Modeling and Analyzing Non-Functional Properties to Support Software Integration

Henning Agt[1], Gregor Bauhoff[2], Ralf-D. Kutsche[1], and Nikola Milanovic[2]

[1] Technische Universität Berlin
{henning.agt,ralf-detlef.kutsche}@tu-berlin.de
[2] Model Labs GmbH
{gregor.bauhoff,nikola.milanovic}@modellabs.de

Abstract. Software integration is one of the major needs as well as cost driving factors in the software industry today. Still, very few established methodologies exist, especially those addressing integration with respect to non-functional properties. Industry studies show that disregarded and hidden non-functional incompatibilities between systems and their interfaces are the constant source of errors and costly workarounds. We introduce a model-based process that allows dynamic definition of non-functional properties in the context of software integration, present a NFP taxonomy, and propose a method for formal analysis of interface incompatibilities with respect to these properties.

1 Introduction

Software and data integration practice is usually focused on structural and communication compatibility conflicts that require transformation of data types and structures and connection of communication channels. Integration frameworks and tools exist which address these issues to some extent (e. g., [1,2,3]). However, non-functional properties (NFP), such as reliability, availability, security, timeliness and cost play the crucial role in software integration when it comes to satisfying business process requirements. Their analysis is either neglected or informal, following best practices. Sometimes this is not enough as non-functional incompatibilities may compromise not only the quality of integration solution, but also limit its functionality.

For these reasons, as part of the research project BIZYCLE[1], we investigate in large-scale the potential of model-based software and data integration methodologies, tool support and practical applicability for different industrial domains (the consortium comprises of six system integrators from health, publishing and media, facility management, production and logistics sectors). We have developed an MDA-based methodology [4,5], we call it the BIZYCLE integration process, to systematically model integration projects at different levels of abstraction in order to achieve automatic analysis, code generation and integration. In this section we cover the basic aspects of the process and in the

[1] This work is partially supported by the Bundesministerium für Bildung und Forschung BMBF under grant number (Förderkennzeichen) 03WKBB1B.

C. Salinesi and O. Pastor (Eds.): CAiSE 2011 Workshops, LNBIP 83, pp. 149–163, 2011.

remainder of the paper we concentrate on how to use models of non-functional properties to facilitate integration compatibility checks.

The essence of the BIZYCLE integration process is to provide multilevel integration abstractions and to make software integration partially automatic. The levels of abstraction are computational independent model (CIM), platform specific model (PSM) and platform independent model (PIM) level. The CIM level captures the integration business process. Existing systems to be integrated, interfaces and their properties realizing an integration scenario are described at the PSM level. In contrast to the usual MDA philosophy, all PSM models are transformed to the PIM level, which represents a common abstraction level. The PIM level is used to perform an integration conflict analysis to discover incompatible interfaces. Appropriate connector (mediator) model and code for required connector components are generated based on results of the conflict analysis. Our conflict analysis addresses structural, behavioral, communication, semantic and non-functional property mismatches. For example, the semantic conflict analysis [6] is carried out using ontology annotations and reasoning with Prolog.

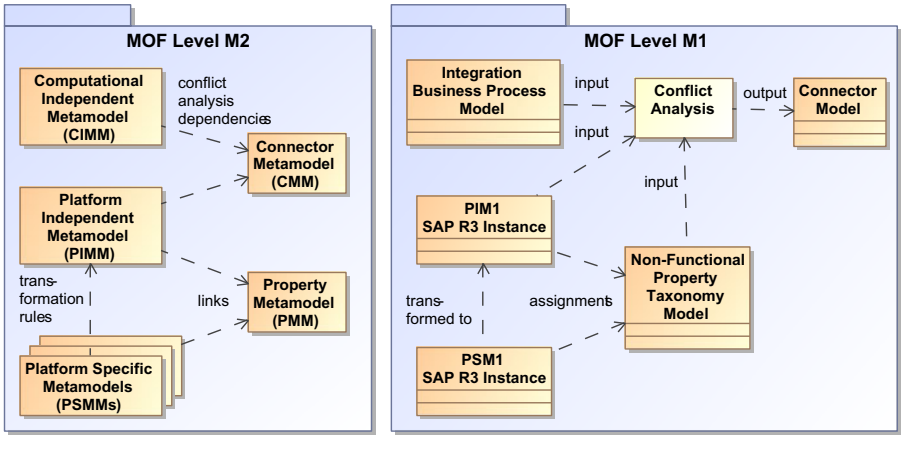

Fig. 1. BIZYCLE modeling methodology overview (excerpt)

Figure 1 presents part of our modeling methodology [7,5] and how NFP modeling is integrated. Dependencies between the framework's metamodels at the MOF M2 level are shown on the left side of the figure. Example models at the M1 level are given on the right (due to space reasons the figure includes only one PSM and PIM model, usually integration projects involve two or more systems). Our integration framework offers several metamodels and respective model editors on platform specific level to describe systems interfaces (PSMMs). We currently support SAP R/3 ERP systems, J2EE applications, SQL-based relational database systems, Web Services and XML Schema based files. A set of transformation rules for each platform is used to perform model transformations to common abstraction level with a single metamodel (PIMM). The PIMM

and all PSMMs are linked to the Property Metamodel (PMM) presented in this paper to be able to associate NFPs to model elements at these abstraction levels. The Connector Metamodel (CMM) provides means to express conflict analysis' results in terms of Enterprise Application Integration (EAI) patterns [8].

