Combining Goal-Oriented and Problem-Oriented Requirements Engineering Methods

Kristian Beckers, Stephan Fassbender, Maritta Heisel, Federica Paci

Combining Goal-oriented and Problem-oriented Requirements Engineering Methods *

Kristian Beckers[1], Stephan Faßbender[1], Maritta Heisel[1], Federica Paci[2]

[1] paluno - The Ruhr Institute for Software Technology – University of Duisburg-Essen
{firstname.lastname}@paluno.uni-due.de
[2] Department and Information Engineering and Computer Science, University Trento
{firstname.lastname}@unitn.it

Abstract. Several requirements engineering methods exist that differ in their abstraction level and in their view on the system-to-be. Two fundamentally different classes of requirements engineering methods are goal- and problem-based methods. Goal-based methods analyze the goals of stakeholders towards the system-to-be. Problem-based methods focus on decomposing the development problem into simple sub-problems. Goal-based methods use a higher abstraction level that consider only the parts of a system that are relevant for a goal and provide the means to analyze and solve goal conflicts. Problem-based methods use a lower abstraction level that describes the entire system-to-be. A combination of these methods enables a seamless software development, which considers stakeholders' goals and a comprehensive view on the system-to-be at the requirements level. We propose a requirements engineering method that combines the goal-based method SI* and the problem-based method Problem Frames. We propose to analyze the issues between different goals of stakeholders first using the SI* method. Our method provides the means to use the resulting SI* models as input for the problem frame method. These Problem Frame models can be refined into architectures using existing research. Thus, we provide a combined requirements engineering method that considers all stakeholder views and provides a detailed system specification. We illustrate our method using an E-Health example.

Keywords: requirements engineering, SI*, Problem Frames

1 Introduction

Eliciting and analyzing requirements of a system is important, because it is hard to built the right software if you do not know what right is. Several methods exist that support the eliciting and analyzing requirements. Fabian et al. [8] present a survey of security requirements engineering (SRE) methods and one of their finding is that the integration of different SRE is worthwhile. The reason is that these all produce different artifacts and support different views and abstraction levels. Hence, a combination of these results can improve the quality of the requirements.

* This research was partially supported by the EU project Network of Excellence on Engineering Secure Future Internet Software Services and Systems (NESSoS, ICT-2009.1.4 Trustworthy ICT, Grant No. 256980).

We propose to combine goal-based and problem-based requirements engineering methods. We use the SI* modeling language [18] as an example for a goal-based method and problem frames [12] as an example for a problem-based requirements engineering method.

The goal-based methods consider the views towards the system-to-be via its stakeholders' goals. The software to be developed is considered only as needed to achieve the goals. For example, on the one hand, a hospitals wants to provide treatment for a patient and requires monitoring data of the patient to administer the treatment. In this case only the monitoring data is modeled. This, however, is sufficient to analyze possible goal conflicts. On the other hand, a researcher wants to use this monitoring data to conduct a medical study. This can be in violation against the privacy goals of the patient. Hence, goal-based methods provide the means to detect and resolve these conflicts. However, the gap to a design description of the overall system is significant, because the system-to-be is only modelled in some artifacts that have a relation to a goal.

The problem-based methods follow a fundamentally different approach. These methods focus on the problem that shall be solved by the system-to-be. Problem-based methods model the environment and the system-to-be, called *machine*, in it. They describe requirements as an effect the system-to-be has on the environment. Refinements allow one to split the problem into sub-problems. Problem-based methods provide a description of the machine and the interaction with its environment. The sub-problems can be further refined into a design description and a system architecture, e.g., via the *ADIT* method [6, 9]. Hence, the abstraction level of problem-based methods is lower than the abstraction level of goal-based methods.

Problem-based methods do not consider the views of all stakeholders and do not consider goals. Thus, they cannot detect or resolve goal-based conflicts. We propose to combine SI* with problem frames in order to create a method that considers all stakeholder views and can resolve goal-based conflicts, while also creating a description of the machine to be built and a structured way to create an architecture.

The rest of the paper is organized as follows. In the next section we discuss related work. In Section 3, we introduce the problem frame method and the SI* modeling language. Section 4 presents our method for seamless software development from goals to software architectures. In Section 5, we apply our method to the E-Health example. Section 6 discusses our results and Section 7 concludes the paper.

2 Related work

Massacci et al. [17] have investigated the relations between Secure Tropos, Problem Frames and general security concepts e.g. assets. They have proposed a unified ontology to reach a shared understanding of the domain of security requirements, and also take advantage of multiple techniques to model and analyze security requirements. The ontology amalgamates the security ontologies of SI* [18] and Problem Frames [12], and accounts for rather nebulous security concepts, such as those of vulnerability and threat. The method differs from our own, because the authors do not investigate how

to combine these methods but just explore the relationship between SI* and Problem Frames concepts.

Fabian et al. [8] propose a conceptual framework for security requirements engineering. The authors also define a common terminology for the framework. The purpose of the framework is to compare security requirements engineering methods. For example, goals are one concept in the framework, and the authors investigate if methods support this concept or not and if the concept is used under a different name. Our method uses the idea of this work that requirements are a refinement of goals and that the SI* method uses a higher abstraction level than Problem Frames. However, the authors do not show a method that actually combines methods.

Liu and Jin [15] propose relations between SI* and Problem Frames. The authors propose to introduce a domain actor and domain constraints into SI* in order to link SI* and problem frame models. This method differs from our own, because we propose to use the methods in sequence, first SI* than Problem Frames. The authors use both methods at the same time.

Supakkul and Chung [19] introduce the concept of a soft-goal from SI* into the Problem Frame method. This method provides support for modeling stakeholder goals in Problem Diagrams. This method differs from our own, because we try to use both methods and not to integrate one into the other. The advantage is that a development is analysed separately on different abstraction layers.

