Monitoring and Adaptation of Service-oriented Systems with Goal and Variability Models

Benedikt Burgstaller[2], Deepak Dhungana[3], Xavier Franch[1], Paul Grünbacher[2,3], Lidia López[1], Jordi Marco[1], Marc Oriol[1], Rafael Stockhammer[2]

Universitat Politècnica de Catalunya (UPC)[1] Barcelona, Spain

Johannes Kepler Universität[2] Institute for Systems Engineering and Automation Linz, Austria

Johannes Kepler Universität[3] Christian Doppler Laboratory for Automated Software Engineering Linz, Austria

bb@sea.uni-linz.ac.at, dhungana@ase.jku.at, franch@lsi.upc.edu, pg@sea.uni-linz.ac.at, llopez@lsi.upc.edu, jmarco@lsi.upc.edu, marc.oriol@gmail.com, rafael.stockhammer@students.jku.at

Report de Recerca LSI-09-8-R

Març 2009

Abstract

Variability modelling and service-orientation are important approaches for achieving both flexibility and adaptability required by stakeholders of software systems. In this paper, we present the MAESoS approach that utilizes goal and variability models to support runtime monitoring and adaptation of service-oriented systems. We illustrate our approach using two scenarios and present a tool architecture that integrates a monitoring tool and an existing tool for defining and executing variability models.

1. Introduction

Software systems today are characterized by the heterogeneity of platforms and networks they operate on; the diversity of stakeholders with changing and ephemeral needs; and the dynamicity of their operating environment. Stakeholders demand flexible systems that can be adapted rapidly and reliably after requirements changes, performance changes, technological updates, new (types of) stakeholders, new regulations, etc. In many environments, systems need to evolve even at runtime to deal with such changes. The service-oriented computing (SOC) paradigm offers capabilities for designing flexible and evolvable systems [1]. Services are open components that can be composed rapidly and often at low cost. Despite these benefits adapting a service-oriented system to different environments and contexts remains challenging.

Researchers and practitioners are increasingly using models to support the definition and adaptation of software systems and to guide and automate changes at runtime. For instance, stakeholder requirements can be analyzed and defined in goal models to guide later system adaptation. Also, variability models can help to determine which alternative services may replace an existing service under certain circumstances, e.g., in case of failure or low performance. Researchers have demonstrated the use of variability models in the context of service-oriented systems to support runtime evolution and dynamism in different domains [2][3][4]. Using models at runtime for adapting software relies on capabilities to evaluate the system's behaviour. E.g., monitors can be deployed to measure properties of available services. If the evaluation indicates a deviation from the desired level of performance defined in the goal model, or when better functionality or performance is available, a system can adapt its own behaviour [4] (e.g., by replacing one service with another service).

We present a model-based approach that utilizes goal and variability models to define design-time and runtime elements of service-oriented systems. Variability models are built from the analysis of goal models that capture stakeholders' needs. While goal models allow relating runtime monitoring results to stakeholders' goals, variability models define the possible and allowed system changes at the levels of stakeholder needs, architecture, and execution environment.

2. A Model-Based Approach to Monitoring and Adaptation of Service-oriented Systems

Our approach to model-based monitoring and adaptation of service-oriented systems relies on two modelling techniques:

Goal-oriented modelling promotes the use of goals for managing different aspects of the system specification process [5]. A goal is an objective the system under consideration is expected to achieve. Goals provide various benefits to engineers. Remarkably, they are considered to be more stable than requirements, which makes them especially attractive in evolvable environments. Also, they are well suited for conflict detection, analysis, and negotiation. Another benefit is their decomposability: goal models are structured as hierarchies in which high-level goals are decomposed down to the level of measurable requirements. The satisfaction or failure of requirements can then be propagated to high-level stakeholder goals to discern whether they are fulfilled or not.

Variability modelling is used to describe the common and variable features of a set of software systems in a particular domain. Variability models define dependencies and constraints regarding the combination of features when deriving a system from a product line. Well-known examples are feature models [6] or decision models [7]. Variability models are useful in different environments and with different types of technologies. External variability addresses the variability of artefacts that is visible to customers while internal variability remains hidden from customers. In other words, external variability represents variations on requirements, whilst internal variability appears at design, implementation, and runtime level.

We build goal and variability models at design time together with rules determining the adaptation space of service-oriented systems. We use these models at runtime to guide and automate their adaptation. We build goal models using well-known methods and techniques from requirements engineering. In particular, we use *i** [8], a goal- and agent-oriented framework, which allows modelling different types of actors. We analyze and refine *i** models regarding the variability to identify candidate features for the variability model. Both models drive system monitoring and adaptation. Quality goals in goal models can be matched to high-level characteristics of a quality model for services [9] to select those metrics that allow discerning if requirements are satisfied or not. A monitoring system can be continuously fed with runtime information that allows computing these metrics. In such a way the monitoring system can detect possible violations of requirements by services. The variability model can then be used to automatically identify alternatives at both levels of design and runtime. These alternatives may eventually be presented to an engineer that will mediate in negotiation with stakeholders; in other cases, the adaptation of the system may be performed automatically.