In this paper we report our first practical experiences with model-based system integration with respect to NFPs. The rest of the paper is organized as follows: Section 2 presents a metamodel for expressing user-defined non-functional properties, their categories and measurement units. We focus on modeling of non-functional properties in a general way. Compared to the OMG QFTP specification [9] we are not limited to Quality of Service characteristics and UML based models. Additionally, we offer explicit modeling of measurement units to enable model-based comparison and calculation of NFPs. Section 3 describes a taxonomy with non-functional properties which we identified as relevant in the software integration context. It was implemented using our NFP metamodel. Section 4 proposes a method for formal analysis of interface incompatibilities with respect to these properties. Related work and conclusion are given in Section 5 and 6.

2 Non-Functional Property Metamodel

In order to model non-functional properties in our integration framework, we propose a property metamodel (PMM) for expressing NFPs, their categories and units of measurement and assigning them to other model elements. The metamodel is implemented in our integration framework using the Eclipse Modeling Framework. In the following we use `typewriter` font to refer to metamodel classes and attributes and describe examples on instance level with *italic* font.

Figure 2 gives abstract syntax of the property metamodel to create property categories (e. g., *performance*), non-functional properties (such as *throughput*) and units of measurement (such as *Mbit/s*).

Basic structural Features. Properties and measurement units are modeled separately and are grouped into categories. The categories and their elements together form the `PropertyModel`. A `NamedElement` metaclass (not shown in the Figure) passes attributes `+shortName:String[1]` (for abbreviations, e. g., *MTTF*) and `+longName:String[1]` (for full names, e. g., *Mean Time To Failure*) to most of the classes.

Properties. The `Property` metaclass owns the `scope`-attribute to define NFP validity. The `PropertyScope` can be one of the following values:

- `System`-wide (e. g., *mean time to repair* of a database system)
- `Interface`-wide (e. g., *availability* of a Web service)
- `Function`-related (e. g., *cost per invocation* of a Web service operation)
- `Parameter`-related (e. g., *accuracy* of a J2EE component method's return value)

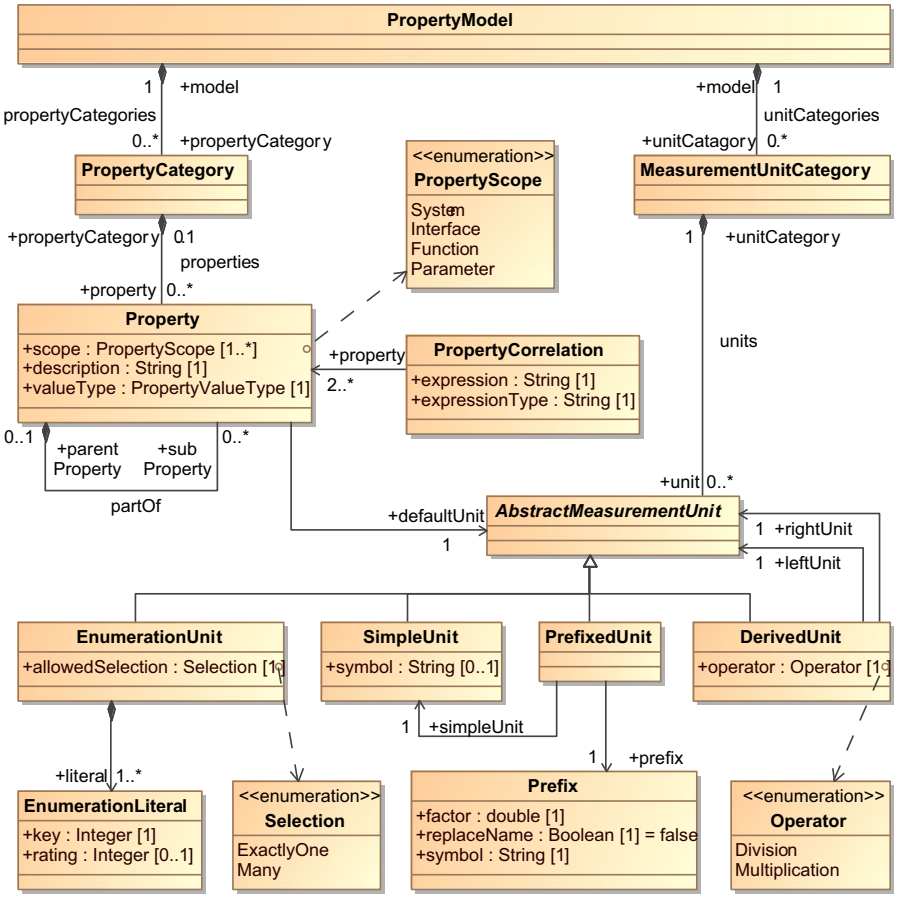

Fig. 2. Property Metamodel - Properties and units of measurement

The `description`-attribute includes additional explanatory text of the property. The attribute `valueType` constraints the value kind of a property to support comparison: character-based values (`String`), floating point numbers (`Double`), integer-based representations (`Long`) and `Boolean`-values. Properties can be nested with the `partOf`-relation. A property is associated with a default unit, which is the standard unit for property assignment.

