Chapter 24
An Automated Approach for Architectural Model Transformations

Grzegorz Loniewsli, Etienne Borde, Dominique Blouin, and Emilio Insfran

Abstract Software architectures are frequently represented as large models where many competing quality attributes have to be taken into account. In this context, there may be a large number of possible alternative architectural transformations that the architecture designer has to deal with. The complexity and dimensions of the solution space make that finding the most appropriate architecture considering several quality attributes is a challenging and time-consuming task. In this paper, we present a model transformation framework designed to automate the selection and composition of competing architectural model transformations. We also introduce a case study showing that this framework is useful for rapid prototyping through model transformations.

Keywords Model Transformations • Architecture Refinement • NFR

G. Loniewsli (✉) • E. Borde
Institute Telecom, TELECOM ParisTech,
LTCI—UMR 514, 46, rue Barrault, 75013 Paris, France
e-mail: grzegorz.loniewski@telecomparistech.fr; etienne.borde@telecomparistech.fr

D. Blouin
Lab-STICC, Universite de Bretagne-Sud, Centre de Recherche,
BP 92116, 56321 Lorient Cedex, France
e-mail: dominique.blouin@univ-ubs.fr

E. Insfran
Department of Computer Science and Computation, Universitat Politècnica de València,
Camino de Vera, s/n, 46022 Valencia, Spain
e-mail: einsfran@dsic.upv.es

24.1 Introduction

Since software systems are constructed to satisfy business goals, the design activities and especially architecture design must also be responsive to those business goals [1]. Business goals are described in [2] as *high-level objectives of the business, organization, or system that capture the reasons why a system is needed and guide decisions at various levels within the software development life cycle*. According to [3], the view that the rationale for a Non-Functional Requirement (NFR)[1] can be found in business goals gives software and system architects a new lens through which to examine and realize the NFRs of software systems.

The dimensions of the architectural models along with the number of alternative architectural strategies to be applied lead to a situation, where architects have to be given support in the candidate architectures exploration. Otherwise, architects manual work is time consuming, and essential design decisions are not always well motivated.

Model-driven Development (MDD) emphasizes the use of models and models transformations as the primary artefacts for automating the software production process. However, in a previous study [4], we showed that there is still a lack of MDD approaches, which starting from requirements, benefit from this automation. Most of the MDD approaches only consider system functional requirements, without properly integrating NFRs into their development processes [2]. Also, approaches which make use of the knowledge about architectural strategies implemented as transformations in the context of design space explorations are lacking.

There exist several partial solutions to deal with design space evaluation with respect to NFRs [5–7] and NFRs in design decisions [3, 8, 9]. However, approaches which consider these two dimensions in an integrated MDD environment are not common.

In this paper, we introduce a MDD approach, tailored for embedded software architecture design, where architecture refinements are implemented as model transformations driven by NFRs and goals. The approach is fully automated making use of: (1) *higher-order transformations* (HOT) in ATL with which to generate model refinements for specific design model; (2) extended goals specification from an extension of the Requirements Definition and Analysis Language (RDAL) [10]; (3) and finally the knowledge about the impact of particular model transformations of different quality properties.

The remainder of this paper is structured as follows. Section 24.2 introduces the problem statement, assumptions and challenges. Section 24.3 introduces our approach giving details about the use of MDD knowledge to select the architectural transformation, and the composition of architecture refinements. Section 24.4 presents a case study using a theoretic example to show the feasibility of the approach. Finally, Sect. 24.5 presents the conclusions and further work.

[1] Some authors use different names, remarkably "quality attribute requirement" or "quality requirement" as a synonymous of NFR.

24.2 Problem Statement and Challenges

This section discusses in details the problem that we address in this paper. First, we present the inputs and assumptions of our contribution. Then we enumerate the challenges raised by current limitations in interpreting this knowledge.

