Goal-driven Adaptation of Service-Based Systems from Runtime Monitoring Data

Xavier Franch[1], Paul Grünbacher[2], Marc Oriol[1], Benedikt Burgstaller[3], Deepak Dhungana[4], Lidia López[1], Jordi Marco[1], João Pimentel[1,5]

[1]GESSI-UPC
Barcelona, Spain
franch@essi.upc.edu
{moriol,llopez,jmarco}
@lsi.upc.edu

[2]SEA-JKU
Linz, Austria
paul.gruenbacher@jku.at

[3]ObjectBay Software
& Consulting GmbH
Linz, Austria
benedikt.burgstaller
@objectbay.com

[4]Siemens AG Austria
Corporate Technology,
Vienna
deepak.dhungana
@siemens.com

[5]CIn-UFPE
Recife, Brazil
jhcp@cin.ufpe.br

***Abstract*—Service-based systems need to provide flexibility to adapt both to evolving requirements from multiple, often conflicting, ephemeral and unknown stakeholders, as well as to changes in the runtime behavior of their component services. Goal-oriented models allow representing the requirements of the system whilst keeping information about alternatives. We present the MAESoS approach which uses *i** diagrams to identify quality of service requirements over services. The alternatives are extracted and kept in a variability model. A monitoring infrastructure identifies changes in runtime behavior that can propagate up to the level of stakeholder goals and trigger the required adaptations. We illustrate the approach with a scenario of use.**

Keywords-goal-oriented requirements engineering; service-oriented system; runtime adaptation; variability modelling; iStar.

I. INTRODUCTION

Service-based systems (SBS) 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, or new stakeholders. In many environments, systems need to evolve even at runtime to deal with such changes. The service-oriented computing paradigm offers capabilities for designing flexible and evolvable systems as services can be composed rapidly and often at low cost [1]. Despite these benefits, adapting an SBS 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 have been proposed to determine which alternative services may replace an existing service under certain circumstances, e.g., in case of failure or low performance, to support runtime evolution and dynamism in different domains [2]. 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 becomes available, a system can adapt its own behaviour (e.g., by replacing one service with another service).

We present an approach that puts together goal models, quality models and variability models to define design-time and runtime elements of SBS:

- *Goal-oriented modeling* promotes the use of goals for managing different aspects of the system specification process [3]. The benefits of goals such as stability; support for conflict detection, analysis, and negotiation; decomposability; and the availability of techniques are widely known. We choose the *i** framework [4] for our approach.
- *Quality modeling* is about defining a hierarchy of quality attributes (also called quality factors). Quality models have been used mainly for quality assessments and customizations are available for several contexts. For instance, several proposals exist for service-orientation. Among them, we have chosen the S-Cube quality model [5], proposed in the context of a European Network of Excellence that involves most of the major European research groups on SBS.
- *Variability modeling* is used to describe the common and variable features of a set of software systems in a particular domain. External variability addresses the variability of artifacts that is visible to customers (i.e., variations on requirements) while internal variability remains hidden from customers. From the wide range of proposals available, we have chosen to adopt decision models [6][7] to define variability in a set of rules. Variability models are essential for SBS as variability plays an important role at all levels of requirements and the technical components.

II. A MODEL-BASED APPROACH TO MONITORING AND ADAPTATION OF SERVICE-BASED SYSTEMS

Our model-based approach MAESoS (**M**onitoring and **A**daptation **E**nvironment for **S**ervice-**o**riented **S**ystems) covers issues ranging from stakeholder goals to low-level aspects of system composition and monitoring. It addresses design-time and runtime aspects and covers monitoring as well as adaptation.

A. MAESoS at Design-time

MAESoS defines three layers of requirements, architecture, and deployment that contain key elements of our metamodel (see Figure 1)., Three types of models are used in these layers: *i** models [4] represent stakeholder goals as well as SBS architectural actors; quality models for services [5] define the measures used to assess the measurable goals; and variability models [6] define the points and rules of adaptation.