3. A Model Integrating Design and Runtime Aspects

Our model-based approach covers issues ranging from stakeholder goals to low-level aspects of system composition and monitoring. It addresses design-time and runtime, including monitoring and adaptation aspects. Our MAESoS[1] framework is shown in Figure 1. It defines three layers that contain key elements of our meta-model.

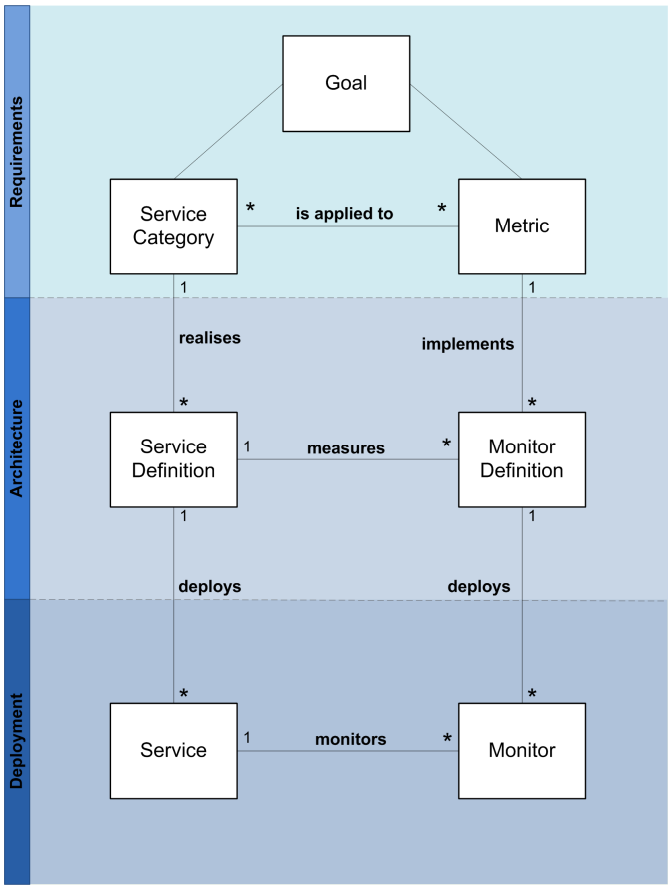

Figure 1: The MAESoS 3-layered framework.

At the requirements layer, stakeholders' needs are represented as *Goals* that involve actors. Actors at this layer represent both system stakeholders and groups of related functionalities, named *Service Categories*. Goals can also represent system quality characteristics like efficiency, accuracy, etc. which are used for generating conditions over service categories: first, quality characteristics are matched with high-level quality factors of the quality model for services; then, the desired *Metrics* that are part of the decomposition of these factors are chosen and *applied to* the service categories stating the condition to be measured.

[1] Monitoring and Adaptation Environment for Service-oriented Systems.

At the architecture layer, the goals are mapped to architectural concepts and real world elements. There are two parts: (i) *Service Definitions* describe those software services whose functionality *realise* service categories (an example is a WSDL interface of a web service). MAESoS supports atomic services as defined by [10], the composition of service is handled by our tool architecture as will be shown later. Although there might be variability within services, we focus on the granularity level of whole services as usually the implementation details of services are not available to external developers. (ii) *Monitor Definitions implement* the different metrics that are applied to services in a way that conditions over that metrics may be checked. Since the set of metrics to be monitored is not large and does not vary much, monitor definitions may be kept in a repository ready to be reused.

Finally, the deployment layer includes *Services* and *Monitors* that *deploy* the corresponding definitions obtained in the previous layer. Each service may be *monitored* by several monitors.

Goals, actors, services and their relationships can be represented using *i** constructs. The quality model is used to guide the identification of metrics. Variability appears mainly in the connection between layers. For instance, several alternative service definitions may be available for a given service category. This is especially true when service categories represent types of services that are available in the service marketplace. Also, several services may deploy a given service definition, typically this represents implementation of a service definition in different nodes of the system. The rules that encode which service or service definition to choose are part of the variability model itself.