To our understanding, MDE processes for embedded systems are keen on providing the level of information considered as the main inputs of the method:

- The model of a system architecture that represents software and hardware components, along with the bindings between them. Such model constitutes a candidate solution that could be refined or improved through applying tailored architectural model transformations.
- System NFR and goals that must be met by the system. For example, constraints related to: end-to-end data flow latency; services response time; availability; and power consumption are part of usual NFR of embedded systems. In addition, the definition of quality goals refines some of the NFR into an objective of minimizing or maximizing a quality attribute. For example, minimizing power consumption. The main difference between a quality goal and a NFR is that the latter gives a strict definition when to approve or discard a solution while the former aims at improving the characteristic of a solution (as long as all the requirements are met).
- A set of model transformations that implement refinements or improvements of the source model, each of them focusing on increasing the satisfaction of a quality objective (reducing power consumption, increasing availability, etc.).

In the following, we explain the challenges raised by merging this information in order to automate design decisions.

As explained in the Introduction section, our objective is to take advantage of the formalization of (1) requirements, (2) sensitivities, and (3) model transformations in order to automate the design space exploration towards an architecture that satisfies requirements and reach goals at best.

The first challenge raised by this objective comes from the necessity of solving tradeoffs between competing goals. For example, availability can be increased by redundancy patterns to the price of increasing the system's power consumption. To the best of our knowledge, there are no existing approaches that would describe how to solve the tradeoff between quality attributes expressed as design pattern (as a model transformation). When a design pattern is implemented as a model transformation improving a quality goal, solving the competition between these goals leads to selecting and composing these model transformations. Indeed, transformations are usually defined to be generic (like design patterns) whereas they will compete with other transformations when used in practice. As a consequence, a combination of competing model transformations should be used during the design of an embedded system.

The second challenge comes from the size of the design space to explore in order to study such tradeoffs. To answer this challenge, it is thus of prime importance to automate the selection and composition of model transformations. Of course, the quality of the exploration algorithm is very important in this context, as well as the implementation of a framework to enable the selection and composition of model transformations. The evaluation and optimization of the exploration algorithm is left as a future work of this contribution, which focuses first on automating the selection and composition of model transformations.

Finally, the last main challenge of this approach is the management of dependencies between model transformations, as model transformations can be used sequentially and potentially with an imposed order. In this paper, we consider model transformations that apply to a given element as competing and independent transformations: only one of them can be selected and applied to this element independently of other transformations.

24.3 Method Overview

At coarse grain, model transformations are grouped into two main categories of model transformation languages: (1) rule-based transformation languages, or (2) imperative transformation languages [5]. Rule-based transformation languages, such as ATL and QVT-R, are of great interest in our context since they express model transformation logics in a way that is easy to interpret and adapt. Indeed, modifying the pattern matching part of a transformation rule is sufficient to modify the set of elements this rule applies to. In addition, superimposition helps to combine different sets of rules [11]. Contrasting with rule-based formalisms, the adaptation of imperative transformation languages requires a deep understanding of the control flow graph that leads to the execution of one or another transformation. Model transformations we deal with in this paper are endogenous (AADL [12] to AADL model transformations) in order to refine or improve the architecture of an embedded system while representing the result of this refinement in an analyzable model (using AADL as an output language).

To facilitate aforementioned pattern matching modifications, thus creating new model transformation rules in an automated manner, the artifacts described hereafter have been designed.

The Transformation Rules Catalog (TRC) was designed to gather MDD knowledge, describing not only available model transformations, but also their impact on quality attributes. Figure 24.1a presents defined metamodel for the TRC artifact. Quality attributes are defined in the RDAL specification. The impact of a model transformation on a given quality attribute is specified by an integer value ranging from −5 (very negative effect on the quality attribute) to +5 (very positive impact on the quality attribute). It is thus of the responsibility of the transformations specifier to define the impact of each of the transformations on the defined quality attributes.

The Transformations Iteration Plan (TIP) is an artifact with which to describe which transformation rules should be applied and how to produce the final architecture refinement rules. Its metamodel is presented on the Fig. 24.1b.