Dependencies between properties can be declared with `PropertyCorrelation`. It is evaluated at runtime and can contain mathematical or boolean `expressions`. For instance, the calculation rule for availability ($A = \frac{MTTF}{MTTF+MTTR}$) can be expressed with MathML Content Markup [10] using property names as variables.

Measurement Units. The metamodel offers four units types to build units of measurement. The `SimpleUnit` is used to create basic SI-units [11] or ISO 80000

units such as *second* or *bit*. Additionally to `shortName` and `longName` attributes, a `symbol`ic representation of a unit can be specified, if available.

`PrefixedUnit`s are created from simple ones by combining them with `UnitMultiple`. Usually a prefix is added in front of the unit name (e. g., metric prefix *kilo*, binary prefix *kibi*). A prefix of a unit is represented by a `factor` and a `symbol`. Contrary, there exist also non-SI measurement units that replace the whole unit name (e. g., *minute*). The change of the name can be controlled by the `replaceName`-attribute.

Composed units are modeled with the `DerivedUnit`[2] metaclass. It combines left- and right-handed units with an `Operator`. Reuse and nesting of simple units, prefixed units and derived units is allowed to build all kinds of measurement units (e. g., kg/m^3).

Finally, the `EnumerationUnit` describes possible values of a property as a set of `EnumerationLiteral`s for non-numeric NFPs. The `allowedSelection`-attribute declares whether a property assignment must use exactly one literal or can use many literals. The `key`-attribute of the `EnumerationLiteral` is an additional unique numeric code. The `rating` attribute can assess a literal (e. g., to assess encryption algorithms). For example, we use enumerations to model ISO 4217 currency names.

Assigning Properties. On meta level, the `PropertyAssignment` (Figure 3) is the link between the actual non-functional property and elements of other metamodels that shall be annotated with NFPs. The `type`-attribute handles whether the property is offered/provided by the system or interface (e. g., *provided uptime*), or expected/required from other systems (e. g., *required network encryption*). Property values are either given as a single value (`PropertyValue`), as `PropertyValueRange` (e. g., minimum and maximum) or as a set of single values (`PropertyValueSet`). In case the default unit shall be overridden, the `unitModifier` is used (constrained to use different prefixes only). Enumeration unit literals are chosen with `enumLiteralSelection` (e. g., selecting *EUR* currency for a cost property from the currency code enumeration).

Example of a Property Assignment. In this paragraph, we demonstrate how the connection between different metamodels works. We have developed a platform specific metamodel [5] to describe interfaces of SAP R/3 enterprise resource planning systems (system access, communication channels, method signatures, parameter types, etc.). Upper part of Figure 4 presents the established link between the property metamodel and the SAP metamodel.

The SAP metamodel contains a metaclass `SAP_R3` that represents a whole SAP R/3 system. `SAP_R3_Interfaces` are used to access certain parts of the system. We establish an association between `PropertyAssignment` and `SAP_R3`

[2] In the SI unit system and related ISO specifications the term *derived unit* usually refers to a fixed set of measurement units that have been derived from the SI base units. Here we use the term in a more general object-oriented way. Derived units are units that can be composed from other units using a formula.

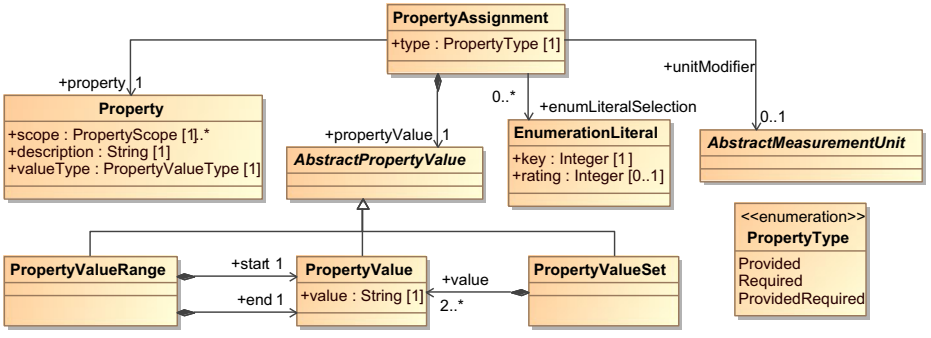

Fig. 3. PMM - Property assignment

metaclass to be able to assign any user-defined NFP to a concrete SAP installation. In our implementation based on the Eclipse Modeling Framework we use attributes that link between different Ecore files.

Lower part of Figure 4 shows an example of a property assignment on model level. We use object diagram notation to represent the instances. Instances *Server 1* and *purchase management* on the left side belong to a platform specific model of a concrete SAP R/3 system. Right part of the figure depicts excerpt of our property taxonomy for *Throughput* and the property assignment. The model contains two simple units *bit* and *second*. The prefixed unit is *Mbit*. Finally, the derived unit constitutes *Mbit/s*, the default unit for the *Throughput* property. The assignment links *Throughput* to *Server 1* and associates a value of *100*.