We illustrate the framework with an example, a distributed system provided by Travel Services Inc. (TSI), a company offering services to travellers for searching and booking trips online. While some of these services are developed by TSI, most of them are provided by third party service providers. The complexity and granularity of services ranges from very simple (e.g., currency conversion) to highly complicated (e.g., travel service). Various Travel Agencies (TA) from Austria and Spain[2] contract TSI's software solution to offer a customized online travel platform to their customers so there are some characteristics of a software product line. The framework presented in Figure 1 adapted to part of the example is shown in Figure 2.

[2] To simplify the example, we assume that both TA and deployed services are restricted to Austria and Spain.

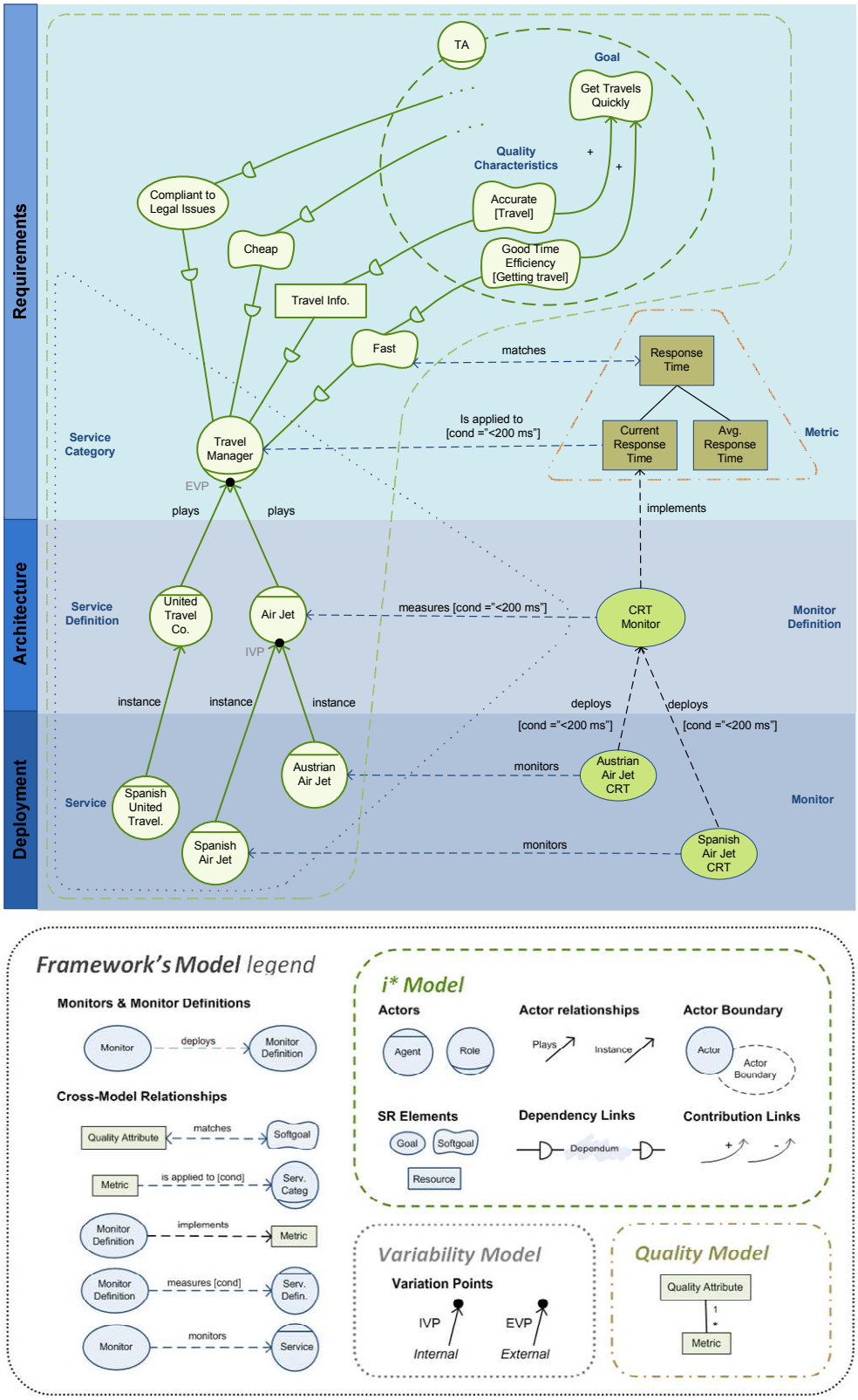

Figure 2: Partial model for the TSI example.

At the requirements layer we show two *i** actors, the main "TA" actor and "Travel Manager", a service category that calculates the travels that may be offered to customers in response to their demands. When we explore the needs for "TA", we identify the goal "Get Travels Quickly". The goal is further analysed and two quality characteristics are identified as contributing positively, "Good Time Efficiency[Getting Travel]" and "Accuracy[Travel]". Both require the use of "Travel Manager" to obtain the "Travel Info" in a "Fast" way, so dependencies are recorded in the model. Other dependencies over "Travel Manager" from other goals may also appear, like "Cheap" and "Compliant to Legal Issues".

