

Towards a requirements specification multi-view framework for self-adaptive systems

Juan C Muñoz-Fenández, Gabriel Tamura, Raúl Mazo, Camille Salinesi

▶ **To cite this version:**

Juan C Muñoz-Fenández, Gabriel Tamura, Raúl Mazo, Camille Salinesi. Towards a requirements specification multi-view framework for self-adaptive systems. CLEI electronic journal, 2015, Best Papers from CLEI 2014 Special Issue, 18 : 2015 (2, August 2015). hal-01215245

HAL Id: hal-01215245

https://paris1.hal.science/hal-01215245

Submitted on 2 Nov 2015

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Towards a Requirements Specification Multi-View Framework for Self-Adaptive Systems

Juan C. Muñoz-Fernández
Facultad de Ingeniería, Universidad Icesi, Cali, Valle del Cauca, Colombia.
CRI, Université Paris 1 Panthéon - Sorbonne, Paris, France.
jcmunoz@icesi.edu.co

Gabriel Tamura
Facultad de Ingeniería, Universidad Icesi, Cali, Valle del Cauca, Colombia.
gtamura@icesi.edu.co

Raúl Mazo and Camille Salinesi
CRI, Université Paris 1 Panthéon - Sorbonne, Paris, France.
{*raul.mazo,camille.salinesi*}*@univ-paris1.fr*

Abstract

The analysis of self-adaptive systems (SAS) requirements involves addressing uncertainty from several sources. Despite advances in requirements for SAS, uncertainty remains an extremely difficult challenge. In this paper, we propose REFAS, a framework to model the requirements of self-adaptive software systems. Our aim with REFAS is to address and reduce uncertainty and to provide a language with sufficient power of expression to specify the different aspects of self-adaptive systems, relative to functional and non-functional requirements. The REFAS modeling language includes concepts closely related to these kind of requirements and their fulfillment, such as context variables, claims, and soft dependencies. Specifically, the paper's contribution is twofold. First, REFAS supports different viewpoints and concerns related to requirements modeling, with key associations between them. Moreover, the modeler can define additional models and views by exploiting the REFAS meta-modeling capability, in order to capture additional aspects contributing to reduce uncertainty. Second, REFAS promotes in-depth analysis of all of the modeled concerns with aggregation and association capabilities, especially with context variables. Furthermore, we also define a process that enforces modeling requirements, considering different aspects of uncertainty. We demonstrate the applicability of REFAS by using the VariaMos software tool, which implements the REFAS meta-model, views, and process.

Keywords: requirements engineering, requirements specification, self-adaptive systems, uncertainty, modeling process.

1 Introduction

Software systems built today are more complex in their functionality, supporting more sophisticated infrastructure and providing greater manageability than in the past. For example, users today make up a large global community that demand more variability in functionalities. A growing percentage of systems used by the community make use of elastic computing clouds, which have varying levels of fulfillment on quality attributes. An alternative means of tackling this growing complexity is the use of self-adaptive software (SAS) systems. These are systems that react at runtime to changing internal and external factors to accommodate their behavior by themselves [1]. These changing factors include context variables that affect the accomplishment of functional and non-functional requirements in unanticipated ways, that is, involving uncertainty.

In this context, self-adaptive systems inevitably face uncertainty from several sources [2]. In particular, some approaches manage the uncertainty in SAS from a selected set of sources, such as Rainbow [3], RELAX [4], FLAGS [5] and FUSION [6]. Despite the achieved progress, uncertainty remains an extremely difficult challenge. In this paper, we consider two sources of uncertainty. First, the simplified assumptions that modelers make for the in-breadth and in-depth analysis of the requirements. Second, the omission of relevant dependencies of the model. From these sources of uncertainty, we address two challenges. First, we consider the adaptation analysis in the broad aspects and different views with a language for specifying requirements, defined with clear syntax and semantics. Each of these views includes their concepts and associations and addresses different concerns. Second, we consider modeling the aggregation relationships of particular concerns into more general or global concerns, as well as the horizontal and hierarchical associations between these concerns. In addition, we define a modeling process with several phases, to systematically address different aspects of uncertainty. Each phase gradually introduces a different modeling view. The phases helps to manage both the complexity of the process, and the discovery of different aspects of uncertainty that can disrupt the accomplishment of the system requirements.

To address the aforementioned challenges, we propose a multi-view framework for Requirements Engineering For self-Adaptive Software systems (REFAS), supporting the specification of functional and non-functional requirements. In this paper, we present the specification language and its abstract syntax, defined through: a meta-model, the concrete syntax, semantics and the modeling process. The language defines claims, soft dependencies and context concepts, related to functional and non-functional requirements. Moreover, the language is extensible to model aspects not considered in the default definition of concepts and associations.

The REFAS language supports breadth analyses, as required by the first challenge, with its multiple views and the capability to be extended. Additionally, the language supports fulfillment levels of quality attributes expressed as concerns, as well as their associations to perform in-depth analyses. In-depth analyses support the second challenge, with aggregation capabilities over concerns and its implied variables. To demonstrate our approach, we apply the modeling process to a running example using the VariaMos software tool[1][7]. This tool implements the REFAS meta-model, its five default views, and its process. It also implements additional operations, such as simulation on the resulting models.

The rest of this paper is structured as follows. Section 2 presents a running example used to illustrate our approach and the addressed challenges. Section 3 presents a background on basic concepts used in our framework. Section 4 presents the meta-model and details the language syntax and semantics. Section 5 presents the modeling process with the set of default views. Section 6 presents the preliminary evaluation of REFAS as applied to the running example. Section 7 exposes the related work on requirements engineering for self-adaptive systems and discusses the advantages and limitations of REFAS. Finally, Section 8 presents conclusions of this paper and defines a research agenda.

2 Motivation

In this section, we introduce the running example that serves to illustrate the application of the REFAS framework, its modeling capabilities, and the challenges we address in this paper.

2.1 Running Example

This paper uses an extension of the GridStix case study [8][9][10], which has been used previously to exemplify the characteristics of self-adaptive software systems.

GridStix faces the problem of monitoring and issuing alerts about a river prone to flood. On-time and accurate alerts help to reduce human and material losses without incurring unnecessary costs. From the technical viewpoint, GridStix counts with a network of wireless sensor nodes deployed in the target river. These nodes perform initial flooding analysis and send information using wireless data networks such as GPRS or GSM. The wireless sensor nodes are battery-operated; thus they require energy optimization to prolong the system operation.

GridStix faces many requirements that are characteristic of dynamically reconfigurable (i.e., self-adaptive) software systems. Originally GridStix focuses on three aspects of dynamic reconfiguration. First, the communication between sensor nodes uses Wi-Fi or Bluetooth. Wi-Fi has lower latency and is more robust compared to Bluetooth, but it requires more energy [9]. Second, the transmission uses one of two routing strategies: shortest path and fewest hops. The first requires less energy, but it compromises performance when compared to the second [9]. Third, the nodes can execute the preliminary analysis on the river flood

[1] http://variamos.com

from camera images using centralized or distributed algorithms [11]. Distributed algorithms can improve the analysis but require more energy [9].

This running example also considers the following aspects inspired by the work of McMillan et al. [12]. First, the monitoring should support different sections of the river, including multiple branches or different locations of the same branch. Each section of the river periodically communicates with a master control using GPRS/GMS. Second, the sections should analyze whether the formulas for flow calculation require calibration. The system requires calibration due to changes in; graver-bed configuration, vegetation inside the river and flow distribution or other sources of uncertainty. A special analysis of the camera images supports identification and determines whether the above changes require calibration.

In the next section, we identify the challenges we address in this paper for modeling requirements.

2.2 Addressed Challenges

Self-adaptive systems face uncertainty in situations generated from several sources [2]. Many of these situations arise from the disrupting effect that context conditions of execution produce in the system's satisfaction of non-functional requirements. For modeling these requirements, in this paper we focus on two of these sources. First, simplified assumptions result in model abstractions that are not accurate enough for the addressed problem. These simplifications occur both in breadth and depth of the requirements analysis. Second, as a result of the simplified assumptions, important dependencies of the model are omitted. Among them, dependencies of the model on context variables constitute a particularly elusive source of uncertainty situations and events.

Considering the aforementioned two sources of uncertainty, in this paper we address the following challenges:

- C1: to model adaptation requirements from different viewpoints and concerns, that is, considering adaptation in the breadth aspects and views. These views should include their concepts and relationships in a clear language with well-defined syntax and semantics. For instance, uncertainty can be more controlled if the requirements model includes not only variability views (i.e., feature models), but also others such as context variables, dependency views, and goals views - referring to functional and non-functional requirements.

- C2: to model the sub-specification of adaptation requirements that result from the in-depth analysis of the different viewpoints and concerns identified by the breadth analysis, and their inter-relationships. That is, modeling not only the aggregation of particular concerns into more general or global concerns in a given problem, but also the horizontal and hierarchical relationships between these concerns. The hierarchical levels should consider the concepts and variables that represent significant context. In our running example, the river should be modeled in its appropriate fragments (e.g., incoming branches, significantly thick and deep sections). Each fragment should be modeled with its characteristics (e.g., water flow rate) and its effects on others fragments' characteristics (e.g., summed-up water flow rate).

In both challenges, originated from breadth and depth simplifications, context variables have a fundamental role. In the first case, for instance by considering different types of context, such as temperature, humidity, and weather (e.g., rainy) conditions. In the second case, for instance in the effect that context variables can have on others (e.g., combined low flow-rates from different river branches can produce a flood, even if the independently monitored low flow-rates would not produce a flood alert).

Unanticipated events occur, and it is practically impossible to anticipate and handle at design time all of the types of events that a system can face in its execution. Thus, by including context dependencies in more depth and breadth in requirements analysis, self-adaptive systems can reduce the adverse effects of uncertainty.

3 Background

In this section, we present the foundational concepts used in the definition of our modeling framework for self-adaptive software requirements: goals, soft-goals, claims, variability, operationalizations, and constraints.

Goals correspond to the reasons that justify the existence of the system from the stakeholders perspective. The exact definition of a goal has variations between different approaches. Dardenne et al. [13] defines a goal as an objective whose achievement depends on the interactions of multiple actors of the system. The main Goal-oriented Requirements Engineering (GORE) approaches, according to Lapouchnian [14], are NFR [15], KAOS[13][16], i*/Tropos [17][18], and GBRAM. Others are Adaptive RML [19] and Sawyer et al. [20]. GORE approaches share concepts to represent functional and non-functional requirements. In this

paper we focus on approaches for requirements engineering of self-adaptive systems are mostly based on goals modeling proposed by Dardenne et al. [13]. In this conception, modeling represents a system with goals, agents, relationships, entities, actions, and constraints. In addition, operationalizations represent an alternative to satisfy an specific goal. A goal can be satisfied by different operationalizations.

Soft goals (SG) correspond to the non-functional requirements of the system. Borrowed from NFR, SG fulfillment is not mandatory. Thus, SG are said to be "satisficed", instead of satisfied; that is, their satisfaction can be approximated by levels, for instance deny (- -), hurt (-), equals(=), help (+) and make (++). It is also common to use soft goals satisficing, instead of satisfaction, in reference to the approximated levels of a soft goal.