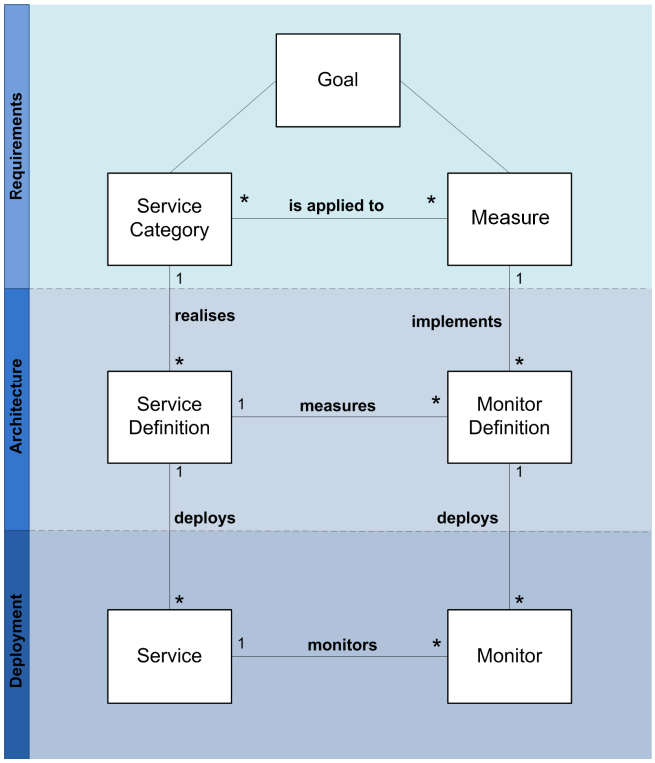

Figure 1. The MAESoS 3-layered framework: metamodel.

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.

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). Although there might be variability within services, we focus on the granularity level of whole services because the implementation details of services are rarely available to external developers.

(ii) *Monitor Definitions implement* the different measures that are applied to services in a way that conditions over that measures may be checked. Since the set of measures 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 measures. Variability appears often in the connection between two 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 the implementation of a service definition in different nodes of the system. The rules defining which service or service definition to choose are defined in the variability model.

B. MAESoS at Runtime

A monitoring system puts together all monitors in a single infrastructure which continuously collects runtime information from the SBS. This allows computing measures to detect possible violations of requirements by services. The variability model is then used to automatically identify alternatives that can be presented to an engineer to mediate negotiation with stakeholders; in other cases, the adaptation of the system may be performed automatically.

We illustrate the framework with a fictitious distributed system provided by Travel Services Inc. (TSI), a company offering services to travellers for booking trips online. Most of these services are provided by third party service providers. Various Travel Agencies (TA) from Austria and Spain contract TSI's software solution to offer a customized online travel platform to their customers. The metamodel presented in Figure 1 instantiated to part of the example is shown in Figure 2.

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 "Accurate[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". Note how the *i** model allows expressing different types of constraints, e.g. how an actor may depend on

a type of service, or how a quality characteristic influences a high-level goal.

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, while 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 are the *i** counterparts of the "realises" and "deploys" concepts defined in the general framework (see Figure 1).

Candidate variation points are searched in the *i** model according to some rules defined elsewhere [8]. The "plays" link induces a variation point that is external ("EVP" in Figure 2) since the decision of the service definition to use is made considering dependencies over "Travel Manager".

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

Let's now 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". The TSI engineer identifies "Current Response Time" (CRT) as the most appropriate measure. Finally, the concept of "Fast" is refined into 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 ("IVP"), 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 location (e.g., "Austrian Air Jet" for an Austrian TA). Services are constantly monitored to check the stated condition. Thus, one monitor observes 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 are shown below in pseudo code[1]:

```
(R2) internal decision Air Jet
  // location of service currently selected
       depends on TA::loc
       alternatives Austrian Air Jet (AAJ),
                    Spanish Air Jet (SAJ)
       defined as
```