At the architecture layer, a market exploration shows that the "Travel Manager" service category can be covered by several existing services. Among them, "United Travel Co." and "Air Jet" satisfy the dependencies generated in the requirements level over "Travel Manager". The first service is deployed only in Spain, whilst the second is deployed in Austria and in Spain. All these actors and relationships are modeled using the appropriate *i** constructs (in particular, the "plays" and "instance" relationships) that are the counterparts in *i** of the "realises" and "deploys" concepts that were defined in the general framework (see Figure 1, *i** model part).

The *i** model is then explored for searching possible variation points following some rules given in [11]. The "plays" relationship induces a variation point that is external, since the final decision of what service definition to use is taken considering dependencies over "Travel Manager". Therefore, the variability rule can be written as a decision:

```
(R1) external decision Travel Manager
          alternatives United Travel Co., Air Jet
          criteria Cheap, Fast, Compliant to Legal Issues, Travel Info.
```

From now we focus on "Fast". The exploration of the service quality model reveals that "Response Time" is the high-level quality factor in the quality model closest to "Fast", and the TSI engineer decides that "Current Response Time" (CRT) is the most appropriate metric. Finally, the concept of "Fast" is refined by the condition "CRT < 200ms." over "Travel Manager". Eventually, this refinement may be used as an additional assessment for the "Fast" criterion in the rule R1 above.

Let's assume that the TSI engineer chose "Air Jet" when applying R1. "Air Jet" is identified as a new variation point coming from the one-to-many *i** "instance" relationship. In this case the variation point is internal, and depends on how the services satisfy the constraint on CRT: rules are incorporated into the variability model ensuring that the selected service fulfills the constraint. Whenever possible, the service is selected according to the TA's country (e.g., "Austrian Air Jet" for an Austrian TA). Services are constantly monitored checking the stated condition. Thus, one monitor monitors CRT for each deployed service; these monitors are

deployed from the CRT monitor type definition. Also, a rule is needed to report failure when both services fail. Some of the rules (expressed in pseudo-code) are shown below[3]:

```
(R2) internal decision Air Jet
     depends on TA::location
     alternatives Austrian Air Jet, Spanish Air Jet
     defined as
            (R2.1) TA::location == Austrian & Austrian Air Jet::CRT ≤ 200ms
                                  → Air Jet = Austrian Air Jet
            (R2.2) TA::location == Spanish & Spanish Air Jet::CRT ≤ 200ms
                                  → Air Jet = Spanish Air Jet
            (R2.3) TA::location == Austrian & Austrian Air Jet::CRT > 200ms
                         and Spanish Air Jet::CRT ≤ 200ms
                                  → Air Jet = Spanish Air Jet
            (R2.4) TA::location == Spanish & Spanish Air Jet::CRT > 200ms
                         and Austrian Air Jet::CRT ≤ 200ms
                                  → Air Jet = Austrian Air Jet
            (R2.5) Austrian Air Jet::CRT > 200ms & Spanish Air Jet::CRT > 200ms
                                  → failure(Air Jet)
```

4. MAESoS Environment for Using Models at Runtime

We have been developing a set of tools that support monitoring and runtime adaptation of service-oriented systems. The MAESoS tool architecture depicted in Figure 3 is divided into three levels (note that although highly related, these three levels do not directly correspond to the three layers in Figures 1 and 2):

The *model level* provides capabilities to define models at design time and to execute them at runtime. *i** domain models are encoded using *i*StarML, an emerging XML-based standard allowing model exchange among existing tools for *i** [12]. The models may be created using any *i** editor that generates this format, we are currently using J-PR*i*M [13]. For service monitoring and service adaptation we extended the DOPLER product line tool suite [14] with additional software components, e.g. a component to analyse the *i** model at design time and present candidate variation points to the engineer as suggested in [11]. Adaptation of a service-oriented system can often not be fully automated as user feedback will be required in many cases. The DOPLER tools thus provide a user interface to either (i) manually adapt a service-oriented system guided by model's variability rules or to (ii) confirm changes automatically suggested by the reasoning capabilities of our tool architecture. Similar to work reported in [15] we are adopting the DOPLER component Configuration-Wizard for this purpose. The DOPLER tools communicate with external *i** model checkers to evaluate the consequences of failures and decisions on stakeholder goals. The engineer uses this information to take informed decisions when the system needs to adapt to changes.

[3] A parameterized version would be required in the general case; we have preferred to present this ad-hoc version for clarity.

