Assumption-Based Risk Identification Method (ARM) in Dynamic Service Provisioning

Alireza Zarghami, Eelco Vriezekolk, Mohammad Zarifi Eslami, Marten van Sinderen, and Roel Wieringa
Department of Electrical Engineering, Mathematics and Computer Science, University of Twente
Enschede, The Netherlands
{a.zarghami,e.vriezekolk,m.zarifi,m.j.vansinderen,r.j.wieringa}@utwente.nl

***Abstract*—In this paper we consider service-oriented applications composed of component services provided by different, economically independent service providers. As in all composite applications, the component services are composed and configured to meet requirements for the composite application. However, in a field experiment of composite service-oriented applications wef found that, although the services as actually delivered by the service providers meet their requirements, there is still a mismatch across service providers due to unstated assumptions, and that this mismatch causes an incorrect composite application to be delivered to end-users. Identifying and analyzing these initially unstated assumptions turns requirements engineering for service-oriented applications into risk analysis.**

In this paper, we describe a field experiment with an experimental service-oriented homecare system, in which unexpected behavior of the system turned up unstated assumptions about the contributing service providers. We then present an assumptions-driven risk identification method that can help identifying these risks, and we show how we applied this method in the second iteration of the field experiment. The method adapts some techniques from problem frame diagrams to identify relevant assumptions on service providers. The method is informal, and takes the "view from nowhere" in that it does not result in a specification of the component services, but for every component service delivers a set of assumptions that the service must satisfy in order to contribute to the overall system requirements. We end the paper with a discussion of generalizability of this method.

***Index Terms*—Risks, Requirements engineering, Composite applications, Dynamic service provisioning, Homecare systems.**

I. Introduction

A. Problem

To reduce development cost in todays' competitive environment businesses use Service-Oriented Architecture (SOA). With SOA, programmers can compose applications by reusing existing services provided by different organizations. A composite application can be built without knowing the internal implementation mechanisms of the services.

A composite service-oriented application composed of component services provided by independent providers, is implemented as a network of actors (including service providers and end-users) that, as a whole, must satisfy application requirements, that in turn are motivated by overall goals.

Each of the contributing service providers makes assumptions about its environment (consisting of the other component service providers, and the end-users) and delivers a service that, if those assumptions were correct, would satisfy the specification of that component service. In practice, these assumptions are mostly unstated, and some of them are incorrect, or at least mutually inconsistent across different service providers.

In a field experiment of composite applications, which will be described later on, these unstated assumptions created unexpected behavior of the applications, which violated the application requirements. The interactions between component services that made incorrect assumptions about their environment had been unforeseen, and were unwanted.

This problem does not simply go away by specifying in detail what each component service must do, and then matching provided services with this specification. The specifications of component services are often not fully available to the programmer creating the composite application.

Component services are subject to many requirements in addition to those that derive from one particular application. This has influenced their internal implementation decisions, their interpretation of our requirements on their service, and the assumptions they implicitly make on their environment (i.e. the other component services and the end-users). This leads to a situation in which the service providers and application programmer all believe, mistakenly, that provided services are required services.

The unpredictability of composing services provided by independent providers increases with dynamic service provisioning. In *dynamic service provisioning*, a composite application can be reconfigured. Dynamicity increases the range of possible application behaviors and therefore also increases the range of possible unexpected interactions between component services.

In this paper we focus on dynamic service provisioning for safety-critical systems, in particular home care systems in which the life and well-being of patients are at risk [1]. The goal of *home care systems* is to allow elderly to live an independent life as long as possible (and to save costs at the same time), by providing them with smart self-care medical aids. It is important for these systems to avoid unexpected behavior and so requirements engineering for these systems contains a risk assessment. In the literature, different definitions of risks have been given in different domains [2], [3], [4]. We will stay close to the dictionary definition of risk by defining it as "*the possibility of loss or disadvantage to*

978-1-4673-5765-4/13/$31.00 © 2013 IEEE

end-users due to composite application behavior". The loss or disadvantage to end-users of an application can happen when that application either acts in a way that is not desired by the end-users, or fails to act when it should have responded to environmental changes.

We will *not* assume that this risk is quantifiable. In home care, care-givers such as nurses are aware of safety risks but cannot quantify them.

If all the assumptions for all possible environmental changes can be stated explicitly during design time, and all provided services can be shown to satisfy the composite application's requirements under these assumptions, there would be no need to do a risk assessment. The first condition is not met in dynamic service provisioning in general, and the second condition is not met because of unstated assumptions and specifications.

B. Objective

In this paper, we propose a method to identify potential risks of using a dynamic service provisioning approach due to incomplete assumptions made, by the application programmer or by independent service providers, on component services.

Our method emphasizes identifying the various requirements on the service providers and how these requirements can affect their assumptions. The method is based on some elements of the Jackson's problem frame technique [5] and on Seater & Jackson's extension of this [6]. However, in contrast to Seater & Jackson's extension, the method is informal and can be applied without any tool support. A second important difference is that we do not use the method to derive a system specification. Our method assumes a "view from nowhere" in which assumptions on component services are derived from application requirements.

C. Related Work

Garlan et al. [7], [8] have identified the problem of incorrect/conflicting assumptions for building a system out of existing subsystems. As one of the solutions to alleviate this problem, they have proposed to make these assumption explicit. In our approach, we aim to make the assumptions on the component services as explicit as possible. To do so, we investigate how and why a service provider might interpret the implicit parts of an assumption differently, and how these different interpretations could lead to a risk.

Knowledge Acquisition in automated specification (KAOS) [9] emphasized on identifying obstacles that prevent the achievement of a goal such as, safety, security, or user-friendliness goals. If these obstacles are unrecognized or underestimated, the requirements on the system and its environment would be inadequate and incomplete. Then the system is exposed to a variety of risks. The obstacle analysis in KAOS is formal and can be applied for any types of goals. While the ARM method is informal and limited to the safety and security risks.