Classen et al. [5] propose to integrate feature diagrams and the Problem Frame method. The authors propose to use these methods in sequence. The Problem Frame method is used first for requirements elicitation and analysis. Feature diagrams contain the solutions for the elicited requirements. The authors also do not change the existing methods and integrate problem frames with another existing method. This method can complement our own.

Letier and van Lamsweerde [14] propose a method to construct software specification from high-level goals. The method uses a formal patterns that guide the operationalization. The method differs from our own, because it results in a specification of pieces of the software. Our work results in the specification of the entire architecture.

3 Background

We introduce the notations we combine in this work in this section. We explain the goal-based notation SI* in Sect. 3.1 and the problem-based notation Problem Frames in Sect. 3.2.

3.1 SI*

The SI* modeling language has been proposed to capture security and functional requirements of socio-technical systems. SI* is founded on the concepts of *agent*, *role*, *goal*, *task*, *resource*. An agent is an active entity with concrete manifestations and is used to model humans as well as software agents and organizations. A role is the abstract characterization of the behavior of an active entity within some context. They are graphically represented as circles. Assignments of agents to roles are described by the

play relation. [3] A goal is a state of affairs whose realization is desired by some actor (objective), can be realized by some (possibly different) actor (capability), or should be authorized by some (possibly different) actor (entitlement). Entitlements, capabilities and objectives of actors are modeled through relations between an actor and a goal: *own* indicates that an actor has full authority concerning ac- cess and disposition over his entitlement; *provide* indicates that an actor has the capabilities to achieve the goal; and *request* indicates that an actor intends to achieve the goal. A task specifies the procedure used to achieve goals. A resource represents a physical or an informational entity without intentionality. A resource can be consumed or produced by a task. In the graphical representation, goals,tasks and resources are respectively represented as ovals, hexagons, and rectangles. Own, provide, and request are represented with edges between an actor and a goal labeled by **O**, **P**, and **R**, respectively. Goals and tasks of the same actor or of different actors are often related to one another in many ways. AND/OR decomposition combines AND and OR refinements of a root goal into subgoals. However, neither such goals might be under the control of the actor nor the actor may have the capabilities to achieve them. *Contribution* relations are used when the relation between goals is not the consequence of a deliberative planning but rather results from side-effects. The impact can be positive or negative and is graphically represented as edges labeled with + and -, respectively. Finally, tasks are linked to the goals that they intend to achieve using *means-end* relations. The relations between actors within the system are captured by the notions of *delegation* and *trust*. Assignment of responsibilities among actors can be made by *execution dependency* (when an actor depends on another actor for the achievement of a goal) or *permission delegation* (when an actor authorizes another actor to achieve the goal). Usually, an actor prefers to appoint actors that are expected to achieve assigned duties and not misuse granted permissions. SI* adopts the notions of *trust of execution* and *trust of permission* to model such expectations. In the graphical representation, permission delegations are represented with edges labeled by **Dp** and execution dependencies with edges labeled by **De**. Finally, trust of permission relations are represented with edges labeled by **Tp** and trust of execution relations with edges labeled by **Te**.

3.2 Problem Frames

Problem frames are a means to describe software development problems. They were proposed by Jackson [12], who describes them as follows: *"A problem frame is a kind of pattern. It defines an intuitively identifiable problem class in terms of its context and the characteristics of its domains, interfaces and requirement."*. It is described by a *frame diagram*, which consists of domains, interfaces between them, and a requirement. We describe problem frames using class diagrams extended by stereotypes as proposed by Hatebur and Heisel [11]. All elements of a problem frame diagram act as placeholders, which must be instantiated to represent concrete problems. Doing so, one obtains a problem description that belongs to a specific class of problems.

[3] For the sake of simplicity, in the remainder of the paper we use the term *actor* to indicate agents and roles when it is not necessary to distinguish them.

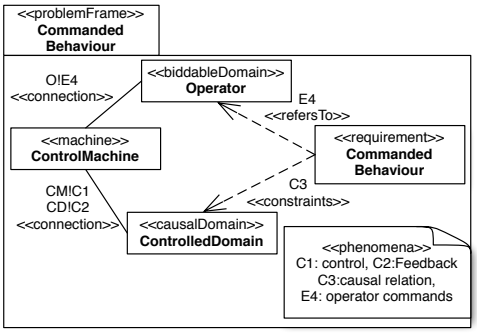


Fig. 1: *Commanded Behaviour* problem frame

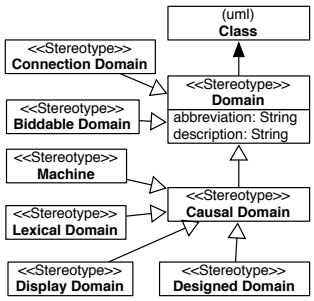


Fig. 2: Inheritance structure of different domain types

Figure 1 shows an example of a problem frame. The class with the stereotype machine represents the thing to be developed (e.g., the software). The classes with some domain stereotypes, e.g., causalDomain or biddableDomain represent *problem domains* that already exist in the application environment. Jackson distinguishes the domain types *causal domains* that comply with some physical laws, *lexical domains* that are data representations, and *biddable domains* that are usually people. We use the formal meta model [11] shown in Fig. 2 to annotate domains with their corresponding stereotype.

Domains are connected by interfaces consisting of shared phenomena. Shared phenomena may be events, operation calls, messages, and the like. They are observable by at least two domains, but controlled by only one domain, as indicated by an exclamation mark. For example, in Fig. 1 the notation *O!E4* means that the phenomena in the set *E4* are controlled by the domain ***Operator***. These interfaces are represented as associations, and the name of the associations contain the phenomena and the domains controlling the phenomena.

In Fig. 1, the ***ControlledDomain*** domain is constrained and the ***Operator*** is referred, because the ***ControlMachine*** has the role to change the ***ControlledDomain*** on behalf of the ***Operator***'s commands for achieving the required ***Commanded Behaviour***. These relationships are modeled using dependencies that are annotated with the corresponding stereotypes.

