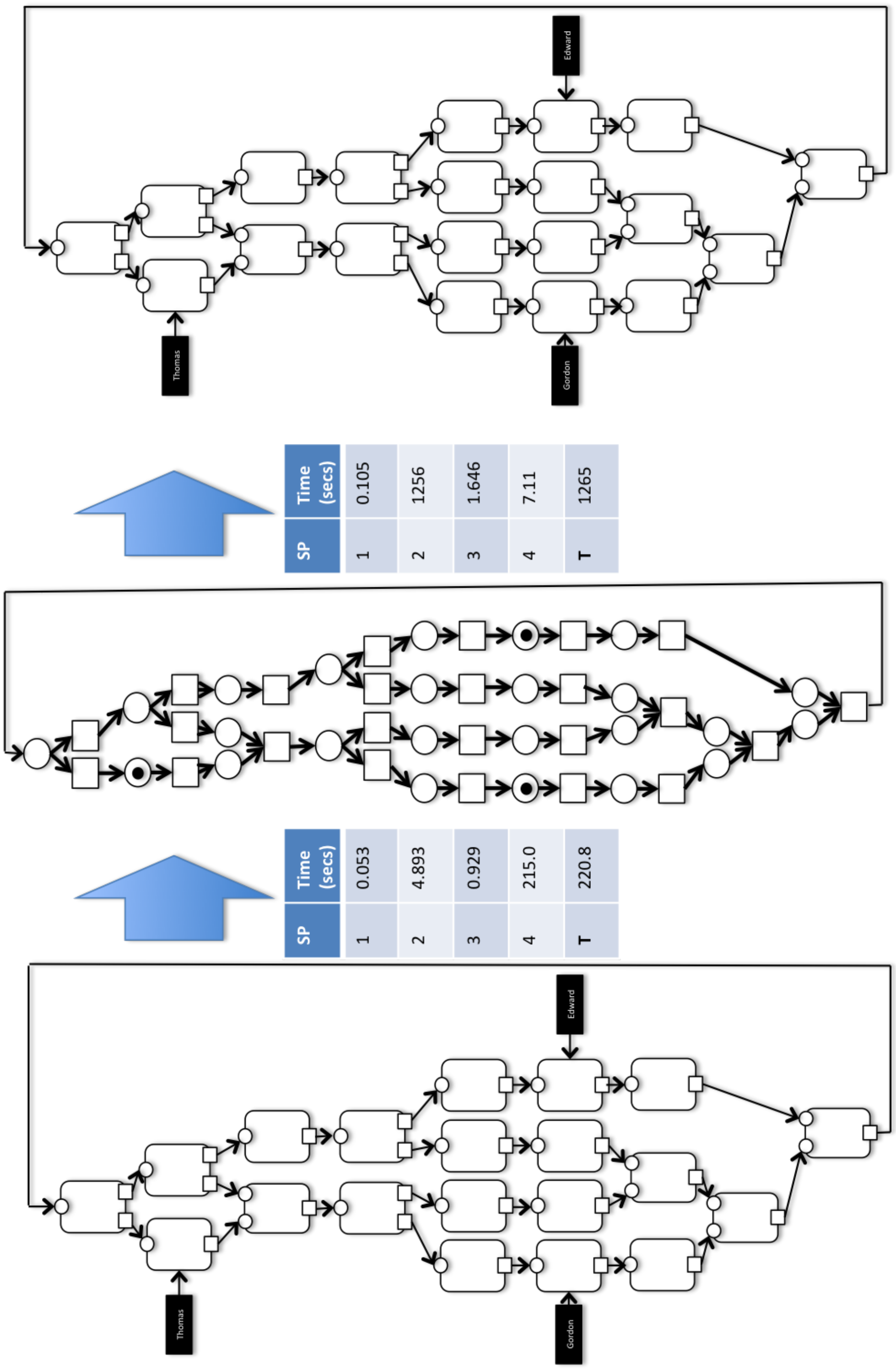
SP	Time (secs)
1	0.038
2	0.496
3	0.249
4	7.348
T	**7.597**

SP	Time (secs)
1	0.052
2	8.951
3	0.355
4	0.837
T	**10.20**

SP	Time (secs)
1	0.039
2	0.517
3	0.28
4	18.95
T	**19.79**

SP	Time (secs)
1	0.065
2	805
3	0.426
4	0.873
T	**806.7**

Figure 18 – Train Model 15, Forward and Reverse Transformations using the Callow specification (top) and Greenyer-derived specification (bottom)



SP	Time (secs)
1	0.105
2	1256
3	1.646
4	7.11
T	1265

SP	Time (secs)
1	0.053
2	4.893
3	0.929
4	215.0
T	220.8

Figure 19 – Train Model 16, Forward and Reverse Transformations using the Callow specification

References

[Bai05] C Bailey. Developing Coherent, Concise And Comprehensive User Requirements Using The MoD Architectural Framework (MODAF). *DTIC Document*, 2005.

