# Requirements Modelling for Multi-Stakeholder Distributed Systems: Challenges and Techniques

Roger Clotet* Xavier Franch* Paul Grünbacher‡ Lidia López* Jordi Marco* Michael Quintus† Norbert Seyff†
*Software Department
Universitat Politècnica de Catalunya (UPC)
08034 Barcelona, Spain
Email: {rclotet|franch|llopez|jmarco}@lsi.upc.edu
†Institute for Systems Engineering and Automation
Johannes Kepler Universität (JKU)
A-4040 Linz, Austria
Email: nseyff@sea.uni-linz.ac.at, michael.quintus@students.jku.at
‡Christian Doppler Laboratory for Automated Software Engineering
Johannes Kepler Universität (JKU)
A-4040 Linz, Austria
Email: paul.gruenbacher@jku.at

***Abstract*— Multi-stakeholder distributed systems (MSDS) are distributed systems in which subsets of the nodes are designed, owned, or operated by distinct stakeholders. New computing paradigms such as service-oriented computing mean that challenges posed by MSDS will be more dominant in the future. These challenges have particular implications for requirements engineering (RE). For example, in MSDS decisions about the system architecture are increasingly shifted from system design to system operation. In this paper we discuss the characteristics of MSDS and present a framework for structuring the MSDS research issues. Using an example we illustrate that existing RE approaches for goal modelling, variability modelling, and negotiation techniques can be used successfully if used in an integrated manner to address the identified challenges.**

## I. Introduction

Multi-stakeholder distributed systems (MSDS) have been described as distributed systems in which subsets of the nodes are designed, owned, or operated by distinct stakeholders [1]. MSDS are quickly gaining importance in today's networked world by several technology trends such as service-oriented computing, web services, etc. MSDS pose a number of critical challenges to both researchers and practitioners and the consequences of MSDS for system development, operation, and evolution are manifold.

MSDS are often composed from already existing services and components. This means that in the future, requirements engineering (RE) approaches need to be aware of existing technological solutions [2], [3]. This shifts many RE decisions from design-time to runtime when, e.g., services are often already in operation when being considered for integration. Also, as MSDS are operated by distinct stakeholders they can change uncontrollably and rapidly. If one stakeholder modifies some part of an MSDS, this can have severe implications on the users of other nodes. Providing adequate knowledge on how to deal with such changes is thus another challenge. In MSDS, negotiation processes ensuring stakeholder agreement are important to provide and maintain mutually satisfactory solutions. This also means that understanding the goals of different stakeholders and resolving potential conflicts becomes essential.

Dealing with these challenges requires the adoption of proven RE approaches in an integrated manner. Here we present an approach combining the strength of three consolidated disciplines that fit well with the needs of MSDS: goal-oriented requirements modelling [4], decision-oriented approaches and variability modelling [5], [6], as well as negotiation techniques [3], [7]. The approach is based on our high-level framework of RE for MSDS presented in [8].

In Section II we discuss a typical MSDS example to illustrate the problems and challenges of this type of systems. In Section III we briefly introduce our framework for MSDS. In Section IV we present our approach integrating concepts from requirements modelling, requirements negotiation, and product line engineering. In Section V we apply this framework to the MSDS example from Section II. In Section VI we make a preliminary assessment of our approach. In the last Section we present conclusions and further work.

## II. Illustrative Example

Our example is a distributed system provided by *Travel Services Inc. (TSI)* that allows travellers to search for and book trips online. While some of the required services are developed by TSI, most are provided by third party service providers. Services range from very simple to highly complex offering large functionality. For example, the system relies on a payment service provider offering payment services to TSI. The system also relies on a number of travel services, e.g., for booking flights or checking the availability of hotel rooms. *Travel Agencies (TA)* contract TSI's software to offer a customized online travel platform to their customers. These TA have different expectations and needs. For example, *University_TA*

*(UTA)* is a travel agency specialized to support researchers in planning trips while *Family_TA (FTA)* is focusing on trips for families. TSI is interested to support further types of travel agencies in the future.

The following scenario is used to highlight selected problems stemming from several interoperating services and diverse stakeholders using those services: A regular customer of NorthStar Travel Agency reports an issue by e-mail: *"I've recently read in a magazine that CheatCard credit service is blamed for selling private data of customers for advertising organizations"*.

1) NorthStar TA looks into the details of their contract with TSI and finds out that the CheatCard credit service is used in the web portal. The contract is not explicit with respect to using private data for advertising.
2) After an internal discussion, NorthStar TA agrees to report this problem to TSI because they are unaware of how TSI has actually implemented the payment service. They are even unsure whom to contact to resolve the issue.
3) TSI is initially unaware of the problem but promises to investigate the issue immediately.
4) TSI is trying to locate the problem. However, due to the limited information available this turns out to be extremely challenging. A lengthy investigation finally reveals that CheatCard is indeed violating privacy laws by selling customer data to third parties. TSI's CEO requests from TSI to replace the service immediately.
5) The software architect starts searching for alternative services according to the customers' goals.
6) TSI's CEO and software architect, together with major customers, discuss three candidate services that could replace CheatCard.
7) The software architect replaces CheatCard credit service with the Securitas service.
8) TSI sends updated contracts to all TA with Securitas as the new service.

This simplified scenario reveals some typical issues in MSDS:

- The CheatCard credit service is operated by a distinct company. While there is an incentive for CheatCard to sell customer data to increase profit this clearly violates the privacy concerns of other stakeholders in the MSDS.
- It is virtually impossible for stakeholders to understand how system nodes are linked in the MSDS to achieve the desired purpose. E.g., it is not transparent for NorthStar TA that TSI is using third-party payment services.
- In case of problems the limited information about the network of adopted services means that it is hard to locate the problem or to find a point of contact for resolving it. It is also difficult to react to problems, e.g., by replacing nodes, due to the unclear impacts of such a change. As customer goals and preferences as well as properties of nodes are not known explicitly it is hard to guide problem resolution.
- Changes to a system node can happen at any time with undesired impacts on other stakeholders in the MSDS. A local change or behaviour can clash with the goals of other stakeholder in the MSDS. It is difficult to monitor and observe nodes to make sure stakeholder goals are not violated. Such monitoring activities can be very complex or even impossible.

## III. A Framework for MSDS

The example shows that MSDS challenges are not purely technical. Rather they cover different layers ranging from stakeholder goals to low-level aspects of system composition and monitoring. In an earlier paper [8] we have discussed these different concerns using a framework. The current version of this framework is shown in Fig. 1. It addresses both design-time synthesis/analysis as well as runtime monitoring/adaptation aspects and covers the following layers:

*Stakeholder Needs:* This layer addresses the needs of stakeholders in the MSDS domain. It is typically hard to model stakeholder needs. Goal modeling provides a language to express stakeholder goals and dependencies. For instance, customers of TAs may express their goal to keep personal data private, and this goal will depend on the correct behavior of other stakeholders such as the TA itself. If stakeholder needs are not modeled explicitly they remain tacit. E.g., TSI may be unaware of the strong privacy goals of TA customers.

*Architecture Prescription:* The Architecture Prescription layer is introduced as an intermediate layer to map negotiated requirements onto architectural concepts [9]. It defines the specification of services and components and their relationships to stakeholder needs. For example, to satisfy the stakeholder goal "Provide Travel Information", a weather forecast service may be defined on this layer.