Failure Mode, Effects and Criticality Analysis (FMECA) [10], identifies the risks which would happen if a component fails and prioritizes them based on their severity, frequency of occurrence, and detectability. However, risks of incomplete assumptions are not limited to the failure of one of the component services. In the cases for which our method is intended, component services work properly with respect to their corresponding assumptions, but incomplete or incorrect assumptions causes a risk. This is the core observation that motivated the STPA hazard analysis method [11], [12]. However, the STPA approach is designed for a distributed system with dedicated components and one actor who is responsible for the entire system design, while we have a provisioning platform with no dedicated component services, and we should identify the risks based on incompletely specified services offered by independent service providers.

Argumentation technique combined with the Jackson's problem frame can be used to investigate the security requirements on components of a system and the assumptions behind them[13]. However, ARM emphasizes on identifying the mismatch among several component services due to their unstated assumptions instead of focusing on one component service and its corresponding assumptions.

HAZards and OPerability studies (HAZOP) [14] identifies a set of risks that can arise due to deviation of the system from the intended design specification by a set of parameters and guide words, such as 'more', 'late' and 'no'. The guide words of HAZOP are not directly applicable to identification of risks of incomplete assumptions. Our work is similar to an extended version of HAZOP for programmable electronic systems [15]. However, we tailor risk identification even further to our specific problem domain, as we investigate how requirements of a service provider can affect its corresponding assumptions which are represented as natural-language-like descriptions.

ARM is similar to the RISA method for identifying security requirements in a network of components [16]. However, RISA focuses on security requirements and uses public databases of security vulnerabilities, where we focus on safety and use methods derived from safety engineering. And where RISA extends problem frames with the Toulmin argumentation technique, we use an informal reasoning approach based geared to sharing the risk assessment with stakeholders who are not computer experts.

Human factors such as end-users exceptional behaviors, have been considered for identifying risks [17]. Similarly, we have investigated the risks of incorrect assumptions on end-users in our previous paper [18]. We consider our proposed method as complementary to the existing risk management approaches to identify new risks. Moreover, we do not talk about risks prioritization since it can be done using the existing approaches.

The rest of the paper is structured as follows. In Section II, we define a generic dynamic service provisioning system including its functionalities and actors. In Section III, we describe a homecare system field experiment that showed unexpected behaviors due to unstated assumptions. In Section IV, we describe our assumption-based risk identification

method (ARM). Section V shows how we applied ARM in the second iteration of the field experiment, and the risks that were identified. Finally, in Section VI, we discuss the results and lessons learned from the field experiment.

II. Dynamic Service Provisioning Platform

In our home care platform, application logic can be defined by a service plan. A *service plan* consists of one or more service building blocks and describes the configuration and composition of instances of these service building blocks with respect to run-time circumstances.

A *service building block* (SBB), in turn, defines a set of functionalities in the abstract level that can be implemented by alternative component services. For instance a SBB of medicine dispenser can be implemented by different medicine dispenser devices which might be provided by different vendors. The service plan specifies the behavior of a composite application at runtime by specifying which component service will be selected for a SBBs and how the selected service will be configured and orchestrated [19] .

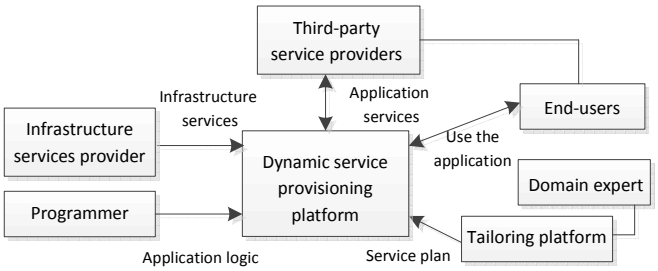

Fig. 1. Actors in a dynamic service provisioning system

Our service provisioning platform is *dynamic*. We define a dynamic service provisioning platform as an adaptive, tailorable and evolvable application service provisioning platform.

An application is *adaptive* if it monitors foreseen changes and reacts to them based on predefined application logic. For example, our application can monitor the blood pressure of an elderly. The application may adapt its behaviour by stopping the current activity and sending an alert to nurses if the blood pressure goes too high/low.

By a *tailorable* platform we mean that an end-user monitors for changes, and adapts the application behavior by reconfiguration. For example, if an elderly develops hearing impairment over time, a nurse can increase the the default volume of audio reminders to remind the elderly to measure his/her blood pressure.

By an *evolvable* provisioning platform we mean that the platform facilitates the manual update of application logic (i.e., reducing required manpower and system resources) to address unforeseen changes. For example, after introducing our system in the field experiment (see Section III), we found that a care-receiver measured his blood pressure much earlier than expected, at 3 AM instead of its scheduled time at 8 AM. Then, if the blood pressure measurement value is too high/low, the application should send the alert immediately at 3 AM. However, the application only starts just before 8 AM. To address this unforeseen change, the programmer of the application should adapt the application logic.

In our dynamic service provisioning system, as illustrated in Fig. 1, a service provisioning platform composes different types of component services: infrastructure services, application services (provided by third-parties) and internal application services (provided by the platform). Infrastructure services are application-scenario independent while application services are used for a specific application scenario.

In addition, there are stakeholders such as programmers, domain experts and end-users. In our case, domain experts are nurses, or more generally care-givers. To decouple the concerns, we assume that there is a separate tailoring platform that takes care of the service plan creation and tailoring, and eventually deploys the service plan to the provisioning platform for the execution.

The end-users use the composite applications running on top of the platform. In our case, they are people needing homecare applications, such as patients or some elderly people, generally called care-receivers in this paper. End-users interact with the applications either directly with the platform through its internal component services or through the 3rd-party component services. They are not aware of this difference.

A domain-expert can be an end-user too if the application is supposed to interact with him/her during its execution. This decentralized architecture is not restricted to homecare systems and we regard it as representative of all decentralized service-oriented systems.