[CCG10] Jordi Cabot, R Clarisó, and E Guerra. A UML/OCL Framework for the Analysis of Graph Transformation Rules. *Software and Systems Modeling*, 2010. doi:10.1007/s10270-009-0129-0.

[CCGM07] M Cadoli, D Calvanese, G De Giacomo, and T Mancini. Finite Model Reasoning on UML Class Diagrams via Constraint Programming. In *Proceedings of the 10th Congress of the Italian Association for Artificial Intelligence on AI*IA 2007: Artificial Intelligence and Human-Oriented Computing*, pages 36–47, Rome, Italy, 2007. AI*IA '07. doi:10.1007/978-3-540-74782-6_5.

[CCR08] Jordi Cabot, R Claris, and Daniel Riera. Verification of UML/OCL Class Diagrams Using Constraint Programming. In *Proceedings of the 2008 International Conference on Software Testing Verification and Validation Workshop*, pages 73–80, Barcelona, 2008. doi:10.1109/ICSTW.2008.54.

[CDREP10] Antonio Cicchetti, Davide Di Ruscio, Romina Eramo, and Alfonso Pierantonio. JTL: A Bidirectional and Change Propagating Transformation Language. In *SLE'10: Proceedings of the Third international conference on Software language engineering*. Springer-Verlag, October 2010. doi:10.1007/978-3-642-19440-5_11.

[CFH$^+$09] K Czarnecki, J Foster, Z Hu, R Lämmel, A Schürr, and J F Terwilliger. Bidirectional Transformations: A Cross-Discipline Perspective. *ICMT2009 - International Conference on Model Transformation*, pages 260–283, 2009. doi:10.1007/978-3-642-02408-5_19.

[CH06] K Czarnecki and S Helsen. Feature-based Survey of Model Transformation Approaches. *IBM Systems Journal*, 45(3):621–645, 2006. doi:10.1147/sj.453.0621.

[CKWO11] Glenn Callow, R Kalawsky, G Watson, and Y Okuda. Addressing Systems Verification of Autonomous Systems through Bi-Directional Model Transformations: A Systems Model Driven Approach. *Proceedings of the 6th IEEE International Conference on System of Systems Engineering*, 2011. doi:10.1109/SYSOSE.2011.5966616.

[Fal96] Emanuel Falkenauer. A Hybrid Grouping Genetic Algorithm for Bin Packing. *Journal of heuristics*, 2(1):5–30, 1996. doi:10.1007/BF00226291.

[Gas85] SI Gass. *Linear Programming: Methods and Applications.* Dover Publications Inc., 5th edition, 1985.

[GdLK$^+$11] Esther Guerra, Juan de Lara, Dimitrios Kolovos, Richard Paige, and Osmar dos Santos. Engineering Model Transformations with transML. *Software and Systems Modeling*, pages 1–23, 2011. doi:10.1007/s10270-011-0211-2.

[GdLKP10] E Guerra, J de Lara, D Kolovos, and R Paige. transML: A Family of Languages to Model Model Transformations. In *Proceedings of the*

13th international conference on Model driven engineering languages and systems, pages 106–120, Oslo, Norway, 2010. MODELS '10. `doi:10.1007/978-3-642-16145-2_8`.

[GK10] J Greenyer and E Kindler. Comparing relational model transformation technologies: implementing Query/View/Transformation with Triple Graph Grammars. *Software and Systems Modeling*, pages 21–46, 2010. `doi:10.1007/s10270-009-0121-8`.

[Hau88] J Hauser. The House of Quality. *Harvard Business Review*, 66(3):63–73, 1988.

[HKA11] F Heidenreich, J Kopcsek, and U Aßmann. Safe Composition of Transformations. *Journal of Object Technology*, 10:7:1–20, 2011. `doi:10.5381/jot.2011.10.1.a7`.