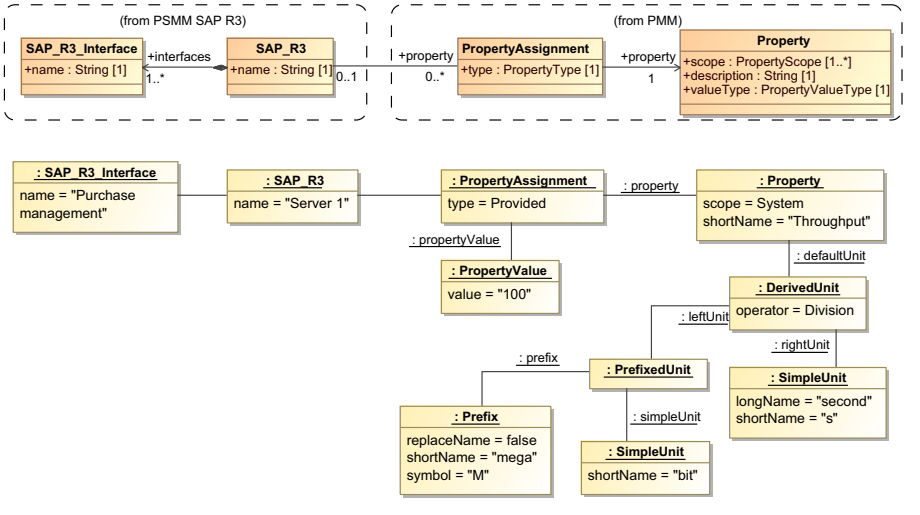

Fig. 4. Assignment example - SAP R/3 Server 1 provides 100 Mbit/s throughput

3 Non-Functional Properties in Software Integration

In this section we present a taxonomy of NFPs which we identified as relevant in the software integration context (excerpt is given in Figure 5). The taxonomy is modeled using the metamodel described in the previous section. It is not intended to be complete, but will be used here as a starting point to discuss modeling of non-functional aspects of software systems, interfaces, parameters and connectors, evaluate integration solutions, compute overall properties, detect non-functional mismatches and rank integration alternatives.

Entries of taxonomy's first level are property categories that thematically group NFPs defined at the second level. The third level (not shown in the picture) characterizes each property with description, type (e. g., float values), scope and the default unit (e. g., milliseconds for latency or currency codes for cost per invocation). Units are described according to standards such as ISO 80000-3:2006 (time), ISO 80000-13:2008 (IT units), ISO 8601 (date) or ISO 4217 (currency codes). In the following we provide more details on each property category.

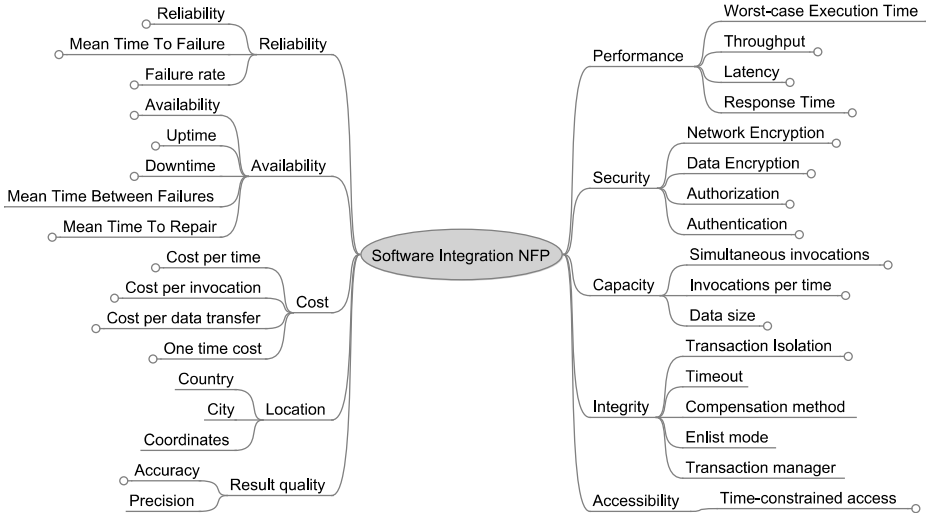

Fig. 5. Taxonomy of software integration non-functional properties (excerpt)

Reliability incorporates properties related to the ability of a system or component to perform its required functions under stated conditions for a specified period of time. Reliability is the probability that a system operates without failure for a given time t. Usually, steady-state reliability is used to characterize integrated systems, where observed interval is the system's lifetime. Reliability attributes such as *failure-rate* (λ) and *mean time to system failure (MTTF)* can also be specified.

Availability is similar to reliability, with one important exception: it allows for a system to fail and to be repaired. Thus, additional properties appear in this category: *mean time to repair (MTTR)*, *mean time between failures (MTBF)* as well as average *uptime/downtime* per year. Using this category, fault-tolerant lifecycle of integrated systems and their components can be analyzed.

Cost related NFP are used to compare different software integration alternatives. For example, the comparison of *cost per data transfer*-properties is achieved by specifying their unit in terms of a currency code in relation with the base unit byte with metric or binary prefix. Values with different currencies and different amounts of data volumes can be compared by determining the current exchange rate and by converting the units.

Performance category includes properties that are either bandwidth or timing related. They enable detection of possible integration bottlenecks and allow investigation of connector features such as caching as well as timing constraints.