[1] A parameterized version would be preferable in the general case; we have opted to present this version for simplicity.

```
    (R2.1) TA::loc == Austrian &
           AAJ::CRT ≤ 200ms
           → Air Jet = Austrian Air Jet
    (R2.2) TA::loc == Spanish &
           SAJ::CRT ≤ 200ms
           → Air Jet = Spanish Air Jet
    (R2.3) TA::loc == Austrian &
           AAJ::CRT > 200ms & SAJ::CRT ≤ 200ms
           → Air Jet = Spanish Air Jet
    (R2.4) TA::loc == Spanish &
           SAJ::CRT > 200ms & AAJ::CRT ≤ 200ms
           → Air Jet = AAJ
    (R2.5) AAJ::CRT > 200ms & SAJ::CRT > 200ms
           → failure(Air Jet)
```

Lastly, a set of policy rules is used to assign priorities for the possible failures and to define conditions on which they can be ignored. Thus, the most critical failures may be handled first. In our scenario, we consider it as a problem if there are more than three consecutive detections of this failure. If it happens less than three times it is considered as a temporary problem that can be ignored. For homogeneity purposes, the policy rules are encoded similar as the adaptation rules:

```
(R3) failure decision Air Jet
       defined as
    (R3.1) isAllowedToFailAtMost(3)
```

III. System Adaptation

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 XML-based interchange format allowing model exchange among existing tools for *i** [9]. The models may be created using any *i** editor that generates this format, we are currently using H*i*ME [10]. For service monitoring and service adaptation we extended the DOPLER product line tool suite [7][11] with additional software components for analysing the *i** model at design time and presenting candidate variation points to the engineer. 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 adapt a service-oriented system manually guided by a model's variability rules; or to confirm changes automatically suggested by the underlying rule-based reasoning capabilities. Similar to work reported in [11] we use the DOPLER ConfigurationWizard 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.

