

Dealing with Changes in Service-Oriented Computing Through Integrated Goal and Variability Modelling

Roger Clotet ¹	Deepak Dhungana ³	Xavier Franch ¹	Paul Grünbacher ^{2,3}	Lidia López ¹	Jordi Marco ¹	Norbert Seyff ²
Universitat Politècnica de Catalunya (UPC) ¹		Johannes Kepler Universität ²			Johannes Kepler Universität ³	
Barcelona, Spain		Institute for Systems Engineering and Automation			Christian Doppler Laboratory for Automated Software Engineering	
		Linz, Austria			Linz, Austria	

Abstract

Variability modelling and service-orientation are important approaches for achieving both the flexibility and adaptability required by stakeholders of software systems. In this paper we present an approach that integrates domain models captured in the i^ modelling framework with variability models to support runtime monitoring and adaptation of service-oriented systems. We believe that approaches integrating goal-oriented modelling and variability management are needed to build, operate, and evolve such systems. We illustrate our approach using two scenarios and present a tentative tool architecture based on an existing product line engineering tool suite.*

1. Introduction

Software-intensive systems are characterized by the heterogeneity of the platforms and networks they operate on; the diversity of stakeholders with changing needs; and the dynamicity of their operating environment [7]. Stakeholders of these systems demand properties such as flexibility and adaptability [27] to allow rapid evolution caused by requirements changes, service performance changes, service updates, new (types of) stakeholders, new regulations, etc. Often, these systems cannot be fully specified in advance and are under constant development, so as to be continuously adjusted and adapted to emerging and evolving requirements from their various stakeholders [3]. The Service-Oriented Computing (SOC) paradigm offers a powerful technological solution for conceiving such flexible and evolvable systems. Services are open components that support rapid and low-cost composition of distributed applications.

Adapting a complex software system to different environments and contexts is also a prime goal of variability modelling in product line engineering [23][24]. Variability modelling is an approach fostering software reuse and

rapid customization of systems. Not surprisingly, researchers have started to explore the integration of service-oriented systems and variability modelling to support run-time evolution and dynamism in different domains [18][20][26][27]. Integrating variability modelling and service-orientation is seen promising to achieve both flexibility and adaptability.

Dealing properly with changes in large systems, as software-intensive systems are, demands a thorough knowledge of the rationale of decisions, alternatives considered, as well as traceability between stakeholders' goals and technical solution elements. Goal-oriented approaches [19] have been recognized as a powerful technique in this direction [6][13]. Among several existing approaches, the i^* framework [28] is gaining popularity to model service-oriented and agent-based systems [22]. Researchers have started to explore new ways to enlarge the framework with variability modelling capabilities [21].

Pursuing similar goals, we have been using the i^* framework to model a service-oriented multi-stakeholder distributed system in the travel domain [7]. The goal was to validate the usefulness of i^* in this context and to gain a deeper understanding of the dependencies between goal modelling and variability modelling. In an earlier workshop paper [17] we have presented some initial results. In this paper we show how variability can be modelled on different levels of abstraction. We present a set of rules that allow to identify variability in i^* models. These rules, formulated over corresponding metamodels, help to convert i^* models into variability models which can then be refined to allow monitoring and adaptation of service oriented systems. We illustrate the approach using examples and present a tool architecture based on our existing work on meta-tools for variability modelling.

2. Change Scenarios

Our fictitious example is a distributed system provided by Travel Services Inc. (TSI), a company offering services to travellers to search for and book trips online. While some of these services are developed by TSI, most are provided by third party Service Providers. Services range from very simple (e.g., currency conversion) to highly complex ones offering large functionality. For example, the system relies on a payment service provider offering payment services to TSI. Further, a number of travel services, e.g., for booking flights or checking the availability of hotel rooms are used. Various Travel Agencies (TA) contract TSI's software solution to offer a customized online travel platform to their customers.

It is obvious that changes play an important role in such a system. Changes such as new requirements, new types of stakeholders or changes in the environment (e.g., new regulations) occur "top-down" while other changes such as service performance variations or service updates happen "bottom-up".

The following scenarios highlight a top-down and a bottom-up change and illustrate subsequent system adaptation. The first scenario highlights how changing needs of stakeholders make it necessary to adapt the system.

Top-down stakeholder-driven change: TSI has so far been used only by European travel agencies, which rely on a payment service for money transfer within Europe. TSI has won a new TA Transworld Travels that interacts with clients in all continents and therefore needs a world-wide payment solution offering more options despite expected higher costs.

1. The new requirement of Transworld Travels might provoke a change affecting other customer TAs serviced by TSI.

2. TSI's software architect analyses whether the new global payment service is interfering with the intentions of other TA's. He finds out that other TAs and the Payment Service Provider (PSP) are affected by this request.

3. TSI negotiates the change with the other customer TAs. The affected TAs and the Payment Service Provider are invited for discussions. The other TAs concur that they are not interested in the new global service, but accept to use a new PSP if it will not increase their costs.

4. The World-Wide Payment System Berne (WWPS-Berne) is willing to provide a more advanced service to satisfy the requested needs and Transworld Travels is willing to pay the extra cost for this new service.