Security category describes encryption and access control related properties. For example, *network encryption* property consists of three sub-properties *strength*, *protocol* and *technology*. Encryption strength can be directly compared, while protocols have to be evaluated based on sets of provided and required enumerations, correlations or rating (e. g., SHA vs. RIPEMD).

Capacity properties are considered to avoid system overloads during integration. For example, *data size* property specifies the maximum amount of data that can be passed to a system at once (value type: integer, default unit: megabyte). Together with *throughput*, duration of integration runs can be thus analyzed.

Integrity describes transactional behavior. *Isolation level* supports read uncommitted, read committed, repeatable read and serializable levels. *Timeout* defines system or interface timeout which can be used to discover timing mismatches, e. g., if a service with 1 hour timeout is waiting for a service with 1 day WCET, there is a timing conflict.

Location category describes geographical system/service location, as it may be necessary to determine validity of an integration scenario. For example, confidential data storage may be restricted to particular countries, because of legal considerations. To enable this, we include country name and its ISO code, as well as city and GPS coordinates of the system location.

Result quality describes attributes of the data/messages produced by a system. *Accuracy* represents calculation correctness (e. g., number of decimal places or maximal guaranteed computation error), while *precision* is the measure of the system output quality which may be gathered and evaluated statistically over a period of time, potentially also by users (in form of a reputation scale).

Accessibility is expressed by *time-constrained access*: it represents concrete time intervals or periodic time slots in which a system is accessible for integration tasks (e. g., Extract-Transform-Load tasks on Sundays 0:00-5:00 a.m.).

Figure 6 shows our NFP editor implementation using the plug-in and view architecture of Eclipse and EMF API to manipulate the models. Here we show the annotation of an SAP R/3 system interface with different availability and reliability NFPs and their values.

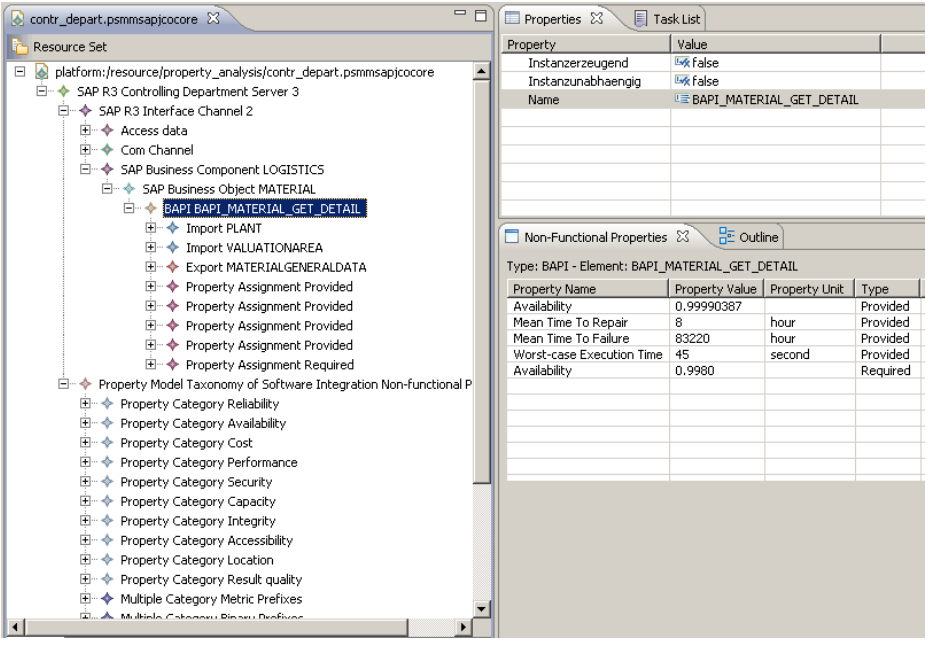

Fig. 6. Non-functional properties editor

4 Non-Functional Property Conflict Analysis

The task of NFP analysis is to determine if there are conflicts (i.e. incompatibilities or mismatches) between systems with respect to modeled non-functional properties. It is especially important in those integration cases where systems are functionally compatible, but hidden non-functional incompatibility compromises the integration solution. We provide the following case study: Figure 7 shows an instance of a so called connector model that describes message-based communication between the systems. The connector model is used in the last phase of the BIZYCLE integration process and is mostly generated from the models at the CIM level (integration business process) and the PIM level (systems and interfaces on a common abstraction level) and from the other conflict analysis phases (e. g., semantic or data type analysis [6,7]). In that last phase we treat all data exchanged between the systems as messages.