Problem frames support developers in analyzing problems to be solved. They show what domains have to be considered, and what knowledge must be described and reasoned about when analyzing the problem in depth. Other basic problem frames besides the commanded behavior frame shown in Fig. 1 are *required behaviour*, *simple workpieces*, *information display*, and *transformation* [12].

Software development with problem frames proceeds as follows: first, the environment in which the machine will operate is represented by a *context diagram*. Like a frame diagram, a context diagram consists of domains and interfaces. However, a context diagram contains no requirements. Then, the problem is decomposed into subproblems. If ever possible, the decomposition is done in such a way that the subproblems fit to given problem frames. To fit a sub-problem to a problem frame, one

must instantiate its frame diagram, i.e., provide instances for its domains, phenomena, and interfaces. The instantiated frame diagram is called a *problem diagram*.

Since the requirements refer to the *environment* in which the machine must operate, the next step consists in deriving a *specification* for the machine (see [13] for details). The specification describes the machine and is the starting point for its construction.

Problem frames are an appropriate means to analyze not only functional, but also dependability and other quality requirements [10, 1].

The UML4PF framework provides tool support for this method. A more detailed description can be found in [7].

4 A Method for Goal- and Problem-based Software Engineering

In this section we present our method for combining goal- and problem-based software engineering in Sect. 4.1. The method uses a mapping from the SI* notation to the Problem Frame notation, which is shown in Sect. 4.2. Section 4.3 presents some consistency checks for our method.

4.1 Method

We propose the following method that integrates the SI* and the Problem Frames method, depicted in Fig. 3.

1. **SI* model instantiation**. The aim of this step is to draw an SI* model that captures the system's stakeholders goals. This means identifying the actors and their goals of the system-to-be. The goals are analysed next and all resources or tasks that are a means to to fulfil them are included in the model, as well. It is also possible that one stakeholder requires the resource of another to fulfil his/her goal. In this case, we need to add trust relationships, so that one stakeholder is entrusted with the usage of the resource of another (see Sect. 3).
2. **Goal conflict analysis**. The second step of our method conducts a conflict analysis on the SI* models and resolves these conflicts. We analyse the impacts one goal has on another. The first question, has a goal any impact on another. If this is the case, we have to analyse if one goal contributes to fulfil another or if it is an obstacle for fulfilling it. The resulting relation is included into the SI* model as described

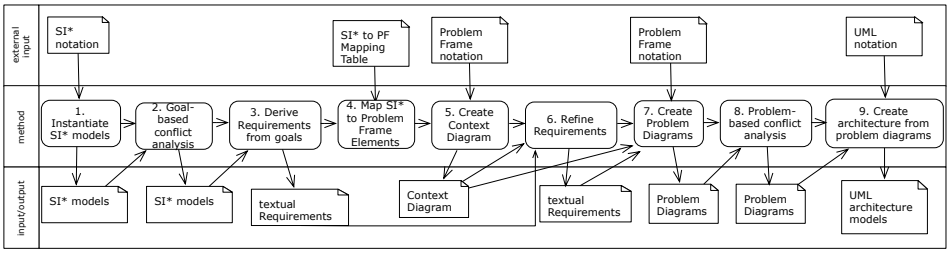


Fig. 3: A method for the integration of SI* and Problem Frames

in Sect. 3. It results in an evolved SI* models, which contain the information how goals relate to one another.

3. **Derive requirements from goals**. The third step demands to write down requirements, which are based upon the goals in the SI* models. We analyse each goal of every stakeholder and formulate a text that consider all the information in the SI* models relevant for that particular goal. For example, we should name the stakeholder the goal belongs to in the text, the resources and tasks required to fulfil it and further relations to other stakeholders, e.g., trust relations.
4. **Mapping SI* to Problem Frames**. This step consists of the mapping of the elements in the SI* models to elements of the problem frame notations. We use Tab. 1 to guide the mapping. The table will be explained in the following subsection. The table states what elements of problems to consider, when mapping an SI* element. This requires in several cases decision by a human expert, because the table does not present a one to one mapping, but offers choices of problem frame notation for most of the SI* elements. It is critical for the requirements engineer to understand the meaning behind every instantiated SI* element in order to execute the mapping. This shall also stimulate a discussion between the customer and the requirements that helps understanding the system-to-be better and validates the instantiated SI*. This validation is based on the knowledge of the customer and is not to be confused with the validation executed by the requirements engineer alone, which is described in Sect. 4.3.
5. **Create context diagram**. We use the mapping of the previous step to create our fist diagram using the problem frame notation. This diagram is our context diagram (see Sect. 3), which describes the context of the development problem. While the SI* models model the actors and their goals, and resources were shown only if they can fulfil a goal. Problem frames is based on the machine to be build. Hence, we have to create the machine domain first and check if it already exist in the set of domains resulting from the mapping in the previous step. If not we have to add it to the context diagram. Next, we add the domains, which whom the machine domain has a relation and exchanges phenomena. We check the set of domains resulting from the mapping and resting phenomena. We include domains and phenomena if these are missing. We complete the context diagram if the entire environment of the machine has been considered and added to the context diagram.
6. **Requirements refinement**. We consider the requirements specified earlier from the goals in the SI* models and refine these into requirements related to the context diagram. These requirements refer to the domains and phenomena in the context diagram. While on the abstraction level of the requirements derived from goals, we might state that a some data should be kept confidential. The same statement would have to be refined. For example, the data has to be kept confidential within a certain database, represented as a causal domain. We have to refine each requirement and check carefully if further requirements need to be added. This can be the case, because the context diagram might contain elements and relations that were not present in the SI* models.
7. **Create problem diagrams**. We create at least one problem diagram for each requirement defined in the previous step. The problem diagrams contain all elements the requirements refer to from the context diagram. It is also possible to add fur-