5. Based on this decision, TSI's software architects confirm that Transworld Travels will use WWPS-Berne. All other TAs will also be switched to WWPS-Berne, but for them the service will only provide reduced functionality. The architect updates the system configuration to address this change.

The second scenario illustrates how a system change can be triggered by a highly dynamic software environment. We assume that customer TAs are either Austrian or Spanish.

Bottom-up monitoring-driven change: In order to ensure high availability of the travel services to its customers, TSI has decided to use two world-wide available Amadeus travel services instances, a Spanish and an Austrian Amadeus server. The system configuration specifies that each Spanish TA normally redirects customer requests to the Spanish Amadeus server and Austrian TAs primarily use the Austrian Amadeus Server. According to the system configuration requests of Austrian customers can be redirected to the Spanish Amadeus if necessary and vice versa.

1. The following day Austria qualifies for the Football World Cup to be held at Barbados next summer. All Austrian TAs are experiencing high load on their website as many Austrian customers are eager to book trips. As a result the Austrian Amadeus server is under heavy load and the average response time is increasing considerably. Finally, monitoring tools detect that the average response time is higher than documented by the Service Level Agreement (SLA) between TSI and its customer TAs.

2. The TSI operator is automatically informed that the system is not satisfying the specification and that it is recommended to automatically redirect some Austrian traffic to the Spanish Amadeus server.

3. After a confirmation by the TSI operator the system automatically updates its configuration and redirects some Austrian traffic to the Spanish server.

4. Four hours later, the monitoring tools detect that the load on the Austrian Amadeus server is again stabilized and that its response time satisfies the SLA.

5. The TSI operator is automatically informed by the monitoring system that the redirection of traffic is no longer needed. After the operator's confirmation the system again redirects all traffic to the Austrian Amadeus server.

The two scenarios show that TSI software architect needs support to deal with top-down and bottom up changes. For instance, he needs to understand the common and specific goals of different TSI customers. He also needs to know the current service configuration of customers. In order to determine the best way to proceed, the architect relies on information about alternatives (e.g., the alternative for a certain service in case it fails). The architect also needs traceability information to comprehend the dependencies among goals, service types, and service instances. An up-to-date and detailed representation of the system at different levels, from stakeholders' needs down to the concrete system configuration, is needed to ensure a speedy and correct reaction after top-down or bottom-up changes [7].

We will show that the integrated use of *i** and variability modelling techniques allow to model these different layers together with information traceability and variability information to facilitate system adaptation. In the next section we thus provide the necessary background about *i** and variability models.

3. Background

3.1 The *i** Framework

In [7] we have shown the use of the *i** framework to model all the levels of our distributed system: stakeholder needs, software architecture and running system. The different models are built with similar constructs, although emphasis is different in each case. Figure 1 provides a simplified view of the *i** metamodel [1] only depicting the elements that are relevant for variability modelling.

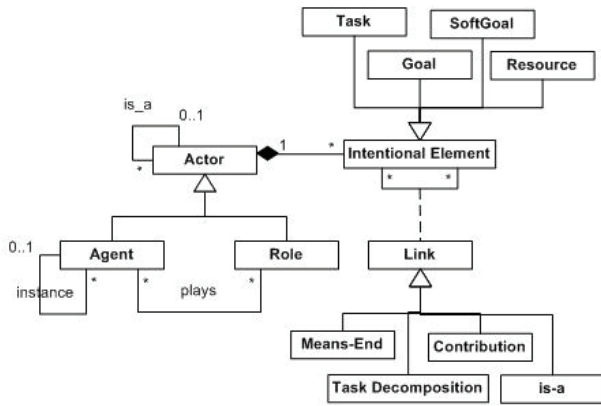


Figure 1. Part of *i metamodel relevant for variability modelling.**

The *i** framework supports the modelling of systems as a network of actors together with their rationale. The network of actors is described using a Strategic Dependency (SD) diagram which shows how actors depend on each others, whilst the rationale is described through a Strategic Rational (SR) diagram for each actor.

Our simplified metamodel describes two kinds of actors: *Roles* are used to represent the system stakeholders and the different “parts” of the system. *Agents* are used to represent the real software components such as services and their instances. Agents can *play* roles and can be *instances* of other agents. There is also an *is-a* relation to create actor hierarchies.

An actor is composed by its *intentional elements*, which are responsible to describe the actor’s needs/requirements or responsibilities. There are four kinds of intentional elements: *goals*, *softgoals*, *tasks* and *resources*. The intentional elements inside an actor can be related between them using different kinds of links:

means-end, *task decomposition* and *contribution* (these links will be used depending on the intentional element type). We allow a fourth link type when there are two actors related by an *is-a* relation: when the actor *A* is-a actor *B*, some intentional elements of actor *A* can be related to intentional elements of actor *B* using *is-a* link (see [8] for a precise definition).