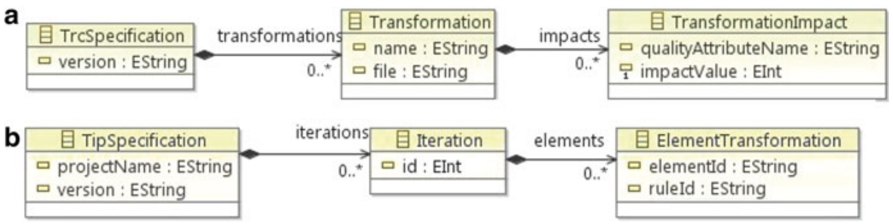


Fig. 24.1 Metamodels (**a**) transformation rules catalog, (**b**) transformations iteration plan

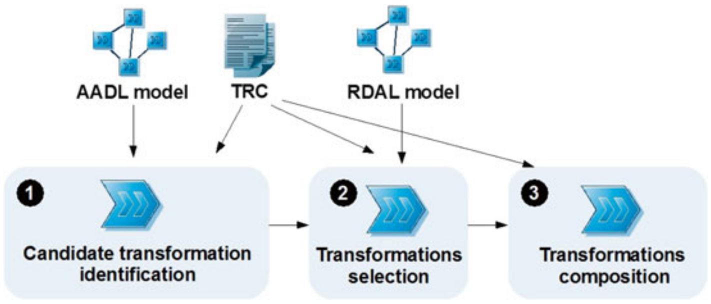

AADL model
TRC
RDAL model
1
Candidate transformation identification
2
Transformations selection
3
Transformations composition

Fig. 24.2 Process schema

In the following subsections, we present the overall process that we have implemented in order to automate the specific architecture refinements generation. Figure 24.2 illustrates this process and its three main activities.

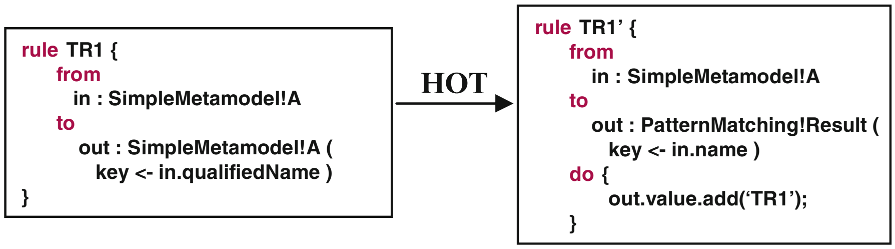
24.3.1 Candidate Transformation Identification

In order to identify the transformations that can be applied to a particular element of the input model, we introduce a method based on the steps presented hereafter.

Step 1. Application of the HOT that takes as input the generic model transformations referenced in the TRC and generates new model transformations that after their application on the architecture model produce lists of transformations that can be applied to each element of the input model.

Step 2. Execution of transformations generated in step 1. Data collected after executing all of the newly created transformations allow creating a set of tuples <*element, list of transformations*>. It constitutes the output of the pattern-matching phase of the transformations execution engine;

Figure 24.3 illustrates the ATL implementation of the HOT where rule *TR1* is transformed into a new rule *TR1'*. Rule *TR1* originally transforms an input element of type *A* (that conforms to *SimpleMetamodel*) into an output element of type that conforms to the same metamodel. On the right side of Fig. 24.3 the output of the



```
rule TR1 {
    from
      in : SimpleMetamodel!A
    to
      out : SimpleMetamodel!A (
        key <- in.qualifiedName )
}
```

```
rule TR1' {
    from
      in : SimpleMetamodel!A
    to
      out : PatternMatching!Result (
        key <- in.name )
    do {
        out.value.add('TR1');
    }
```

Fig. 24.3 Schema of the higher-order transformation for pattern matching

HOT is a rule *TR1'* that transforms all the elements of type *A* (same pattern matching clause as the original rule *TR1*) into an output element *out* of type *Result* from the *PatternMatching* metamodel (defined by us). The output element is then initialized in the following manner: *key* attribute receives the name (a unique identifier) of the element that was matched by this rule, and *value* attribute is initialized with a list of names of the matched rules that apply, in this case *TR1*. The execution of the rule *TR1*' on a given source model conforming to *SimpleMetamodel* outputs a XMI file containing tuples of all possible rules (among those referenced in TRC) that can be applied to each source model element.