Figure 7 depicts different types of components: *Application Endpoints* representing the integrated systems and Message Processors, equivalent to EAI patterns [8] such as *Aggregators*, *Routers* or *Splitters* that mediate between application endpoints and route and transform messages. The components have *Ports* (P) and communicate via *Channels* (C). A complete language description can be found in [12]. The given connector model describes integration scenario with 4 systems: S_1, S_2, S_3 and S_4 which are annotated with (among others) non-functional descriptions, as explained in the previous sections. Based on the

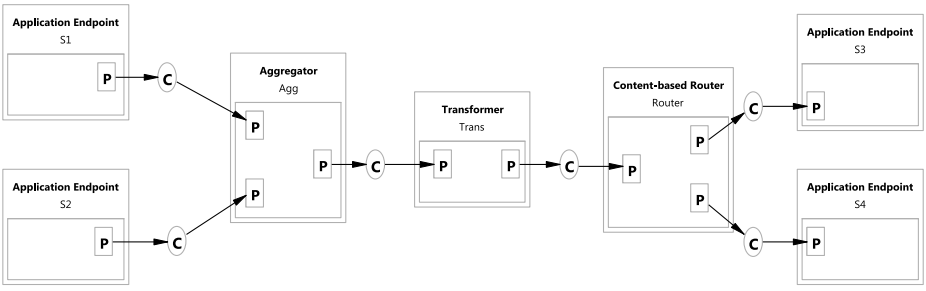

Fig. 7. Case study connector model

connector model, in the process of model transformation we generate an analysis model to investigate NFP mismatches.

The analysis model describes structural dependencies between connector components, with respect to NFP. The connector model is treated as a graph where application endpoints and message processors are vertices and channels are edges. Sets of source (S) and destination (D) vertices are identified first (in our example $\{S_1, S_2\} \in S$ and $\{S_3, S_4\} \in D$). Then all paths between S and D are discovered, using the standard algorithm for the detection of all paths between two vertices in a graph [13]. For each independent path that is discovered, a Boolean expression is derived: all vertices and edges on the path are connected with $\wedge$ if they belong to the same path. If more independent paths exist, they are connected with $\vee$. In our example, generated Boolean expression is:

$$S_1 \wedge S_2 \wedge Agg \wedge Trans \wedge Router \wedge (S_3 \vee S_4) \tag{1}$$

Note that we did not include channels in this analysis, assuming that they are ideal. After this step, Boolean expressions are minimized using the idempotence, associativity and distributivity rules of the Boolean algebra. Minimized equations may then be used for verification of several non-functional properties, such as cost, reliability and availability. We demonstrate how to analyse the latter.

For the purpose of availability analysis, Boolean expression is further transformed into reliability block diagram (RBD), which is the graph with following semantics: if elements are placed in series, they are statically dependent on each other. Otherwise, if they are placed in parallel, they are independent. The rules for this transformation are:

- The blocks of RBD model are all terms appearing in the minimized Boolean expression.
- If two terms are connected with the operator $\wedge$, RBD blocks are generated for both terms and placed into serial configuration.
- If two terms are connected with the operator $\vee$, RBD blocks are generated for both terms and places into parallel configuration.

After all block elements have been generated, each model element is parameterized with $MTTF$ and $MTTR$ parameters from the NFP annotation model.

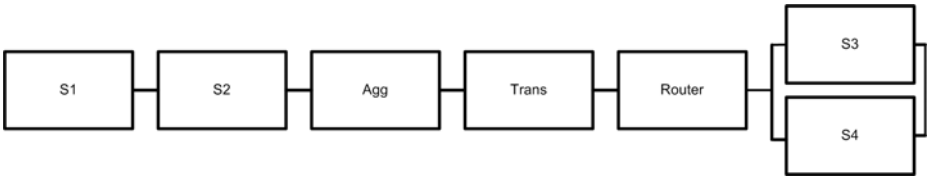

Fig. 8. Reliability block diagram model

This enables calculation of availability A of each model element (Table 1). The RBD equivalent of the connector model from Figure 7 is given in Figure 8.

Availability of each model element is calculated as:

$$A = \frac{MTTF}{MTTF + MTTR} \tag{2}$$

Availability of series (A_S) and parallel (A_P) configuration of N elements (A_i) is calculated as (we do not address further issues regarding availability models as it is out of scope of this paper):

$$A_S = \prod_{i=1}^{N} A_i \, , A_P = 1 - \prod_{i=1}^{N} (1 - A_i) \tag{3}$$

Exponential distribution is further assumed. Note once again that MTTR and MTTF parameters come from the non-functional annotation model, which has to be created manually as shown in Figure 6 or generated automatically based on historical and statistical data for systems/connector components. There are also numerous industrial studies listing these parameters for various system/software classes [14,15].

Instead of solving the model manually, for the purpose of availability analysis we generate RBD models for the external solver (Sharpe [16]), and obtain evaluation results for the connector availability shown in the row *Connector* of Table 1. This is the provided availability of the entire connector component which can now be compared with the required availability, if one was specified in the requirements specification phase.

Based on the analysis we can assert that the connector will run on the average for more than 2000 hours before it fails. Furthermore, connector repair will take more than 3 hours on the average. This also means that the connector based on the properties of its constituent parts and integrated systems will experience additional 12 hours of unplanned downtime per year. This is critical information for the integration scenario, because although the systems S_1, S_2, S_3 and S_4 may be functionally compatible, the requirement of the integration scenario may be that no more than 10 hours of unplanned downtime is allowed. In this case non-functional incompatibility would be detected.