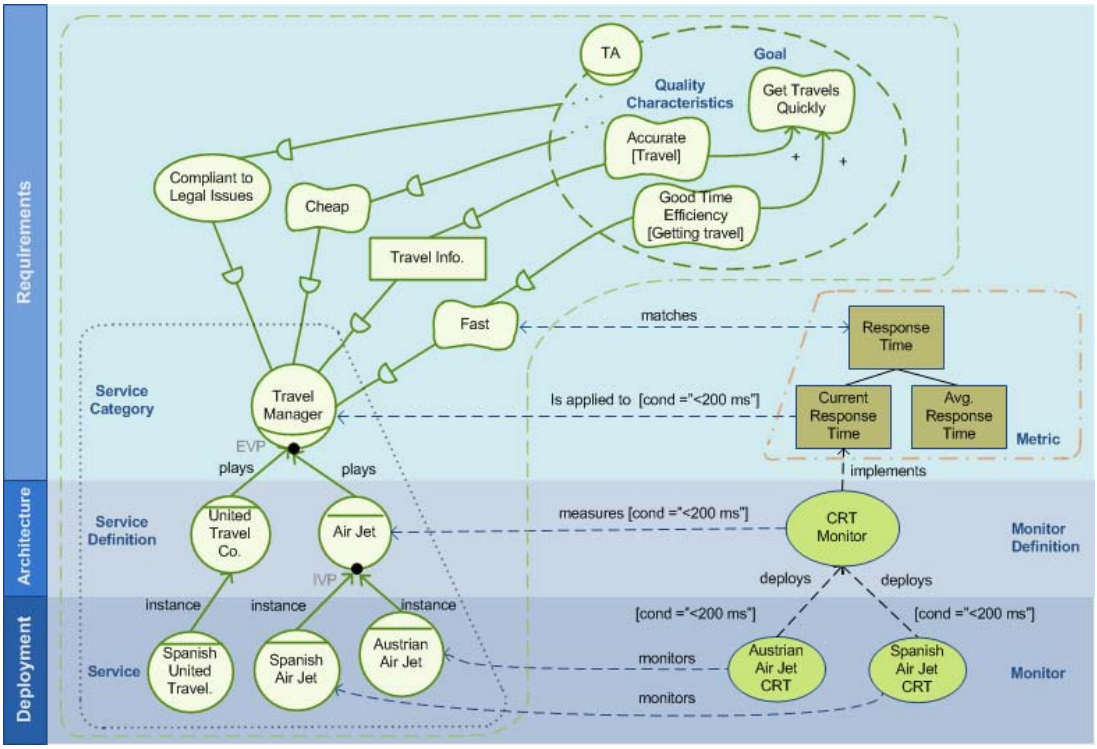

Figure 2. Partial Model for the TSI Example.

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*, the *FAST Analyzer* and the *Adaptor*. SALMon [12] 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 FAST Analyzer [13] assesses the policy rules to determine the priority of a failure and whether it can be ignored. The Monitoring Controller is mainly responsible for the communication between SALMon and the other components. It prepares monitoring rules defined in the variability model for the SALMon Analyzer to detect rule violations and failure conditions. Additionally, it enables and manages the handling of domain-specific measures by using domain-specific monitors in the DOPLER tools to calculate domain-specific measures. The Adaptor executes changes from the variability model on the target application, automatically or after confirmation by the engineer as shown in the right side of Figure 3.

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

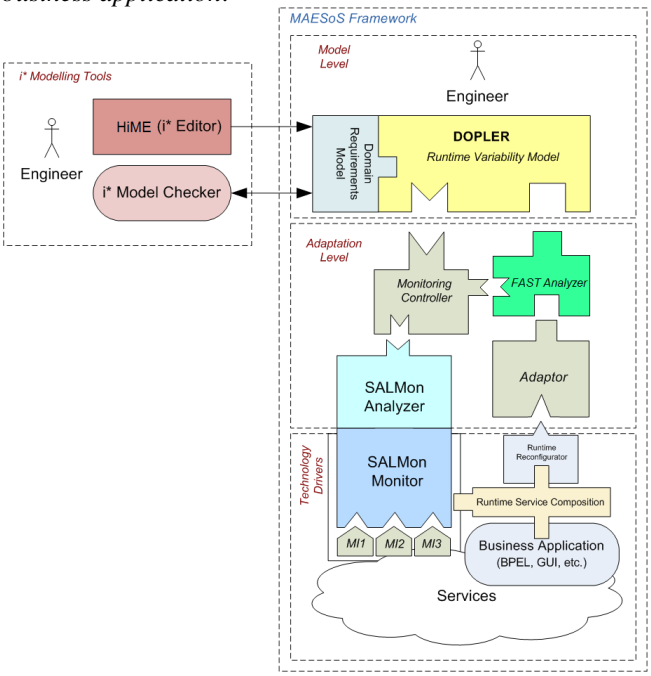

Figure 3. MAESoS Framework: System Architecture.

MAESoS aims at supporting two types of adaptations: *Runtime adaptation* is procured by the system based on performance variations or service updates while *requirements*

adaptations are actively triggered by stakeholders. For space reasons, we will focus just on the first of them.

In Section 3 we explained that TSI 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 for 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 2 and are represented as Measure Instruments in MAESoS. We now consider the following scenario:

1. The Austrian network experiences problems that lead to a dramatic increase Current Response Time of the Austrian server, as reported to SALMon Monitor by the "Austrian Air Jet CRT" measure instrument. SALMon Analyzer detects a condition violation as the current response time of the "Austrian Air Jet" service exceeds the specified threshold and thus notifies the Monitoring Controller which 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 guaranteeing the quality of service is possible. Depending on the system configuration, the TSI engineer needs to either confirm the change which can otherwise also be performed automatically by the MAESoS framework. Let's for now assume the second option, i.e., that 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 automatically updates its configuration and redirects Austrian traffic to the Spanish server. To do so, the Adaptor compares the configurations of the model and the business application and detects that the Spanish service is used instead of the Austrian service. 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 ways:

Subscenario 5.a) Several hours later, the SALMon components detect that the load on the Austrian server is back to normal again as the 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 may be executed automatically depending on the configuration. The system then again redirects all traffic to the Austrian server by applying R2.1. The TSI engineer is automatically informed by the monitoring system that the redirection of traffic is no longer needed.