Table 1: Mapping of Concepts from SI* models to Problem Frames

SI* Element	**PF Element**
Role or Actor	Biddable Domain or Causal Domain
Goal	textual Requirement
Task	basis for at least one Phenomenon
Resource	Lexical Domain or Causal Domain
Means end(Goal, Task)	*Actor!Task* shared between *Machine* and *Actor* Biddable Domain
Means end(Goal, Resource)	*Resource* Lexical Domain constrained by *Requirement* which refines *Goal*
Means end(Task, Resource)	*Actor!Task* shared between *Actor* Biddable Domain and Machine of the problem diagram of the related Requirement. Machine!Task between Machine and the Lexical or Causal Domain representing the *Resource*.
De(Depender, Dependee, Dependum)	If *Dependum* is a Goal the Biddable Domain *Dependee* is constrainted by the Requirement which refines *Goal* If *Dependum* is a Task *Dependee!Task* If *Dependum* is a Resource *Dependee!Resource*
Dp(Depender, Dependee, Dependum)	*Depender!grantResource*, *Dependee!accessResource*

ther domains to the problems diagrams. We consider the set of resulting domains from the mapping of SI* elements to elements from the problem frame notation. At the completion of this step we check if all problem frame elements from the mapping have been used at least one problem diagram. If this is not the case we might have missed an problem diagram and a requirement. We have to correct this by include further requirements and problem diagrams for the missing problem frame elements.

8. **Problem-based conflict analysis**. This step focuses on a conflict analysis based upon the problem frame models. The aim of this analysis is to detect and resolve problems that can be identified on the abstraction level of problem frames. This includes technical analysis considering the differences between domains and phenomena. For example, if a phenomenon is directed in only one direction we have to ask ourselves if we really do not want a phenomenon in the other direction. We also have to consider if connections between domains are missing or if problem diagrams contradict each other. For example, if two problem diagrams add further domains and these are not compatible to one another.
9. **Create architecture**. The last step of our method is the creation of software architectures in UML from the problem frames as explained in [4].

4.2 Mapping

We present the rules to map SI* elements to Problem Frames elements in Tab. 1. Note that not all concepts and relations of SI* can be mapped to Problem Frames elements. We map a role or agent either to a biddable domain or to a causal domain. *Roles* or *Agents* map to a *Biddable Domain* if they represent human actors. If they represent entire businesses, hardware or even software, they are mapped to *Causal Domains*.

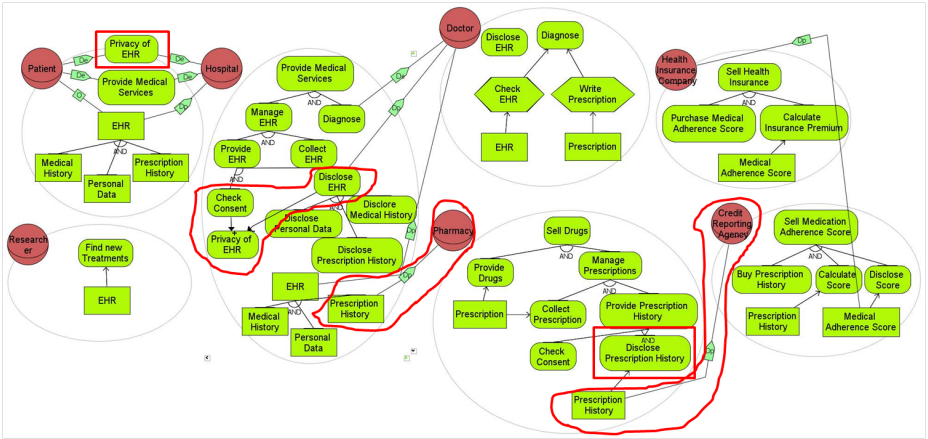
There is no representation of *Goals* in the Problem Frames notation. *Requirements* are means to fulfil goals [8] and are represented as text in the Problem Frame method, as well as graphical element in problem diagrams. *Tasks* in the SI* notation give rise to *Phenomenon* in the Problem Frames notation. The *Phenomenon* mapped to the *Task* will be under the control of the *Biddable Domain*, which has been mapped to the *Actor* providing the *Task*. A *Resource* in SI* is mapped to a *Lexical Domain* if it is just data. In the case it is hardware, software or any other mean with well defined behavior it is mapped to a *Causal Domain*.

Different mapping rules are applied to translate the relation *means_end* into Problem Frames elements based on the type of services (goal, task, resource) that are linked. If *means_end* is a relation between a *Goal* and a *Task* which is *provided* by an *Actor*, *Task* will be mapped to a *Phenomenon* under the control of the *Biddable Domain* representing *Actor*. The phenomenon belongs to the interface between the machine *Machine* and *Actor*. In addition, *Actor* is referred by the *Requirement* which refines the goal *Goal*. If *means_end* is a relation between a *Goal* and a *Resource*, *Resource* will be mapped to a *Lexical Domain* which is constrained by the *Requirement* that refines *Goal*. If *means_end* is a relation between a *Task* and a *Resource*, *Resource* is mapped to a lexical or causal domain. Additionally, there has to be Actor!Task between *Biddable Domain Actor* that represents the actor providing *Task* and the *Machine*. And Machine!Task between *Machine* and *Domain* representing *Resource* has to be added. To map the *delegation of execution* relation we use different rules according to the type of the Dependum. If the Dependum is a Goal, the Domain representing the *Dependee* Actor in the delegation of execution relation is *constrained* by the *Requirement* which is a refinement of Dependum *Goal*. If the Dependum is a Task or a Goal, the delegation of execution dependency is mapped to two Phenomenon under the control of the Biddable Domain representing the *Dependee* Actor. Furthermore, the *delegation of permission* relation having as Dependum a *Resource* is mapped to two *Phenomenon* under the control of the *Biddable Domain* representing the actors *Depender* and the *Dependee* because both of them have access to the *Lexical Domain* representing the *Resource*.