[HSST11] Z Hu, A Schürr, P Stevens, and J F Terwilliger. Dagstuhl Seminar on Bidirectional Transformations (BX). *SIGMOD Record*, 40(1):35–39, 2011. `doi:10.4230/DagRep.1.1.42`.

[JK06] F Jouault and I Kurtev. On the Architectural Alignment of ATL and QVT. *Proceedings of the 2006 ACM symposium on Applied computing*, page 1195, 2006. `doi:10.1145/1141277.1141561`.

[KW07] E Kindler and R Wagner. Triple Graph Grammars: Concepts, Extensions, Implementations, and Application Scenarios. *University of Paderborn Technical Report tr-ri-07-284*, 2007.

[Mak10] A Makhorin. Modelling Language for GNU MathProg for GLPK Version 4.45. December 2010.

[MIM09] AEF Muritiba, M Iori, and E Malaguti. Algorithms for the Bin Packing Problem with Conflicts. *INFORMS Journal on Computing*, 22(3):401–415, 2009. `doi:10.1287/ijoc.1090.0355`.

[MM03] J Miller and J Mukerji. MDA Guide v1.0.1. *OMG Document omg/03-06-01*, June 2003. Available from: `http://www.omg.org/cgi-bin/doc?omg/03-06-01`.

[OMG08a] OMG. Meta Object Facility (MOF) 2.0 Query/View/Transformation Specification. *OMG Document formal/2008-04-03*, April 2008.

[OMG08b] OMG. MOF Model to Text Transformation Language, v1.0. *OMG specification formal/2008-01-16*, pages 1–48, February 2008.

[SBM08] S Sen, B Baudry, and JM Mottu. On Combining Multi-Formalism Knowledge to Select Models for Model Transformation Testing. In *Proceedings of 1st International Conference on Software Testing, Verification, and Validation*, April 2008. `doi:10.1109/ICST.2008.62`.

[SBM09] S Sen, Benoit Baudry, and Jean-Marie Mottu. Automatic Model Generation Strategies for Model Transformation Testing. *Theory and Practice of Model Transformations, Lecture Notes in Computer Science*, 5563:148–164, 2009. `doi:10.1007/978-3-642-02408-5_11`.

[SBPM09] D Steinberg, F Budinsky, M Paternostro, and E Merks. *EMF: Eclipse Modeling Framework 2.0.* Addison-Wesley, 2009.

[SBV10] S Sen, B Baudry, and Hans Vangheluwe. Towards Domain-Specific Model Editors with Automatic Model Completion. *Simulation: Transaction of the Modeling and Simulation Society*, 86(2):109–126, February 2010. doi:10.1177/0037549709340530.

[Sch95] Andy Schürr. Specification of Graph Translators with Triple Graph Grammars. *Graph-Theoretic Concepts in Computer Science*, 903:151–163, 1995. doi:10.1007/3-540-59071-4_45.

[Ste11] P Stevens. A Simple Game-Theoretic Approach to Checkonly QVT Relations. *Software and Systems Modeling*, 2011. doi:10.1007/s10270-011-0198-8.

About the authors

Glenn Callow received his BSc (Hons) in computer science from the University of York in 1999. He is currently working towards an Eng.D. in Systems Engineering at Loughborough University. He was previously employed at the BAE Systems Advanced Technology Technology in the area of intelligent and autonomous systems, and has led the development of a number different platforms. His main research interests include system modelling, architectures and improved engineering for autonomous systems.

Glenn is a member of the British Computer Society. Contact him at g.m.callow@lboro.ac.uk

Roy Kalawsky (PhD (Hull 1991), MSc (Hull 1984), BSc (Hull 1978), C.Eng, MIET, FRSA) is Director of the Research School of Systems Engineering at Loughborough University, UK.). He has extensive industrial and academic experience in systems engineering spanning over 32 years. He spent over 17 years working for BAE Systems as a systems engineer and was responsible for Advanced Crew Station research across the Military Aircraft Division. He joined Loughborough University in 1995 and established the Advanced VR Research Centre to specialize in advanced systems, modelling and simulation, synthetic environments and advanced visual analytics. He is the founding Director of the Research School of Systems Engineering. He is adjunct professor at the University of Southern Australia (Adelaide). Contact him at r.s.kalawsky@lboro.ac.uk

Acknowledgments The work reported in this paper was funded by BAE Systems and the Systems Engineering for Autonomous Systems (SEAS) Defence Technology Centre established by the UK Ministry of Defence. Additionally, the authors would like to thank Prof. Charles Dickerson for his input and guidance with this work.