The *adaptation level* contains the monitoring and adaptation components that are independent of the actual implementation technology. The main components on this level are the *Monitoring Controller*, the *SALMon Analyzer*, and the *Adaptor*. SALMon is a service-oriented monitor that uses *measurement instruments* (MI) to monitor services and an analyzer component to decide, based on conditions that come from rules in the variability model, whether measurement results are forwarded to the upper level components or discarded. The Monitoring Controller is mainly responsible for the communication between the variability model and SALMon. It prepares monitoring rules defined in the variability model for SALMon Analyzer to detect rule violations. Additionally, it enables and manages domain-specific metric handling by using domain-specific monitors in the DOPLER tools to calculate domain-specific metrics. On the right hand side of our architecture, the Adaptor executes changes from the variability model on the target application, after confirmation by the engineer or automated system rules.

On the level of the *technology driver*, the *SALMon Monitor* and the *Runtime Reconfigurator*, a technology-dependent part of the Adaptor, are situated. SALMon Monitor is the part of SALMon that manages the single measurement instruments and deploys them. The Runtime Reconfigurator uses technology-dependent information to adapt the Runtime Service Composition of the business application. The *Runtime Service Composition*, also residing on this level, is the central composition of services used by the business application. It keeps track of technology-dependent information of services in the *business application*.

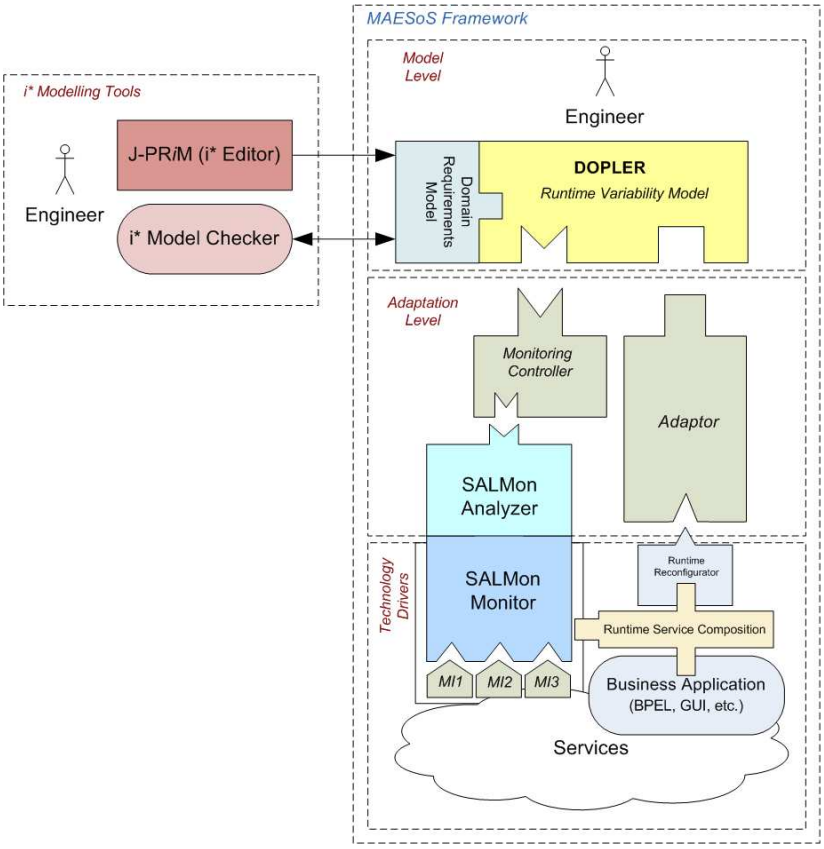

Figure 3: MAESoS Tool Architecture

MAESoS aims at supporting two types of changes: *Runtime changes* are procured by the system based on performance variations or service updates while *requirements changes* are actively caused or driven by stakeholders. The following scenarios highlight a runtime and a requirements change and illustrate subsequent system adaptation, which can be inferred by referring to the goal and variability models at runtime.

4.1 Runtime (system-driven) change

In Section 3, TSI has decided to use the third-party flight services provided by "Air Jet" situated in Spain and Austria. Different options available for fulfilling this request can be found by evaluating the internal decision Air Jet (Decision Rule R2). The model specifies that each Spanish TA normally redirects customer requests to the "Spanish Air Jet" server and Austrian TAs primarily use the "Austrian Air Jet" server. Additionally, redirection between the Austrian and Spanish servers is possible, if required. The travel services are monitored by "Spanish Air Jet CRT" and "Austrian Air Jet CRT" monitors as described in Section 3, represented as Measure Instruments in MAESoS. We now consider the following scenario (see Figure 4 for illustrating the impact on MAESoS's components):

1. In a certain moment, the Austrian network experiences some problems that provoke a dramatic increase of the Austrian server Current Response Time, as reported to SALMon Monitor by the "Austrian Air Jet CRT" measure instrument. SALMon Analyzer detects a condition violation: the current response time of the "Austrian Air Jet" service is higher than the specified threshold and notifies the Monitoring Controller. The Monitoring Controller itself sets the reported values in the services and domain-specific monitors represented in the model.