4.3 Consistency

We propose several consistency checks for our method:

- The *backtracking* from a problem frame diagram to a requirement to the goal has to be possible
  - The actors, which have provided a relation to a goal, have to appear in the problem diagram.
  - If a resource has a means-end relation to a goal, the resource has to appear in the problem diagram.
  - If a task has a means-end relation to a goal, at least one phenomenon related to that task has to be in the problem diagram
- If *changes* are made in the goal model these have to be also applied to the problem frame model



Legend: Circles represent roles (e.g Patient), ovals denote goals (e.g Provide Medical Services), exagons model tasks (e.g Write Prescription), while rectangles model resources (e.g EHR).

Fig. 4: SI* instantiation of the EHR scenario

5 Application of our Method

Application scenario To illustrate our method for combining goal- and problem-based requirements engineering methods, we use a scenario taken from the health care domain provided by the industrial partners of the EU project NESSoS [4]. It concerns the management of electronic healthcare records. The scenario focuses on providing medical services to patients, reading and updating their healthcare record and providing the results of examinations and treatment to authorized externals entities.

1. Instantiate SI* Models Fig. 4 shows the SI* model for the management of electronic health records (EHRs). The model consists of seven actors: *Patient*, *Hospital*, *Doctor*, *Pharmacy*, *Credit Reporting Agency*, *Health Insurance Company* and *Researcher*. The *Patient owns* the resource Electronic Health Care Record (EHR) which consists of *Medical History*, *Prescription History*, and *Personal Data*. The *Patient* wants *Privacy of EHR* and *Provide Medical Services* to be guarantee and depends on the *Hospital* for these goals to be achieved. To achieve the goal *Provide Medical Services*, *Hospital* needs to *Manage EHR* and the *Diagnose* the *Patient*. The goal *Manage EHR* is decomposed in the subgoals *Provide EHR* and *Collect EHR*. *Provide EHR* is further decomposed into subgoals *Check Consent* and *Disclose EHR*. The goal *Check Consent* positively contributes to the achievement of the goal *Privacy of EHR*, while the goal *Disclose EHR* negatively affects the fulfillement of the goal. To satisfy the goal *Diagnose* the *Hospital* depends on the *Doctor*. The *Diagnose* goal is achieved by the tasks *Check EHR* and *Write Prescription*. The task *Check EHR* requires the resource *EHR* to be accomplished and thus the *Hospital* delegates to the *Doctor* both the permission of accessing *EHR* and of *Disclose EHR*. The *Pharmacy* is responsible for *Sell Drugs* to patients. The goal *Sell Drugs* consists of *Provide Drugs* to be delivered according to

[4] http://www.nessos-project.eu/

the *Prescription* received from the *Patient* and *Manage Prescriptions*. The goal *Manage Prescriptions* requires to *Collect Prescription* and *Provide Prescription History*. This goals requires to *Check Consent* and *Disclose Prescription History*. The *Credit Reporting Agency* wants to *Sell Medical Adherence Score* that indicates how well patients handle drug prescriptions. To this end, *Credit Reporting Agency Buy Prescription History* to *Calculate Score* and *Disclose Score* to third parties like *Health Insurance Companies*. *Health Insurance Company Purchase the Medical Adherence Score* to calculate the *Health Insurance Premium*. Finally, the information in the *EHR* is used by the *Researcher* to *Find new Treatments*.

Figure 4 shows a fragment of the SI* model in Fig. 4 where a conflict occurs between the main goal of the *Patient* and *Pharmacy*. The conflict arises between the main goal of *Patient*, *Privacy of EHR* and the *Pharmacy* goal *Disclose Prescription History* (wrapped by red rectangles). The *Patient* has delegated the execution of the goal *Privacy of EHR* to the *Hospital*. The fulfillment of the goal *Privacy of EHR* is negatively affected by the goal *Disclose EHR* which is decomposed in the sub-goals *Disclose Personal Data*, *Disclose Medical History* and *Disclose Medical History*. The *Hospital* has not delegated to the *Pharmacy* the permission of *Disclose Prescription History* but just the permission of accessing *Prescription History*. The *Pharmacy*, instead, grants access on *Prescription History* to *Credit Reporting Agency*. Thus, the *Pharmacy* goal *Disclose Prescription History* is in conflict with the goal *Privacy of EHR* of the *Patient*. The information about which goals have a negative impact on each other is domain knowledge. This cannot be derived automatically and has to be an input to the model. However, the detection of these goal conflicts is not possible in the Problem Frame notation, because stakeholder goals are not part of the notation. Thus, we can only detect these goal conflicts when using SI* (or another goal-based notation). We resolve the conflict by removing the goal *Provide Prescription History* to the *Pharmacy*.

3. Derive Requirements from Goals We refined the goals from the SI* models into requirements. The goals depicted in Fig. 4 are marked in bold in the following requirements:

RQ1 Provide the means for the doctor to **Diagnose** patients
RQ4 Provide access to monitoring data for researchers, so that the researcher can **Find new Treatments**
RQ5 Provide means to enable doctors to **Asses Treatment** plans of other doctors

4. Map SI* to Problem Frame Elements We provide a mapping of the SI* elements to the PF elements in Tab. 2.

5. Create Context Diagram In Fig. 5 we present a context diagram of the electronic health system (***EHS***), the machine to be built. The EHS has connections to the lexical domains ***EHR***, the ***Privacy Policie***s of the system, and the ***System Log***s. The Privacy Policies state the privacy preferences of each patient, e.g., to which doctors the patient has given a consent to use her EHR. The EHS is connected to the ***Patient*** using the ***Browser Patient***, a ***Mobile Device***, which is further connected to ***Sensor***s that are in turn attached to the Patient. A ***Monitor*** or the ***Browser Care Provider***s connects the machine to the health care professionals, in our example ***Doctor***s.