Sawyer's et al. [20] proposed a language based on GORE. In addition to the concepts above-mentioned, Sawyer's et al. include the soft dependencies, and claims with context variables. First, soft dependencies express the required satisficing level of a soft goal. Second, a claim defines the rationale of the relation between operationalization and soft goals, representing the positive or negative influence of an operationalization selection over the satisficing of a soft goal. Sawyer's et al. modifies the way to express claims concepts, compared to previous approaches [15][21][22].

The associations between of goals, and between goals and operationalizations, support the definition of variability.

4 The REFAS Language for Modeling Requirements

Our framework for self-adaptive software systems' modeling requirements, REFAS, comprises several subsystems, being the REFAS language the core of the framework. In this section, we present the REFAS language definition, by focusing on three main aspects of it. First, to allow the modeler to define arbitrary views and express their corresponding concerns, we conceived a generic meta-model. Generic means that we can meta-model any modeling language built with concepts and relationships, such as feature models and goal models. Second, we show how we use this meta-model to define the abstract syntax of our modeling language, and its corresponding semantics. Third, we explain five of the modeling views for self-adaptive systems that REFAS defines by default to specify requirements models. For each view, we present the concrete syntax of its concepts and relationships, illustrating it with instances of our running example.

4.1 Meta-model

REFAS represents the abstract syntax of the language through a generic meta-model. The meta-model specifies the concepts, associations and their relationships to define the requirements model of the specification of self-adaptive systems. By default, REFAS includes the definition of concepts and associations for five modeling views to represent the different aspects of the requirements model. Guerra et al. [23] applied the idea of multiple views with a unique meta-model in other domains. The meta-model is a foundation for formalizing the abstract syntax of a specification language.

Figure 1 presents all of the concepts and association of the meta-model, supported in four stereotypes. First, the *MetaConcept* stereotype represents a concept. Second, the *MetaOverTwoAssociation* stereotype represents an over-two association, i.e. a group association. Third, the *MetaPairwiseAssociation* represents a pairwise association, i.e. direct association. Fourth, the *MetaExtendsAssociation* represents a direct extension association. Arrows represent directional connections between *MetaConcept*s and *MetaPairwiseAssociation*s or *MetaExtendsAssociation*s. The figure also delimits the five default modeling views, identified by letters (A) to (E), which are built with the four stereotypes and directional connections. Some views share concepts as in the case of the *Operationalization* MetaConcept (required in three views) and the *Soft goal* MetaConcept (required in two views).

The meta-model presented in this paper is based on the initial version introduced in Muñoz-Fernández et al. [24], with several improvements. We identified these improvements in two application scenarios along the last year, yielding a major revision of the meta-model published earlier. First, we identified conceptual problems in the generalized application of the language to different case studies and examples. Second, we analyzed these issues during the implementation of a tool, VariaMos [7] [2], to support modeling using REFAS. The following are the most important improvements in the revised meta-model:

i. Simplification of the definition of conditional expressions, claims, and soft dependencies, without losing the semantics already supported.
ii. Removal of the functional requirement concept to avoid confusion with goals and operationalizations.

[2] http://variamos.com

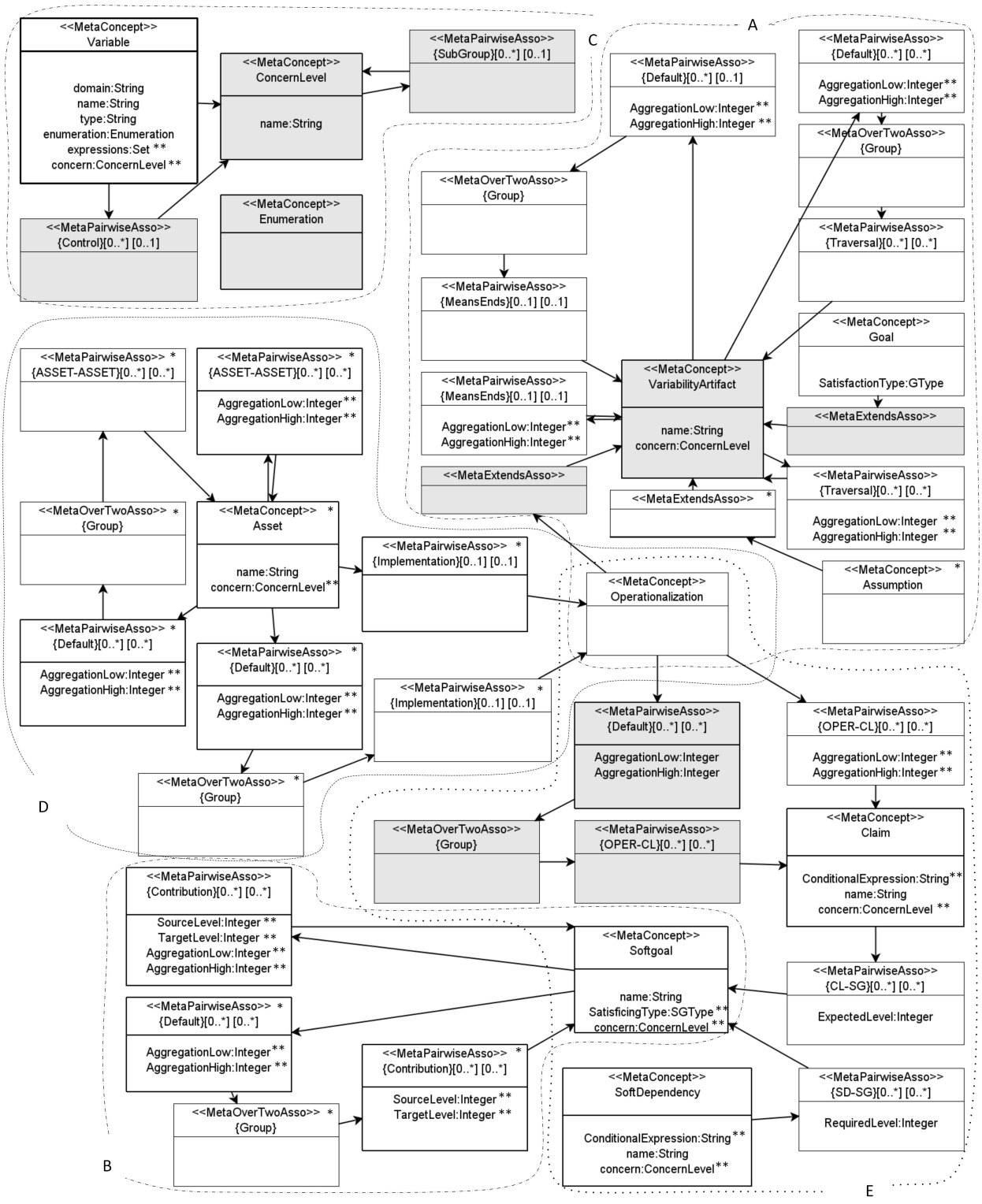


Figure 1: The REFAS meta-model with the five default views delimited. (A) Variability view; (B) soft goals view; (C) context view; (D) assets view; (E) soft goal satisficing view. New concepts introduced in REFAS are highlighted with a gray background. Concepts not existent in Sawyer et al. [20] and borrowed from other approaches have an asterisk on the right of the name (*). New attributes introduced by REFAS have a double asterisk on the right (**).

iii. Separation of the many to many (i.e. MetaOverTwoAssociation) and one to one (i.e. MetaPairwiseAssociation) association concepts to support better both types of associations. The revised version supports associations including ranges and exclusions (i.e. mutex).
iv. Extension of the soft goals view with contribution associations between soft goals with a clear semantics.
v. Inclusion of four new elements of MetaOverTwoAssociation between different concepts for modeling. Also, inclusion of many new relationships of MetaPairwiseAssociation.
vi. Homogenization of associations using *MetaPairwiseAssociation*, *MetaExtendsAssociation* and *MetaOverTwoAssociations*, to simplify the verification of their correct use and implementation.
vii. Inclusion of the association concepts between concern levels and aggregation goals from a specific concern level to a more general or global concern.

In the following section we introduce the definitions of the concepts that are involved in the five default views defined in REFAS.

4.2 Abstract Syntax and Semantics

A soft goal (*Softgoal* MetaConcept) defines a goal of the system from the stakeholder perspective that represents a non-functional aspect, for instance, *Energy efficiency*. Soft goals are recognized by a name and include a scale of levels of satisficing/denial. The satisficing level by default are integers between zero and four. A soft goal also defines a value to ponder between soft goals and a type of achievement (as high, as low or as close as possible). Soft goals exist in approaches such as the work of Chung et al.[15], Dardenne et al. [13] and Giunchiglia et al. [17].

A soft goal (SG) can contribute (*Contribution* MetaPairwiseAssociation) to other soft goals. The contribution includes expected level of the source and the target soft goals. If the expected level on the source is achieved, then the expected level on the target is obtained. It is possible to have many contribution associations between two soft goals.

A variability artifact (*VariabilityArtifact* MetaConcept) groups functional concepts (i.e., relative to functional requirements) and defines the name attribute. The meta-model considers three types of variability artifacts. First, a goal (*Goal* MetaConcept) represents functional aspects of the system from the stakeholder perspective, such as *Predict flooding*. The goal defines a satisfaction type attribute (maintain, achieve, optimize, avoid or cease). Second, an operationalization (*Operationalization* MetaConcept) represents a way of partial or complete satisfaction of a goal (e.g., through a software component), for instance, *WiFi*. Third, an assumption (*Assumption* MetaConcept) represents an external action or situation that must be maintained for the satisfaction of a goal or an operationalization, for instance, *GPRS/GSM network available*. The model guarantees the satisfaction of the goal/operationalization only if the assumption is true.

The variability artifact supports two types of associations. First, a variability artifact can be refined (*MeansEnds* MetaPairwiseAssociation) in one or many variability artifacts. Second, a variability artifact can have traversal associations (*Traversal* MetaPairwiseAssociation) with other variability artifacts. The traversal association types are: conflict, required, alternative and preferred.

A variable (*Variable* MetaConcept) is an abstraction of key information from the system, the environment or the model. From the system, variables can represent the system state, or a value defined by the administrator or a user. From the environment, variables are externally sensed, and finally from the model, variables are calculated from expressions. Expressions combine values of variables with logical and numeric expressions to obtain the values of the model variable. A variable has a name, a type of data, and optionally a domain or enumeration of allowed values. An example of a variable is the *Battery State* (low/high).

An enumeration (*Enumeration* MetaConcept) defines the allowed values for variables. For example, *FloodState* allowed values are: 0 - Normal, 1 - Low and 2 - High. It is worth noting that each textual value has an integer value associated for comparison purposes.

An asset (*Asset* MetaConcept) represents a software asset (e.g., a software component) of the system, for example, *WifiDataTransmit*. The assets may have functional and structural dependencies between them (*ASSET-ASSET* MetaPairwiseAssociation).

An Asset may implement (*Implementation* MetaPairwiseAssociation) an operationalization. One or many assets may be required to satisfy an operationalization. Only assets related to operationalizations or other assets are included to keep the diagram as clear as possible.

A claim (*Claim* MetaConcept) represents an expected satisficing level of a soft goal conditioned to the selection of operationalizations or satisfaction of a conditional expression. The expected satisficing level includes an integer value from zero to four. Syntactically, a claim is defined by an instance of Claim MetaConcept and includes: an instance of OPER-CL MetaPairwiseAssociation from at least one operationalization; and an instance of CL-SG MetaPairwiseAssociation to at least one soft goal. For example, a claim with *Bluetooth* incoming relationship means and affects *EnergyEfficiency* (level=4) that the claim is activated

when the selected operationalization is *Bluetooth*. Claims that neither supported the conditional expressions nor multiple operationalizations were used in Sawyer et al. [20] and other approaches [15][21][22].