*Solution Architecture:* This layer addresses the mapping of architecture prescription elements to real world elements. Based on the specifications from the prescription layer, real world services and components are selected. While a payment service has been specified in the prescription layer, CheatCard and Securitas are concrete services that can be selected for the solution architecture.

*Open System:* The MSDS including components, services, and connectors is represented in this layer. Knowing the location and other physical properties of the system's elements is fundamental for monitoring its behavior. In an MSDS, an individual stakeholder's goal may conflict with goals of other stakeholders, even more than in other type of systems. For instance, the goal to ensure privacy of data may collide with the TSI goal of advertising their services to a wide network of potential customers. Negotiation is essential for resolving such requirements conflicts. In the lower layers, negotiation is a pre-requisite for selecting adequate services or components that meet mutually satisfactory agreements of stakeholders during design-time. If such agreements become obsolete due to design-time or runtime changes of services or components, stakeholders will have to re-negotiate.

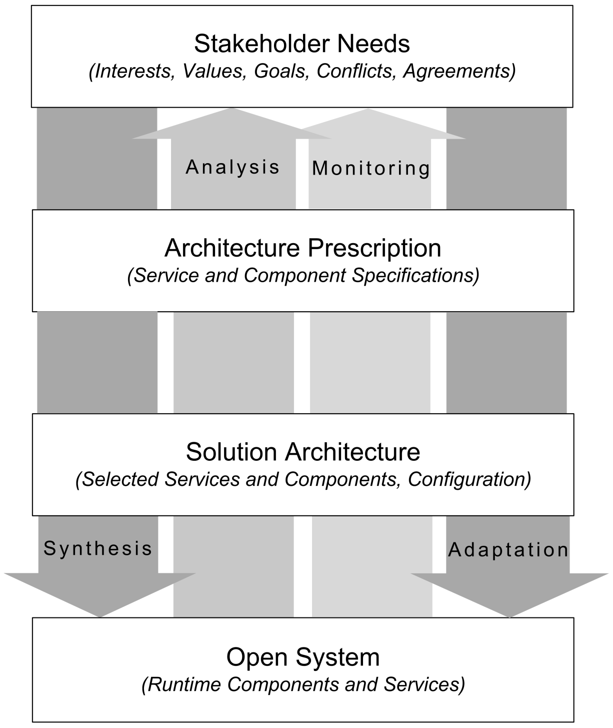
Stakeholder Needs
(Interests, Values, Goals, Conflicts, Agreements)
Analysis
Monitoring
Architecture Prescription
(Service and Component Specifications)
Solution Architecture
(Selected Services and Components, Configuration)
Synthesis
Adaptation
Open System
(Runtime Components and Services)

Fig. 1. Framework for MSDS RE (adapted from [8]).

It must be noted that the layers in the framework are highly intertwined and complement each other. To move from one layer to another we have identified two pairs of related processes. During *synthesis*, the definition of the open system takes place by selecting and composing services and components based on high-level stakeholder needs. The *analysis* of the MSDS is continuously performed during synthesis, to check emergent properties of the system. At runtime, *monitoring* (i.e., feedback from the system during operation back to the stakeholder) allows *adapting* the system to changes of various kinds for ensuring satisfaction of stakeholder goals and alerting about the violation of those goals.

## IV. USING MODELLING TECHNIQUES IN THE MSDS FRAMEWORK

The layers and processes outlined in the MSDS framework need to be refined with concrete languages, techniques, methods, and tools. In this section we present these conceptual elements and explore their application in the framework.

### *A. Modelling Techniques*

We have been experimenting with different approaches supporting the framework. In the MSDS domain we rely on methods that allow the elicitation and modeling of stakeholders and their goals. We also need approaches supporting the negotiation and reconciliation in case of conflicts between stakeholders. Finally, we need to be able to deal with the dynamism of MSDS by modeling variability on all layers of the framework.

Our approach is based on the *i** modelling language. The approach is extended and complemented with variability modelling techniques from product line engineering and negotiation techniques known from requirements engineering.

*The i* language:* Goal-oriented models [10] are a good choice because they allow expressing high-level requirements, recording the rationale behind requirements, and decomposing them at the required level of detail. An interesting choice is the *i** notation [4] because it is not only goal-oriented but actor-oriented too, thereby supporting the assignment of responsibilities to system actors. At this level, actors are mainly used to represent stakeholders, while in lower levels they stand for services and components. There are two types of models in *i**: Strategic Dependency (SD) models declare the actors, their relationships (e.g., specialization and composition) and how they depend on each other. Strategic Rationale (SR) models state the main goals of these actors and their decomposition using some refinement constructs. Both types of models together provide a comprehensive view of the system.

*Negotiation techniques:* A significant number of approaches are available to support negotiations [3]. The win-win negotiation approach [11], for example, allows expressing stakeholder goals as win conditions. In case of conflicting win conditions, risks, or uncertainties an issue is created. Such issues are resolved by proposing options and alternatives to overcome the identified problems. The goal is to cover all win conditions by agreements. Refinements of the win-win approach are available to support stakeholder involvement, prioritization, and trade-offs in negotiations [7].

*Variability modelling:* Product line concepts such as variability modelling are increasingly used to support run-time evolution and dynamism in different domains. There exist various tools and techniques to deal with variability on different levels, i.e., at the requirements, design, architecture, implementation, application, and runtime level. For instance, John and Schmid [6] describe a customizable approach for variability management which is independent from specific notations. Their approach is based on decision models, which are built by specifying decision variables and constraints between them. Bachmann *et al.* [5] propose a conceptual model of variation to represent variability in product family development. In their model, a variation point represents an explicit engineering decision to allow for several alternative variants with regard to selected assets of system development. Authors have also distinguished between external variability, i.e., the variability of artefacts that is visible to customers, and internal variability, i.e., the variability of domain artefacts that remains hidden from customers [12].

### *B. Synthesis and Analysis*

The following two sections describe how synthesis and analysis can be realized according to the layers of the framework (see Fig. 1).

*Stakeholder Needs Layer:* In this layer, the main objective is to develop a requirements model capturing the needs of the various stakeholders. According to the analysis undertaken in Section II, understanding the needs of stakeholders is essential for dealing with MSDS challenges such as conflicting goals or reacting to changes properly. It becomes clear that identifying the right set of actors covering these stakeholders is a key point for modeling. Therefore, methods that provide guidance for this aspect are needed; we may mention the R*i*SD method for building *i** models [13] and we also remark that win-win emphasizes identification of stakeholders as the first step to be carried out in the process. During the synthesis of this requirements model, different analyses may be undertaken. For instance, the workability of the different actors [4] or predictability of the model [14] may be analyzed and the model may be tuned to achieve the required results.

As remarked in Section II, the usual problem of conflicting requirements becomes especially important in MSDS and we need specific techniques to address with this issue. An excellent starting point for analyzing potential conflicts is the information gathered on the stakeholder needs layer, i.e., the *i** models expressing goals of different stakeholders.