2. On the other hand, "Spanish Air Jet CRT" shows that the Spanish service still fulfils the condition. This information is also propagated to the Monitoring Controller.

3. The DOPLER tool engine notices the changes above and revaluates the model. In particular, it verifies that the guard of R2.1 fails whilst the guard of R2.3 holds, meaning that a change that guarantees the quality of service is possible. Depending on the system configuration, the TSI engineer may be requested to confirm the change or else the change will be performed automatically by the MAESoS framework and reported to the TSI engineer for information only. In Figure 4 we assume the second option, i.e. the TSI engineer is informed via the user interface that the system is not satisfying the specification and that it will adapt itself to automatically redirect Austrian traffic to the "Spanish Air Jet" server.

4. The system then automatically updates its configuration and redirects Austrian traffic to the Spanish server. To do so, the Adaptor compares the configuration of model and business application where it finds that the Austrian service is no longer used in the model but the Spanish service is used instead. For technology-dependent adaptation, the Adaptor uses the Runtime Reconfigurator to adapt the Runtime Service Configuration of the business application accordingly.

5. Once the adaptation has been completed, the system may evolve in two different ways:

Subscenario 5.a) Four hours later, the SALMon components detect that the load on the Austrian server is stabilized again and that its response time satisfies the specification. Therefore, the Adaptor is notified by the DOPLER tool engine to perform a switch from the "Spanish Air Jet" service to the "Austrian Air Jet" service, which it may execute automatically depending on the configuration. The system then again redirects all traffic to the Austrian server applying R2.1 in a similar way than above. The TSI engineer is automatically informed by the monitoring system that the redirection of traffic is no longer needed.

Subscenario 5.b) At some moment, also the monitor for the "Spanish Air Jet" service detects that the CRT condition is violated (this subscenario is illustrated in Figure 4).

5.b.1) Then the guard for R2.5 holds which means that the *i** "Air Jet" agent is marked as "failure".

5.b.2) Using goal-modelling analysing techniques, the "failure" mark is propagated backwards reaching the quality characteristic "Good Time Efficiency" through the dependency "Fast".

5.b.3) The system displays to the TSI engineer that both services (Spanish and Austrian Air Jet) have a current response time greater than 200 ms. and as a consequence, the "Good Time Efficiency" of the TA

is affected. This failure cannot be handled automatically by MAESoS, and therefore human intervention is needed, e.g. to allow temporally a violation of quality of service, to apply R1 to choose a different service definition, etc.

5.b.4) Once the TSI engineer has taken a decision, she informs the Configuration Manager that implements the required action (not shown in Figure 4).