A soft dependency (*SoftDependency* MetaConcept) represents the required satisficing level of a soft goal given the fulfillment of a conditional expression. Syntactically, a claim is defined by an instance of SoftDependency MetaConcept and include: an instance of SD-SG MetaPairwiseAssociation to at least one soft goal. For example for GlobalState="Normal" requires a level satisficing level equals to four for the *Energy Efficiency* SG.

A conditional expression represents a valid logical expression with a string. The expression combines variables and values in logic and numeric expressions. A Claim's conditional expression uses configured system and model variables. In contrast, a soft dependency's conditional expression uses sensed environmental and system variables.

A concern level (*ConcernLevel* MetaConcept) allows the in-depth analysis of concerns, including instances of concepts, i.e. groups requirements and context that can be partially analyzed independently and may have one or more instances. Examples of concern levels are instances of logical nodes, physical nodes, and user sessions. All meta-concepts except the *Enumeration* include a *ConcernLevel* attribute to define their membership optionally, as shown in the meta-model. An example of a concern level is *River Section*, meaning its group variables for sensing a section of the river. A concern level has a name defined according to the characteristics of the variables related. The model supports the association of concern level with another concern level (*SubGroup* MetaPairwiseAssociation).

Many concepts support many to many (MetaOverTwoAssociations) associations. Group Associations (*Group* MetaOverTwoAssociation) represent different MetaOverTwoAssociations with a cardinality. The cardinality defines the number of concepts required: all ("and"); at least one ("or"); only one ("mutex"); and range ([n,m]).

Connected to the Group Association, a Default Association (*Default* MetaPairwiseAssociation) relates the source concepts to the Group Association. This association also defines the aggregation required to satisfy the relationship that applies only to associations between concerns of different levels (only a relationship from a specific to a general concern level is allowed). The aggregationLow and aggregationHigh attributes define the minimum, and maximum values required for aggregation purposes.

The meta-model also supports aggregation in other pairwise associations. In this case, the MetaPairwiseAssociation includes the aggregationLow and aggregationHigh attributes.

A Control Association (*Control* MetaPairwiseAssociation) relates a *Variable* with a calculated value to a *Concern Level*. The expressions required to calculate the value can be defined independently for each instance of the concern level or the same for all the instances. In the first case, the number of instances must be known at design time. In both cases, the value is calculated from a single expression or a set of conditional expressions for each different value. The Control MetaPairwiseAssociation includes an attribute to define the number of instances.

Some meta concepts of the meta-model were taken from Sawyer's et al. [20] language. The added meta concepts compared to Sawyer's et al. are the *Assumption* MetaConcept, *Enumeration* MetaConcept, *ConcernLevel* MetaConcept, *Asset* MetaConcept and *VariabitilyArtifact* MetaConcept. The *Assumption* MetaConcept and *Asset* MetaConcept already exists in other approaches such as Qureshi et al. [19] and **??** respectively. Furthermore, the meta-model introduces several new association concepts and attribute additions in the meta-model. The new association concepts and attributes are marked in the Fig.1.

4.3 Meta-model Views and Concrete Syntax

The concrete syntax of REFAS and its notational elements have a direct correspondence to the abstract syntax of elements, except the *Variability Artifact*. Given the correspondence between the two syntax, we use the same meta-model to support the concrete syntax of our language. In particular, the four meta-model stereotypes include attributes to define the concrete syntax of the elements. These include the drawing style, the default size, the associated image, if the element is abstract and if the element is resizable.

The REFAS language relies on five views defined by default to model requirements. These views are represented by an additional stereotype called MetaView. This stereotype is not included in the abstract syntax. Instead, its instantiations are views, and are represented graphically by their limits, and named with letters (cf. Fig 1). Each view serves as a logical container of visible concepts and their associations. The views allow a designer to specify particular concerns of the requirements model from a specific viewpoint, that is, considering different adaptation aspects as broadly as possible, thus addressing our first stated challenge.

The concrete syntax for concepts and associations uses lines and texts in black. The concepts use a combination of rectangles, hexagons, parallelograms, trapeziums and ovals. The direct associations use different kinds of arrows. Also, the background of concepts combine white and two shades of blue. The colors are useful to increase the semantic separation between concepts, following Moody's recommendations

for models [25]. The four colors, including the black, are correctly identified even in grayscale printing. The combination of background colors and shapes defines the concepts syntactically.

4.3.1 Soft Goals View

The *soft goals view* represents the non-functional requirements, i.e. soft goals, and their contribution associations. This view includes the *Softgoal* MetaConcept, the two *Contribution* MetaPairwiseAssociation, a *Group* MetaOverTwoAssociation, and a *Default* MetaPairwiseAssociation. All of them presented on the bottom left of the Figure 1 (enclosed with a line of dashes and called B).

- Group Association Element (GAE): A circle with a white background represents the association from the origin of the many to many association (i.e. MetaOverTwoAssociation) to the destination concepts. Above the circle, the element includes the cardinality of the association ("and","or", "mutex", [n,m]) as shown in Fig 2A.
- Default Association: A directed arrow without additional information connects a MetaConcept instance to a GAE, as shown in Fig. 2D. The semantic is given by the GAE and the association between the GAE and the target concept.
- Soft Goal (SG): A distorted and softened rectangle with a thin black border surrounding it represents the soft goal (Figure 2B). The SG includes its name and the achievement type (as high, as low or as close as possible) in the figure's center.
- Contribution Association: A solid arrow represents the contribution association between soft goals, from a contributing SG to a target SG or from a GAE to a target SG. Figure 2C and Figure 2D present examples of contribution associations. Using a GAE, it is possible to represent contributions from multiple SG. The contribution association includes the level needed to activate the contribution association (expected level on the source SG) and the level assigned to the target SG (expected level on the target SG).

  Figure 3 shows an example of a soft goal with contribution relationships.

The remaining views define in a similar way the first two associations of this view, except the context view.

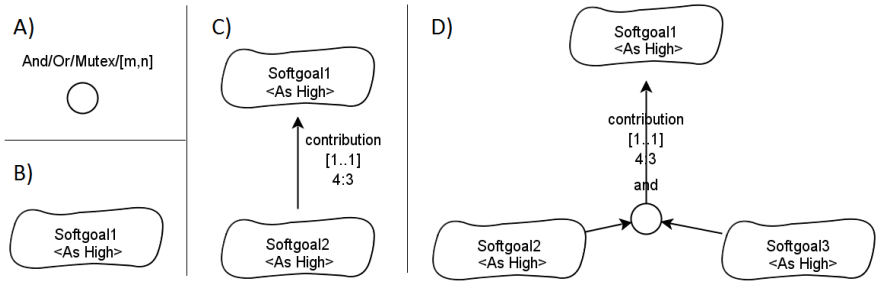


Figure 2: A)Group Association element, B) Soft goal, C) contribution pairwise association, D) Group and default associations with a contribution association

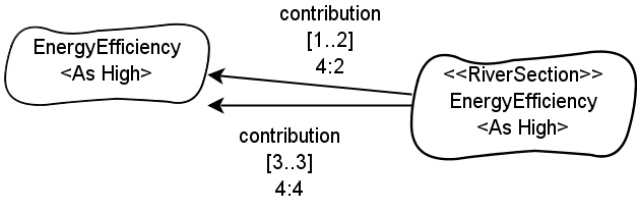


Figure 3: Part of the soft goals view of the running example

4.3.2 Variability View

The variability view relates goals and operationalizations to handle variability at two different levels. First, at the problem space level with alternatives between refinements of general goals into more specific ones. Second, at the solution space with alternatives between goals and operationalizations and between operationalizations.

This view has the *Goal* MetaConcept, the *Operationalization* MetaConcept, and the *Assumption* MetaConcept which extend from the *Variability Artifact* MetaConcept. All of them presented between the center and top right in Fig. 1 (enclosed with a dotted line and called A). The *Variability Artifact* MetaConcept is abstract therefore it does not have a concrete syntax.

- Goal: A parallelogram with a thick black border represents a goal (Figure 4A). The goal includes their the satisfaction type (maintain, achieve, optimize, avoid or cease) and its name in the figure's center.
- Operationalization: A hexagon with dark thick black borders on the left and right sides represents an operationalization (Figure 4B). The operationalization includes its name in the figure's center.
- Assumption: A rectangle with thin black borders represents an assumption (Figure 4C). The assumption includes its name in the figure's center.
- Means-Ends Association: The associations supported are one to one (i.e. MetaPairwiseAssociation) and many to many (i.e. MetaOverTwoAssociation) associations. Both are represented by a directed arrow to the target concept (Figure 4D). The one to one association starts from the source concept as presented in 5B. The many to many association starts from a group association element and include default associations from the source concepts as presented in Fig. 5A and Fig. 5C.
- Traversal Association: The associations supported are one to one (i.e. MetaPairwiseAssociation) and many to many (i.e. MetaOverTwoAssociation) associations. Both are represented by an arrow directed to the target concept (Figure 4E. The one to one association starts from the source concept, the many to many starts from a group association element and include default associations from the source concepts in the same way as presented for Means-Ends associations 5C.

Any variability artifact may include the concern type (*ConcernLevel* MetaConcept) when it applies..

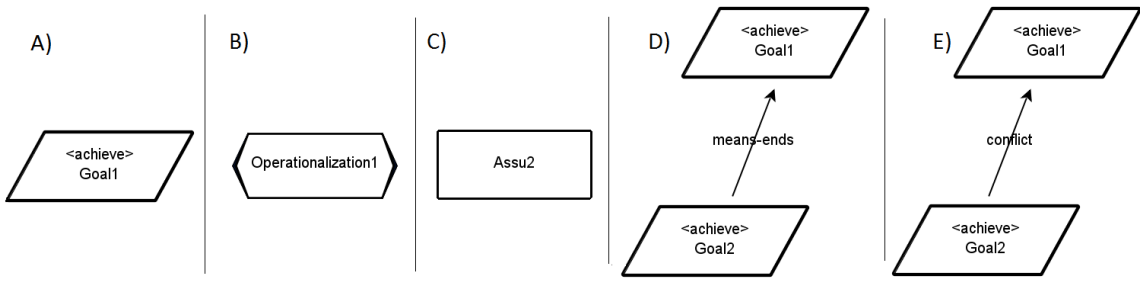


Figure 4: A) Goal, B) Operationalization, C) Assumption, D) Means-Ends association, E) Traversal association (conflict)

4.3.3 Context View

The context view represents the levels for in-depth analysis of the requirement model and the definition of the variables. A variable represents part of the context or the configuration of the target system. This view has the *Concern Level* MetaConcept, the *Variable* MetaConcept, and the *Enumeration* MetaConcept shown at the top left of the Figure 1 (enclosed with a dotted line and called C). Figure 6 presents the concrete syntax of the concepts. The view also includes two *MetaPairwiseAssociation*.

- Concern Level: A rectangle with oval sides and a background combining white (in the center) and dark blue (on the sides) represents the concern level (Figure 6A). The concern level includes its name in the figure's center.
- Variable: A rectangle with oval sides and a white background represents the variable (Figure 6B). The variable includes its name, and the data type or an enumeration of allowed values inside the figure. The string representation of expressions to calculate the value of the variables is not shown in the view.