The domain expert can define the behavior of the application by assigning values to the configuration parameters of the service plan. In response, the composite application updates its behaviour based on its service plan. However, addressing unforeseen changes might need new configuration parameters, values or even new orchestrations. This requires IT-knowledge that the domain expert usually does not have. In this case, we assume a programmer who can do arbitrary IT-specific tasks to define or modify the application logic. We assume that the programmer acquires the requirements from the domain expert and accordingly updates the application logic. The programmer must also update the tailoring platform to enable the domain expert to use the new/modified service plan.

III. Our Field Experiment in the Homecare Domain

We conducted a field experiment at Orbis, a care-institution in the Netherlands [1]. This institution consists of residential blocks where elderly can live and receive care services that are provided by professional care-givers. The aim of this institution is to provide round-the-clock services to their care-receivers and at the same time to enable them to live an independent life as much and as long as possible. As part of the U-Care (User-tailored Care services platform) project[2],

[1] http://www.orbisconcern.nl/
[2] http://www.utwente.nl/ewi/ucare/

we developed a prototype of an IT-based homecare service provisioning platform [19].

The field experiment has been done in two iterations of two months each, with one month in between to improve existing applications and to add new applications. The problems that we have faced in the first iteration motivated us to design the ARM risk identification method, which we then applied in the second iteration of the field experiment.

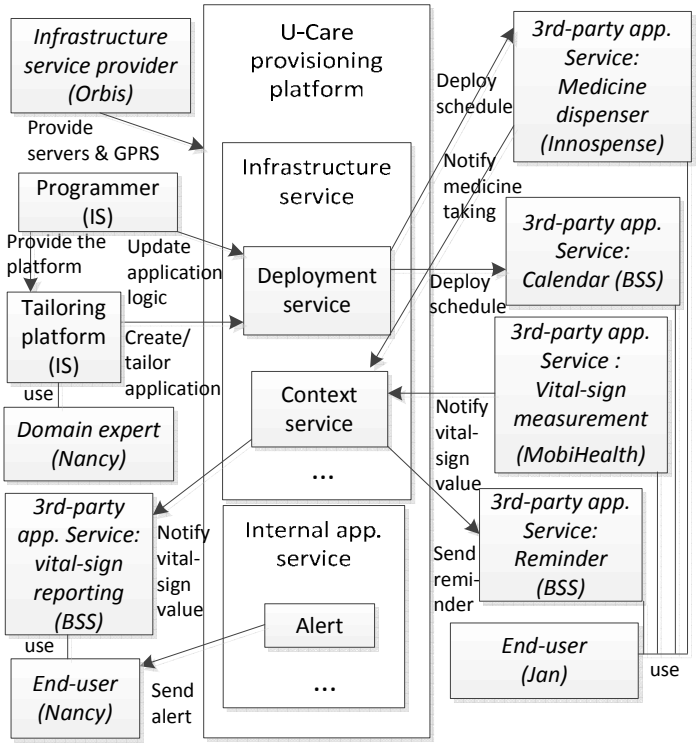

Fig. 2. Part of the architecture of the U-Care system.

Fig. 2 shows some of the infrastructure and application services of the U-Care system. It is an instance of the architecture shown in Fig. 1. More details and justification of this architecture has been given elsewhere [20], [19].

In this section, we explain the vital-sign monitoring (VsM) application of the U-care system to motivate the need for the ARM method. In Section V, we explain the medication monitoring (MdM) application to validate the ARM method in the second iteration of our field experiment.

Fig. 2 shows all the component services which are employed by VsM and MdM applications. There are 3rd-party application services which are used by the VsM application: calendar, reminder, vital-sign measurement and vital-sign reporting and medicine dispenser. The medicine dispenser service is only used by the MdM application and will be described later.

The calendar, reminder and vital-sign reporting services are provided by the Biomedical Signals and Systems (BSS) group of the University of Twente[3]. These services are running on end-user Tablet PCs available to the care-givers and care-receivers.

[3]http://www.utwente.nl/ewi/bss/

The tailoring platform is provided by the Information System (IS) group of the university of Twente[4]. It is running as an application on end-users Tablet PC available to the care-givers.

If the application logic needs to be updated manually to address unforeseen changes, a programmer of IS modifies the application logic and accordingly, updates the tailoring platform. The tailoring platform, through the deployment service of the provisioning platform, deploys a schedule to the calendar service based on the deployed service plan.

The vital-sign measurement service is provided by the MobiHealth[5] company. Care-receivers use a vital-sign measurement device at home, which is connected to a server in MobiHealth. The MobiHealth server forwards vital-sign measurement values (e.g., blood pressure), which it receives from the measurement devices, to the context service.

We also show an internal application of the provisioning platform. The alert service sends an alert to a care-giver's PDA. This internal application service and the infrastructure services are running on top of the infrastructure which are provided by Orbis as the care center in our field experiment.

Orbis also provides communication infrastructure for the 3rd-party application services and the PDA/Tablet PC used by the care-givers and care-receivers.

The U-Care system architecture illustrates some of the problems with the composition of services provided by independent providers. We explain this using a scenario in which Jan, a care-receiver, measures his vital signs at home, and in which Nancy is the care-giver responsible for Jan. Nancy creates the vital sign monitoring (VsM) service plan for Jan. The service plan helps Jan to measure his vital-signs (e.g., weight) on time.

After the service plan deployment, Jan can see his schedule for vital-sign measurement on the calendar service running on his Tablet PC. The VsM application starts based on a the scheduled time, and reminds Jan, possibly several times, to measure his vital-signs such as blood pressure. Vital-sign measurement is actually an interaction with MobiHealth, who notifies the provisioning platform of the measurement. If Jan does not measure it on time, or if his vital-signs are not in the normal range, the application sends an alert to Nancy. To monitor whether a vital-sign measurement is not on time, the U-Care system uses the vital-sign reporting service provided by BSS.