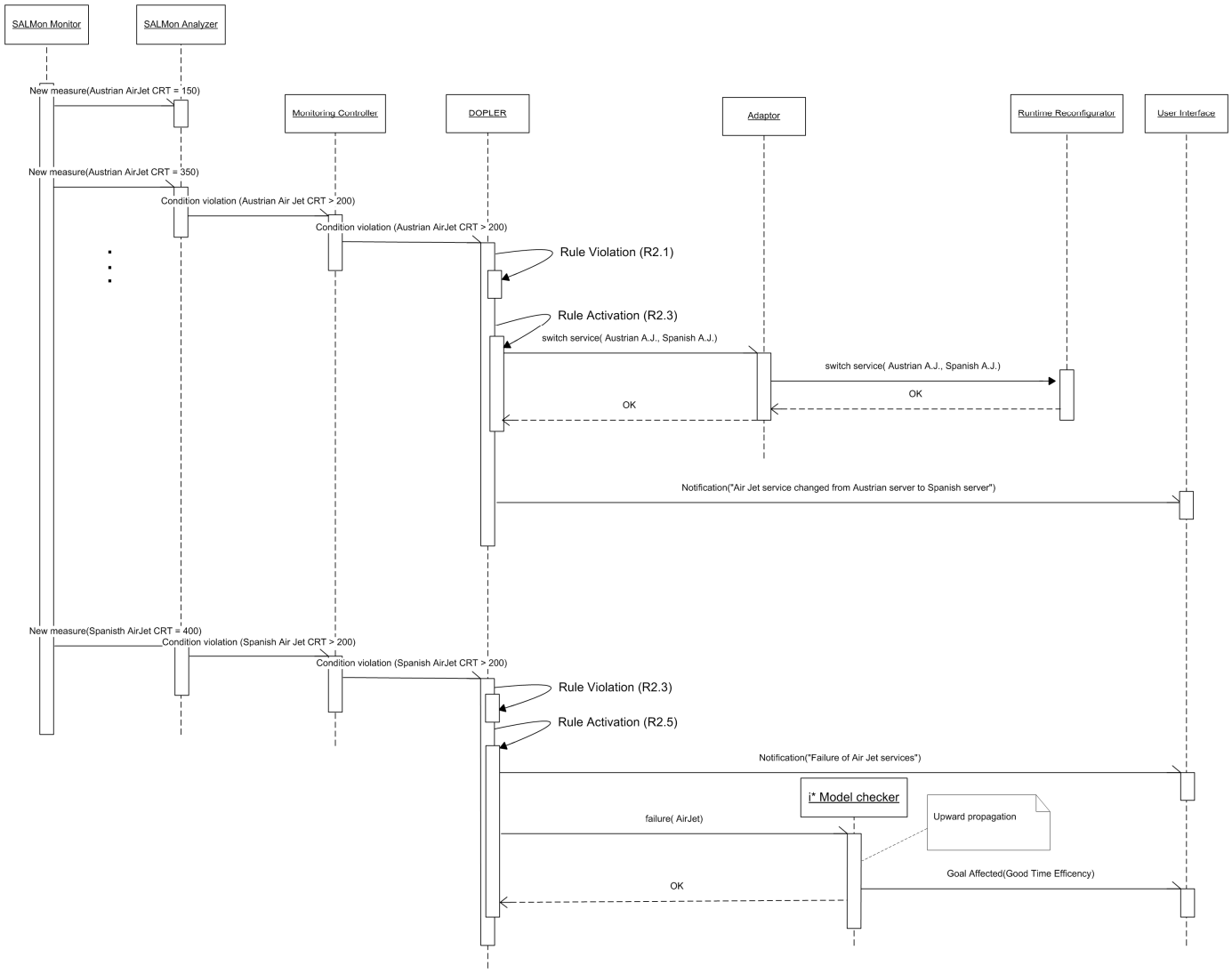

Figure 4: Sequence diagram showing the MAESoS behaviour in the runtime scenario (with variant 5.b)

4.2 Requirements (stakeholder-driven) change

TSI has won a new TA named Business Travel Agency (BTA). Due to some legal regulations, BTA raises several requirements not yet covered by "Air Jet" and therefore the dependency "Compliance to Legal Issues" from TA to Travel Manager (see Figure 2) fails.

1. The TSI engineer is informed by DOPLER that Compliance to Legal Issues is one of the criteria that apply on the external variability rule (R1). Therefore, this rule needs careful analysis.
2. DOPLER communicates to the external *i** Model Checker tool about the failure of Compliance to Legal Issues. The checker propagates these changes backwards which may provoke the failure of some other stakeholder goals (not shown in the figure).
3. The TSI engineer is notified about these failed goals and also the other criteria that influence R1.
4. After some consultations with BTA, they agree that Travel Manager may be changed to UTC despite its higher price.
5. Before considering this decision definitive, the TSI engineer analyses whether the change is interfering with the intentions of other TAs by analyzing the *i** goal model. She finds out that the other TAs are affected by changing to UTC due to higher price since all TAs are in fact instances of the same TA actor in the model.
6. TSI negotiates the change with the other customer TAs. The affected TAs and "Air Jet" are invited for discussions. "Air Jet" is not interested in covering the new legal regulations and the other TAs concur that they are not interested as well, but accept to use UTC if it will not increase their costs.
7. TSI is willing to assume the extra cost for this new service and switches to the new service. Since the other TAs are not affected, goal evaluation in their *i** models will not change.
8. Using MAESoS, the TSI engineer therefore adds the new service to the variability model in the DOPLER tools. The Monitoring Controller evaluates the newly added services, and passes the new rules to SALMon Analyzer.

5. Conclusions and future work

We have presented a model-based approach to managing and adapting service-oriented systems based on two types of models, goal-oriented models and variability models. The use of both models together links the problem space represented by goals and the solution space represented by variation points and decision rules. The main benefits are:

- The use of goal models adds a powerful reasoning mechanism especially suited for exploring the consequences of alternative adaptations and also deducing corrective actions to help during negotiation processes. What-if questions are naturally supported.