3.2 Variability Modelling

Variability modelling is a key technique in product line engineering to define how various products in a product line can be distinguished from each other. Orthogonal variability modelling [2] is based on complementing existing models and artifacts with variability information rather than using specific notations or languages for variability modelling. For instance, John and Schmid [24] have proposed a decision-oriented approach that supports orthogonal variability modelling for arbitrary artifacts independent from a specific notation. Their work is based on earlier work in the Synthesis project [25]. The benefits of decision-oriented approaches are the flexibility gained and traceability established by using one variability mechanisms for different artifacts at the requirements, design, architecture, implementation, application, and runtime level. Our work is based on the meta-meta-model presented in [10] which uses decision and assets as key modelling elements (see Fig. 2). An *asset model* describes the system elements and their dependencies. The decision model describes the variability of the system through a set of decision variables that are used to adapt a system, i.e., to derive a product from the product line. The *included-if* relationship determines which assets will be part of the system depending on the values of the decision variables.

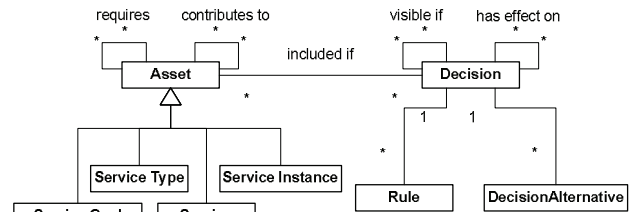


Figure 2. Metamodel for Variability Management.

The first step of the approach presented in [10] is the development of a domain-specific metamodel by identifying the relevant assets and dependencies among them. For this context we identified the following asset types: service goal, service type, service, and service instances: A *service goal* establishes the objective of a service (e.g., “Offer travels”). Different *services types* contribute to fulfilling these goals (e.g., “Travel services provider”). Available services realizing a service type are modelled as

a *service* (e.g., “Amadeus”). Finally, available runtime implementations of services can be modelled as *service instances* (e.g., “Spanish Amadeus Server”). We also identified two kinds of relationships between the assets: The *requires* relationship is used whenever the selection of a certain asset leads to the selection of another asset. This can be a result of logical dependencies between goals, conceptual relationships between service types, relationships between services or functional dependencies between service instances. The *contributesTo* relationship is used to capture structural dependencies between assets of different levels. Service instances for example contribute to services. Services contribute to service types which contribute to goals. It is however also possible for goals to be split up into sub-goals. Such compositional relationships between goals can also be modelled using the *contributesTo* relationship.

A *decision model* is used to model the variability of the system and to describe dependencies between the variation points (cf. Table 1). Decision alternatives describe the range of available options when taking a decision. For example, decision alternatives can be an enumeration of available services. The event of taking a decision triggers the evaluation of attached rules. This includes checking relevant conditions and identifying actions which have to be executed. For example, deciding which travel service shall be used could depend on the average response time of the available services. A condition checking whether the average response time is higher than a predefined threshold could influence the service selection and the identification of actions relevant for service configuration.

Table 1. Partial Decision Model.

<i>Decision Variable</i>	<i>Decision Alternative</i>	<i>Has-effect-on Rule</i>
typeOfCustomer-Assistance	synch, asynch	
typeOfTravelPayment	credit card, transfer, worldwide	
typeOfService Travel-Provider	flight, hotel, camping	if (typeOfService TravelProvider == camping) then whichTravel Service:=Amadeus
whichTravelService	Amadeus, Vivaldi	
whichCredit CardService	CheapCard, Securitas, NorbSecureCredit, FastAndCheap	if (whichCredit CardService == Securitas) then typeOfIdentification:=FingerPrint
whichAmadeusService	Austrian, Spanish	
AustrianAmadeus-AverageResponseTime	Metric [ms]	if AustrianAmadeus AverageResponse Time > 200 then whichAmadeus Service:=Spanish

4. Modelling the Variability of Service-Oriented Systems with i^*

In the previous section we have presented two different metamodelling frameworks that describe the main concepts of the i^* framework and decision modelling. We use each approach to model the same system capturing overlapping but also different information. To guarantee model consistency and traceability we first integrate the two metamodelling frameworks and then introduce the rules for identifying candidate variation points in i^* models and their transformation to decision models.

4.1 Metamodel Integration

Our decision-oriented variability modelling approach has two main elements: *assets* and *decisions*. Assets are included in the deployed system depending on the decisions that the user has taken, i.e., the values of the decision variables. The different kinds of assets have a direct relation with i^* model elements which is shown in the rest of the section. The decision model is not connected to the i^* metamodel since it is dealing with variability which is not kept in the i^* model (cf. Table 2 this information).

Table 2. Integration of Metamodels.

<i>Variability meta-model element</i>	<i>i^* metamodel element</i>	<i>Constraints</i>
Service Goal	Intentional Element	It is not a resource
Service Type	Roles	Software role
Service	Agents	Is playing a role and is not an instance of another agent
Service Instance	Agents	Is an instance of other agent

4.2 Identification of Candidate Variation Points

The i^* framework has not specific constructs for modelling variability. Some authors have tackled this issue by extending i^* with explicit constructs (see Sections 6 and 7). Others consider variability as implicit in i^* models depending on the types of modelling constructs used. We adhere to this perspective and aim at identifying candidate variation points by analyzing the very structure of the i^* model. From our analysis of the Travel Agency example, we have identified six different cases of candidate variation points. We present next the different cases classified by the type of i^* construct.