Another possibility is to directly compare provided availability of source systems (S_1, S_2) and connector parts and determine if required availability of target systems (S_3, S_4) is satisfied. This is easy to do using the generated model

Table 1. Availability analysis results

	MTTF	MTTR	A
S_1	8760	0.6	0,99993151
S_2	8760	9	0,99897365
Agg	14400	2	0,99986113
$Trans$	18000	2	0,99988890
$Router$	9000	2	0,99977782
S_3	83220	8	0,99990387
S_4	83220	15	0,99981978
$Connector$	**2153.2**	**3.38**	**0.99854460**

by removing S_3 and S_4 from the model, reevaluating it and comparing the obtained value with required availability for S_3 and S_4. Steady-state availability of the system thus obtained is 0.99854462. Note that it does not differ greatly from the overall provided connector availability, because S_3 and S_4 are highly-available redundant data stores. Nevertheless, assume that required availability of S_3 and S_4 is 0.9980 and 0.9990 respectively. While availability requirements of S_3 are satisfied, it is obvious that S_4 will not guarantee correct operations as the rest of the system is not stable enough. This kind of incompatibility is very difficult, if not impossible, to determine using current system integration methods. As already noted, this approach may be used for equivalent calculation/matching of other non-functional properties, such as costs or accuracy. Properties, such as WCET or throughput, can be analyzed using a dynamic analysis model (i. e., Petri nets).

5 Related Work

Non-functional aspects in software integration relate to several research areas, i.e., requirements engineering, software engineering, service-oriented architecture, component-based development and model-driven development, in which non-functional properties/requirements (NFP/NFR), quality of service (QoS) attributes, and constraints are described, computed and analyzed. Glinz [17] surveys definitions of theses terms in relation to requirements engineering. We addressed non-functional properties with respect to software integration. Similar efforts in other contexts have been made: [18] discusses definitions and taxonomies of dependability and security in the context of faults, errors and failures. A Quality of Service catalog with regard to component-based software development is given in [19]. Several NFP ontologies exist, for example [20] compares existing solutions, such as OWL-QoS and QoSOnt.

Software integration research is today closely coupled with Web services. Several proposals have been made for NFP extensions of Web service descriptions, either XML-based [21,22] or as UML profiles to support graphical modeling of non-functional aspects in SOA [23]. Similar extensions are used for Web service discovery, matchmaking (e. g., DIANE framework [24]) and selection (a survey can be found in [25]). The tool Ark [23] also supports application code generation with respect to non-functional properties.

Description and analysis of non-functional properties plays an important role in software development based on MDA. [26] and [27] contribute frameworks that propose additional analysis models on CIM, PIM and PSM levels to validate non-functional properties on different levels of abstraction.

Apart from Web service extensions, metamodels for non-functional property specification exist. A survey of existing solutions for general non-functional property specification can be found in [28]. The OMG QFTP specification [9] provides an UML profile for QoS modeling that enables association of requirements and properties to UML model elements. Currently QFTP only offers abstract syntax for QoS models and is intended to be used in real-time domain (as well as the related specification MARTE). In the context of development of distributed components [29] defines the Component Quality Modelling Language (CQML), relates its usage to several different development scenarios and provides computational support for QoS managing and monitoring. Both CQML and QFTP have very limited support for modeling units of measurement (informative only and string-based respectively).

Similar to our approach, [30] provides a framework for NFP in component-based environment. It extends interface (IDL) and architecture (ADL) description languages to describe NFP at design time and then compare provided and required NFP at runtime. Only three properties are supported (performance, reliability, availability). [31] analyzes NFR of provided and required component interfaces. Additionally they provide tactics to resolve mismatches. The framework is process-oriented (manual steps done by a developer) in which capabilities of components are expressed and compared as goals.

Currently, we are not aware of an integrated solution that supports all the features we presented. Our solution is not restricted to Web service technology, UML profiles (assuming system models in UML) or textual IDLs and supports automatic analysis of non-functional mismatches as well as overall NFP computation. We provide ready-to-use property model for assignments based on the NFP metamodel that can be easily used and extended in model-based environments. It also overcomes weak unit modeling support of other solutions.

6 Conclusion

We presented part of the BIZYCLE integration framework which addresses system integration with respect to non-functional properties using model-driven methods and tools. The main advantage of the proposed solution is the ability to dynamically create new NFP and include them into analysis. Based on the NFP metamodel, it is possible to define relevant information necessary for further formal analysis – we presented an example of transforming integration models into reliability block diagrams to analyze availability. The project, in its present form, has been included into our Model-based Integration Framework (MBIF) tool and is undergoing industrial evaluation within our project consortium.

One challenge that remains however, is to perform analysis of multiple (possibly conflicting) NFP for a single scenario. The idea is to be able to derive optimal system configuration with respect to more than one NFP (e.g., cost, execution time and availability) for a given integration setup. Additional problems related to optimization with multiple criteria and goals arise that we plan to address using the proposed framework in the future work.