Table 2: Mapping of SI* elements to PF elements

SI* Element	**PF Element**
Actors	
Patient	Biddable Domain *Patient*, because it represents a group of human actors
Hospital	The *EHS* machine offers no functionality to the hospital directly. Thus, we do not require a domain for it in the context diagram. The functionality is actually provided to the *Doctor*.
Doctor	Biddable Domain *Doctor*, because it represents a group of human actors
Researcher	Biddable Domain *Researcher*, because represents a group of human actors
Pharmacy	The *EHS* machine offers no functionality to the pharmacy directly. Thus, we do not require a domain for it in the context diagram. The functionality is actually provided to the *Pharmacist*.
Pharmacist	Biddable Domain *Pharmacist* is introduced, because these group of actors use the *EHS* machine.
Credit Reporting Agency	The *EHS* machine offers no functionality to the credit reporting agency. Thus, we do not require a domain for it in the context diagram. The prescription information is forwarded from *Pharmacists* with other means.
Health Insurance Company	The *EHS* machine offers no functionality to the health insurance company. Thus, we do not require a domain for it in the context diagram.
Tasks	
MonitorVitalSigns	A sequence of phenomena where the machine records the vital signs of the patient: *P!VitalSigns* and the according sequence of phenomena to the machine: *S!VitalSigns, MD!CreateEHR*
ProcessMonitoring-Data	The processing of the data is machine internal, but for processing the data has to be read first: *EHS!RequestEHR*
Notify about Treatment	A phenomenon that informs the nurse or doctor about a treatment to be applied and the corresponding sequence of phenomena: *EHS!sendEHR, M!sendEHR*
Record Appliance	A sequence of phenomena that sends an EHR with the treatment entry from the nurse / doctor to the machine: *N!sendEHR, D!sendEHR, BCP!sendEHR*
Check Health Data	A phenomenon that enables the doctor to request an EHR: *D!requestEHR, BCP!requestEHR*
Write Prescription	A phenomenon that enables the doctor to write an prescription: *D!send, BCP!sendEHR*
Provide Prescription	A phenomenon that enables the pharmacist to write an prescription: *P!provideDescription, BP!providePrescription,E!sendPrescription*
Collect Prescription	A sequence of phenomena that enables the pharmacist to collect a prescription: *P!collectPrescription, BP!collectPrescription,EHS!collectPrescription*, *E!sendPrescription,EHS!sendPrescription,BP!sendPrescription*
Resource	
Mobile Device	Causal Domain *Mobile Device*, because it is a device containing hard- and software. It is also a Connection Domain *Mobile Device*, because it connects the sensor and the EHS.
MonitoringData	Lexical Domain *Monitoring Data*, because it is data without a physical device.
Sensor	Causal Domain *Sensor*, because it is a device containing hard- and software.
Electronic Health Record (EHR)	Lexical Domain *EHR*, because it is data without a physical device.
Prescription	Lexical Domain *EHR*, because it is data without a physical device.

The ***Researcher*** uses the ***Browser Researcher*** to access the ***Research Database Application***, which is in turn connected to the EHS. The hospital from the SI* diagram is not mapped to in the context diagram, because it has no direct connection to the machine. The reason is that the hospital delegates all goals to other actors. Hence, a direct connection to the machine is not necessary. The Problem Frame method demands that the connections between domains and the machine are made explicit. Hence, our model got enriched by several connection and causal domains, e.g., ***Browser Researcher***. These are not present in the SI* models.

6. Refine Requirements We identified 19 preliminary functional requirements for the EHS, which were refined into 34 functional requirements and corresponding problem diagrams. For reasons of space, we focus on the following refined requirements for the remainder of the paper:

RQ1 Provide the means for the doctor to diagnose patients

RQ1.1 Store ***EHR***, which is created by care providers.
RQ1.2 Display ***EHR*** to care providers as needed
RQ1.3 Store and process vital signs of patients in ***EHR*** for care providers
RQ4 Provide access to monitoring data for researchers, so that the researcher can find new treatments
RQ4.1 Provide a functionality to request medical data for ***Researcher***s
RQ4.2 Release medical data to ***Researcher***s

7. Create Problem Diagrams The problem diagram for RQ1.1 describes creating and storing of ***EHR***s (depicted in Fig. 6). The ***Patient*** is connected to a ***Sensor*** that reports the ***Patient***'s vital signs to the ***EHR Create & Store Machine*** using a ***Mobile Device***. The machine stores the ***EHR***. In addition, the ***Patient*** can use the ***Browser Patient*** to create an ***EHR***. ***Doctor***s and ***Nurse***s can use the ***Browser Care Providers*** to command the machine to create ***EHR***s.

The release of medical information to researches described in RQ4.2 and depicted in the problem diagram in Fig. 7. ***Researcher***s can use the ***Browser Researcher*** to request medical data from the ***Research Database Application***. This application requests the data in turn from the ***ReleaseMedicalDataMachine***, which releases it the ***Research Database Application***. The application sends the information to the browser, where it is shown to the ***Researcher***s.

The remaining problem diagrams are drawn in a similar manner.

8. Problem-based conflict analysis We proposed a pattern-based method for identifying laws [3] based upon software requirements specification. Using this method for our case study gives the result that data protection law, among others, is relevant. For example, the the German data protection act (BDSG). This law demands a detailed privacy analysis of a system. In earlier works we proposed the ProPAn method [2] for computer-aided privacy threat analysis. The method is based upon the problem frame method and