Each of the transformations declared in the TRC, when applied the aforementioned steps 1 and 2, outputs the result of the pattern matching condition checking applied on the model, giving the set of candidate transformations for a given design element.

24.3.2 Transformations Selection

We distinguish the following inputs to the selection process: (1) Set of tuples <*element, list of transformations*> resulting from the candidate transformations identification; (2) RDAL specification—information about the quality attributes to optimize along with responsible for them architecture elements (sensitivities); (3) TRC artifact—information on the judged impact that each transformation has with respect to all the quality attributes which are important for the system's architecture.

The output of the selection process is the Transformations Iteration Plan (TIP), which defines which model transformations should be used in an iteration of the architecture refinement. It consists of the actual tuples <*element, transformation*> to be applied. Figure 24.1b shows the ecore metamodel of the TIP artifact. *Element7ransformation* meta-class defines aforementioned tuples describing which transformation (attribute: *ruleld*) is going to be applied on which model element (attribute: *elementld* is the element's qualified name which identifies every element in deterministic manner). After the iteration planning, the architecture refinement composition is to be performed.

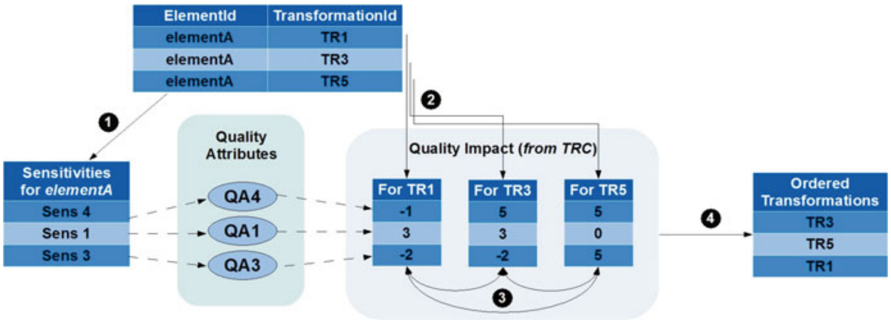
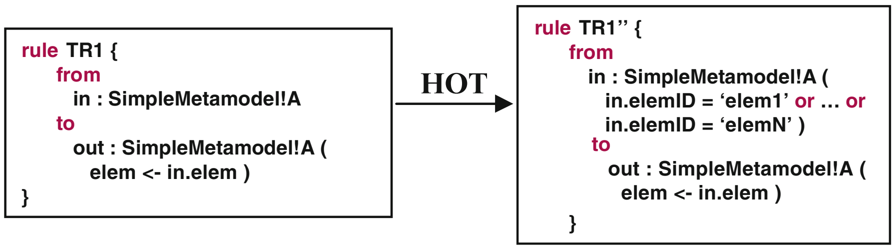


Fig. 24.4 Transformations selection process

The inputs presented above (set of tuples, RDAL specification, and TRC) combines several factors to be considered while analyzing possible architectural refinements. These factors can not be analyzed in isolation, as their complex relationships have to be considered all together. For this purpose, different multi- criteria decision making methods are known (e.g. AHP [7]). They can be easily integrated to our approach to analyze provided input data from the goals' specification and MDD knowledge. We propose to use simple rules with which to select transformations of the best possible impact on targeted quality goals.