4.2.1 Means-end variability

A means-end link is used to describe different ways to achieve a goal or a task, thus it may be describing a varia-

tion point, usually related to external variability [23], i.e., the variability of artifacts that is visible to customers.

Means-end links are composed as an OR, so at least one of the means should be attained to achieve the end. This OR can be interpreted as a variation point when the customer can decide the way she wants the system to achieve his goal. For instance, customer assistance can be either provided using asynchronous or synchronous support (see Fig. 3).

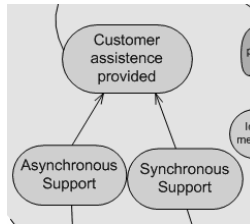


Figure 3. Variation point: Means-end case.

4.2.2 Plays variability

Agents allow modelling real services or components, and these agents play roles. At this point we can find some internal variation points [23] (i.e., the variability of domain artifacts that remains hidden from customers), because the architect can decide among the different services (agents) that can play a role. This variability is represented using the *plays* link. In Figure 4 we find the Amadeus and Schubert agents playing the same role (Travel Services Provider) which means that the architect has to choose between them when deploying the system.

4.2.3 Instance variability

The *i** framework also allows modelling which services instances can be deployed. The example shown in Figure 4 highlights that there are two agents – the Spanish and the Austrian Amadeus Server – as instances of the agent Amadeus. The variability is modelled using the link *instance*.

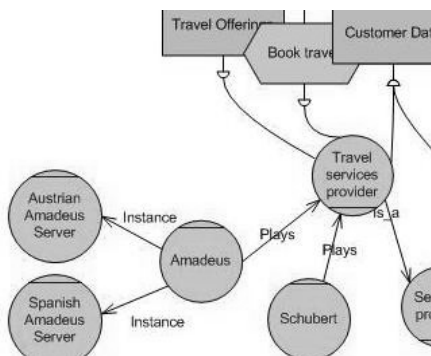


Figure 4. Variation point: Plays and instance cases.

4.2.4 Role inheritance variability

Variability related with architectural features is found in the relationship between actors. In an *i** model, actors can represent the different roles our system has to include. These roles can be classified as a hierarchy using the *is_a* link. For example, Fig. 5 shows a classification for *Travel Payment* role. This example is also an external variation point because the TAs will be able to select a kind of payment they will provide to their customers.

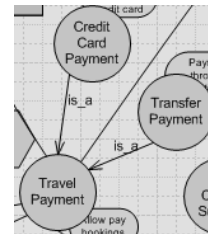


Figure 5. Variation point: Inheritance case.

4.2.5 Intentional element inheritance variability

At a finer-grained scale, a superactor may have different intentional elements refined onto the subactor. This inheritance relationship may take different forms (see [8] for further details). In the case of having more than one heir, the intentional element in the superactor becomes a variation point. For instance, Figure 6 shows how the task *Select Destination* (which is a subtask of task *Buy Travel*) in the general actor *Customer* is refined into its heirs as *Select Destination Country* in *Family* heir and *Select Destination Conference* in *Researcher* heir. In this case there are 2 ways of achieving the task *Select Destination*.

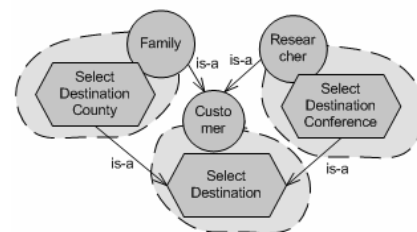


Figure 6. Variation point: Intentional element inheritance.

4.2.6 Softgoal variability

Since the satisfaction of a softgoal is not uniquely defined, we may imagine several criteria acceptable at different moments or contexts. For instance, **Fehler! Verweisquelle konnte nicht gefunden werden.** shows the softgoal *Secure*.

Table 3. Rules for identifying variability in i^* models.

Rule identifier	Type of variation point	Formulation	Additional Restrictions	Decision Variable	Decision Name Prefix	Decision Alternatives	Cardinality	Asset Type
ME-VP	means-end	$\{x_1, \dots, x_n\}$ are means of y	$n > 1$ AND (is-goal(y) OR is-task(y)) AND (not is-resource(x_i))	y	TypeOf y	$\{x_1, \dots, x_n\}$	Min: ≥ 0 Max: $\leq n$	Service Goal
P-VP	play	$\{a_1, \dots, a_n\}$ play r	$n > 1$ AND is-role(r) AND is-agent(a_i)	r	Which r	$\{a_1, \dots, a_n\}$	Exactly 1	Service
I-VP	instance	$\{a_1, \dots, a_n\}$ instance a	$n > 1$ AND is-agent(a) AND is-agent(a_i)	a	Which a	$\{a_1, \dots, a_n\}$	Exactly 1	Service Instance
RI-VP	role inheritance	$\{r_1, \dots, r_n\}$ is-a r	$n > 1$ AND is-role(r) AND is-role(r_i)	r	TypeOf r	$\{r_1, \dots, r_n\}$	Min: ≥ 0 Max: $\leq n$	Service Type
IEI-VP	IE inheritance	$\{x_1, \dots, x_n\}$ is-a y	$n > 1$ AND is-IE(y) AND is-IE(x_i)	y	TypeOf y	$\{x_1, \dots, x_n\}$	Exactly 1	Same as inherited element
SG-VP	softgoal	is-softgoal(y)		y	LevelOf y	Metrics available as fit criterion	Exactly 1	Service Goal

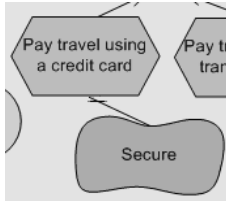


Figure 7. Softgoal variability.