A win-win negotiation may be carried out as a two-step process. First, all members of a particular stakeholder group (e.g., all the representatives of travel agencies) may negotiate their wishes and needs. In the general case, they will agree in some points but they will also have different goals with respect to other issues. In the *i** model, we will represent each group with a hierarchy of actors using the specialization (*is-a*) construct. The root will stand for the actor (e.g., TA) and will include the general needs, while there will be an heir for each stakeholder group (e.g., University TA) with their specific needs.

Next, a general negotiation among all stakeholders is performed to agree on the final model. In fact, this negotiation may be seen as the basis of the analysis performed in this layer, meaning that the results obtained usually impact in the requirements model. As widely known in requirements engineering, modeling and negotiation are two highly intertwined processes. These two aspects play an important part in this layer of our MSDS framework.

As stated above, after negotiation we may end up with different hierarchies of actors corresponding to specializations of different groups of stakeholder in case stakeholders were unable to agree on one system. This is similar to product line modelling [12], in which we have a common part with specific additions for each group of stakeholders (e.g., specific customers). The goals that are refined in different ways in the heirs stand for external variation points in the system.

*Architecture-related Layers:* In the synthesis process we need to evolve the negotiated and agreed requirements into architectural solutions. As presented in Section III, we have distinguished three different layers. A goal in our approach is to describe and use these architectural layers using the same models and techniques as in the first layer.

We have shown in previous works [2] that *i** supports modeling the mapping of stakeholder goals onto architectural concepts on the architecture prescription layer. The different model elements of *i** (e.g., roles, agents, etc.) are applicable also in the three lower levels of the MSDS framework. For instance, at the architecture prescription layer, there may be two different actors (expressed as roles in *i**) for the types of services "Flight Ticket" and "Hotel Booking". In the solution architecture, we may choose to cover these roles with one actor "Amadeus" (modeled as an *i** agent since it is a real-world entity). In the open system layer, we may choose the "Amadeus" service hosted on the Spanish site.

The use of product line engineering concepts such as variability models can help specifying, discovering and selecting services or other type of components more precisely. The alternative services and components for the selected one can be recorded for further monitoring (e.g., if one service fails, we can explore the recorded alternatives) as internal variation points [12]. As stated in the previous subsection, we may use *i** constructs to implement these concepts.

Win-win negotiations can be useful in the solution architecture layer to discuss and select possible options and alternatives about how to realize the stakeholders' goals (e.g., which particular services or components are available for which system actors). Also, in the open system layer, negotiation may help to decide on the details of the initial deployment of the system (e.g., whether to mirror services for efficiency reasons).

Analysis in all these layers may be carried out on *i** models using the adequate properties. The architecture prescription model may be analyzed in terms of properties such as security, integrity, etc. In the lower two levels, properties such as performance are likely to be assessed once concrete products and topologies have been chosen. Again, the variability model is important on these lower levels to better model possible adaptations.

## C. Monitoring and Adaptations

In Section I I we have already mentioned MSDS' changing nature. Change is constant in MSDS due to several reasons:

- *Requirements changes:* At any moment, the needs of existing stakeholders may change. Such changes may impact the first layer and will require to partially redo the synthesis and analysis processes.
- *New stakeholder types:* New stakeholders (e.g., new types of customers) may impact the variability analysis carried out during negotiation and propagate to the lower layers.
- *Technology changes:* Today we can observe a rapid appearance of new services and components while at the same time other system elements become obsolete. This may have an impact on higher-levels as new goals may be satisfied with impacts on trade-offs in negotiations.
- *Run-time changes:* Changes may have multiple causes. For instance, a site may provide bad service and thereby degrade the performance of the MSDS as perceived by certain stakeholders. Also, the change may be provoked by some human activity, e.g., dealing with personal data unsafely.

TABLE I
USE OF TECHNIQUES IN THE FRAMEWORK

| | Goal Modelling | Variability Modelling | Negotiation Techniques |
|---|---|---|---|
| Stakeholder Needs | Building stakeholders' SR models<br>Representing stakeholders' dependencies in a SD model<br>Including *is-a* hierarchies | Identifying external variation points<br>Building initial decision table | Identifying stakeholders<br>Eliciting stakeholder' goals<br>Resolving stakeholder' groups conflicts<br>Resolving overall system conflicts |
| Architecture Prescription | Modeling the types of the future system's candidate service and components<br>Decomposing the system into roles | Identifying additional external variation points<br>Refining the decision tables | Evaluating trade-offs of several candidate designs<br>Reconcile information about negotiation model with market information |
| Architecture Solution | Including agents modelling concrete services and components | Identifying internal variation points<br>Refining the decision tables | Ranking possible services and components<br>Attaining agreements on selected services and components |
| Open System | Including agents as instances of former agents | Updating decision tables with selected services and components<br>Selecting a new service or component automatically at runtime | Deciding details on the deployment of the system<br>Signing agreement contracts between stakeholders |

Continuous monitoring is necessary to ensure that requirements of stakeholders remain satisfied after such changes. In our framework we need to combine the ability to recognize both human-triggered and system-triggered changes. For human-triggered changes, we need specific activities in the monitoring process. For system-triggered changes tool support to detect the conflicts and translate them in terms of model elements is required. Equally important, we need to be able to trace from lower models to upper ones to support reasoning and proper reactions. For instance, if the open system level detects that a service fails, traceability allows to identify the affected stakeholders to inform them accordingly.

Reaction and adaptation will be similar to synthesis in the sense that *i** modeling and win-win negotiations play an important part, while keeping the product line scope. For instance, once the stakeholders affected by a service failure are identified, a negotiation may be carried out to assess the criticality of this failure, possible alternatives, etc. As part of this process, alternative services and components that are available may be explored to see how they fulfill requirements and conflict with stakeholder goals. At this level, negotiation may benefit from the use of scenarios [15] which state the change situation and explore the different solutions.

Decision modeling plays an important role to support such adaptations. Decision tables as shown in Section V capture variation points and can be used to inform engineering about possible and meaningful changes that do not violate stakeholder goals. For example, there could be a variation point PaymentService on the Prescription Architecture layer with CheatCard and Securitas as possible services on the Solution Architecture layer.

### D. Summary

Table I summarizes the most important issues about the suitability and relevancy of each used technique. Goal modelling using *i** takes up an important role in the upper layers to establish system requirements expressed in terms of goals. The flexibility and adaptability of *i** permits the modelling of stakeholder needs and high-level architectures. Variability modelling uses the information from *i** to start modelling variation points and alternatives. Our framework keeps this information in order to support traceability of runtime system backward to stakeholder goals and also to support fast adaptability of the system in case requirements change. Negotiation can be used at all layers to resolve conflicts that arise. The importance decreases in lower layers because these are more technical than upper ones and in many cases the possible conflicts could be resolved using defined metrics that minimize the need of negotiation between stakeholders.

## V. Initial Validation

We applied our approach to the travel agency example briefly introduced in Section II. We used the combination of *i** modelling, win-win negotiations, and variability modelling techniques to establish an online platform providing support for travel agencies and their customers. As outlined in Section II this platform is provided by a third actor, a fictitious company called *Travel Services Inc.* We used this example to explore and validate the framework and modelling techniques. Each author was asked to take the role of a stakeholder in the study. We did not involve real-world stakeholders such as representatives from travel agencies so far. We describe how we used the approach on the different levels. We also highlight the highly iterative nature of the different processes and the intertwining of the different layers.