The selection process is illustrated on Fig. 24.4. In Step 1, we iterate on the elements for which some applicable transformations were found (candidate tuples). For each of these elements, we retrieve from the RDAL specification a list of sensitivities. Sensitivity is simply a reference to a quality attribute from an element of the architecture. In the retrieved list, sensitivities are ordered with respect to their priority in RDAL metamodel. On Fig. 24.4, *elementA* is sensitive, by order of importance, to quality attributes *QA4*, *QA1* and *QA3*. In Step 2, for each candidate transformation to be applied on *elementA*, the list of quality impact values is retrieved from the TRC specification. E.g. *TR1* has negative impact (−1) on quality attribute *QA4*, a positive (+3) impact on *QA1*, and a negative impact (−2) on *QA3*. In Step 3, values obtained in step 2 are then used for the selection of rules to be applied. Our tool performs pair-wise comparisons considering the priority of the quality attribute and the impact value. However, different multi criteria decision making methods can be applied here. In Step 4, the output of the selection process is a list of transformations to be applied on particular model element. They are ordered from the best to the worst with respect to the quality properties that should be taken into account. Optimal refinement selection is a very challenging task. However, our selection algorithm along with the refinements generation provides means to review the refinements space in an efficient manner. Identified solutions should be evaluated in the proposed in TIP order, validating the correctness of the identified refinements if the already search solutions are not satisfactory, and alternatives have to be explored.



```
rule TR1 {
    from
      in : SimpleMetamodel!A
    to
      out : SimpleMetamodel!A (
        elem <- in.elem )
}
```

```
rule TR1" {
    from
      in : SimpleMetamodel!A (
        in.elemID = 'elem1' or ... or
        in.elemID = 'elemN' )
     to
      out : SimpleMetamodel!A (
        elem <- in.elem )
    }
```

Fig. 24.5 Schema of the HOT for actual transformations generation

24.3.3 Transformations Composition

The composition of architecture refinement is performed by using the HOT to transform original rules into a new set of rules which are going to be applied on particular model elements. This process takes as input the rules definitions (TRC) and also the refinements configurations (TIP).

Figure 24.5 illustrates the realization of mentioned above refinements generation with an ATL implementation of the HOT where rule *TR1* is transformed into a new rule *TR1"*. Rule *TR1* was originally specified to transform all the elements of type A in source metamodel *SimpleMetamodel. TR1"* uses the transformation mechanism of the original rule but with a limited set of elements for which this rule has been selected: as depicted on Fig. 24.5 (changed pattern matching condition in the output transformation *TR1"*).

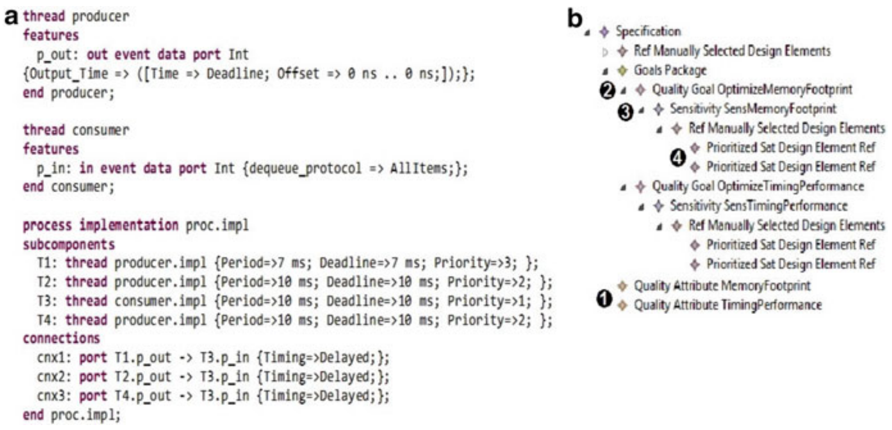
Finally, the framework implemented to support presented process executes generated model transformations on the input AADL model to produce a refined AADL model. If the evaluation of the output model is not satisfactory, the architect has the possibility to continue the design space exploration from step 2 (i.e. transformations selection), to produce output models with different results in terms of quality attribute.

Next section details designed intermediate artifacts and shows their use in the automation of the complete process introduced in this section.

24.4 Case Study

Architectures of embedded systems are frequently very large models where many competing quality attributes have to be taken into account. At the same time, the amount of possible architectural refinements increases as developers are becoming familiar with model-driven techniques. However, the complexity and the dimensions of the solution space make that finding the most correct solution is a challenging, time-consuming task.