This application was tested in the first iteration of the field experiment. The test revealed several problems, of which the following three are illustrative. Fig. 3 zooms in on the part of the architecture of Fig. 2 that supports the VsM scenario. Each box represents a service which is provided by a service provider mentioned between the parentheses. To explain the problems, the internal implementation components of each service are also shown. Fig. 3 shows that MobiHealth gives a vital-sign measurement device and an smart phone to the care-receiver. The smart phone communicates with the measurement device by Bluetooth and forwards the measurement

[4]http://www.utwente.nl/ewi/is/
[5]http://www.mobihealth.com/

data to a web server. Then the web server of MobiHealth, forwards the data to the context service of Orbis running on an application server (i.e., App server). Orbis stores the data and then forwards the data to the vital-sign reporting service of BSS. This service is running on an application server that communicates with Nancy's Tablet PC.

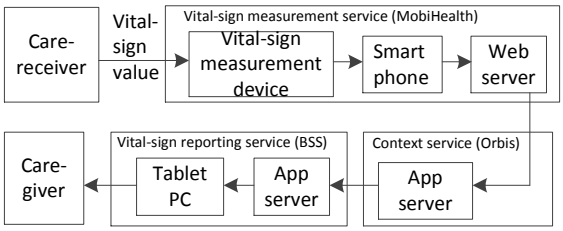

Fig. 3. The part of the U-Care architecture supporting the VsM scenario.

We classify the problems into three kinds, so that we can refer to them later.

- *Service availability problem:* The Orbis application server is down for maintenance every night at 3 a.m. Data received in downtime would be lost. However, unexpectedly, some care-receivers measured their blood pressure around 3 a.m.
- *Data transportation problem:* The Bluetooth connection between the measurement device and the smart phone of the care-receivers resends data when the sender does not receive an acknowledgment on time. This turned out to happen sometimes, and then leads to data duplication not discovered by the rest of the network, i.e. all other services did not recognize that data was duplicated.
- *Data storage problem:* All the data must be hashed before transportation among the component services. Due to hashing algorithm used by MobiHealth, the hashed care-receiver's ID (identification number) exceeds the maximum size of care-receiver's ID defined by BSS. Therefore, Orbis faced an error when it forwards the hashed care-receiver's ID to BSS. Because BSS can not store them and sends an error back to Orbis.

These cases of architectural mismatch can be traced by lack of knowledge of each others internal implementations and operational processes, as well as lack of knowledge about how end-users behave. Some implementation decisions are made for good reasons, others are bad decisions, but none of them are under the control of a central coordinator.

IV. The Proposed Rsk Identification Method

In this section we first give an overall description of the method, using the VsM application as an example. We then provide step-by-step instructions on how to apply the method in general.

A. Overall Description

We assume that our risk identification method is used by the *owner* of the composite application. The owner is the actor specifies the requirements for, and is responsible for the delivery of the composite application.

We do not assume that the owner is in charge of any of the component services used to compose the composite application. Therefore, the owner cannot specify or implement any of these services.

The owner has the "view from nowhere" because he does not take any of the component service providers' point of view.

The owner is responsible for selecting component service providers, and composing them into a composite application that satisfies his composite application requirements. To discharge of its responsibility, the owner must be able to give an argument of the form:

> If service providers $S_1, ..., S_n$ behave like this: $A_1, ..., A_n$, respectively, then the composite application satisfies its requirements.

This is called a frame concern by Jackson [21], but because we are reasoning about components services and not about problem domains, we call it a *contribution argument*. Note that instead of requirements on service provider S_i, the contribution argument makes *assumptions* about the service provider meeting his requirements. We therefore have two types of assumptions: assumptions made by service providers about the environment of their service, and assumptions made by the application owner $(A_1, ..., A_n)$ as part of his contribution argument. The assumptions made by the owner take the form of descriptions of the behavior of the components as inferred or desired by the owner.

Fig. 4 extends the architecture diagram of Fig. 3 with annotations that illustrate parts of the contribution argument. We borrow from problem frame diagrams the notation to represent requirements in a dashed ellipse, connected by dashed lines to the actors who are the subject of the requirements. Fig. 4 contains the requir description ement R1 on the care-giver and care-receiver:

> R1 The care-receiver shall provide vital-sign values to the care-giver according to the service plan for vital-sign measurement. The care-giver shall respond to situations in which the care-receiver does not follow the service plan, and situations where the measurements are outside the safe range as stated in the service plan.

This requirement can be fulfilled if the care-giver is permanently present at the care-receiver's location. The U-care system is introduced to fulfill the requirement even when the care-giver is not permanently present.

The owner of the vital-sign monitoring (VsM) application service has designed an architecture of independent service providers, and translated requirement R1 into a set of assumptions about services provided by these service providers, such that these assumptions jointly satisfy R1. The contribution argument now becomes

> If the service providers in the architecture of Fig. 4 satisfy the assumptions listed in Fig. 4, the composite application satisfies R1.

Because the owner is responsible for more U-Care applications, this architecture may contain components that are not optimal for implementing R1. For example, a simpler

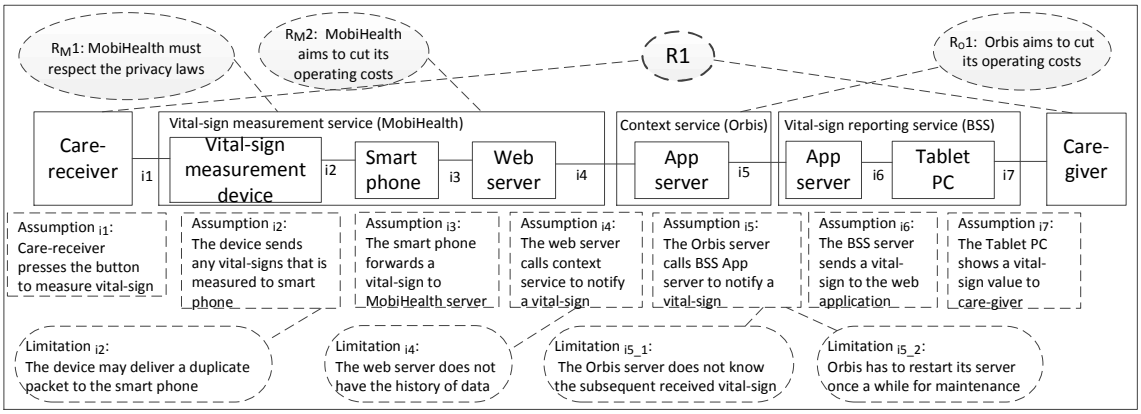

Fig. 4. The network for requirement 1 to be satisfied by vital-sign monitoring (VsM application).

architecture from the standpoint of implementing R1 only, would be to have the vital-sign reporting service be provided by MobiHealth instead of by BSS. However, there are other requirements that call for an independent vital-sign reporting service, which motivates the choice made in Fig. 4.