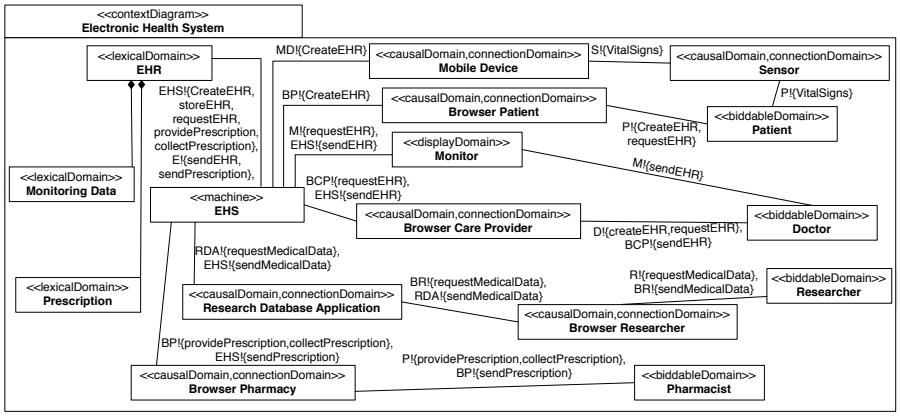


Fig. 5: Context Diagram

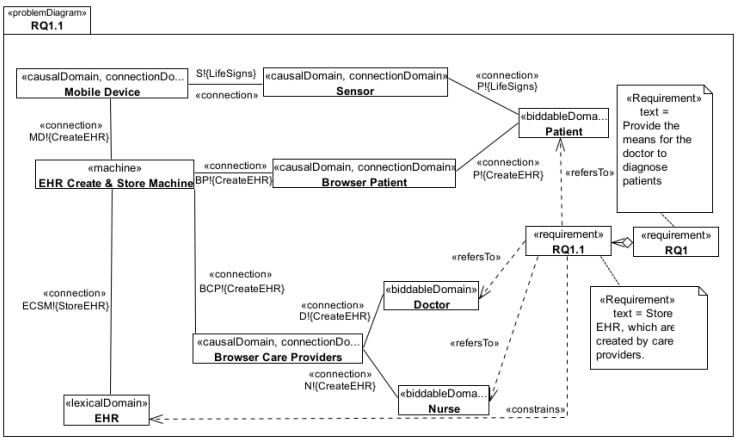


Fig. 6: Problem Diagram for Requirement RQ1.1

results in a set of requirements that have a privacy conflict. In our case study the results are that RQ4 has a conflict with other requirements, because the *Researcher*s gain access to personal information of the *Patient*s without an informed consent. We resolve this issue by anonymising the data transferred to the researchers. The ProPAn method only uses the problem frame approach and relies upon an analysis of the domains and their connections. Hence, the complete modelling of the machine and all the domains in the environment allows to trace personal information through the system. This complete tracing is difficult to achieved with SI*, because the system is only modelled in relevant artifacts. Hence, it might be possible to detect a privacy violation, if a related goal is modelled. To describe a tracing, however, is rather difficult to master in SI*, because the system is only partially modelled.

After changing the problem diagram of RQ4, we have to execute consistency checks on the other problem frame models and the SI* models. This also results in a Feedback loop to the goal model and we have to update the goal model accordingly. This results in the model shown in Fig. 8. In this model, we added goal the *protect personal data* of the new established actor *Legislator* (red part with the lightning). The *Hospital* has to comply to this law. Hence, we have to include the information that the data in the EHR have to be anonymised, before sending to the *Researcher*s (green part with the exclamation mark).

9. Create Architecture from Problem Diagrams For space reasons, we refer to the works of Choppy et al. [4] for creating software architectures from problem diagrams.

6 Discussion

The integration of a goal-based and a problem-based requirements engineering method can be achieved either by integrating the concepts of a goal-based method into a problem based (and vice versa) or by creating relations between a goal-based and a problem-

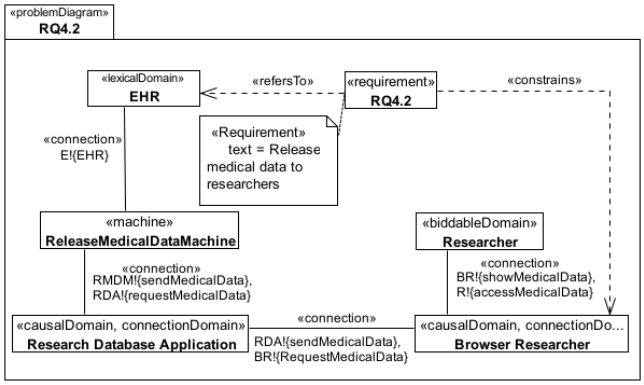


Fig. 7: Problem Diagram for Requirement RQ4.2

based method. We did experiments with the integration of goals and more explicit stakeholder views into the Problem Frame approach. However, the results were difficult to process, because we discovered that the goals are on a higher abstraction level the Problem Frame method. For example, creating a goal domain and instantiating it with the goal *Earn Money* is feasible. However, each domain in the Problem Frame approach has to have relations to other domains, and phenomena towards other domains or the machine. We could not come up with meaningful phenomena between goals and Problem Frame domains. Moreover, the investigation of goal conflicts is very difficult, when the relations between stakeholder goals are expressed in phenomena. Hence we abandoned the approach of integrating concepts.

Nevertheless, we agree that a intensive analysis of stakeholder goals is valuable for every software development problem. Thus, we decided to work on relations between the two requirements engineering methods. This has the drawback that two time consuming methods have to be executed. However, we recognized that it is important to analysis conflicts on different levels of abstractions. Goal-based methods have the advantage of only considering the technical aspects of the software if they have a direct relation to a stakeholder goal. In addition, goal-based methods only consider the specific parts of a software that has this relation. The problem-oriented world works on a lower abstraction level and tries to describe a complete context of the machine. This has the advantage of being able to derive software architectures from this context description.

7 Conclusions

In this paper we have presented a method to integrate goal- and problem-based requirements engineering methods. We presented a particular method for the integration of the SI* and Problem Frames methods. This method provides the ability for software engineers to analyze and resolve goal conflicts among stakeholders on a high abstraction level and to transfer the resulting goal models to problem models that focus on the particular software design problem.