### A. Synthesis and Analysis

*Stakeholder Needs:* As mentioned in Section IV.B, we applied a process for building the initial *i** model prescriptively. Our inspiration was the R*i*SD methodology [13] but with one fundamental change: whilst R*i*SD focuses on the whole system, we focused on the needs of each group of stakeholders in the MSDS separately. In practical terms, this means that once the critical stakeholders for the project were identified, we built several SR models to discover the individual needs of stakeholders. As part of this process the dependencies of each stakeholder on others appeared, giving light to a first version of the SD model. This process was in fact iterative

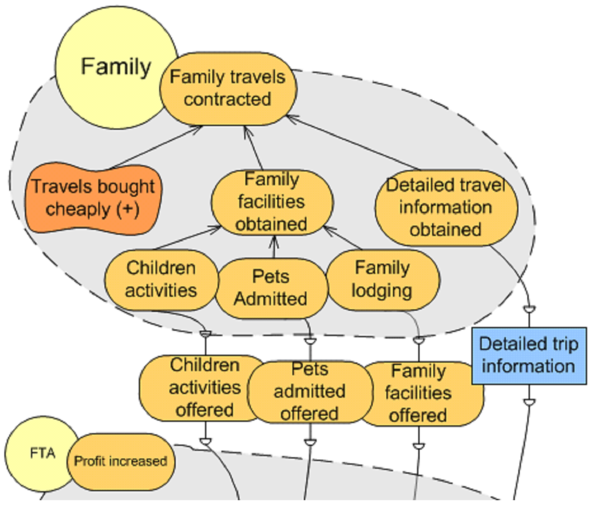
Family
Family travels contracted
Travels bought cheaply (+)
Family facilities obtained
Detailed travel information obtained
Children activities
Pets Admitted
Family lodging
Detailed trip information
Children activities offered
Pets admitted offered
Family facilities offered
FTA
Profit increased

Fig. 2. Partial SR model for the Family actor.

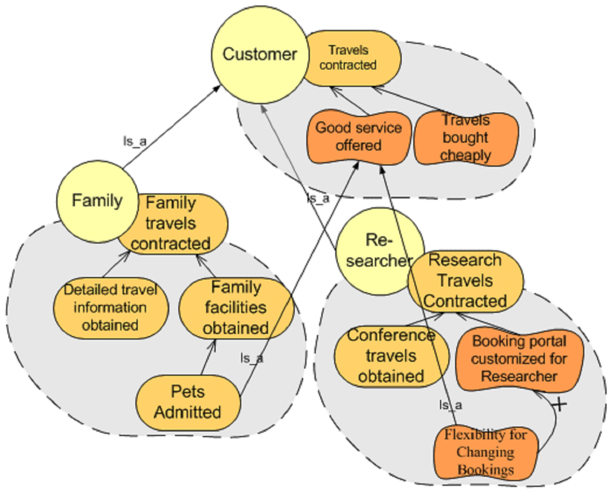
Customer
Travels contracted
Good service offered
Travels bought cheaply
Is_a
Is_a
Family
Family travels contracted
Detailed travel information obtained
Family facilities obtained
Pets Admitted
Is_a
Re-searcher
Research Travels Contracted
Conference travels obtained
Booking portal customized for Researcher
Is_a
Flexibility for Changing Bookings

Fig. 3. Subtyping a stakeholder-related actor using the *is-a* relationship.

and therefore, some of the obtained dependencies could not be linked to existing actors; instead, we needed to identify new success-critical stakeholders and introduce the corresponding actors in the model.

In our example, we identified initially three stakeholder groups, namely TSI, TA, and Customer. For each group, some critical stakeholders were identified to collect system requirements and to invite them to the negotiation. For TSI only one stakeholder was identified to represent the company needs, but for the other two groups two different stakeholders were identified, TA and Customer interested in trips for researchers (UTA and Researcher) and TA and Customer interested in trips for families (FTA and Family).

We started to model the needs of the TSI stakeholder, whose main goal is to contract as many agencies as possible. To attain this goal it is decomposed into subgoals, e.g., providing good service, ensuring privacy, and keeping trustworthiness. After further decomposition we observed that TSI is unable to achieve all its goals itself, in particular the goals referred to obtaining information about travels and contracting travels. This gave rise to a new stakeholder group Service Provider with stakeholders representing Travel Service Providers for looking for travels and Payment Service Providers for paying travels during the booking process. This situation illustrates the iterative nature of our process as mentioned above. Also, these new stakeholders are examples of unavailable stakeholders that cannot participate in a negotiation, so their needs and abilities need to be modelled by a representative (in a typical process, by the requirements engineer driving this first layer). Similarly, we developed SR *i** models for both types of TA's and Customers. Fig. 2 shows part of the specific SR for the Family actor and some dependencies stemming from it.

After building individual SR *i** models, the two Customers were requested to participate in a negotiation to discern if they were able to agree on one SR *i** model representing their needs and desires. Family and Researcher share similar goals,

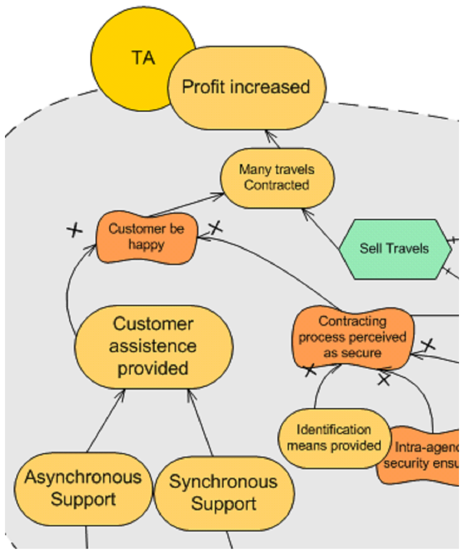

Fig. 4. Example of variation point: Customer Assistance Provided.

but each group has some individual interests. For example both want cheap travels, a good level of service and some privacy provided by the TA. Families are interested in more specific facilities like activities for children and pet admittance, while Researchers are interested in travelling to conferences or flexibility for changing bookings. Therefore stakeholders could not agree on one combined model and the negotiation led us to a model with common goals located in the general actor Customer and specific goals placed in Family and Researcher actors. Family and Researcher are linked to Customer using an *is-a* relationship (see Fig. 3). A similar situation appeared when considering UTA and FTA.

When developing an SR *i** model normally more requirements are gathered than can be realized within limited schedule and budget. So we also asked stakeholders to reflect about the priorities of their goals regarding project success and feasibility. The obtained priorities were used to support negotiation activities in subsequent steps.