There may be several strategies for satisfying this softgoal. One option could be use a website that implements SSL v3 to transfer credit card information. Another option could be to make a phone call to a TA operator and provide the credit card information. If we choose SSL, it may be sufficient to use an 80-bit key. Advances in cryptology could make 80-bit keys obsolete.

4.3 Building decision tables from i^* models

We have identified a set of rules to generate the corresponding excerpt of a decision model from variation points found in the i^* model. We define this correspondence in terms of the metamodel. The result is summarized in Table 3. Also, in table 1 (see section 3.2) we compile the excerpts of decision models built up from the i^* models presented in 4.2.

Means-end variability rule (ME-VP). This rule is applicable when a goal or a task is the end for more than one means-end link. Only goals, softgoals and tasks are taken into account to know if there are more than one means;

we consider that a resource as a means is an information needed to attain the end, not one way to achieve it.

Plays variability rule (P-VP). This rule is applicable when a role is played by several agents in the model.

Instance variability rule (I-VP). This rule is applicable if an agent is instantiated by several other agents in the model. This means that there exist different deployments of the same type of service that may be selected, typically according to SLA clauses.

Role inheritance variability rule (RI-VP). This rule is applicable when there is a classification of roles. This means that there are different kinds of agents, representing the same role. So one or more of the heirs will be chosen depending on the user characteristics.

Intentional element inheritance variability rule (IEI-VP). This rule is applicable for actor classifications using inheritance if some inherited intentional elements are modified in their heirs. In the three cases of inheritance identified in [8] (extension, refinement and redefinition), the intentional element placed in the parent has different ways to be achieved. In the case of extension, the new features are considered as alternatives to the parent. In the other cases, each intentional element declared as an heir is considered a way to achieve the intentional element in the parent.

Softgoal variability rule (SG-VP). This rule is applicable for every softgoal of the i^* model. Since softgoals are high-level concepts, we need here some more concrete fit criterion, e.g. metrics or qualitative reasoning arguments for the particular softgoal. A catalogue of such metrics and techniques would be helpful, and then the different items of the catalogue would be the possible decisions.

It is important to mention that in all of the cases, the rule will be applied only when the decision is relevant (concept of relevance of a variation point). For the sake of an example, consider a role R played by two agents A and B, and assume that the agent A has two instances C and D. Then, the rule I-VP over A is applied only if the decision variable Which R equals A.

5. Tool Architecture

We are currently developing a set of tools for supporting the type of scenarios discussed in Section 2 using the introduced concepts. We are using software components that are part of the DOPLER product line tools suite [11] and plan to extend them with software components for service monitoring and service adaptation. We have shown that we use *i** to create a domain model of our service oriented system. Such domain models can be encoded using *iStarML*, an emerging XML-based standard allowing model exchange among existing tools for *i** [5].

We assume that monitoring and adaptation of a service-oriented system should never be fully automated as user feedback will be required in most domains. This means that we need to provide an interface that enables system administrators to (i) manually modify a service-oriented system by exploiting the known variability or to (ii) confirm changes suggested by the reasoning capabilities of our tool architecture (e.g., when replacing one service with another). Instead of automating the procedure for taking decisions, the proposed tool architecture therefore provides a user console which is able to perform the discussed tasks in an intuitive manner. Similar to work reported in [27] we are adopting the DOPLER component ConfigurationWizard for this purpose.

5.1 Architecture overview

The *variability management engine* is at the heart of our tool architecture. The encoded *i** domain model is used as input for DecisionKing [9], a meta-tool for variability modelling. We are adopting DecisionKing in our architecture to support variability modelling as described in Section 3 [10]. DecisionKing allows managing a variability model of available service instances, services, and stakeholder goals together with trace links. Decisions are used to represent variation points. The rule engine JBoss Rules¹ used by DecisionKing supports reasoning needed to compute the impact of changes stemming from monitored service behaviour or user-triggered adaptations. It is important to note that we do not aim at creating a complete domain-service variability model from an existing *i**

model. We can however create candidate variation points based on the rules described in Section 4. These initial rules can be refined using DecisionKing's rule language to express dependencies and constraints more precisely.

The *Monitor component* acquires the monitoring parameters from the variability management engine. It provides measure instruments (MI) for measuring and monitoring different runtime parameters such as response time or availability of services. The *Adaptor component* performs the changes that are required based on the actions carried out by the user. It performs the actual update of the service-oriented system. Both monitors and adaptors are unaware of the concrete technologies used to develop and compose the service-oriented system. These software components define generic extension points that can be implemented to address technology-dependent aspects. Plug-ins are used to connect the generic monitoring and adaptation components to concrete service implementation technologies (e.g., BPEL, WSDL) We are currently developing initial examples of these plug-ins.