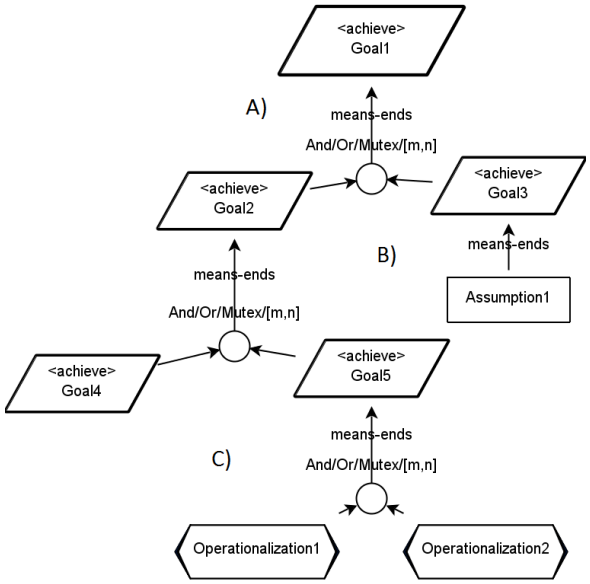


Figure 5: A) Means-Ends association with group association between a goals, B) Means-Ends association from an assumption to a goal, C) Traversal (MetaPairwiseAssociation) association between an operationalization and a goal.

- Enumeration: An enumeration features two rectangles. A rectangle on top shows the name of the enumeration and a second rectangle includes the values of the enumeration (Figure 6C).
- Control Association: A dashed arrow pointing the concern level represents the association between a variable and the concern level as shown in Fig. 6D.
- SubGroup Association: A solid arrowhead pointing the more general concern level represents the association between two concern levels as shown in Fig. 6E. The association includes the number of instances of the specific concern. '*' represents an undetermined number of instances.

  Figure 7 shows examples of an enumeration, a concern level, and some variables.

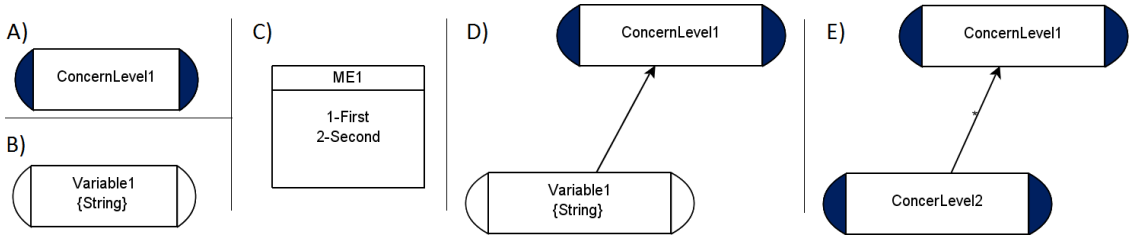


Figure 6: A) Concern level, B) Variable, C) Enumeration, D) Control association, E) SubGroup Association

4.3.4 Assets View

The assets view represents the assets of the model, e.g. system components, and their association with the variability view, i.e. the operationalizations. This view has the asset and the operationalization concepts presented in the center of the Figure 1 (enclosed with a line of big dashes and called D). This view also includes the associations between assets and the associations with operationalizations.

- Asset: A rectangle surrounded by two small rectangles on the left border represents an asset (Figure 8A) using the standard syntax for components.
- Implementation Association: A dashed line with a closed arrow from an asset to its operationalization represents the implementation association as shown in Fig. 8B.

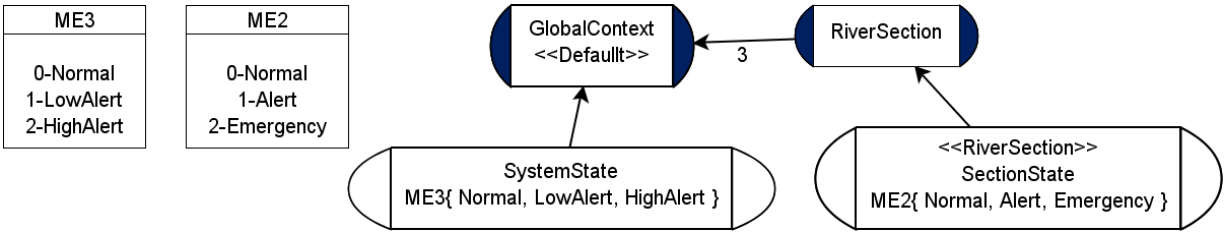


Figure 7: A subset of the context view of the running example

- Assets Association: An arrow from the origin asset to the destination asset represents the assets association, as shown in Fig. 8C.

  Figure 9 shows examples of assets with implementation and assets associations.

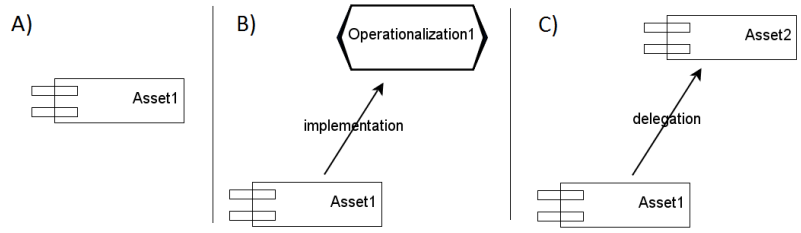


Figure 8: A)Asset, B) Implementation Association, C) Assets Association

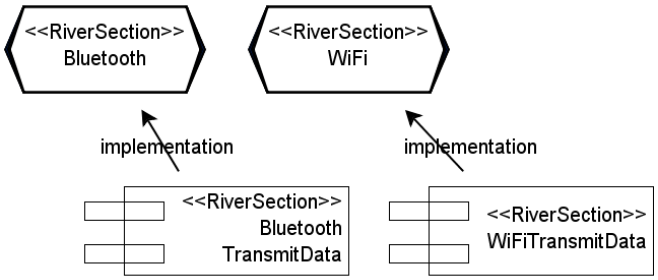


Figure 9: A subset of the asset view of running example

4.3.5 Soft Goals Satisficing View

The soft goals satisficing view supports the associations between the operationalizations of the variability view, and the soft goals of the soft goals view through expected satisficing level. This view also includes the required satisficing levels of soft goals for defined conditions.

The soft goal satisficing view has the *Claim* MetaConcept, the *Soft Dependency* MetaConcept, a *Group* MetaOverTwoAssociation and five meta pairwise associations presented on the left of the Figure 1 (enclosed with a dotted line and called E). This view also has the soft goal and operationalization already presented.

- Claim: A trapezium with a light blue background and solid lines with arrows pointing to the affected soft goals represents the claim. The claim has its name, follow by a conditional expression (optionally). The conditional expression may include values, variables and arithmetic and logical operators with a resulting logical value. Figure 10A presents an example of the associations defining a claim. In the arrow, the qualitative value should be specified (0, 1, 2, 3, 4). At least one operationalization or a conditional expression is mandatory for the claim definition. The group association for claims include an additional type, *None*, to represent whenever none of the operationalizations is selected to activate the claim.

- Soft dependency (SD): An oval with a light blue background surrounding a conditional expression and dotted arrows to the soft goals. Figure 10B presents an example of a SD with the required association. The arrow specifies the required satisficing level between zero and four, where zero is the minimum and four is the maximum. The conditional expression is compulsory and is defined in the same way as explained for the claim.

  Figure 11 shows examples of claims and soft dependencies from the running example.

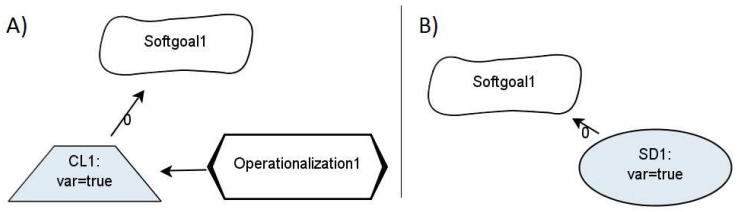


Figure 10: A) Claim, B) Soft dependency

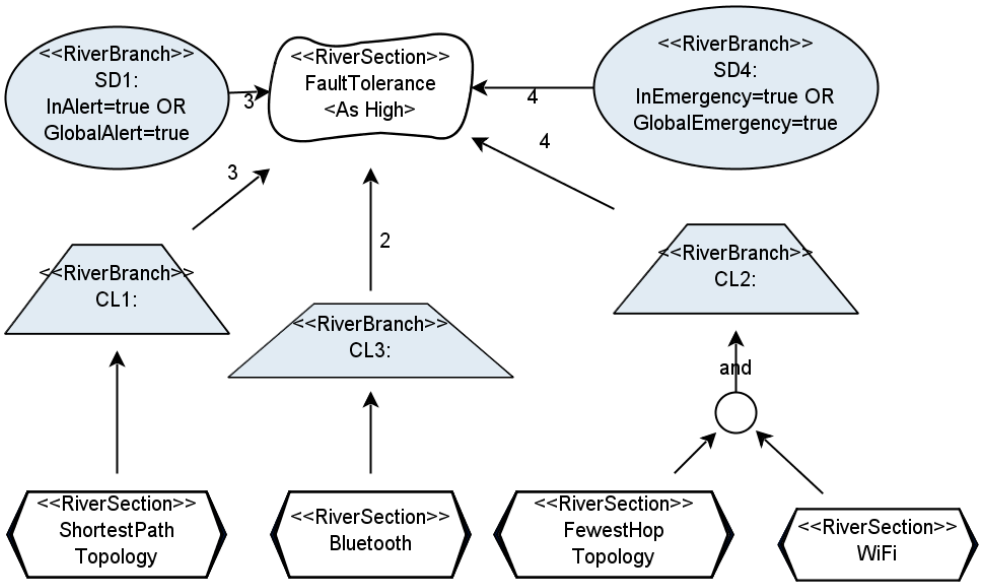


Figure 11: A subset of the SG satisficing view of running example

5 The REFAS Process for Modeling Requirements

The modeling process prescribes a sequence of phases to produce the requirements model, according to the modeling views. The process facilitates the correct and methodical definition of the requirements model according to the different concerns to achieve the first challenge.

The revised process presented here is based on our initial process [24] with several improvements. First, the revised process is consistent with the modifications presented for the meta-model's concepts and associations. Second, the revised process presents the tasks of the six phases and decisions not presented before. Third, the revised process includes a general process diagram and diagrams for each phase not presented before. Fourth, we present the running example of Section 2.1 following the defined tasks.

The first phase of the process identifies the initial concern levels. Next, other concepts relating to these concern levels to support the in-depth analysis of all the meta-model views as required by the second challenge.

Phase 1: This phase identifies the concern levels and the soft goals of the system. Figure 13 presents this phase divided into six tasks. First (1.1.), identify whether the model requires different concern levels as defined in Section 4.3.3. Create the appropriate concern level and relate specific concern to general ones.