As soon as the different stakeholder groups agree on a general model the next important step is to analyze the existing dependencies between actors. Some dependencies were previously identified but they need to be confirmed whilst new ones may appear. Since a dependency from an actor $A$ to an actor $B$ means that $B$ responsible with respect to $A$, negotiation should take care that $B$ is aware of and agrees with that. In our framework, dependencies are modelled with an SD model. In some cases, existing SR *i** models have to be extended because of such dependencies. Key stakeholders can meet face-to-face to discuss issues and the requirements engineer can use these discussions to develop the final SD *i** model. An example of issue in our case was the incompatibility between payment services provider goal "Customer data sold to third parties" and Customer goal "Personal data not sent to third parties". This conflict was identified when analyzing the individual *i** SR models. A negotiation in such a situation may help to find options for resolving the issue, e.g., via alternative service types or services. It might also be revealed, however, that there are non-resolvable conflicts between SR models. A requirements engineer would then point out the problem to stakeholders and ask them to reconsider their goals. In this layer, we determine the initial scope of the product line. Some external variation point candidates were detected applying heuristics like:

- Intentional elements appearing in a general actor which are refined by an *is-a* relationships. For instance, the "Good Service Offered" goal that belongs to Customer is refined into several elements in Family like "Pets Admitted" and into others in Researcher as "Flexibility for Changing Bookings" (see Fig. 3).
- Goals and softgoals that appear as leafs in the model. This means that they admit different ways for being attained. For instance, the goal "Identification Means Provided" can later be implemented using different types of services or components (see Fig. 4).
- Means-end decompositions that, unlike the previous bullet, state the alternatives explicitly. For instance, the "Customer Assistance Provided" goal in TA that admits several ways, e.g. Asynchronous Communication and Synchronous Communication (see Fig. 4).

These and other external variation points can be collected in a decision table including useful information for later monitoring and reaction to changes. For instance, the external variation point "Customer assistance provided" leads to decisions such as "What kind of customer assistance do you need?" or "What should a component support?" (see Table II).

The output of that layer is one SD *i** model, including all the specific SR *i** models, and an initial decision model collecting external variation points.

*Architectural Prescription:* While in the previous layer we have modelled the stakeholder needs constructing the social system model, we evolve this model into a socio-technical system by mapping stakeholders' goals and some social actors

TABLE II

PARTIAL DECISION MODEL FOR THE STAKEHOLDER NEEDS LAYER.

| Decision Variable | Question | Selection Type | Cardinality | Link to *i** element |
|---|---|---|---|---|
| Type of customer assistance | What kind of customer assistance do you need? | Set {synch, asynch} | 1:2 | Customer assistance provided |
| Degree of customer assistance | How many hours per day should the hotline be available? | Value [0..24] | 1 | Customer assistance provided |

onto architectural concepts in this layer. This activity is done in highly interactive workshops due to the need to reconcile information from the stakeholder model with market knowledge. The new *i** model includes on the one hand, the main system-to-be as an actor, i.e., in our case is the Web Portal for TSI (hereafter, TSI-WP). This system is decomposed into subactors that capture the main capabilities embraced by the system, e.g. "Hotel Booking", "Travel Payment", and "Data Management". These subactors are modelled as roles in *i**. On the other hand, we find types of services or components available in the market, some of them coming from the stakeholder model because they already play a part whilst eliciting needs, others appearing here for the first time. For instance, we may identify at this layer that Payment Service Provider may support one or more of different payment types, such as credit card or transfer. The TSI-WP and these services form together the MSDS subject of study. Dependencies among these new actors and social actors appear, they may be new or they may be existing ones that are reallocated to adapt to the new configuration of the system.

Fig. 5 shows an excerpt of the resulting architectural prescription model. We decomposed the TSI-WP and explored different architectural options. Concerning the relationships among system elements, we have found the following typology:

- *Linking stakeholder needs with TSI-WP roles.* These dependencies show how the different parts of the TSI-WP support the attainment of stakeholder needs. Typically, one or more intentional elements from a stakeholder SR model are linked with one or more roles. For instance, the goals "Asynchronous Support" and "Synchronous Support" from TA depend on the "Message Support" and "Call Center Support" roles, respectively. Also, the "Personal Data kept isolated" and "Personal Data not send to 3rd parties" goals are linked with the "Privacy Manager" role.
- *Linking TSI-WP roles with external types of services or components.* These dependencies show how the TSI-WP relies on existing services and components to satisfy the delegated needs. One or more TSI-WP roles are linked with one or more external actors. For instance, the "Travel Payment Support" role is linked with the "Payment Service Provider" actor. Also, the "Travel Finder", "Hotel Booking" and "Flight Ticket" roles rely on the "Travel Service Provider" type of service.

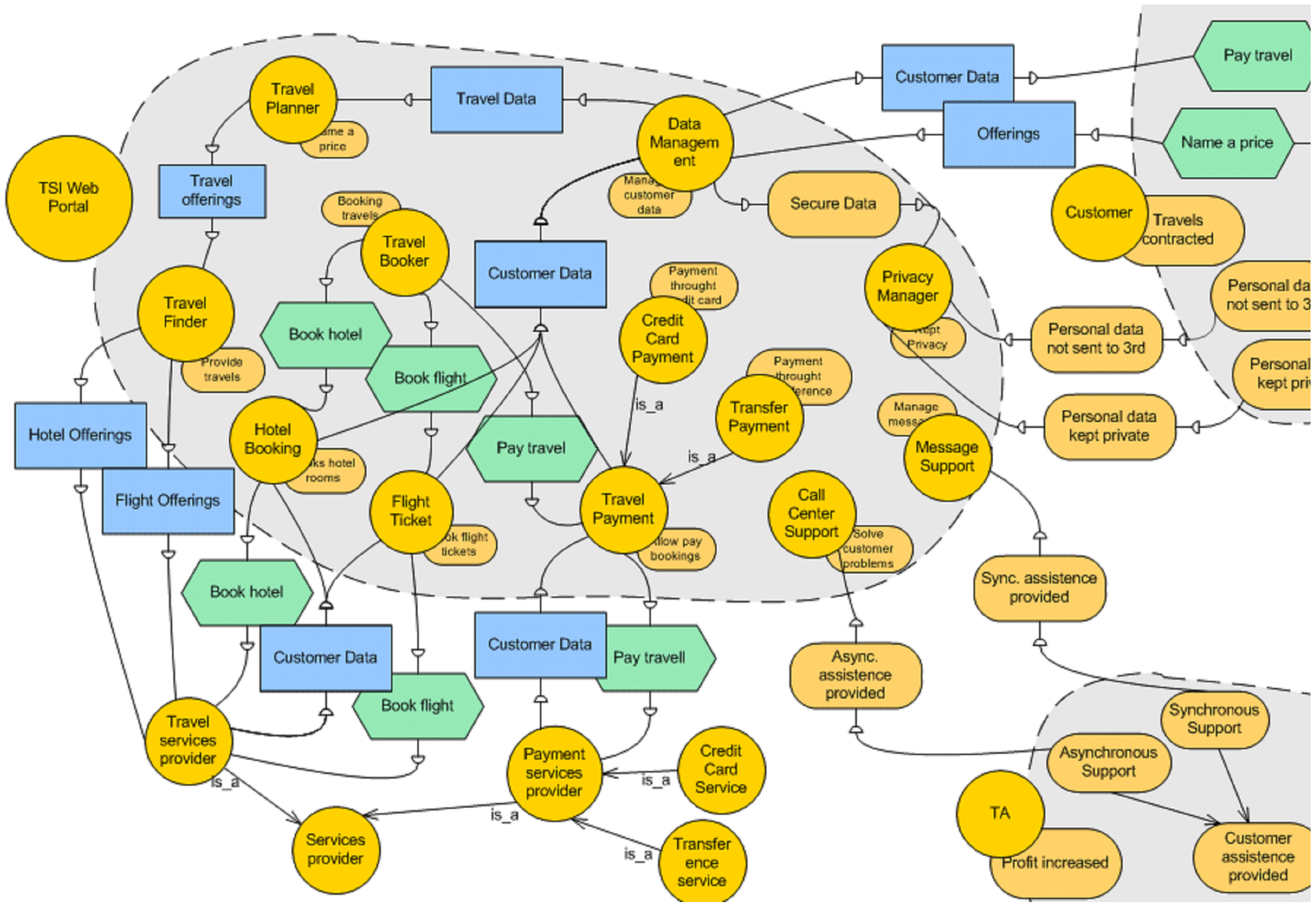

Fig. 5. Excerpt of the Architectural Prescription Model.