The *Services* represents existing deployed services of the service-oriented system. It is important to note that these services are often developed, maintained, and deployed by 3rd party service providers.

Suggestions for actions are presented to the user for dealing with changes in the services. The suggestions are inferred from the variability management engine, which is constantly updated with current monitoring parameters.

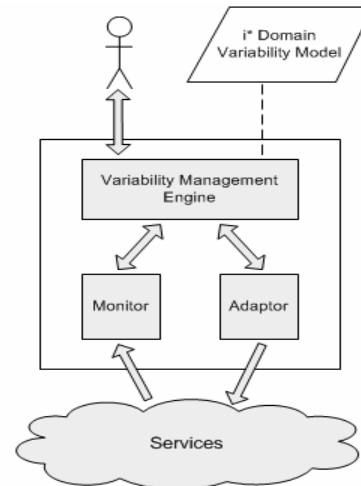


Figure 8. Tool Architecture Overview.

5.2 Revisiting the scenarios

Let's consider the impact of the framework and the usage of the tool by revisiting the bottom-up monitoring-driven change scenario presented in Section 2. We explain in more detail how the framework and tools work.

¹ <http://www.jboss.com/products/rules>

1. *In order to ensure high availability of the travel services to its customers TSI uses two world-wide available Amadeus travel services instances concurrently (only one is used at the time, each TA has a primary one it is using)*
 - a. The technology independent information about the Spanish Amadeus server and Austrian Amadeus server (e.g., service location, service name, etc) is defined in the i^* Domain Variability Model (see Figure 8).
 - b. The architect configures the Variability Management Engine (VME) by adding information about these two services.
 - c. The architect configures the Monitor by specifying rules and deploying Measure Instruments (MI) to check the running system. The decision model with rules for load balancing is integrated in the Monitor and also in the Adaptor.
2. *The day after Austria classifies for the Football World Cup to be held at Barbados next summer, all Austrian TAs are experiencing high load on their website as many Austrian customers are eager to book trips. As a result the Austrian Amadeus server is under heavy load and the average response time is increasing considerably.*
 - d. The corresponding MI inside the Monitor detects a sharp increase of the response time.
 - e. The Monitor discovers that the average response time is higher than the threshold established in the SLA.
 - f. The Monitor sends a notification to the VME.
 - g. The VME reevaluates all rules after the notification to compute the effects of the change. The VME suggests and “automatic” switch from Austrian to Spanish Amadeus server for Austrian TA.
3. *The TSI operator is automatically informed by the system about the suggested automatic switching of the service. The operator confirms the switch in his configuration console.*
 - h. The TSI operator confirms the change.
 - i. The Adaptor is reported to make the change.
 - j. The current configuration is automatically updated by the Adaptor to address the service switch. Some of the new requests are redirected to Spanish Amadeus.
 - k. In a general case, we may need to inform to Analyzer about new services or new rules for monitoring.
4. *Four hours later, the Austrian Amadeus server is again stabilized and its response time is satisfying the SLA.*
 - l. The Monitor, following low-level MI measures, reports a decrease of the response time of the Austrian Amadeus server.
 - m. The Monitor realises that the average response time is now below the level established in the SLA.
 - n. The Monitor sends a notification to the VME.
 - o. The VME reevaluates all rules after the notification to compute the effects of the change. The VME suggests an “automatic” switch from Spanish to Austrian Amadeus server.
5. *The TSI operator is automatically informed by the system the service switching is no longer needed.*
 - p. The TSI operator confirms the change to the VME. (He might also decide to update the VME rules to better react to the situation. In this case an updated list of rules is automatically sent to the Analyzer).
 - q. The Adaptor is notified to perform the change.
 - r. The current configuration is automatically updated by the Adaptor to address the service switch. Every new request is redirected to Austrian Amadeus server.

6. Related work

Managing variability in i^* models is an emergent research issue. Some authors are interested in including the variability information in i^* models, while others focus on analyzing the implicitly captured variability in these models to create the variability model:

For instance, Bibian *et al.* [4] include decision boundaries for the easy identification of goals and the corresponding features. Yu *et al.* [29] include new constructors to distinguish the different types of features (Mandatory, Optional, Alternative and Or). Some authors also include some information about variability constraints by adding new constructs to the i^* language. Specifically, Liaskos *et al.* [21] add new links to model for representing constraints among goals, at level of satisfaction.

Other approaches suggest analyzing models to discover variability. For example, Baxuali *et al.* [15] show how studying the OR-trees goals and the contribution to softgoals can be used for the study of variants (functional behavior) depending on customer priorities (non-functional attributes). Liaskos *et al.* [21], besides introducing constraints to the language, show how goal models can be used to capture variability, constructing variability frames studying the goals semantics.

Even, Baxuali *et al.* present a new idea about using aspect-orientation concepts in goal models to deal with variability [16]. In this paper they also add new constructs such as crosscutting links to the i^* language.