In this section, we present a case study where given candidate architecture is refined in order to ensure the satisfaction of two quality attributes: *MemoryFootprint*



a
```
thread producer
features
  p_out: out event data port Int
{Output_Time => ([Time => Deadline; Offset => 0 ns .. 0 ns;]);};
end producer;

thread consumer
features
  p_in: in event data port Int {dequeue_protocol => AllItems;};
end consumer;

process implementation proc.impl
subcomponents
  T1: thread producer.impl {Period=>7 ms; Deadline=>7 ms; Priority=>3; };
  T2: thread producer.impl {Period=>10 ms; Deadline=>10 ms; Priority=>2; };
  T3: thread consumer.impl {Period=>10 ms; Deadline=>10 ms; Priority=>1; };
  T4: thread producer.impl {Period=>10 ms; Deadline=>10 ms; Priority=>2; };
connections
  cnx1: port T1.p_out -> T3.p_in {Timing=>Delayed;};
  cnx2: port T2.p_out -> T3.p_in {Timing=>Delayed;};
  cnx3: port T4.p_out -> T3.p_in {Timing=>Delayed;};
end proc.impl;
```

b
```
Specification
  Ref Manually Selected Design Elements
  Goals Package
❷   Quality Goal OptimizeMemoryFootprint
❸     Sensitivity SensMemoryFootprint
        Ref Manually Selected Design Elements
❹         Prioritized Sat Design Element Ref
          Prioritized Sat Design Element Ref
    Quality Goal OptimizeTimingPerformance
      Sensitivity SensTimingPerformance
        Ref Manually Selected Design Elements
          Prioritized Sat Design Element Ref
          Prioritized Sat Design Element Ref
❶ Quality Attribute MemoryFootprint
  Quality Attribute TimingPerformance
```

Fig. 24.6 Case study artifacts: (**a**) AADL specification, (**b**) RDAL specification

and *TimingPerformance*. ***Memory footprint*** refers to the amount of memory resources that the running system needs for proper execution. It is one of the characteristics of embedded systems which possess limited memory resources. Optimization of this quality property is often a challenging task. ***Timing performance*** is one of the most important quality properties of embedded real-time systems. At design time, architects have to ensure that it is possible to satisfy the imposed on the system timing constraints. The application of the process presented in this paper is automated by JAVA/ATL implementation.

This case study bases on a theoretical example to show how our approach deals with the complexity of embedded systems' architectures design. In this example, where input model consists of 4 elements, 2 transformations apply to 3 of them, and another 2 transformations apply to the fourth element. Thus we get ten alternative possible architectures. Even in this very simple example, the automation of architectural refinement proves beneficial, as the different refinement combinations are not always easy to be configured manually.

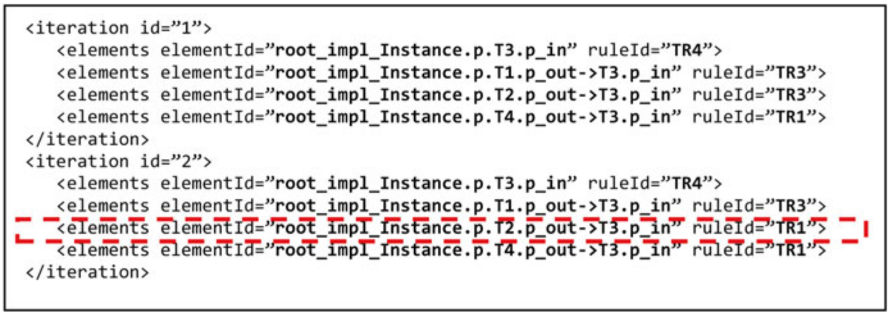
Figure 24.6a presents a simple system specification where three threads of type *Producer* (*T1*, *T2*, *T4*) interact with one *Consumer* thread (*T3*) through event data ports. Moreover, the specification consists of the definitions of a system, one processor with one process running.