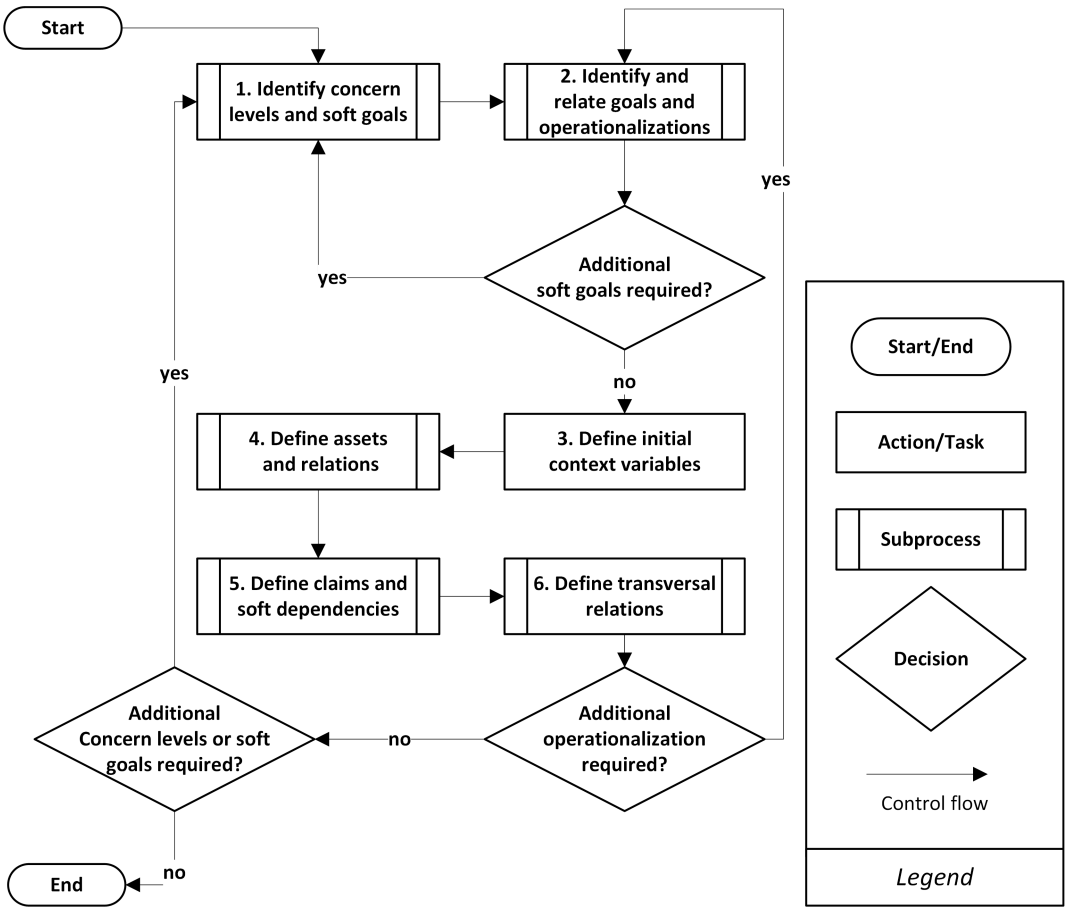


Figure 12: General flowchart for requirements modeling

Second (1.2.), define a Soft Goal (SG) of the system following the definition in Section 4.3.1. If the SGs are concrete enough to relate them to other concepts, then go straight to task 1.4. Third (1.3.), refine the SG into a more specific SG, and then return to last decision. Fourth (1.4.), review contributions of the new SG over other SGs already defined and in the opposite direction. Also, include contribution associations between SG. Fifth (1.5.), review the completeness of SG model. If more SG are required, then return to task 1.2.

In the running example, we identified the *RiverSection* concept as a *ConcernLevel* MetaConcept, as shown in Fig. 14. We consider three sections of the river for the example. The first two sections are affluent of the last section. In Figure 14, the global concern receives the association to represent the three instances.

For the running example, we identified three concepts: *EnergyEfficiency*, *PredictionAccuracy*, and *FaultTolerance* at the global level instances of *Softgoal* (SG) MetaConcept. The four SGs apply at the *RiverSection* concept. Thus, we define additional four SGs at this level. The running example considers three instances of the *RiverSection* and the three contributions relationships between each pair of SG concepts as shown in Fig. 15.

Phase 2: This phase identifies all the instances of *Goal* MetaConcept and *Operationalization* MetaConcept of the system. Figure 16 presents this phase divided into seven tasks. First (2.1.), identify an instance of the *Goal* MetaConcept of the system ($L_{i=1}$). Second (2.2.), refine the goal (L_i) in a set of more specific goals ($L_{i=i+1}$) that satisfies the main goal individually or alltogether. Third (2.3.), specify the type of association between the new defined goals and the more general goal using "and", "or", "mutex", or a range [n,m] in the association (as shown in Fig. 3A). If all goals are specific enough to be operationalized, return to task 2.2. Fourth (2.4.), review whether the model requires an additional goal and return to task 2.1. Fifth (2.5.), define the operationalizations associated with one specific goal as shown in Fig. 5B. Sixth (2.6.), define the type of association between the operationalizations and the specific goal using "and", "or", or "mutex" in the association. If more specific goals require operationalizations, return to task 2.6. Seventh (2.7.), the soft goals are reviewed to identify additional soft goals. Return to Phase 1 to add new soft goals if required.

In the running example, we defined a main goal (*Predict Flooding*) and two more specific goals (*Predict Section Flooding* and *Evaluate Weather*). The goal *Evaluate Weather* also has two more specific goals

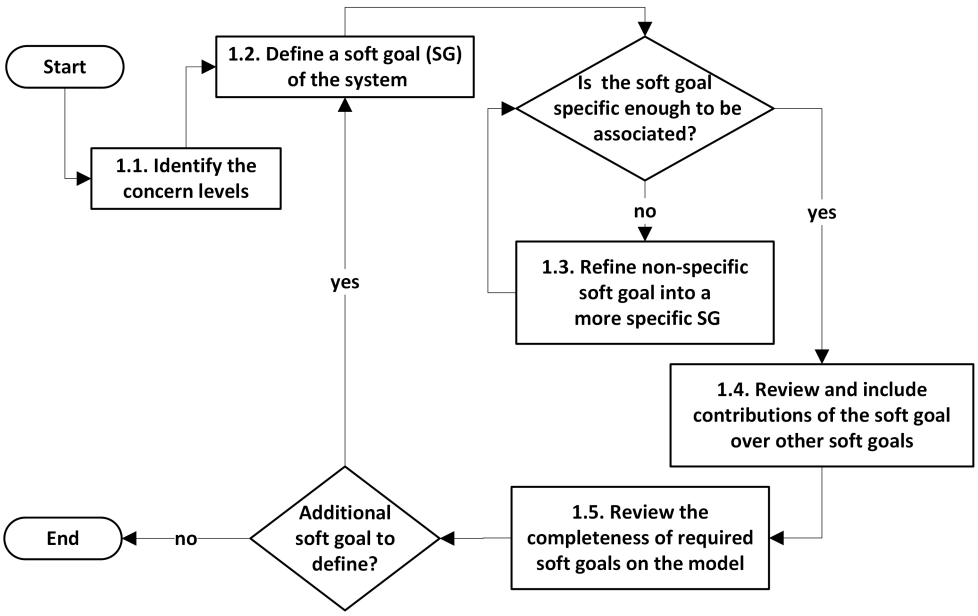


Figure 13: Phase 1: Identify concern levels and soft goals

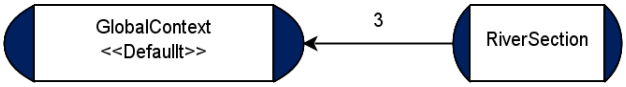


Figure 14: Concern Levels

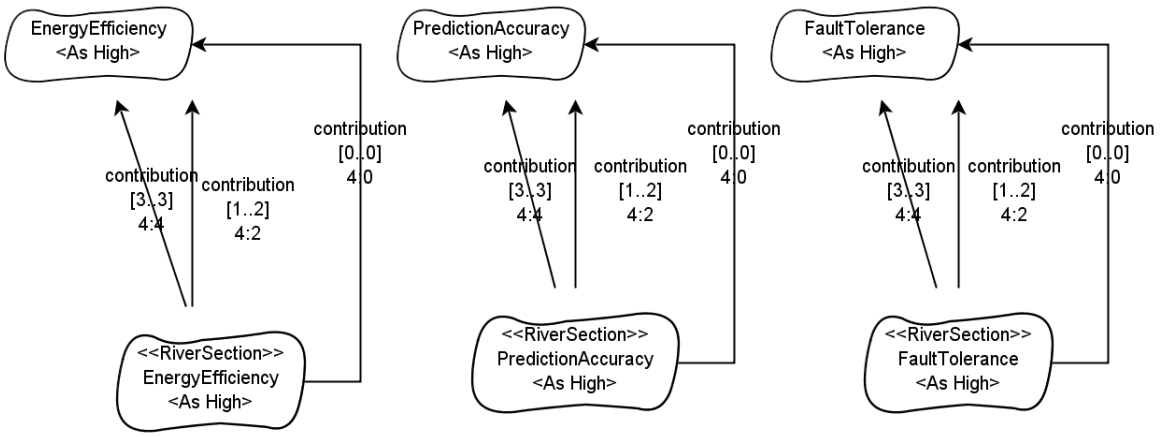


Figure 15: Soft Goals view

(*Receive Weather* and *Process Weather*). Both refinements require all the goals ("and") as shown in Fig. 17.

The *Predict Section Flooding* is at the *River Section ConcernLevel* including three specific goals. The default association from *Predict Section Flooding* includes the cardinality three to three ([3..3]) to represent that the three sections are required to satisfy the global goal. We define the operationalizations for three of the specific goals.

The goal *Organize Network* seeks to achieve the topological organization of the network and has two operationalizations. The operationalizations are *Fewest Hop Topology* and *Shortest Path Topology*. The two operationalizations imply exclusion ("mutex").

The goal *Calculate Flow Rate* has two operationalizations. This goal seeks for the calculation of the river flow using a camera available in each selection of the river. The defined operationalizations are *Single Node*

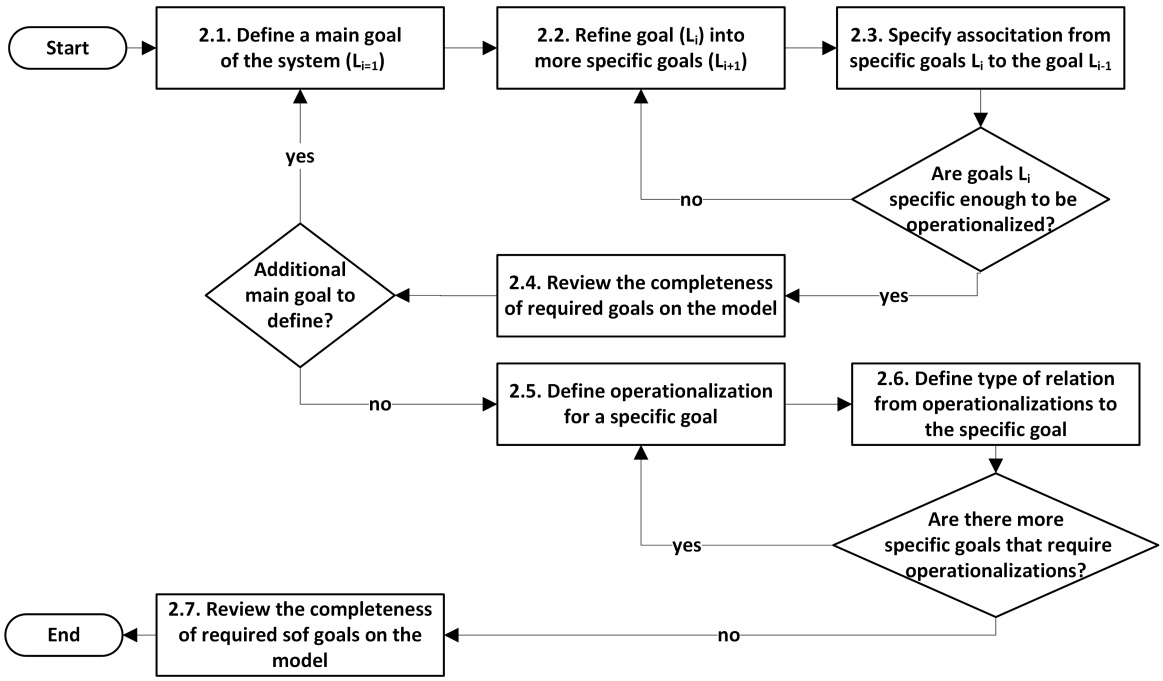


Figure 16: Phase 2: Identify and associate goals and operationalizations

Processing and *Distributed Node Processing*. The operationalizations have a mutual exclusion association ("mutex") with this goal.