Numerous researchers from different areas have developed approaches and tools contributing to runtime adaptation of systems: In the area of requirements engineering, researchers have explored runtime deviations of systems

from original requirements. In [12] an approach based on goal models specified in the formal language KAOS is presented. The approach adopts a set of agents to monitor runtime behaviour of systems and to suggest either automated or runtime adaptation of the systems. Variability is expressed via alternative refinements in goal models. In [3] different levels of requirements engineering for dynamic adaptive systems have been explored. Their aim is to provide a general framework bridging human-centred requirements and machine-centred adaptation mechanisms. Other approaches combine product line engineering and runtime adaptation of systems. For instance, [18] presents a feature-oriented approach for dealing with runtime adaptation which is based on identifying binding units in feature models that serve as the basis for later reconfiguration. The work of [26] shows how product line architectures can be used to support feature adaptation in the area of Web system personalization.

7. Conclusions and future work

Understanding the dependencies and interactions between goal modelling and variability modelling is important to support runtime adaptation of service-oriented systems. This paper presents how a variability model can be obtained from a goal model without including new language constructs to the goal model to avoid unnecessary complexity.

We have explored the possibility of including the variability explicitly in these kinds of models for instance by adding some graphical information to show which groups corresponds to variation points. Similar to the OR and AND labels for means-end links in Tropos [14] the modeler could include the label VP for variation points. We realized however that it is impossible to add variability information without negative effects on model comprehensibility. We thus decided to complement i^* models with an orthogonal variability modelling technique based on decision models. Some variability can be found in i^* models although it is not explicitly modelled. In our approach we thus decided not to include variability in the goal model but to use a set of rules to extract initial variability models by discovering variability in goal models. While other authors analyze the variability inside actors [15][21][16] our approach focuses on variability stemming from actor relations which are particularly important in service-oriented systems. We present a preliminary set of rules for identifying the variation point candidates. Most of these rules apply to links between actors (is-a, plays, instance). The rules allow deriving an initial variability model (mainly decisions and their alternatives). To define a complete variation model we need to manage dependencies between decisions, rules, and cardinalities which we manage in a separate model outside i^* . Model

synchronization is supported by iStarML, a XML-based exchange format for goal models [5].

We are currently developing different elements of the tentative tool architecture presented in Section 5. A critical task in the near future is to validate the architecture using reasonably complex examples. This will also require the development of service monitors for different quality aspects. An interesting challenge is to make the framework technology-independent such that it can be used with different service technologies and monitoring environments.

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] Ayala, C.P., Cares, C., Carvallo, J.P., Grau, G., Haya, M., Salazar, G., Franch, X., Mayol, E., and Quer, C. "A Comparative Analysis of i^* -Based Goal-Oriented Modeling Languages". In: Proceedings of The Seventeenth International Conference on Software Engineering and Knowledge Engineering (SEKE'05). 14-16 July, 2005. Taipei, Taiwan, Republic of China. Pages: 43-50.
- [2] Bachmann, F., Goedicke, M., Leite, J., Nord, R., Pohl, K., Ramesh, B. and Vilbig, A., "A Meta-model for Representing Variability in Product Family Development", in *Lecture Notes in Computer Science: Software Product-Family Engineering*. Siena, Italy: Springer Berlin / Heidelberg, 2003, pp. 66-80.
- [3] Berry, D., B. Cheng, and J. Zhang, "The Four Levels of Requirements Engineering for and in Dynamic Adaptive Systems", 11th International Workshop on Requirements Engineering: Foundation for Software Quality (REFSQ'05), Porto, Portugal, June 2005.
- [4] Bidian, C., Yu, E.S.K. "Towards Variability Design as Decision Boundary Placement". Anais do WER07 - Workshop em Engenharia de Requisitos, Toronto, Canada, May 17-18, 2007, pp 139-148.
- [5] Cares, C., Franch, X., Perini, A., Susi, A. "Introduction to iStarML". Research report ITC/IRST, 2007.
- [6] Castro, J., Kolp, M., Mylopoulos J. "Towards Requirements-Driven Information Systems Engineering: The Tropos Project". *Information Systems*, vol. 27, 2002.
- [7] Clotet, R., X. Franch, P. Grünbacher, L. López, J. Marco, M. Quintus, and N. Seyff, "Requirements Modeling for Multi-Stakeholder Distributed Systems: Challenges and Techniques", 1st Int. Conf. on Research Challenges in Information Science