Given an architecture, assumptions are placed on its components so that the contribution argument can be given. This corresponds to adding "breadcrumbs" (assumptions) in terms of Seater & Jackson's method to transform a requirement into a machine specification [6].

The assumptions on service providers are assumptions on *interfaces* of the service providers. An assumption listed under the box representing the service provider is about its interface represented by the line (e.g., in Fig. 4, assumption$_{i2}$ is on "interface i2" of "vital-sign measurement device"). Assumptions made by the composite application owner are not specifications to be implemented by a service provider. Rather, the owner has to check if the service provider has the capability to satisfy the assumptions made on it. The service provider has a range of possible behaviors that we call its *capabilities*. An actor uses its capabilities to provide services, and usually does so in the context of different composite applications. For example, MobiHealth provides services in many other contexts in addition to the U-care context, and does so with the same services as those contributed in the U-Care system.

Every capability implies a *limitation*: every range of possible behaviors implies that there are behaviors *not* in this range. Fig. 4 lists some limitations of capabilities by a dashed oval box, connected to the assumption that they qualify. The assumption states a capability that the actor is assumed to have, and the limitation is a limitation of the actor's capabilities that causes it to fail to meet the assumption. Note that the actual capabilities of an actor are nowhere stated; they are unknown to the owner of the composite application. Rather, the owner states an assumption about these capabilities. A limitation shown in the diagram states a property of the capability that the actor actually has. It tells us why this actor, with this limitation, fails to meet the assumption. The diagram in Fig. 4 shows four limitations, which are four reasons why this architecture with these service providers, fails to meet requirement R1. In other words, the diagram shows four risks of this particular network.

Mitigating these risks involves a choice between accepting the risks (occasional failure on the system), transferring them (taking out insurance against failure), avoiding them (dropping requirement R1), removing it (replacing a service by one that satisfies the assumptions on it) or compensating it (changing the capability of some service provider so that the limitation stops being a case for failing to meet R1). Risk mitigation falls outside of the scope of this paper.

Fig. 4 also lists the reasons why some actors have some limitations in their capabilities. The diagram shows two requirements on MobiHealth and one on Orbis that are extraneous to the U-Care project. The actors are subject to these requirements independently from whether or not they participate in U-Care.

These requirements are themselves motivated by higher-level goals, namely privacy laws in the homecare domain [22] and business goals (cost-effectiveness). The reasons explain why some actors made some implementation choices, that caused the limitations noted in the Fig. 4. They also make clear that these choices will not be changed just to satisfy a U-Care requirement.

Finally, note that the three requirements listed for MobiHealth and Orbis are context-free in the sense that they refer to one actor only. This explains why these requirements are not necessarily mentioned in discussions with other actors or with the owner of the application.

We next classify the limitations that we have found in the first iteration of our field experiment into three kinds, illustrated earlier by three problems.

- *Service availability limitation:* A service provider has a limitation on its capability to a service always running.
- *Data transportation limitation:* A service provider has a limitation on its data transportation capability, such data transportation assumptions made by the composite application owner are not satisfied. (This assumption could be: no later than delivery, no earlier than delivery, at most once delivery, exactly once delivery etc.)

- *Data storage limitation:* The capability of a service to add, store or delete data may differ from that assumed by the owner of the composite application.

We claim that these are all the limitations that the owner should look for when doing a risk assessment.

B. Step-by-step Description of ARM

Our assumption-based risk assessment method ARM consists of three steps. We assume that the requirements are known and that an architecture for the composite application has been designed in which service providers provide the component services. We now identify relevant assumptions, and search for risks that the requirement will not be satisfied.

1) *Translate the main requirements into assumptions on the interfaces of the network actors.* This is done by constructing a contribution argument in which the assumptions on the interfaces of service providers are listed, needed for ensuring that the composite application satisfies its requirement. In our experience we find this step the most difficult one. In our field experiment, the programmer with the help of the service plan (given by a nurse), and service specification and end-user manual (given by the service providers) writes down these assumptions. How complete and accurate are they? This is a difficult question and its answer is outside of the scope of this paper.
2) *Explore the capabilities and limitations of each service provider to satisfy the assumptions on its interface.* That is, we try to identify capabilities that a service provider should have in order to satisfy the assumptions on its interface. These capabilities are in correspondence with the three types of limitations which we have faced in the first iteration of our field experiment. They are mainly related to how a service provider manipulates data.
   Here are sets of questions that we have found useful to ask about each service provider, in order to assess whether it has the assumed capabilities.
   a) *Service availability questions.* What are the operating systems of the internal components used by the provided service? How (and how often) are they maintained? What are the limitations of employed software/hardware? For instance: do they need to restart for maintenance and if yes, how long does it take and how often? How will a software/hardware component be updated (e.g., on-the-fly)? Do they have a second power supply? How long can they continue on the second power supply? If they use a battery, how long can they go on battery power?
   b) *Data transportation questions.* How are the internal components connected to each other? How reliable are services and the interconnections (e.g., error handling, packet lost, duplicate packet, ..)? What are the limitations of employed networks and networks protocols? For instance: do they guarantee the messages will be transfered? Be sent maximum once? Do they send an acknowledge? What happens if before receiving an acknowledge the connection is lost?
   c) *Data storage questions.* How are the internal data stored? What are the primary/foreign keys? How are these data added/edited/deleted? What are the limitations for storing/internal transportation the data values? For instance: does the provider have the permission to store the data? How long can the provider keep the data? What is the minimum level of encryption for storing and transporting the data? Is data hashing sufficient? Can the provider find out who is the owner of the data? Can the provider make a distinction between data based on their owner?