The goal *Transmit Data* seeks for the transmission of data between the nodes deployed on the river section and has two operationalizations. The operationalizations are *WiFi* and *Bluetooth*. The two operationalizations implies exclusion ("mutex").

Figure 17 presents the complete variability view for the running example.

Phase 3: This phase defines the variables for the Requirements model. The variables should be relevant for the adaptation logic, i.e. they affect the required or expected satisficing level of soft goals defined in Phase 1. Each variable requires a name, a type (Boolean, Integer, Float or Enumeration), and a domain. Some variables are controlled by a concern level. Thus, these variables include an expression or set of expressions to define its value.

In the running example, we identified the relevant variables for the soft goals previously defined, as shown in Fig. 18. We identified three global variables, the last two calculated with expressions. First, *Weather State* identifies the state from the weather information. Second, *System State* is calculated according to the failures of nodes and sensors with conditional expressions (left side represents the value and right side the conditional expression):

Normal: (sum(NodeFailures) + sum(NodeFailures)) = 0

LowAlert: (sum(NodeFailures) * 2 + sum(SensorFailures) > 1 AND

(sum(NodeFailures) * 2 + sum(SensorFailures) < 3)

HighAlert: (sum(NodeFailures) * 2 + sum(SensorFailures) > 2

Third, *Global Alert* is calculated according to the flooding state in the river sections with conditional expressions:

Normal: (sum(SectionState <> 0)) = 0

PartialAlert: (sum((SectionState = 1) * SectionWeight) > 0) AND

(sum((SectionState = 1) * SectionWeight) < 2)

GeneralAlert: (sum((SectionState = 1) * SectionWeight) > 1) AND

(sum((SectionState = 2) * SectionWeight) < 3)

Emergency: sum((SectionState = 2) * SectionWeight) > 2

Also, we identified seven variables related to the River Sections, only one calculated with expressions. First, *Battery State* considers the battery state of the wireless sensor nodes. Second, *Sensor Failures* counts the number of sensors with failures in the wireless sensor nodes. Third, *Node Failures* counts the number of nodes with failures. Fourth, *River Flow* represents the river flow from 0 to 10. Fifth, *Flood Predicted* represents the prediction of flooding (true or false). Sixth, *Section Weight* identifies a weight of the section about the other sections. The value of the *Section Weight* variable is one for the first two sections and two

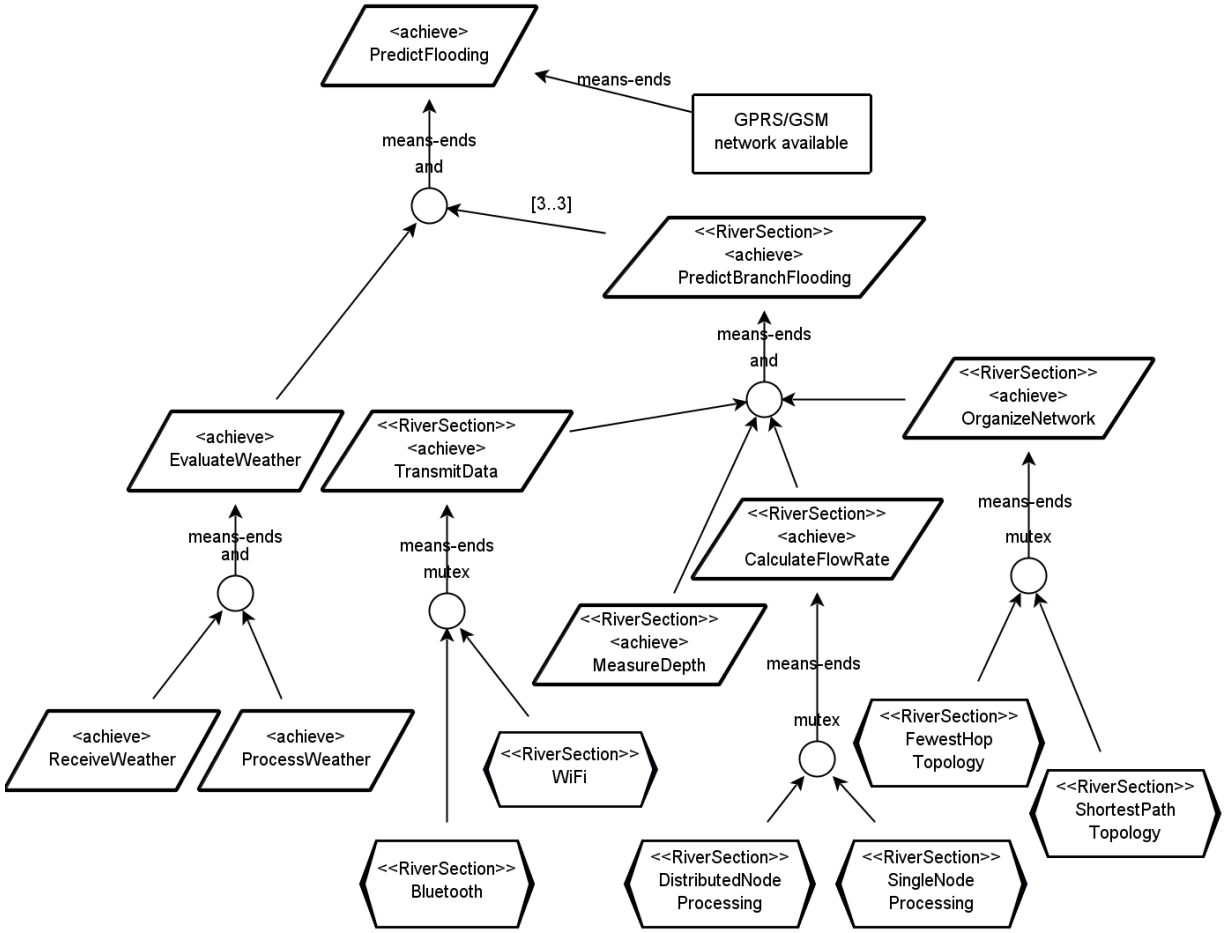


Figure 17: Goals, operationalizations and assumptions

for the third section. Seventh, *Local State* calculates the state of the section, according to the river flow with conditional expressions:

River Section 1 (S1):
Normal: S1.SectionState = 0
Alert: S1.SectionState = 1
Emergency: S1.SectionState = 2
River Section 2 (S2):
Normal: S2.SectionState = 0
Alert: S2.SectionState = 1
Emergency: S2.SectionState = 2
River Section 3 (S3):
Normal: S1.SectionState = 0 AND (S2.RiverFlow + S3.RiverFlow) < 10
Alert: (S1.SectionState = 1 AND S2.SectionState = 1) OR S3.SectionState = 1 OR
((S2.RiverFlow + S3.RiverFlow) > 9 AND (S2.RiverFlow + S3.RiverFlow) < 14)
Emergency: (S1.SectionState = 2 AND S2.SectionState = 2) OR S3.SectionState = 2 OR
(S2.RiverFlow + S3.RiverFlow) > 13

Phase 4: This phase defines the assets that represent the functionality of the system with four tasks as shown in Fig. 19. First (4.1), identify assets for operationalizations. If the asset requires additional assets, then go straight to task 4.4. Second (4.2), define the additional assets. Third (4.3.), define associations between the additional assets. Fourth (4.4.), link implementation with operationalization. If not all the operationalizations have implementations, return to task 4.1., in other case, the phase ends.

In the running example, we connected each asset to the corresponding operationalization with the implementation association between them, as shown in Fig. 20.

Phase 5: This phase defines claims and soft dependencies. The claims identify the expected satisficing level of a SG according to a set of operationalizations or a conditional expression (Figure 7). On the other

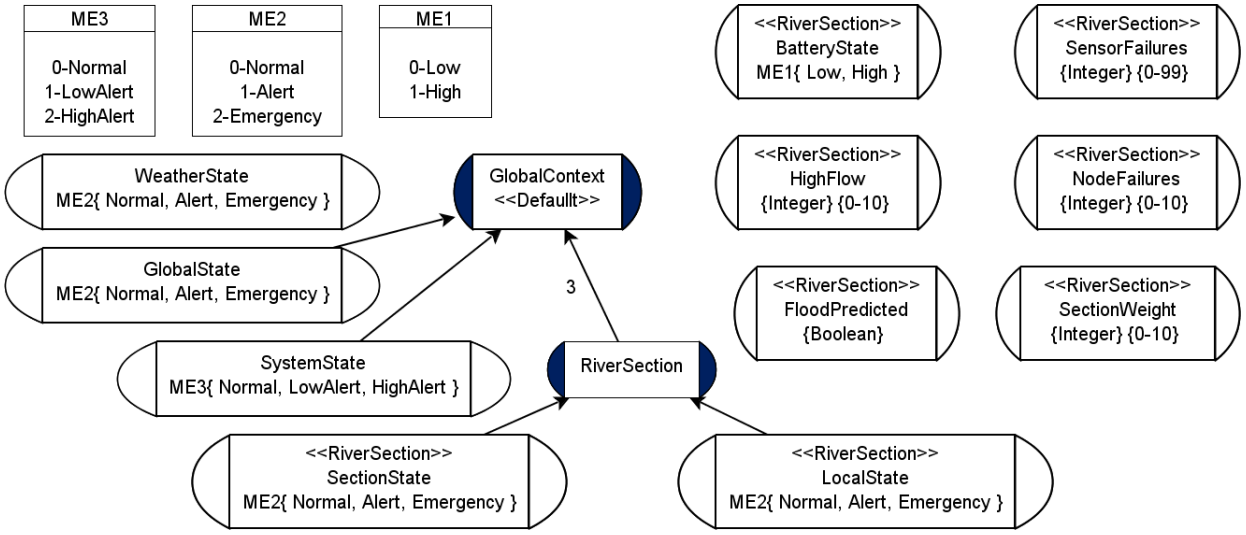


Figure 18: Context view. Variables and concern level.