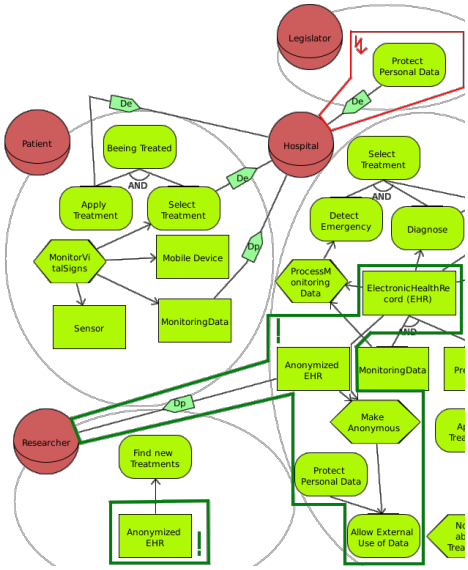


Fig. 8: Patient Monitoring - An example of a revised goal model due to an requirements conflict

Our method offers the following main benefits:

- Combining the SI* and Problem Frames methods without changing the existing methods
- Systematic identification and solving of goal conflicts
- Beginning the Problem Frames method with already resolved goal conflicts
- Using the advantages of goal- and problem-based requirements engineering methods

In the future, we will use or method for further goal-based methods, e.g., KAOS [20]. We will also look into the integration of security- and risk-based requirements engineering methods, e.g., CORAS [16]. We will also investigate the applicability of graph model transformation techniques to semi-automate some steps of our method like the mapping from SI* to Problem Frames elements.

References

1. Azadeh Alebrahim, Denis Hatebur, and Maritta Heisel. A method to derive software architectures from quality requirements. In Tran Dan Thu and Karl Leung, editors, *Proceedings of the 18th Asia-Pacific Software Engineering Conference (APSEC)*, pages 322–330. IEEE Computer Society, 2011.
2. Kristian Beckers, Stephan Faßbender, Maritta Heisel, and Rene Meis. A problem-based approach for computer aided privacy threat identification. In *Privacy Forum*, LNCS. Springer, 2012. to appear.

3. Kristian Beckers, Stephan Faßbender, Jan-Christoph Küster, and Holger Schmidt. A pattern-based method for identifying and analyzing laws. In *Proceedings of the International Working Conference on Requirements Engineering: Foundation for Software Quality (REFSQ)*, LNCS, pages 256–262. Springer, 2012.
4. C. Choppy, D. Hatebur, and M. Heisel. Systematic architectural design based on problem patterns. In P. Avgeriou, J. Grundy, J. Hall, P. Lago, and I. Mistrik, editors, *Relating Software Requirements and Architectures*, chapter 9, pages 133–159. Springer, 2011.
5. Andreas Classen, Patrick Heymans, Robin Laney, Bashar Nuseibeh, and Thein Than Tun. On the structure of problem variability: From feature diagrams to problem frames. In *Proceedings of International workshop on Variability Modeling of Software-intensive Systems*, pages 109–118, Limerick, Ireland, January 2007.
6. Isabelle Côté. *A Systematic Approach to Software Evolution*. Deutscher Wissenschafts-Verlag (DWV) Baden-Baden, 2012. to appear.
7. Isabelle Côté, Denis Hatebur, Maritta Heisel, and Holger Schmidt. UML4PF – a tool for problem-oriented requirements analysis. In *Proceedings of the International Conference on Requirements Engineering (RE)*, pages 349–350. IEEE Computer Society, 2011.
8. Benjamin Fabian, Seda Gürses, Maritta Heisel, Thomas Santen, and Holger Schmidt. A comparison of security requirements engineering methods. *Requirements Engineering – Special Issue on Security Requirements Engineering*, 15(1):7–40, 2010.
9. Denis Hatebur. *Pattern and Component-based Development of Dependable Systems*. Deutscher Wissenschafts-Verlag (DWV) Baden-Baden, September 2012.
10. Denis Hatebur and Maritta Heisel. A foundation for requirements analysis of dependable software. In Bettina Buth, Gerd Rabe, and Till Seyfarth, editors, *Proceedings of the International Conference on Computer Safety, Reliability and Security (SAFECOMP)*, volume 5775 of *LNCS*, pages 311–325. Springer Berlin / Heidelberg / New York, 2009.
11. Denis Hatebur and Maritta Heisel. A UML profile for requirements analysis of dependable software. In *SAFECOMP*, pages 317–331, 2010.
12. M. Jackson. *Problem Frames. Analyzing and structuring software development problems*. Addison-Wesley, 2001.
13. M. Jackson and P. Zave. Deriving specifications from requirements: an example. In *Proceedings 17th Int. Conf. on Software Engineering, Seattle, USA*, pages 15–24. ACM Press, 1995.
14. Emmanuel Letier and Axel van Lamsweerde. Deriving operational software specifications from system goals. *SIGSOFT Softw. Eng. Notes*, 27(6):119–128, November 2002.
15. Lin Liu and Zhi Jin. Integrating goals and problem frames in requirements analysis. In *Requirements Engineering, 14th IEEE International Conference*, pages 349 –350, sept. 2006.
16. Mass Soldal Lund, Bjørnar Solhaug, and Ketil Stølen. *Model-Driven Risk Analysis: The CORAS Approach*. Springer, 1st edition, 2010.
17. Fabio Massacci, John Mylopoulos, Federica Paci, Thein Tun, and Yijun Yu. An extended ontology for security requirements. In *Advanced Information Systems Engineering Workshops*, volume 83, pages 622–636, June 2011.
18. Fabio Massacci, John Mylopoulos, and Nicola Zannone. Security requirements engineering: The si* modeling language and the secure tropos methodology. In Zbigniew Ras and Li-Shiang Tsay, editors, *Advances in Intelligent Information Systems*, volume 265 of *Studies in Computational Intelligence*, pages 147–174. Springer Berlin / Heidelberg, 2010.
19. Sam Supakkul and Lawrence Chung. Extending problem frames to deal with stakeholder problems: An agent- and goal-oriented approach. In *Proceedings of the 2009 ACM symposium on Applied Computing*, SAC '09, pages 389–394. ACM, 2009.
20. Axel van Lamsweerde. *Requirements Engineering: From System Goals to UML Models to Software Specifications*. John Wiley & Sons, 1st edition, 2009.