   Questions like these can be answered by inspecting the contract with a service provider, or, if more detailed information is needed, by inspecting technical documentation provided by a service provider or by interviewing experts inside the service provider organization.
   The answers for these questions might be motivated by the extraneous requirements on the service providers.
3) *Explore in which way each limitation identified in the previous step could cause the service provider fail to meet an assumption.*
   One way to do this is to change keywords in the capability descriptions in a HAZOP-like manner [14] [15]. The assumptions are written in natural language, and so variations in these assumptions can be derived by adding or removing some words (e.g., adverb) in the assumptions.
   For the VsM application, limitation$_{i2}$ and limitation$_{i5_1}$ could be changed into corresponding assumptions as follows:
   - Assumption$_{i2'}$: The device *might* send any vital-signs that is measured to smart phone *more than once*.
   - Assumption$_{i5'}$: The Orbis server *might* call BSS web service to notify a vital-sign *more than once*.

   By knowing these two affected assumptions, we could have identified the *Data transportation problem* of duplication data before the field experiment. Since it is not a serious risk, we could inform care-givers for such application behaviour and asking them to take care of that (transferring the risk.)
   As another example, the limitation$_{i4}$ and limitation$_{i5_2}$ could be changed into corresponding assumptions as follows:
   - Assumption$_{i4'}$: The web server calls context service to notify a vital-sign *and delete the data*.
   - Assumption$_{i5''}$: The Orbis server calls BSS App server to notify a vital-sign *not during the maintenance*.

   By knowing these two affected assumptions, we could have identified the *Service availability problem* and the

risk of *losing vital-sign measurement data* did not occur. It is a serious risk and either Orbis has to remove limitation$_{i5_2}$ or MobiHealth has to remove limitation$_{i4}$.

V. Applying the Risk Identification Method

In the second iteration of our field experiment, we applied the ARM method to the medication monitoring (MdM) application scenario. In this scenario, the care-receiver uses medicines made available by a medicine dispenser provided by the Innospense company[6]. As in the first iteration, we use "Nancy" as short-hand for the care-giver, and "Jan" for the care-receiver.

Nancy creates a service plan, using the tailoring platform, to help Jan to take his medication on time (see Fig. 2).

Similar to the VsM application, if Jan does not take his medication, the MdM application will send a reminder to him, possibly several times (shown on his Tablet PC). If Jan has not taken his medication after a tailored number of reminders, the application will send an alert to Nancy (shown on her PDA).

Fig. 5 zooms in one the part of the architecture of Fig. 2 that supports the MdM application scenario. It shows three sub-networks. In sub-network (1), the medicine dispenser, which is located at Jan's home, communicates with the web portal of Innospense and forwards the timestamp of Jan taking his medication. The web portal then forwards the timestamp to the context service running on an application server of Orbis.

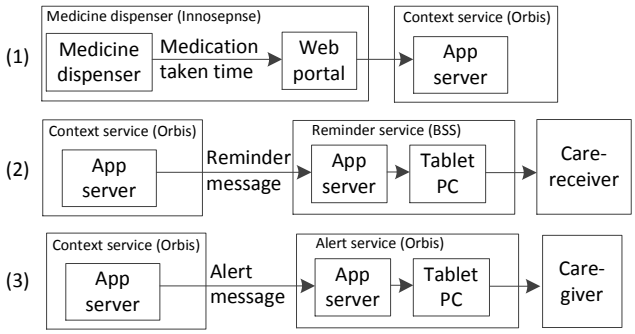

Fig. 5. Part of the U-Care architecture supporting the MdM scenario.

In sub-network (2), the context server checks at the scheduled time for medication if it has recently received the timestamp from Jan's medicine dispenser. If no timestamp has been received, the context service sends a context-aware reminder message (e.g., if Jan is outside his home, the reminder message would be: "please go home and take your medicine") to the reminder service of BSS. This service is running on an application server that communicates with Jan's Tablet PC.

In sub-network (3), if the context service has not received a timestamp from Innospense after having sent several reminder messages, it will send a context-aware alert message (e.g. "Jan is outside his home and has not taken his medicine yet") to the alert service. This service runs on the application server of Orbis that forwards the alert to Nancy's PDA.

[6]http://www.innospense.com

The MdM application must satisfy requirement R2 on the care-giver and care-receiver:

R2 The care-receiver shall take a medicine from the dispenser at the scheduled time according to his service plan. The care-giver shall respond to situations in which the care-receiver does not follow his service plan.

Fig. 6 contains R2 and extends the architecture diagram of Fig. 5. This figure is explained through the steps of the ARM method.

Step 1: *Translate the main requirements into assumptions on the interfaces of the network actors.* Based on the service plan, the programmer as the owner of the MdM application, drew the network of actors for the MdM application according to its service plan. As elaborated in a previous paper [19], the service plan contains a BPMN (Business process modeling language)-like part that shows the combination of the component services.

Then, the programmer wrote down the assumptions on capabilities of each actor. In this case, the end-user manual of Innospense turned out to be useful, since it specified the external behaviour of the medicine dispenser.

As the figure shows, if the service providers satisfy assumptions$_{i8,i9,i10,i11,i12}$, Jan will receive a reminder if he forgot to take his medicine. Besides, if the service providers satisfy assumptions$_{i8,i9,i13,i14,i15}$, Nancy will receive an alert message if Jan does not take his medicine after several reminder messages. Together, these assumptions fulfill requirement R2 even when the care-giver is not permanently present.

Step 2: *Explore the capabilities and limitations of each service provider to satisfy the assumptions on its interface.* As an example, based on interviews with the service providers we identified this requirement for Innospense:

R$_I$1 Innospense aims to reduce its communication costs between devices.