In the requirements specification (RDAL), important quality attributes are assigned to elements of the architecture. This assignment is called *sensitivity*. It indicates the quality attributes to be considered while transforming sensitive elements. In our example, connections *cnx1* and *cnx2*, as well as input event data port *p_in* are sensitive to *MemoryFootprint* while for *cnx3* the priority is assigned to *TimingPerformance* (see Fig. 24.6b).

In our example, TRC consists of four rules. Rules *TR2* and *TR4* are specified to transform an *in event data port* and rules *TR1* and *TR3* apply to a *connection*. These transformations can be used to implement lock-free queues for managing delayed communications between periodic tasks [13]. The underlying theory is

Table 24.1 Quality impact from transformation rules catalog

		Transformation Rule			
		TR1	TR2	TR3	TR4
Quality property	Memory Footprint	–4	–4	2	2
	Timing Performance	5	5	–3	–3



```
<iteration id="1">
   <elements elementId="root_impl_Instance.p.T3.p_in" ruleId="TR4">
   <elements elementId="root_impl_Instance.p.T1.p_out->T3.p_in" ruleId="TR3">
   <elements elementId="root_impl_Instance.p.T2.p_out->T3.p_in" ruleId="TR3">
   <elements elementId="root_impl_Instance.p.T4.p_out->T3.p_in" ruleId="TR1">
</iteration>
<iteration id="2">
   <elements elementId="root_impl_Instance.p.T3.p_in" ruleId="TR4">
   <elements elementId="root_impl_Instance.p.T1.p_out->T3.p_in" ruleId="TR3">
   <elements elementId="root_impl_Instance.p.T2.p_out->T3.p_in" ruleId="TR1">
   <elements elementId="root_impl_Instance.p.T4.p_out->T3.p_in" ruleId="TR1">
</iteration>
```

Fig. 24.7 Identified architectural refinements (TIP artifact)

presented in [14]. Transformations *TR1* and *TR2* implement queues accesses with lookup tables that contain indexes of the queue which a task can access for each of its activation (over a hyper-period). Transformations *TR3* and *TR4* implement queues accesses by computing indexes at runtime. Thus, *TR3* and *TR4* consume more memory but less CPU time in comparison to *TR1* and *TR2*. Details about their influence on particular quality attributes can be found in Table 24.1.

In order to validate the different steps described on Fig.24.3 Sect. (24.3), we have executed the process on the use-case described in previous subsection. Several configurations of the architectural refinements have been automatically identified and necessary tailored transformations have been generated. Figure 24.7 shows the output of that process for two successive iterations of the selection algorithm.

- For iteration 1, the result shows that the architecture refinement will contain different transformations applied to different model elements, since three elements of the input model are sensitive to *MemoryFootprint* quality attribute, they are transformed by the low memory footprint version of available transformations (*TR3*). One input element is transformed by the transformation resulting in lower CPU consumption, as this element is primarily sensitive to *TimingPerformance*.
- For iteration 2, the identified architecture refinement sets the transformation that is preferable for improving the *TimingPerformance* of the architecture (*TR1*) on one more element of the input model in comparison to the iteration 1 (element highlighted by a dashed line on Fig. 24.7).

The resulting architecture refinements produced in each of the iterations are next executed in RAMSES execution framework [13], producing candidate architectures that need to be validated by the analysis tools and domain experts.

The main outcome of this contribution is to automate the design space exploration, based on the selection and composition of legacy transformations in architectural refinement and improvement processes. The performed case study shows design space exploration by the automated production of different candidate architectures. Moreover, it proves to be useful in rapid prototyping and evaluation of embedded systems architectures, with respect to specified quality properties.

24.5 Conclusions and Further Work

It is well accepted in the software architecture community that goals and quality attribute requirements are the most important drivers of architecture design [15]. However, dealing with several competing goals and quality attribute requirements to perform well-informed architectural design decisions is not an easy task. MDD techniques, based on explicit modeling of these goals and quality attribute requirements, and design decisions (this latter as model transformations) will allow not only to perform automated and documented design decisions but to preserve important architectural knowledge. In this paper, we introduced a specific MDD approach tailored for embedded software architecture design with the aim of helping software architects to perform these informed design decisions based on automating the selection and composition of competing model transformations [16]. The main contribution of the paper is to bring together quality attribute requirements to design decisions (represented as model transformations) and to use this information to automate the selection of architectural refinements based on these relationships.