A role that is not linked to any other element will be developed in-house. This is the case, e.g., of "Travel Planner". It is worth to mention that once the main roles have been identified, some of them can be covered by different specializations, which is represented again using the *is-a* relationship inside the system actor. In our case, the role "Travel Payment Support" could be covered by two different specializations "Credit Card Payment" and "Transfer Payment" (the link has not been represented in the drawing). These specializations of a role are identified as external variation points of a possible product line and are added to the decision model of this layer. A final remark about the architecture prescription model is the apparent duplicity among roles and external services. This is due to the necessary separation about which capabilities offer the system-to-be (represented by TSI-WP roles) and what services are available in the market (represented by external roles and, in the following layers, agents). Our future work includes the exploration of *i**-based notational techniques allowing to represent this situation in a more compact form.

*Solution Architecture:* At that point we need to negotiate and agree on an actual solution and identify real services and components. At the Solution Architecture we must choose one or more real services or components for each role of the architectural prescription model bound to some service; we remind that some roles may have been designated to be developed in-house. A fundamental part here is service discovery, i.e., discerning which of the many possible candidates better fit to the stated needs. We have defined in [16] a framework that facilitates this selection process considering selection as a problem of matching among goal-based models: the model of the needs to satisfy and the models of the different services and components, which may fulfil or not the goals, tasks, etc. defined in the role of the corresponding service. During this selection, other two aspects that need to be taken into account are negotiation and service composition. Both techniques together help finding solutions for the usual case in which there is no service in the market covering all the relevant needs.

The result of this activity is a list of candidate services and components to be chosen from. For instance, for covering the Credit Card Service Provider we may obtain two services, CheatCard and Securitas. To choose between them, negotiation again may help if representatives of these services and components are available. In any case, from our product line perspective, if more than one service or component could play the service role, these roles become internal variation

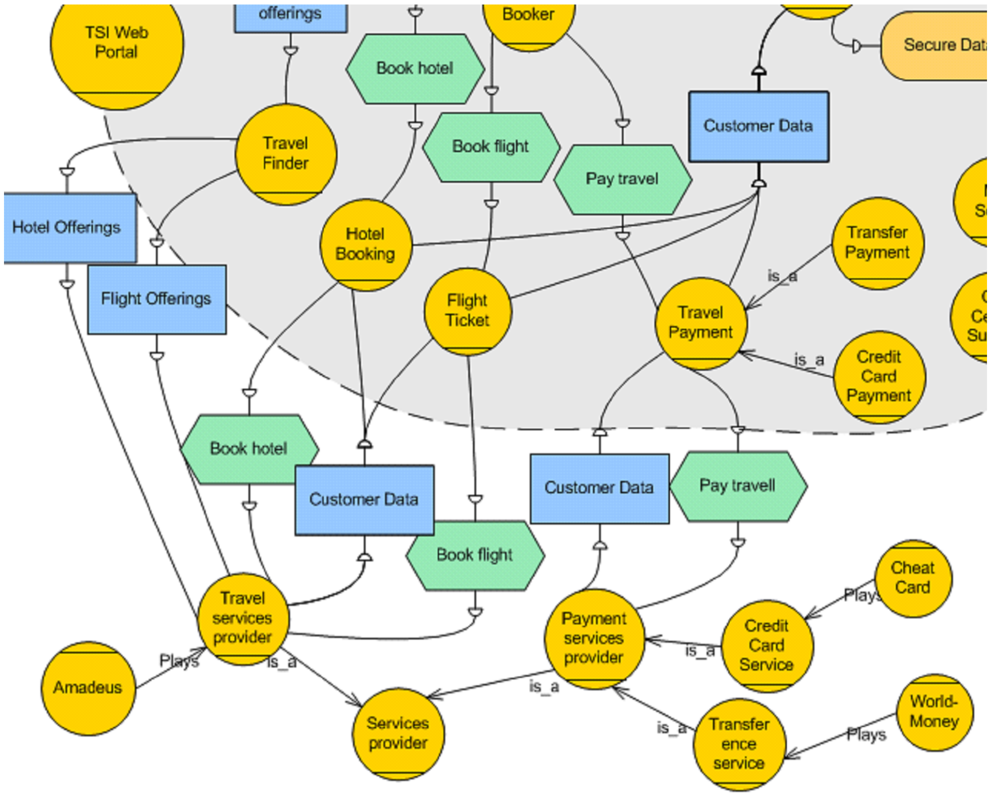

Fig. 6. Excerpt of Solution Architecture Model.

points and will be added to the decision table, as shown in Table III. The information added to the decision table is manifold: make clear the nature of the variation point; state how the services relate to that variation point (i.e., what part of the system is implemented by which service); and show the alternative services available (in our case, CheatCard and Securitas). This information may be used for supporting the monitoring process, because the entries of the table help in identifying triggers for monitoring. Also we may take profit of this information to develop some scenarios for adapting the MSDS in response to these triggers, similar to the one presented in Section II.

TABLE III

PARTIAL DECISION MODEL FOR THE SOLUTION ARCHITECTURE LAYER.

| Decision Variable | Question | Selection Type | Cardinality | Link to *i** element |
|---|---|---|---|---|
| Credit Card Service Provider | What is your preferred service for credit card transactions? | Set {CheatCard, Securitas} | 1 | Credit Card Service Provider |

Once this process ends, the real components selected are included in the model as agents (circles with a segment in the upper part, whilst roles have the segment in the lower part) with an *i** *plays* relationship between them and the roles. Fig. 6 shows an excerpt of the *i** model obtained at this layer. This model does not contain information about the alternatives found because *i** does not offer this feature. Not only the TSI-WP' subactors but also the rest of actors are modeled as roles, because when approaching to the solution space, it becomes more important to distinguish between the different types of actors. The output of that layer is an *i** model of the Architectural Solution and an updated Decision Model.

*Open System:* So far we have developed the model and selected some real services and components for the system, but there is no real system running. At this layer concrete instances of the real services and components modelled as agents in the previous *i** model are used for defining a deployment model. These instances are defined again as agents which, following the *i** guidelines, are instances of the agents obtained in the previous layer; this relationship may be established in the model by using the *i** instance relationship (see Fig. 7) defining the the existing node implementing the Amadeus service.