This requirement in turn motivated Innospense to make some implementation decisions. We uncovered these decisions by asking the three *Service availability, Data transportation, and Data storage questions*. Innospense limits the communication between its web portal (located in the company) and the medicine dispenser device (located at home) to some specific time granularity. As such, limitation$_{i8}$ is imposed.

Step 3: *Explore in which way each limitation identified in the previous step could cause the service provider fail to meet an assumption.* To continue our example, we found the following impact of limitation$_{i8}$ on assumptions$_{i8}$:

- Assumption$_{i8'}$:
  The device sends the timestamp *with a maximum delay of m minutes after* the medicine is taken.

Based on the affected assumption, we identified the following scenarios and risks:

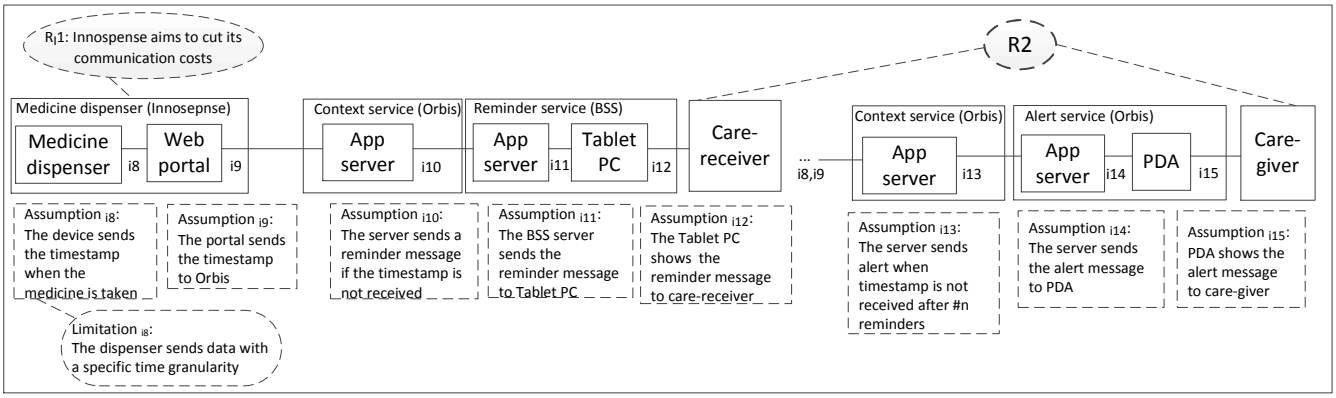

Fig. 6. The network for requirement 2 to be satisfied by medication monitoring (MdM) application.

Risk 1 for assumption$_{i8'}$: we can think of a scenario in which Jan receives a reminder even though he has taken his medication. This happens when the scheduled time for taking the medicine expires before the timestamp is received by the context service, due to the delay introduced by the medicine dispenser.

Risk 2 for assumption$_{i8'}$: we can think of a scenario in which Nancy receives an alert even though Jan has taken his medication. This happens when the scheduled time for sending an alert is reached (after sending the last reminder and) before the timestamp is received by the context service, due to the delay introduced by the medicine dispenser.

There are other assumptions, limitations and risks in addition to those mentioned above, but these have been omitted for brevity. We indeed completed the risk assessment for the MdM application before the second iteration of our field experiment. Therefore, the provisioning of the MdM application was more reliable than the VsM application in the first iteration of our field experiment.

VI. Discussion and Further Work

We have presented and illustrated a method to identify risks in a decentralized system that is composed of component services provided by independent providers. Each component service provider is responsible for the way it provides its service, and the owner of the composite application cannot demand a component service provider to provision a dedicated service. In this situation, the owner of the composite service does not *specify* component services, but must make *assumptions* about them, that reflect agreements made with every component service provider.

The end-users are not computer experts, which limits the kind of assumptions we can make on end-users. At the same time our system is safety-critical because a failure to meet system requirements may harm end-users. Riskiness is increased by dynamicity in the form of adaptability, tailorability and evolvability of the system.

The ARM method starts out with the requirements on a composite application and the component service network used to meet these requirements. It proceeds by listing the assumptions about each component service, checking whether the component service provider really has the capability to meet these assumptions. Then, it explores the impact of any limitations of these capabilities for the ability of each component service provider to meet its assumptions, and hence of the service network to meet the overall requirement. Any risks identified this way are managed by accepting, avoiding, transferring or reducing the risk. Mitigation is typically done in agreement with the stakeholders but is outside the scope of the ARM method.

We have developed and used the method in a field experiment of a homecare system. This test showed that the method can be used and delivers useful results in this case. A possible threat to internal validity could be that the risk identification actually was successful, not because ARM was used, but because we (in particular the first author) were aware of the risks in some other way already. However, this was not the case: The first author was not aware of the risks identified in between the two iterations before he used the ARM method.

An important threat to external validity is that perhaps we (the first author) were able to use this method in this project, but that this is not repeatable in other projects, neither by other people nor by the first author himself. Usability by other people in other projects must be shown in future experiments, by giving this method to other risk assessors in similar projects. The method is arguably repeatable by the first author in similar projects with decentralized service networks because the method explicitly assumes such decentralization and provides means to represent it and reason about it. Moreover, the method might only be feasible for human analysts rather than for automated tools, as the analysis rules and representations are hard to formalize.

We specify the assumptions in a natural-language-like description. Therefore, unlike HAZOP, we are not limited to a set of predefined guide words. Instead, we can more informally investigate the variation of the assumptions by adding/removing any words.

The ARM method can be applied as soon as the design of the composite application has completed. Risks can be

identified before the application is built or the component services have been implemented. ARM can therefore be used in an early stage of application development when mistakes can still be corrected with relatively little cost.

After identifying a risk, we need to reach an agreement among the service providers on how to prevent/mitigate that identified risk. Therefore, one/several actors should pay the cost and to some degree compromise their requirements. To do so, the requirements of actors should be prioritized. Some of them, for instance the privacy law, are compulsory and the service providers must comply. Nevertheless, some others such as the goal of reducing cost can be negotiated. We can even inform the stakeholders, for instance the care-givers, about a risk and if it is acceptable, there would be no need to (re)plan the application behavior. Instead, we plan end-users behavior by making them aware of the risk in advance.