Currently, some improvements are being planned in the architecture refinement framework, among them: integrating an evaluation process for the resulting architecture to quantitatively measure the resulting architecture (regarding the goals and quality requirements stated); reviewing the RDAL metamodel to be able to capture more expressive requirements; and enriching our implementation tool by adding a user-friendly interface to allow architects to easily define and test different architectural transformations. Finally, we are developing a tool framework to automate all this process and we are planning to use this tool to conduct an industrial case study in the context of a medium-size embedded system development project.

Acknowledgments This research work is partially funded by the "Chaire Ingénierie des Systèmes Complexes" and by the MULTIPLE project (MICINN TIN2009-13838).

References

1. Ozkaya I, Bass L, Sangwan R, Nord R (2008) Making practical use of quality attribute information. IEEE Software 25(2):25–33
2. Anton A, McCracken W, Potts C (1994) Goal decomposition and scenario analysis in business process reengineering. In: 6th Conference on advanced information systems engineering (CAiSE'94), Utrecht, Holland

3. Clements P, Bass L (2010) Using business goals to inform a software architecture. In: 18th IEEE Int. Requirements engineering conference (RE'10. IEEE CS), pp 69–78
4. Loniewski G, Insfran E, Abrahão S (2010) A systematic review of the use of requirements engineering techniques in model-driven development. In: 13th MODELS Conf. Springer, Berlin, pp 213–227
5. Elahi G, Yu E (2007) A goal oriented approach for modeling and analyzing security tradeoffs. In: 26th Int. Conf. on conceptual modeling (ER'07). Springer, Berlin, pp 375–390
6. Letier E, van Lamsweerde A (2004) Reasoning about partial goal satisfaction for requirements and design engineering. In: 12th ACM SIGSOFT Int. Symp. on Foundations of Software Eng (SIGSOFT '04/FSE-12, ACM, New York), pp 53–62
7. Svahnberg M, Wohlin C, Lundberg L, Mattsson M (2003) A quality-driven decision support method for identifying software architecture candidates. Int Journal of Software Engineering and Knowledge Management 13(5):547–573
8. Ameller D, Franch X (2012) Linking quality attributes and constraints with architectural decisions. In: CoRR abs/1206.5166
9. Sterritt A, Cahill V (2008) Customisable model transformations based on non-functional requirements. In: IEEE congress on services, CS, Washington, pp 329–336
10. The Requirements Definition and Analysis Language Annex of AADL, https://wiki.sei.cmu.edu/aadl/index.php/Standardization
11. Wagelaar D, Tisi M, Cabot J, Jouault F (2011) Towards a general composition semantics for rule-based model transformation. In: 14th Int. Conf. on model driven engineering languages and systems (MoDELS 2011), LNCS 6981, pp 623–637
12. The Architecture Analysis & Design Language (AADL), version 2, Jan 2010, http://standards.sae.org/as5506a/
13. Cadoret F, Robert T, Borde E, Pautet L, Singhoff F (2013) Deterministic implementation of periodic-delayed communications and experimentation in AADL. In: 17th International symposium on object/component/service-oriented real-time distributed computing, June 19–21, Paderborn
14. Cadoret F, Borde E, Gardoll S, Pautet L (2012) Design patterns for rule-based refinement of safety critical embedded systems models. In: 17th Int. Conf. on Eng. of complex computer systems (ICECCS'12). IEEE CS, Washington, DC, pp 67–76
15. Ameller D, Franch X, Cabot J (2010) Dealing with non-functional requirements in model-driven development. In: 18th IEEE Int. Requirements Engineering Conf. (RE'10). IEEE Computer Society, Washington, DC, USA, pp 189–198
16. Czarnecki K, Helsen S (2003) Classification of model transformation approaches. In: Workshop on generative techniques in the context of model-driven architecture (OOPSLA'03), Anaheim, CA