Once all the layers have been defined, customization for particular types of stakeholders is supported. For example, if a new TA wants to subscribe for TSI services, this can be done with the help of the decision model. The TSI can immediately deliver a model for the customized version of the product to the new TA and in the case that product line engineering has been followed, the customised portal could be effectively delivered with little effort. In some cases the new TA will have special needs which won't be covered by the current product line scope. The TSI and the new TA will then be committed to negotiate the opportunities. After new features or services have

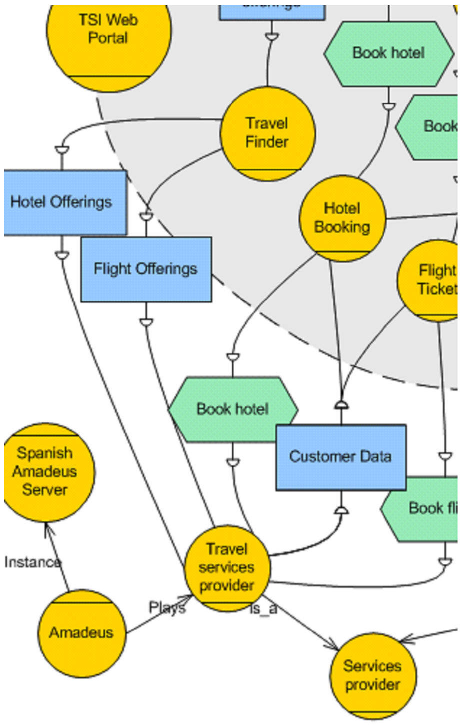

Fig. 7. Excerpt of Open System model.

been identified, they can be incorporated into the model at the upper level and then propagate to the lower ones using the synthesis process. Inviting other TAs can help to support the development, because a new feature can also be interesting for them.

## B. Monitoring and Adaptation

Monitoring and adapting to changes are important characteristics for MSDS as shown in Section II. Many approaches are described in literature for monitoring systems (e.g., [17]). Here we focus on the adaptation aspect and demonstrate how our approach can support changing a system at runtime. Within our framework we distinguish two different ways how this change can emerge: 1) Changes can be "bottom-up", i.e., triggered from the open system layer provoked by factors like new services or components on the market or run-time modifications of existing services or components; 2) Changes can be "top-down" and stem from the Stakeholder Needs layer of the framework. As discussed in section IV.C, new stakeholder or floating requirements of existing stakeholders can provoke changes at this level.

Based on the example presented in Section II we describe a scenario demonstrating how such changes are handled by the different layers of our approach. The first example shows how to react if there is a change on the open system level. Instead of discussing new candidate services with major customers like NorthStar in step 6 of the original example, the variability model provides the solution how to proceed.

1) The software architect of the TSI can recognize prioritized alternative services in the architectural solution layer variability model. In our example the Securitas service, which was already discovered during system synthesis, fulfils all essential stakeholder needs.
2) As the Securitas service is the alternative service with the highest priority the software architect chooses this service to replace the CheatCard service.
3) After the system changes, the software architect uses the *i** system model and the decision model to identify all affected stakeholders. To do so she can use some existing tool support functionalities, e.g. backward traversals as described in [18]. In this case stakeholders have only to be informed that the security issue is resolved and the system is working again as suggested. We are aware that this example is only valid if alternative services can be identified at the time of synthesis. If there are no alternatives available the software architect has to focus on the architectural prescription layer of the framework to resolve the problem.

The second example illustrates how changes on the Stakeholder Needs level can provoke changes of the running system:

1) TransworldTravels TA enters the market and requests TSI to provide an extended payment service supporting world-wide payment solutions.
2) As the need of TransworldTravels might provoke a change on the stakeholder needs level, TSI software architects analyse the high-level *i** goal models.
3) TSI figures out that the goal of TransworldTravel is not included in the *i** goal model so far and that other TAs like NorthStar and the Payment Service Provider (PSP) are affected by this request.
4) According to the framework TSI has to negotiate and reconcile the new need with the other stakeholders. The affected TAs and the Payment Service Provider are invited to discussions. In our case we assume that the other TA's concur that they are not interested in the new service.
5) The Payment Service Provider is willing to provide a more advanced service to satisfy the requested needs as TransworldTravels is willing to pay for this new service.
6) Based on this decision, TSI's software architects have to adapt the *i** and the variability model on all framework levels according to that system change. In this case they have to model variability as TransworldTravels is now using a different payment service as the other TAs. Again, the *i** and variability models support the decision-making process within TSI, as the ways to react to changes are already thought through and explicitly expressed.

## VI. ASSESSMENT OF THE APPROACH

In this section we highlight the main strengths and difficulties of our approach. In the next section we include some further work aimed at solving the identified difficulties.

*The* i* *language:* In our framework, *i** is used in the upper layer to describe stakeholders (including their subtyping relationships) and their needs, to establish how these actors depend one on each other (the *i** SD model), and to record the reasoning behind the decisions taken (the *i** SR model). Using *i** modeling at that layer helps to gather information about the different actors and to make requirements engineers understand the whole system. In the lower three layers, the concepts modeled are different: types of services, services, consumed data, etc. However, and this is a first benefit, the versatility of *i** allows using it in all the layers, providing a uniform and comprehensive framework and facilitating traceability among models using for instance the matching notion defined in [16]. In particular, the use of *i** to generate low level models out of high level ones supports traceability of individual stakeholder goals to real world components of the open system.

A second benefit of using *i** is the possibility to assess properties using metrics. Depending on their context, metrics may be very different in nature, from requirements-oriented (predictability, workability, etc.) to architecture-oriented (security, performance, etc.). Although we have not illustrated this aspect in the paper, we have demonstrated before [2], [14] that it is feasible to define these metrics using *i** elements. Also, having actors in the system allows focusing on their needs independently, which fits with the MSDS characteristic that stakeholders have their own perception of the system.

Finally, there exists currently a large community using *i** which may facilitate knowledge transfer in both directions, i.e. we may port other's work to our MSDS framework whilst making our approach more attractive for this community. Remarkably, there currently exist some tools for editing and reasoning with *i** models which we envision as part of our future tool-support.

At the time being, the most important drawback on the use of *i** is the size and complexity of the resulting models. This has been reported as a main drawback in *i** modeling [19] and MSDS are not an exception. A second difficulty is the lack of expressivity for some needs that are very related to the MSDS domain. We have mentioned the incomplete way in which the *is-a* construct is defined in *i**, but also the absence of a construct to record the discarded alternatives in the solution architecture and other minor issues. We are currently working to ameliorate these two drawbacks.

*Negotiation techniques:* The utility of win-win in the MSDS framework is manifold. The approach emphasizes the involvement of all success-critical stakeholders in a negotiation and supports trade-offs in case of conflicting requirements. Usually stakeholders demand more than can be covered by the project budget. Win-win also entails prioritization techniques to make requirements analysts understand the system's most critical features. It also supports consensus-building techniques by pointing out goals where stakeholders do not agree. By negotiating their needs, stakeholders have the chance to understand the overall system and the needs of other stakeholders. Negotiation is also helpful as a means to identify candidates for variation points [20]. In that paper we also document that win-win is helpful for scoping a product line. An issue that we need to cope with is identify all the stakeholders for a MSDS, Also, for some of the identified stakeholders, it may be hard to involve them in a negotiation process. We need to adapt win-win to overcome these difficulties.