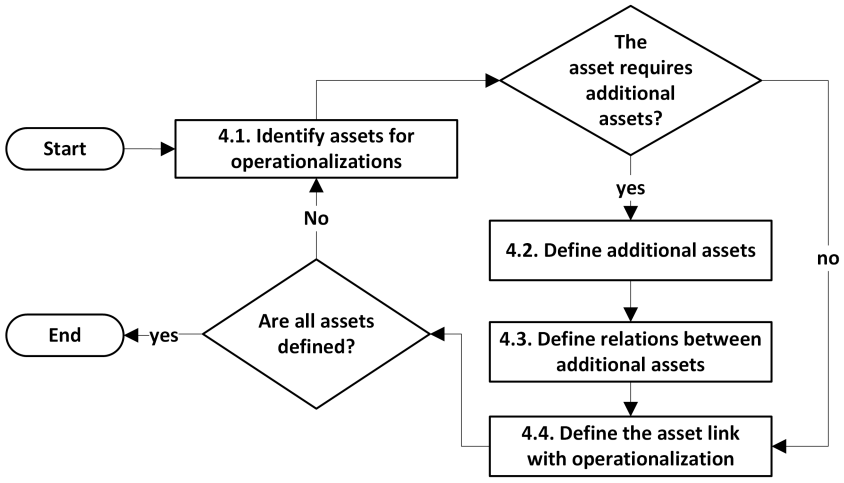


Figure 19: Phase 5: Define assets and associations

hand, SD defines the required satisficing level of a SG when a conditional expression is satisfied. Figure 21 presents this phase divided into eleven tasks. First (5.1), identify a new claim with the operationalization(s) or a condition that affects a SG. If the claim requires a conditional expression, continue to task 5.3. Second (5.2.), review the completeness of required claims of the model. If an additional claim is required, then return to task 5.1., in other case go straight to task 5.7. Third (5.3.), review the required variables for the claim. If all required variables exist, continue to task 5.5. Fourth (5.4), define the additional variables as explained in Phase 4. Fifth (5.5), define the conditional expression for the claim. Sixth (5.6.), define the expected satisficing level of the soft goal, according to the claim. Seven (5.7.), review claims to identify conflicts in the expected satisficing level of each soft goal. A conflict exists when two or more claims can be activated at the same time, and they have different expected levels over the same soft goal. Conflicts represent a blocking problem only if the achievement type of a SG is *as close as possible*. If an additional claim is required, then return to task 5.1. Eighth (5.8), define a new SD, i.e. identify a new conditional expression and its required satisficing level over a SG (Figure 9. If required variables for the condition already exist, then continue to task 5.10. Ninth (5.9), define additional context variables for the new SD. Tenth (5.10.), define the conditional expression for the SD as explained in Section 4.3.5. Eleventh (5.11), define the required satisficing level for the SG. If required, return to task 5.8 to define an additional SD.

In the running example, we defined the claims presented in Fig. 22. The first five iterations identified five claims. The expected satisficing level of *Fault Tolerance* SG varies, according to the operationalizations of the goal *OrganizeNetwork* as defined by CL1 and CL2. The expected satisficing level of *Calculate Flow* SG varies, according to the operationalizations of the goal *Calculate Flow* as defined by claims CL6 and

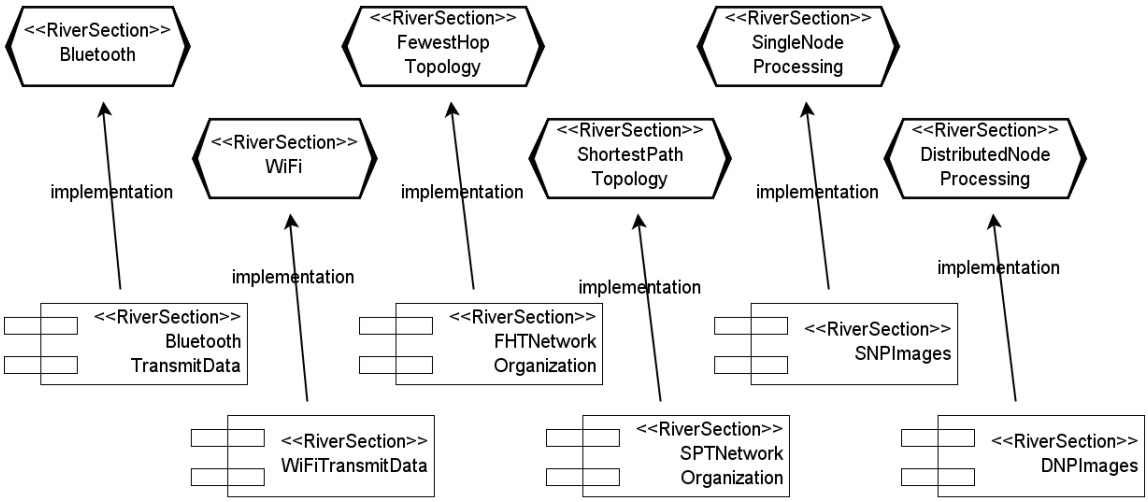


Figure 20: Assets and associations

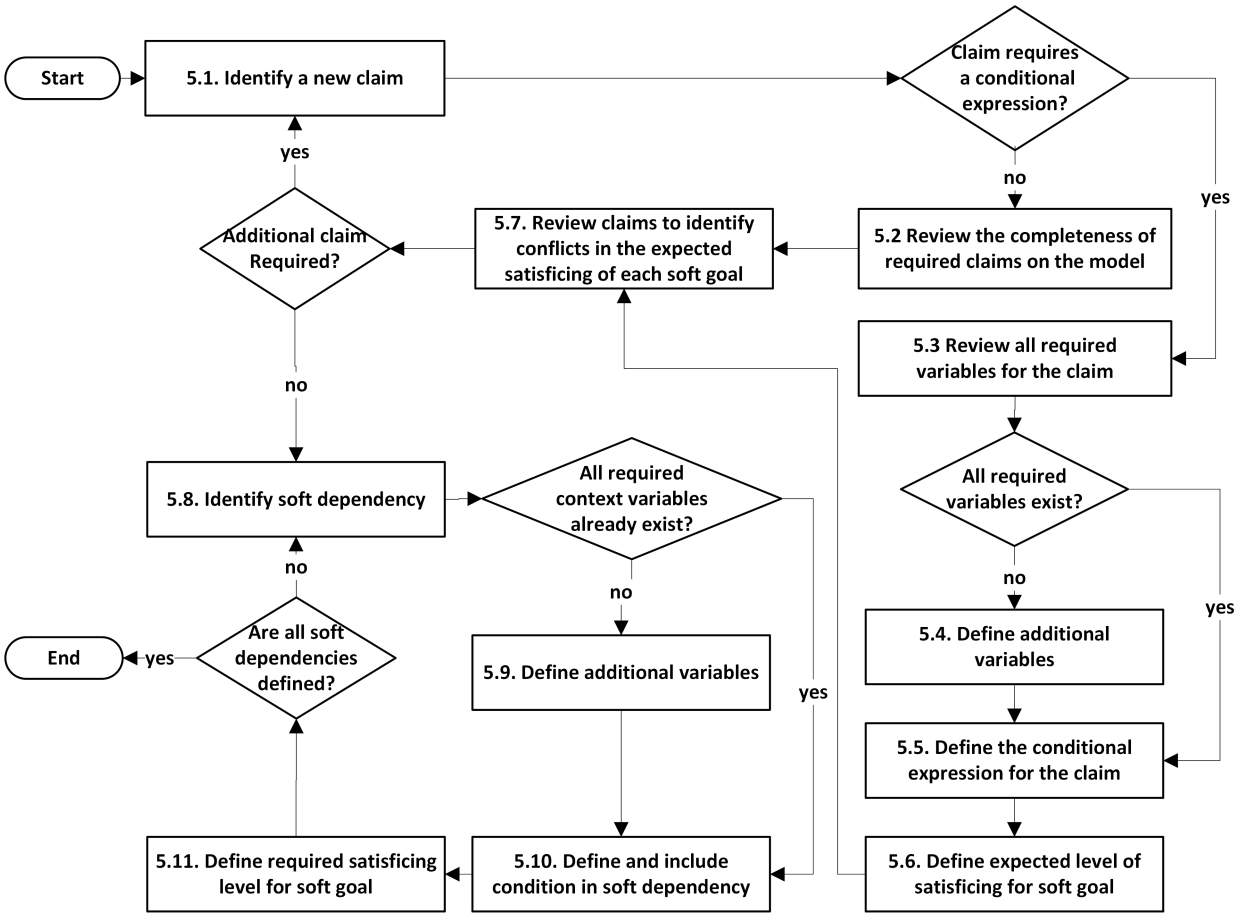


Figure 21: Phase 5: Define claims and soft dependencies

CL7. Only Bluetooth has a good influence on the *Energy Efficiency* SG. Other operationalizations of the River Section has a low expected satisficing level of *Energy Efficiency* SG as shown in CL4 and CL5.

Figure 22 also presents the soft dependencies (SD) for the running example. The first four correspond to SD of the *River Section* (*SD1*, *SD2*, *SD3* and *SD4*). The last three SDs with required levels over the SD of the river section (*SD5*, *SD6*, *SD7* and *SD8*). Each SD has the required satisficing level.

Phase 6: This phase defines the traversal association between variability artifacts (goals, operational-

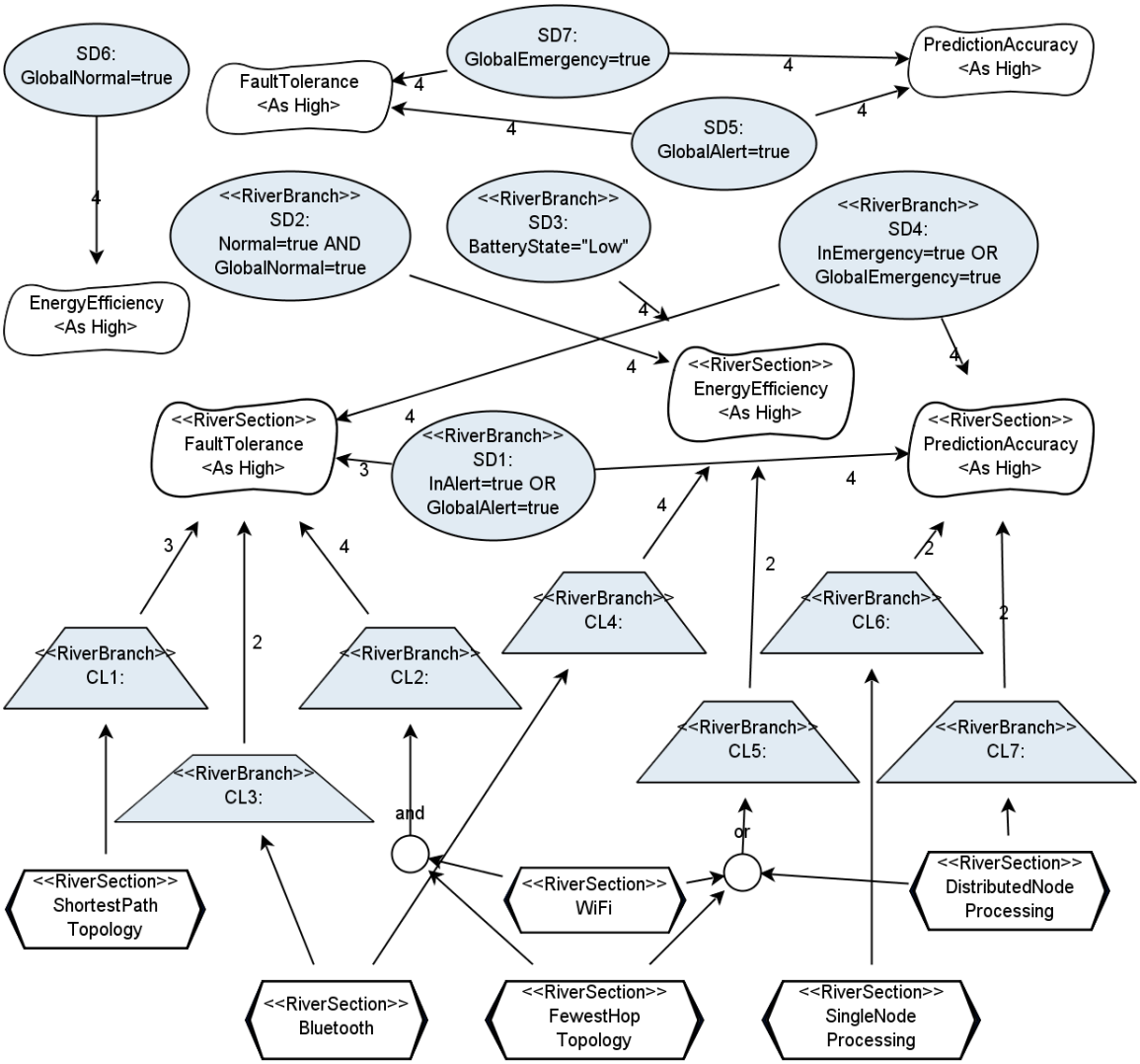


Figure 22: Soft goals satisficing view.