Subscenario 5.b) The monitor for the "Spanish Air Jet" service detects that its CRT condition is also violated.

5.b.1) In this case the guard for R2.5 holds and the *i** "Air Jet" agent is marked as "failure". If this failure is detected the first time it can be ignored (R3). However, in case of the fourth consecutive failure it is considered as persisting and therefore needs be handled.

5.b.2) Using goal-modelling analysis techniques [14], the "failure" mark is propagated through the model reaching the quality characteristic "Good Time Efficiency" through the dependency "Fast".

5.b.3) The system displays to the TSI engineer that both services have a current response time higher than 200 ms affecting the "Good Time Efficiency" of the TA. This failure cannot be handled automatically by MAESoS and requires human intervention to either apply R1 or to choose a different service definition.

5.b.4) Once the TSI engineer has made a decision, she notifies the Configuration Manager to implement the required action.

IV. CONCLUSIONS

The joint use of goal-oriented models, quality models and variability models links the problem space represented by goals and the solution space represented by quality factors, variation points and decision rules. Synchronizing these three types of models is the baseline of our approach, and the paper has illustrated some of the challenges to overcome. The approach ensures traceability from stakeholders' goals to the running system and vice versa. This allows identifying services affected by changes in stakeholder needs, and goals affected by changes in the running system in a straightforward way.

The use of goal models allows reasoning about the consequences of alternative adaptations and eases deducing corrective actions to inform requirements negotiation. For instance, what-if questions are naturally supported. Variability models allow defining elegantly which parts of the system may adapt to changes and which conditions are relevant for evolution.

The symmetric structure among the SBS and the monitoring infrastructure makes the resulting framework conceptually clear. In addition, the involved models structure the problem and solution spaces by defining service categories, quality characteristics, measures, etc. with well-defined relationships between them.

The three layers of requirements, architecture, and deployment clearly separate important concerns of the system, while ensuring consistency by using the same type of models which are used widely in the software engineering community. Unlike other proposals that combine goal and variability models (e.g., [15]) we do not define a single model

but intentionally keep them separated and coordinated to improve model understandability. Consistency is achieved by using the goal model to identify candidate variation points, and by including goals and failure conditions in the decision rules. Concerning variability, it is worth to remark that it appears at several levels, ranging from higher-level goals to service implementations.

The proposed MAESoS platform 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.

Several approaches similar to MAESoS exist in the literature. For example, Dalpiaz et al. propose an approach based on the concept of context applied over *i**/Tropos goal-oriented models [16]. They provide an evaluation algorithm based on some predefined meta-rules. The main difference is on scope, since MAESoS specifically addresses adaptation due to changes in quality of service (hence the use of quality models to operationalize softgoals) whilst Dalpiaz's adaptation conditions are more generic and may eventually require human assistance. Also, the authors deal with adaptive systems in general and do not focus explicitly on SBS. This also applies to the approaches by Morandini et al. [17] focusing on multi-agent systems; and by Cetina et al. [18] also using variability models as a key element for defining adaptation. Similar to our approach, Elkhodary et al. [19] use variability models to represent engineers' knowledge about system adaptation logic and to abstract from different underlying implementation techniques.

A related body of work that is attracting a lot of research now is that of engineering adaptive requirements. In this context, the result of monitoring could be used to evolve the stakeholders' requirements instead of the SBS [20][21]. Our proposal does not address this issue, since in proposes adapting the solution and not the problem. Requirements adaptation in our framework is assumed to be done externally in the basis of goal evaluation from the monitoring results. Putting together these two lines of research and exploring heuristics about when to adapt the SBS and when to adapt the requirements is part of our future work.