- The use of variability models allows stating neatly which are the parts of the system that may adapt to changes, and which are the conditions for evolution.
- The approach ensures full traceability from stakeholders' goals to the running system and vice versa. Therefore, it is straightforward to find out which services are affected by changes in stakeholder needs, and which goals are affected due to changes in the running system.
- The symmetric structure among the service-based system and the monitoring infrastructure makes the resulting framework conceptually clear. In addition, models inside structure the problem and solution spaces by defining service categories, quality characteristics, metrics, etc., with well-defined relationships between them.
- Both types of models are widespread in the software engineering community. In addition, unlike other proposals that combine these two types of models (e.g., [16]), we are not defining a single model but keeping both models separated and coordinated, which improves model understandability. Coordination is implemented by using the goal model to identify candidate variation points, and by including goals and failure conditions in the decision rules.
- The proposed platform, MAESoS, follows a strict separation of technology-dependent and technology-independent aspects, which makes it highly portable. In addition, the use of the iStarML exchange format does not bind the platform to concrete tool support for the goal-oriented models.

As future work we plan to deploy a scalable implementation of MAESoS, including also repositories with service categories, monitors, etc. Also we plan to develop a simulation environment that may allow the engineer to explore the effect of changes without interfering with the running system.

Acknowledgements

This work has been supported in part by the ACCIONES INTEGRADAS program HU2005-0021 supporting bilateral scientific and technological cooperation between Austria and Spain; and the Spanish projects TIN2007-64753 (ADICT) and SODA FIT-340000-2006-312 (PROFIT programme). The development of the DOPLER tools has been supported by the Christian Doppler Laboratory for Automated Software Engineering.

References

[1] M. Papazoglou *et al.*, "Service-Oriented Computing: State of the Art and Research Challenges". *IEEE Computer* 40(11): 38-45 (2007)

[2] S. Hallsteinsen *et al.*, "Using Product Line Techniques to Build Adaptive Systems". *Procs. 10th International Software Product Line Conference* (SPLC): 141–150 (2006).

[3] F. Fleurey *et al.*, "Modeling and Validating Dynamic Adaptation". *Procs. Models@runtime*: 36-46 (2008).

[4] P. Robertson, R. Laddaga, H. Shrobe, "Introduction: The First International Workshop on Self-Adaptive Software". In *Self-*

Adaptive Software, LNCS 1936, Springer: 1–10 (2001).

[5] A. van Lamsweerde, "Requirements Engineering in the Year 00: a Research Perspective". *Procs. 22nd IEEE Intl. Conference on Software Engineering* (ICSE): 5-19 (2001).

[6] J. Lee, K.C. Kang, "A Feature-Oriented Approach to Developing Dynamically Reconfigurable Products in Product Line Engineering", *Procs. 10th International Software Product Line Conference* (SPLC): 131–140 (2006).

[7] D. Dhungana, P. Grünbacher, R. Rabiser, "Domain-specific Adaptations of Product Line Variability Modeling". *IFIP WG 8.1 Working Conference on Situational Method Engineering: Fundamentals and Experiences*: 238-251 (2007).

[8] E. Yu, *Modeling Strategic Relationships for Process Reengineering*. PhD Thesis, Toronto, 1995.

[9] I. Jureta, C. Herssens, S. Faulkner, "A Comprehensive Quality Model for Service-oriented Systems". *Software Quality Journal*, 17(1): 65-98 (2009).

[10] J. Lee, D., Muthig, M. Naab, "An Approach for Developing Service Oriented Product Lines". *Procs. 12th International Software Product Line Conference* (SPLC): 275-284 (2008).

[11] R. Clotet *et al.*, "Dealing with Changes in Service-Oriented Computing through Integrated Goal and Variability Modeling". *Procs. 2nd Intl. Workshop on Variability Modelling of Software-intensive Systems* (VaMoS): 43-52 (2008).

[12] C. Cares *et al.*, "iStarML: An XML-based Model Interchange Format for *i**". *Procs. 3rd Intl'. Workshop on i**: 13-16 (2008).

[13] G. Grau, X. Franch, S. Ávila, "J-PRiM: A Java Tool for a Process Reengineering *i** Methodology". *Procs. 14th IEEE Int'l Requirements Engineering Conference* (RE): 352-353 (2006).

[14] D. Dhungana, *et al.*, "DOPLER: An Adaptable Tool Suite for Product Line Engineering". *Procs. 11th International Software Product Line Conference* (SPLC): 151-152, 2nd Vol. (2007).

[15] R. Wolfinger *et al.*, "Supporting Runtime System Adaptation through Product Line Engineering and Plug-in Techniques". *Procs. 7th IEEE International Conference on Composition-Based Software Systems* (ICCBSS): 21-30 (2008).

[16] S. Liaskos, Y. Yu, E. Yu, J. Mylopoulos, "On Goal-based Variability Acquisition and Analysis". *Procs. 14th IEEE Int'l Requirements Engineering Conference* (RE): 79-88 (2006).