- (RCIS), Ouarzazate, Apr. 23-26, 2007.
- [8] Clotet, R., Franch, X., López, L., Marco, J., Seyff, N., Grünbacher, P., The Meaning of Inheritance in i*. 17th International Workshop on Agent-oriented Information Systems (AOIS-2007), Trondheim, Norway, June 11, 2007.
- [9] Dhungana, D., Grünbacher, P., Rabiser, R., "DecisionKing: A Flexible and Extensible Tool for Integrated Variability Modeling.", 1st International Workshop on Variability Modelling of Software-intensive Systems, Limerick, Ireland, 2007.
- [10] Dhungana, D., Grünbacher, P., Rabiser, R. "Domain-specific Adaptations of Product Line Variability Modeling", IFIP WG 8.1 Working Conference on Situational Method Engineering: Fundamentals and Experiences, Geneva, Sept. 2007.
- [11] Dhungana, D., Rabiser, R., Grünbacher, P., Lehner, K., and Federspiel, C., "DOPLER: An Adaptable Tool Suite for Product Line Engineering", 11th International Software Product Line Conference (SPLC 2007), Kyoto, Japan, Sep. 10-14, 2007.
- [12] Feather, M. S., Fickas, S. van Lamsweerde, A., and Ponsard, C., "Reconciling System Requirements and Runtime Behavior", Proceedings of the 9th international Workshop on Software Specification and Design, Washington, DC, April 1998.
- [13] Franch, X., Maiden, N.A.M. "Modeling Component Dependencies to Inform their Selection". In: Proceedings 2nd International Conference on COTS-Based Software Systems (ICCBSS), Lecture Notes on Computer Science 2580, Springer, 2003.
- [14] Fuxman A., Liu L., Mylopoulos J., Pistore M., Roveri M., Traverso P. "Specifying and Analyzing Early Requirements in Tropos". *Requirements Engineering Journal*, 9 (2), 2004.
- [15] González-Baixauli, B., Leite, J.C.S.P., and Mylopoulos, J. "Visual Variability Analysis with Goal Models". Proc. of the RE'2004. Sept. 2004. Kyoto, Japan. IEEE Computer Society, 2004, pp. 198–207.
- [16] González-Baixauli, B., Laguna, M.A., Leite, J.C.S.P. "Using Goal Models to Analyze Variability". First International Workshop on Variability Modelling of Software-intensive Systems (VaMoS 07), Volume 2007-01, pp. 101–107, Jan. 2007.
- [17] Grünbacher, P., Dhungana, D., Seyff, N., Quintus, M., Clotet, R., Franch, X., López, L., Marco, J.: Goal and Variability Modeling for Service-oriented System: Integrating i* with Decision Models. In: Proceedings of Software and Services Variability Management Workshop: Concepts Models and Tools. Helsinki, Finland, 19-20 April 2007, pp. 99–104.
- [18] Hallsteinsen, S., Stav, E., Solberg, A., and Floch, J., "Using Product Line Techniques to Build Adaptive Systems", Proceedings of the 10th international on Software Product Line Conference, Washington, DC, Aug. 21-24, 2006, pp. 141–150.
- [19] van Lamsweerde, A. "Goal-Oriented Requirements Engineering: A Guided Tour". In *Proceedings 5th IEEE International Symposium on Requirements Engineering (RE 2001)*, Toronto (Canada), 2001.
- [20] Lee, J., and K.C. Kang, "A Feature-Oriented Approach to Developing Dynamically Reconfigurable Products in Product Line Engineering", Proceedings of the 10th International Conference on Software Product Line, Washington, DC, Aug. 21-24, 2006, pp. 131–140.
- [21] Liaskos, S. Yu, Y., Yu, E., Mylopoulos, J. "On Goal-based Variability Acquisition and Analysis". Proc. 14th IEEE Int'l Requirements Engineering Conference (RE'06) (Sept 11-15, 2006). IEEE Computer Society.
- [22] Penserini, L., Perini, A., Susi, A., Mylopoulos, J. "From Stakeholder Needs to Service Requirements". Proceedings of the 2nd International Workshop on Service-Oriented Computing: Challenges on Engineering Requirements (SOCCER), 2006.
- [23] Pohl, K., Böckle, G., van der Linden, F. J., *Software Product Line Engineering: Foundations, Principles, and Techniques*: Springer, 2005.
- [24] Schmid, K., John, I., "A Customizable Approach to Full-Life Cycle Variability Management". *Journal of the Science of Computer Programming, Special Issue on Variability Management*, vol. 53(3), pp. 259–284, 2004.
- [25] Software Productivity Consortium, *Reuse-driven Software Processes Guidebook* (SPC-92019-CMC, Version 02.00.03), Herndon, VA, November 1993.
- [26] Wang, Y., Kobsa, A., van der Hoek, A. and J. White, "PLA-based Runtime Dynamism in Support of Privacy-Enhanced Web Personalization", Proceedings of the 10th international on Software Product Line Conference, Washington, DC, Aug. 21-24, 2006, pp. 151-162.
- [27] Wolfinger R., Reiter S., Dhungana D., Grünbacher P., and Prähofer H.: Supporting Runtime System Adaptation through Product Line Engineering and Plug-in Techniques. 7th IEEE International Conference on Composition-Based Software Systems (ICCBSS), February, 25-29, 2008, Madrid, Spain.
- [28] Yu, E. Modeling Strategic Relationships for Process Reengineering, PhD Thesis, Toronto, 1995.
- [29] Yu, Y., Mylopoulos, J., Lapouchnian, A., Liaskos, S., Leite, J.C.S.P."From stakeholder goals to high-variabilitysoftware designs". Technical Report CSRG-509, University of Toronto, 2005. Available at: <ftp://ftp.cs.toronto.edu/csrgtechnical-reports/509>.
- [30]