References

1. Rahm, E., Bernstein, P.: A survey of approaches to automatic schema matching. VLDB Journal 10(4), 334–350 (2001)
2. InterSystems: Ensemble data transformation language (2009), http://docs.intersystems.com/documentation/ensemble/20091/pdfs/EDTL.pdf
3. E2E Technologies Ltd: E2E Bridge, http://www.e2ebridge.com/en/e2e.asp
4. Kutsche, R., Milanovic, N.: (Meta-)Models, Tools and Infrastructures for Business Application Integration. In: UNISCON 2008. Springer, Heidelberg (2008)
5. Agt, H., Bauhoff, G., Cartsburg, M., Kumpe, D., Kutsche, R., Milanovic, N.: Metamodeling Foundation for Software and Data Integration. In: Proc. ISTA (2009)
6. Agt, H., Bauhoff, G., Kutsche, R.D., Milanovic, N., Widiker, J.: Semantic Annotation and Conflict Analysis for Information System Integration. In: Proceedings of the MDTPI at ECMFA 2010 (2010)
7. Kutsche, R., Milanovic, N., Bauhoff, G., Baum, T., Cartsburg, M., Kumpe, D., Widiker, J.: BIZYCLE: Model-based Interoperability Platform for Software and Data Integration. In: Proceedings of the MDTPI at ECMDA (2008)
8. Hohpe, G., Woolf, B.: Enterprise Integration Patterns. Addison-Wesley, Reading (2003)
9. OMG: Uml profile for modeling quality of service and fault tolerance characteristics and mechanisms (2008), http://www.omg.org/spec/QFTP/
10. W3C Recommendation: Mathematical Markup Language (MathML) Version 2.0, 2nd edn. (2003), http://www.w3.org/TR/MathML2
11. International Bureau of Weights and Measures: The International System of Units (SI), 8th edn. (2006)
12. Shtelma, M., Cartsburg, M., Milanovic, N.: Executable domain specific language for message-based system integration. In: Schürr, A., Selic, B. (eds.) MODELS 2009. LNCS, vol. 5795, pp. 622–626. Springer, Heidelberg (2009)
13. Thorelli, L.: An algorithm for computing all paths in a graph. Scientific Notes, BIT 6 (1966)
14. Yankee Group: Global Server Operating System Reliability Survey 2007-2008 (2008)
15. Scheer, G.W., Dolezilek, D.J.: Comparing the Reliability of Ethernet Network Topologies in Substation control and Monitoring Networks. Schweitzer Engineering Laboratories TR 6103 (2004)
16. Sahner, R., Trivedi, K., Puliafito, A.: Performance and Reliability Analysis of Computer Systems. Kluwer Academic Publishers, Dordrecht (2002)
17. Glinz, M.: On non-functional requirements. In: 15th IEEE International Requirements Engineering Conference, RE 2007 (2007)
18. Laprie, J.C., Randell, B.: Basic concepts and taxonomy of dependable and secure computing. IEEE Trans. Dependable Secur. Comput. 1(1), 11–33 (2004)

19. Brahnmath, G., Raje, R., Olson, A., Bryant, B., Auguston, M., Burt, C.: A quality of service catalog for software components. In: Proceedings of the Southeastern Software Engineering Conference, Alabama, pp. 513–520 (2002)
20. Dobson, G., Sanchez-Macian, A.: Towards unified qos/sla ontologies. In: SCW 2006: Proceedings of the IEEE Services Computing Workshops, pp. 169–174. IEEE Computer Society, Washington, DC, USA (2006)
21. Toma, I., Foxvog, D., Paoli, F.D., Comerio, M., Palmonari, M., Maurino, A.: WSMO Deliverable: Non-Functional Properties in Web Services. Technical report, STI International (2008)
22. Paoli, F.D., Palmonari, M., Comerio, M., Maurino, A.: A meta-model for non-functional property descriptions of web services. In: ICWS 2008: Proceedings of the 2008 IEEE International Conference on Web Services, pp. 393–400. IEEE Computer Society, Washington, DC, USA (2008)
23. Wada, H., Suzuki, J., Oba, K.: A model-driven development framework for non-functional aspects in service oriented architecture. Int. J. Web Service Res. 5(4), 1–31 (2008)
24. Hamdy, M., König-Ries, B., Küster, U.: Non-functional parameters as first class citizens in service description and matchmaking - an integrated approach. In: Di Nitto, E., Ripeanu, M. (eds.) ICSOC 2007. LNCS, vol. 4907, pp. 93–104. Springer, Heidelberg (2009)
25. Yu, H., Reiff-Marganiec, S.: Non-functional property based service selection: A survey and classification of approaches. In: NFPSLA-SOC 2008,CEUR-WS, Ireland, Dublin (2008)
26. Jonkers, H., Iacob, M.E., Lankhorst, M.M., Strating, P.: Integration and analysis of functional and non-functional aspects in model-driven e-service development. In: EDOC 2005. IEEE Computer Society, Washington, DC, USA (2005)
27. Cortellessa, V., Di Marco, A., Inverardi, P.: Integrating performance and reliability analysis in a non-functional MDA framework. In: Dwyer, M.B., Lopes, A. (eds.) FASE 2007. LNCS, vol. 4422, pp. 57–71. Springer, Heidelberg (2007)
28. Colin, S., Maskoor, A., Lanoix, A., Souquiéres, J.: A synthesis of existing approaches to specify non-functional properties. Technical report, Universit Nancy II (2008)
29. Aagedal, J.O.: Quality of Service Support in Development of Distributed Systems. PhD thesis, University of Oslo (2001)
30. Saleh, A., Justo, G.R.R., Winter, S.: Non-functional oriented dynamic integration of distributed components. Electr. Notes Theor. Comput. Sci. 68(3) (2003)
31. Supakkul, S., Oladimeji, E., Chung, L.: Toward component non-functional interoperability analysis: A uml-based and goal-oriented approach. In: IEEE International Conference on Information Reuse and Integration (2006)