*Variability modeling:* Our framework extends the traditional use of variability modeling. While in conventional product line engineering, variability modeling is essential in system design and product configuration, in the MSDS context we have shifted the emphasis on runtime adaptations. Decisions tables allow reacting to system changes in an appropriate way as they can also be used to define the alternatives in case of system adaptations. Variability modeling also allows dealing with the high volatility of the service market. New services may be tight to the architecture in the identified variability points without impacting much on the running system. Variability techniques are also helpful for coping with stakeholder variability, whilst maintaining a common set of features relevant for all stakeholders.

## VII. CONCLUSIONS

In this paper we have presented a framework for RE in MSDS and reported several critical and important research challenges. We have shown that an integrated approach based on goal-oriented RE modelling, negotiation and variability modelling is promising to deal with these issues. These different individual approaches have been successfully used in building high quality software systems in various domains. Complementing each other, they seem to be beneficial for supporting the development and evolution of MSDS.

The issues summarized in Section II highlight the problems of the MSDS domain and confirm the need for an integrated approach. Such an approach is particularly important for providing systems that can be tailored to customer needs and changed with minimal effort during operation as discussed in this paper. The combination of *i** modelling and variability models provides new possibility to better react to changes and to support architectural design in the system operation phase.

Although preliminary, the application of the approach to design a typical MSDS example highlights its strength and benefits. Appling our approach we were able to identify issues and ideas stimulating further research:

- Combining the three different techniques helps stakeholders to understand the overall system vision. However, it is time consuming to use more than one isolated modelling technique. In our further research we have to explore how to provide more integrated support and guidance for applying the approach.
- At the moment the artefact types of the different approaches are not linked to each other clearly. A more formal meta-model describing the relationship of artefacts will be needed to automate information exchange.
- Identification of possible variation points in the *i** model relies in some not validated rules. In our further research we have to explore how to derive variation points in a more guided manner.
- While *i** works fine on the higher levels of modelling the MSDS we intend to refine some concepts (remarkably,

the *is-a* construct) to support the MSDS domain more explicitly.

- So far we have been mainly using two tools. The REDEND tool [18] supports *i** modelling (cf. the figures in this paper). The DecisionKing tool [21] supports the development and enactment of variability models. These two tools are not integrated to so far and we will be providing exchange mechanisms. The EasyWinWin tool suite [7] provides good support for the elicitation and negotiation aspects of our framework. So far we have not been using the negotiation tools but we intend to integrate selected negotiation capabilities in the future.
- We are also working on defining an evolution of the R*i*SD method for building *i** models [13] adapted to some MSDS specificities.

## ACKNOWLEDGMENTS

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 TIN2004-07461-C02 and SODA FIT-340000-2006-312 (PROFIT programme).

## REFERENCES

[1] R. Hall, "Open modeling in multi-stakeholder distributed systems: requirements engineering for the 21st century," in *Proc. First Workshop on the State of the Art in Automated Software Engineering*, U.C. Irvine, Institute for Software Research, 2002.

[2] X. Franch and N. Maiden, "Modeling component dependencies to inform their selection," in *2nd International Conference on COTS-Based Software Systems*, ser. Lecture Notes in Computer Science, vol. 2580. Springer, 2003.

[3] P. Grünbacher and N. Seyff, "Requirements negotiation," in *Engineering and Managing Software Requirements*, A. Aurum and C. Wohlin, Eds. Springer Verlag, 2005.

[4] E. Yu., "Modeling strategic relationships for process reengineering," Ph.D. dissertation, Univ. Toronto, 1995.

[5] F. Bachmann, M. Goedicke, J. Leite, R. Nord, K. Pohl, B. Ramesh, and A. Vilbig, "A meta-model for representing variability in product family development," 2003.

[6] I. John and K. Schmid, "A customizable approach to full-life cycle variability management," *Elsevier Journal of the Science of Computer Programming, Special Issue on Variability Management*, 2004.

[7] B. Boehm, P. Grünbacher, and R. Briggs, "Developing groupware for requirements negotiation: Lessons learned," *IEEE Software*, vol. 18, no. 3, pp. 46–55, 2001.

[8] P. Grünbacher, F. Stallinger, N. Maiden, and X. Franch, "A negotiation-based framework for requirements engineering in multi-stakeholder distributed systems," in *Workshop on "Requirements Engineering and Open Systems (REOS)" at RE'03*. Monterey, CA: http://www.cs.uoregon.edu/ fickas/REOS/, 2003.

[9] P. Grünbacher, N. Medvicovic, and A. Egyed, "Reconciling software requirements and architectures with intermediate models," *Journal on Software and System Modeling*, vol. 3, no. 3, pp. 235–253, 2004.

[10] A. v. Lamsweerde, "Goal-oriented requirements engineering: A guided tour," in *Proceedings 5th IEEE International Symposium on Requirements Engineering (RE'01), August 27-31*, 2001.

[11] B. Boehm, P. Bose, E. Horowitz, and M. Lee, "Software requirements as negotiated win conditions," in *First International Conference on Requirements Engineering*. Colorado Springs, CO, USA: IEEE Computer Society, 1994, pp. 74–83.

[12] K. Pohl, G. Böckle, and F. van der Linden, *Software Product Line Engineering – Foundations, Principles, and Techniques*. Berlin, Heidelberg, New York: Springer, 2005.

[13] X. Franch., G. Grau, E. Mayol, *et al.*, "Systematic construction of i* strategic dependency models for socio-technical systems." *to appear: International Journal of Software Engineering and Knowledge Engineering (IJSEKE)*, 2007.

[14] X. Franch, "On the quantitative analysis of agent-oriented models," in *Proceedings 18th CAiSE Conference*. Springer LNCS 4001, 2006.

[15] I. Alexander and N. Maiden, *Scenarios, Stories and Use Cases*. John Wiley, 2004.

[16] X. Franch, "On the lightweight use of goal-oriented models for software package selection," in *Proceedings 17th CAiSE Conference*, ser. Lecture Notes in Computer Science, vol. 3250. Springer, 2005.

[17] M. Mikic-Rakic, S. Malek, N. Beckman, and N. Medvidovic, "Improving availability of distributed event-based systems via run-time monitoring and analysis," in *Proceedings of Twin Workshops on Architecting Dependable Systems (WADS 2004), Edinburgh, UK, May 25, 2004 and Florence, Italy, June 30*, 2004.

[18] J. Lockerbie and N. Maiden, "REDEPEND: Extending i* modelling into requirements processes," in *Proc. 14th International Conference on Requirements Engineering (RE'06), Minneapolis, MN*, 2006.

[19] H. Estrada, A. Martínez, O. Pastor, and J. Mylopoulos, "An experimental evaluation of the i* framework in a model-based software generation environment," in *Proceedings 18th CAiSE Conference, Luxembourg*. Springer LNCS 4001, 2006.

[20] M. Noor, R. Rabiser, and P. Grünbacher, "A collaborative approach for reengineering-based product line scoping," in *Proceedings 1st International Workshop on Agile Product Line Engineering (APLE'06), Baltimore, USA*, 2006.

[21] D. Dhungana, "Integrated variability modeling of features and architecture in software product line engineering. doctoral symposium," in *21st IEEE/ACM International Conference on Automated Software Engineering, Tokyo, Japan*. IEEE Computer Society, 2006.