Another challenge that we have found in our pilot study is that after introducing the homecare applications, the end-users change their behavior because of the new possibilities. For instance, Jan has not to wait until Nancy comes to his apartment to measure his blood pressure. Therefore, he often measures his blood pressure earlier than scheduled time and goes out to meet his friends. Predicting these type of end-user behavior as *what if* scenarios is not a straightforward task at the design time.

ACKNOWLEDGEMENTS

This work is part of the IOP GenCom U-Care project(http://ucare.ewi.utwente.nl) which is sponsored by the Dutch Ministry of Economic Affairs under contract IGC0816.

REFERENCES

[1] A. Lang, N. Edwards, and A. Fleiszer, "Safety in home care: a broadened perspective of patient safety," *Int. Journal for Quality in Health Care*, vol. 20, pp. 130–135, 2008.

[2] UN-ISDR, "Terminology on Disaster Risk Reduction," Geneva, 2009.

[3] ISO/IEC, "Information technology, Security techniques, Guidelines for the management of IT Security: Concepts and models," Intl. Std. 13335-1, 2004.

[4] J. Duffus, S. Brown, and N. Fernicola, "Glossary for chemists of terms used in toxicology," *International Union of Pure and Applied Chemistry*, vol. 65, pp. 2003–2122, 1993.

[5] M. Jackson, *Software Requirements and Specifications: A lexicon of practice, principles and prejudices*. Addison-Wesley, 1995.

[6] R. Seater, D. Jackson, and R. Gheyi, "Requirement progression in problem frames: deriving specifications from requirements," *Requirements Engineering*, vol. 12, no. 2, pp. 77–102, 2007.

[7] D. Garlan, R. Allen, and J. Ockerbloom, "Architectural mismatch: Why reuse is so hard," *IEEE Software*, vol. 12, no. 6, pp. 17–26, November-December 1995.

[8] D. Garlan and R. Allen and J. Ockerbloom, "Architectural mismatch: Why reuse is *still* so hard," *IEEE Software*, vol. 26, no. 4, pp. 66–69, July-August 2009.

[9] A. Van Lamsweerde, "Goal-oriented requirements engineering: A guided tour," in *Requirements Engineering, 2001. Proceedings. Fifth IEEE International Symposium on*. IEEE, 2001, pp. 249–262.

[10] J. Bowles and C. Peláez, "Fuzzy logic prioritization of failures in a system failure mode, effects and criticality analysis," *Reliability Engineering & System Safety*, vol. 50, no. 2, pp. 203–213, 1995.

[11] S. Pereira, G. Lee, and J. Howard, "A system-theoretic hazard analysis methodology for a non-advocate safety assessment of the ballistic missile defense system," DTIC Document, Tech. Rep., 2006.

[12] T. Ishimatsu, N. Leveson, J. Thomas, M. Katahira, Y. Miyamoto, and H. Nakao, "Modeling and hazard analysis using stpa," in *Conference of the International Association for the Advancement of Space Safety, Huntsville, Alabama*, 2010.

[13] C. B. Haley, J. D. Moffett, R. Laney, and B. Nuseibeh, "Arguing security: validating security requirements using structured argumentation," in *Third Symposium on Requirements Engineering for Information Security (SREIS'05) held in conjunction with the 13th International Requirements Engineering Conference (RE'05)*, 2005. [Online]. Available: http://oro.open.ac.uk/2488/

[14] R. Kennedy and B. Kirwan, "Development of a Hazard and Operability-based method for identifying safety management vulnerabilities in high risk systems," *Safety Science*, vol. 30, no. 3, pp. 249–274, Dec. 1998. [Online]. Available: http://dx.doi.org/10.1016/S0925-7535(98)00025-3

[15] F. Redmill, M. Chudleigh, and J. Catmur, "Principles underlying a guideline for applying HAZOP to programmable electronic systems," *Reliability Engineering & System Safety*, vol. 55, no. 3, pp. 283 – 293, 1997.

[16] V. Nunes Leal Franqueira, T. T. Tun, Y. Yu, R. J. Wieringa, and B. Nuseibeh, "Risk and argument: A risk-based argumentation method for practical security," in *Proceedings of the 19th IEEE International Requirements Engineering Conference, Trento, Italy*. USA: IEEE Computer Society, June 2011, pp. 239–248.

[17] E. Hollnagel, *Human reliability analysis: Context and control*. Academic Press London, 1993.

[18] M. Zarifi Eslami, B. Sapkota, A. Zarghami, E. Vriezekolk, M. van Sinderen, and R. Wieringa, "Risk Identification of Tailorable Context-aware Systems: a Case Study and Lessons Learned," in *Proceedings of the CAiSE'12 Forum at the 24th International Conference on Advanced Information Systems Engineering (CAiSE)*, ser. CEUR Workshop Proceedings, vol. 855. CEUR-WS.org, 2012, pp. 40–49.

[19] A. Zarghami, M. Zarifi Eslami, B. Sapkota, and M. van Sinderen, "Dynamic Homecare Service Provisioning Architecture," in *9th Int. Conf. on Service-Oriented Computing and Applications (SOCA)*, 2011, pp. 292–299.

[20] A. Zarghami, B. Sapkota, M. Zarifi Eslami, and M. van Sinderen, "Decision as a Service: Separating Decision-making from Application Process Logic," in *The 16h IEEE Int.l Enterprise Distributed Object Computing Conference (EDOC)*. IEEE, 2012, pp. 103–112.

[21] M. Jackson, *Problem Frames: Analysing and Structuring Software Development Problems*. Addison-Wesley, 2000.

[22] M. Chan, D. Estve, C. Escriba, and E. Campo, "A review of smart homespresent state and future challenges," *Computer Methods and Programs in Biomedicine*, vol. 91, no. 1, pp. 55–81, July 2008.