As further 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 engineers to explore the effect of changes without interfering with the running system. Applying MAESoS in real projects is part of our future work too. From the method perspective, we will address the adoption and improvement of current adaptability measures [22] and we plan to provide some guidance in the shift from goal orientation to task orientation, by transforming goal models into sequences of tasks with decision points.

ACKNOWLEDGMENTS

This work has been partially supported by the Spanish project TIN2010-19130-c02-01 and the Christian Doppler Forschungsgesellschaft.

REFERENCES

[1] M. Papazoglou, P. Traverso, S. Dustdar, F. Leymann, "Service-Oriented Computing: State of the Art and Research Challenges". *IEEE Computer* 40(11), 2007.

[2] F. Fleurey, V. Dehlen, N. Bencomo, B. Morin, J.-M. Jézéquel, "Modeling and Validating Dynamic Adaptation". Models@runtime 2008.

[3] A. van Lamsweerde, "Requirements Engineering in the Year 00: a Research Perspective". ICSE 2001.

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

[5] A. Gehlert, A. Metzger (eds.), "Quality Reference Model for SBA". S-Cube deliverable #CD-JRA-1.3.2, 2008.

[6] K. Schmid, R. Rabiser, P. Grünbacher, "A Comparison of Decision Modeling Approaches in Product Lines". VaMoS 2011.

[7] D. Dhungana, P. Grünbacher, R. Rabiser, "The DOPLER Meta-Tool for Decision-Oriented Variability Modeling: A Multiple Case Study". *Automated Software Engineering Journal*, 18(1), 2011.

[8] R. Clotet et al., "Dealing with Changes in Service-Oriented Computing through Integrated Goal and Variability Modeling". VaMoS 2008.

[9] C. Cares, X. Franch, A. Perini, A. Susi. "Towards interoperability of *i** models using iStarML". *Computer Standards and Interfaces* 33(1), 2011.

[10] L. López, X. Franch, J. Marco, "HiME: Hierarchical *i** Modeling Editor". *Revista de Informática Teórica e Aplicada*, 16(2), 2009.

[11] R. Wolfinger, S. Reiter, D. Dhungana, P. Grünbacher, H. Prahofer, "Supporting Runtime System Adaptation through Product Line Engineering and Plug-in Techniques". ICCBSS 2008.

[12] M. Oriol, X. Franch, J. Marco, D. Ameller. "Monitoring Adaptable SOA-Systems using SALMon". MoNA+ 2008.

[13] J. Pimentel, E. Santos, J. Castro, "Conditions for ignoring Failures based on a Requirements Model". SEKE 2010.

[14] J. Horkoff, E. Yu, "Finding Solutions in Goal Models: An Interactive Backward Reasoning Approach". ER 2010.

[15] S. Liaskos, Y. Yu, E. Yu, J. Mylopoulos, "On Goal-based Variability Acquisition and Analysis". RE 2006.

[16] F. Dalpiaz, P. Giorgini, J. Mylopoulos, "An Architecture for Requirements-Driven Self-reconfiguration". CAiSE 2009.

[17] M. Morandini, F. Migeon, M.-P. Gleizes, C. Maurel, L. Penserini, A. Perini, "A Goal-Oriented Approach for Modelling Self-Organising MAS". AAMAS 2009.

[18] C. Cetina, P. Giner, J. Fons, V. Pelechano, "Autonomic Computing through Reuse of Variability Models at Runtime: The Case of Smart Homes". IEEE Computer 42(10), 2009.

[19] A. Elkhodary, S. Malek, N. Esfahani, "On the Role of Features in Analyzing the Architecture of Self-Adaptive Software Systems". Models@Runtime 2009.

[20] L. Baresi, L. Pasquale, "Adaptive Goals for Self-Adaptive Service Compositions". ICWS 2010

[21] N. Qureshi, A. Perini, N. Ernst, J. Mylopoulos, "Towards a Continuous Requirements Engineering Framework for Self-Adaptive Systems". Requirements@runtime 2010.

[22] J. Pimentel, X. Franch, J. Castro, "Measuring Architectural Adaptability in *i** Models". CIbSE 2011.