izations, and assumptions). The phase has four tasks shown in Fig. 23. First (6.1.), identify conflicts and traversal associations (conflict, required, alternative or preferred). If no conflict or traversal association is identified, the phase ends. Second (6.2.), define the identified traversal association as tentative. Third (6.3.), evaluate the new association to avoid conflict and redundancies, including the claims and the soft dependencies. Fourth (6.4.), modify or eliminate associations, claims or soft dependencies in conflict or redundant, and then return to task 6.1.

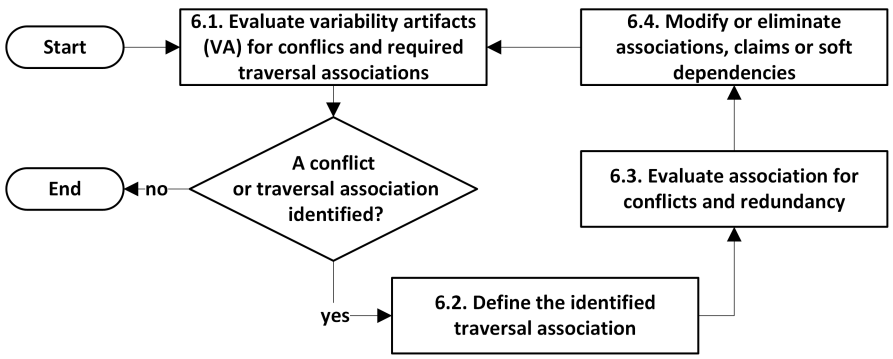


Figure 23: Phase 6: Define traversal associations

In the running example *Bluetooth* and *DistributedNodeProcessing* operationalizations require a conflict association between them, as shown in Fig. 24.

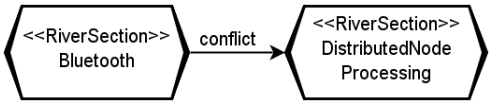


Figure 24: Traversal associations

6 Evaluation

The REFAS language for modeling SAS requirements is based on Sawyer et al. approach [20], borrowing concepts relations and attributes from other approaches. We iterate several times to extend, enrich and clarify our approach. The following are the most relevant advantages found in REFAS through the implementation of VariaMos, the definition of the running example and the feedback obtained from early adopters of REFAS using VariaMos.

Modeling Process The REFAS process enforces the systematic breadth-analysis for dependencies on external factors to mitigate risks due to uncertainty through five default views. The views allow the designer to focus on elements relevant to each concern, reducing the number of concepts and relationships displayed at the same time. The process presented facilitates the systematic and methodical definition of the requirements model.

In-depth Concerns REFAS language supports the in-depth analysis of dependencies with aggregations. Each of the view in REFAS supports the definition of in-depth concerns with relationships at the same level and between levels. These relationships can be direct or indirect through the variables defined by functions.

In the running example, we identify concepts related to a global concern and other concepts related to a particular concern (*River Section*). Goals and soft goals at the global level aggregated other goals and soft goals at the *River Section*. These aggregations support the evaluation of the satisfaction of the system considering the three instances of the river.

Enriched Associations Sawyer et al. [20] supports only exclusive or ("xor") association between goals and its operationalization, and only all ("and") associations between goals; REFAS uses Group MetaOverTwoAssociations in the associations of four of the views. These associations support all ("and"), at least one ("or"), only one ("mutex") and range types of associations. These new types of associations represent more freedom for the designer to define the requirements model. The freedom allows to express to variability in the problem space, i.e., it is possible con consider optional goals. In Sawyer's et al. [20], it is only possible to represent variability at the architectural level (between goals and operationalization), restricted to exclusive associations.

On the other hand, the freedom implies an increased complexity in the verification and simulation. In particular, the or ("or") increases the possible combinations to define valid configurations. We identify that some the valid configurations sets can be treated as a single one. VariaMos should support the automatic definition of those sets. This support requires the adaptation of actual VariaMos implementation.

In the running example, we applied different group associations in two of the claims (CL2 and CL5).

Conditional Expressions Claims and soft dependencies support conditional expressions. With conditional expressions, the language is more expressive compared to Sawyer et al.[20] to represent complex conditions. In the running example, combined expressions allow the definition of complex conditions to define the soft dependencies satisfaction. This kind of definition was not possible in other approaches, including Sawyer et al., were only specific variable values are supported.

Conditional expressions in claims allow the specification of special parameter or configuration for operationalizations thought variables. These conditional expressions avoid the creation of several similar operationalization within the model. In REFAS, conditional expressions in claims and soft dependencies support then reasoning over soft goal satisficing. Except Sawyer et al. [20], other proposals found not do not include the reasoning over soft goals in the way we proposed.

Variable Expressions Variables with calculated values include a set of expression. With these expressions, the language is more expressive to represent more complex variables required to address some problems. In the running example, variable expressions allowed to calculate the state of the system, the state of the river and failures at the global level, according to the characteristics of the concern levels. These calculations are not possible in other approaches such as Sawyer et al. [20].

Meta-model and Process Extensions REFAS meta-model is not rigid in its definition and modeling process, it supports three type of extensions. First, the MetaPairwiseAssociation between concepts can include a new type of association if required by the designer. Also, the designer can define new group associations types. Second, the concepts and associations allow the addition of new attributes. Third, the meta-model itself is extensible. New concepts and associations can be defined as well as even the definition of new views for the requirements model.

The possibility of extending the REFAS meta-model offers an important alternative to address uncertainty. The designer has the control whenever the existing concepts or associations are not enough to represent the appropriate detail level of the runtime system concerns. Hypothetically, if the designer would need an additional group association between operationalizations and claims, where this association activates the claim when no operationalizations are selected, then the designer would add the new type of group association in the meta-model, and would apply it to the appropriate claims.

It is worth noting that the designer must provide the operational semantics for new concepts or associations as they already exist for other concepts and associations of the meta-model. The operational semantics defines the behavior of the requirements model in operations such as verification and simulation. However, its detailed explanation and definition are beyond the scope of this paper.

Final remarks We have reviewed many aspects of our language through the implementation of REFAS in VariaMos. We also received feedback from early users and tried to incorporate the suggestions and concerns at the tool and language level. The initial results suggest that the language can be used to model different kind of SAS systems. Despite the initial results, we consider that more experimental information and formal validation is required to identify whether the meta-model require refinements and evaluate its applicability to complex problems in different environments. These refinements have the purpose of making a more general and complete framework for modeling self-adaptive systems requirements.

We are working to define formal case studies for evaluation purposes. From one of the case studies we are working on, we found relevant to include the support of constraints in transitions between stable configurations in the REFAS language. We are evaluating the best alternatives and the implications of the incorporation of transitions to our language.

Finally, conditional expressions and variable expressions rely on a set of logical and numerical expressions available in VariaMos. We are considering to extend the set of logical and numerical expressions supported. Additional expressions will allow the definition of more complex and relevant expression by the designers.

7 Related Work and Discussion

For analyzing and eliciting requirements for self-adaptive systems, several approaches have been presented. This section presents a discussion of representative ones.

NFR Framework: Chung et al. [15] proposed a framework focused on soft goals (SG) to manage uncertainty in software systems. The framework consists of a design process of SG, broken down into more specific SG and then into operationalizations. Some operationalizations are chosen, and others rejected by the designer. The impact of these decisions on the top soft goals is then evaluated. Between operationalization and SG, positive and negative impact contributions can be identified. In those contributions, the justification of decisions are presented using claims. In REFAS, the claims are not limited with only positive or negative contributions. Moreover, claims in REFAS include conditional expressions to control its activation.

KAOS: KAOS (Knowledge Acquisition and autOmated Specification) proposed by Dardenne et al. [13] has three main aspects. First, conceptual elicitation and modeling of functional and non-functional requirements. Second, the acquisition strategies to requirements modeling. Third, an automated assistant that guides the processes for acquisition strategies. KAOS is considered a methodology, as it provides both the modeling language and the method. KAOS focuses on late requirements and only partially in the early elicitation and architectural design. Late requirements focus on modeling the system within its environment [14]. Additional to these requirements, in REFAS we can model changes in the environment that affect the satisfaction of soft goals. Also in REFAS the improvement in the variability modeling is twofold. First, the refinement supports “mutex” and “range” in addition to “and” and “or” available by KAOS. Second, the

relations available are "prefer", "require" and "alternative" in addition to the conflict relation available in KAOS.

*i**: Yu et al. proposed the i* framework for requirements elicitation in the early stages [26]. This framework includes the concepts of goals, tasks, resources and soft goals, some taken from the NFR framework. i* focuses on early elicitation and partially covers late elicitation. In i*, goals can be refined into more specific goals, to the level of tasks. However, it is not possible to state whether the goals or tasks are optional.

RELAX: Whittle et al. [27][28] proposed RELAX, a language for handling uncertainty in requirements for self-adaptive systems. Uncertainty is represented using requirements whose fulfillment depend on environmental conditions or the changing needs of the same conditions. RELAX defines a vocabulary with a clear syntax for stating such requirements. The proposed language is semantically formalized via a branch temporal fuzzy logic. RELAX provides a syntax and semantics for defining the requirements under uncertainty, without going into specific system views or considering other relevant concepts such as soft goals. An example they propose, "The synchronization process must be run as many times as possible.".

Cheng et al [4] proposed an extension to RELAX for modeling adaptive system using goals. With goals it is possible to identify uncertainty factors and mitigate them through relaxation, the addition of subgoals or the addition of a higher-level goal.

Tropos: is a methodology for software development requirements from early elicitation to implementation. It includes late elicitation, architectural design and detailed design [17], [18]. In the elicitation phase, early and late elicitation phases are differentiated, but they use the same conceptual and methodological approach. In the early elicitation, Tropos defines system goals associated with actors. Then in the late elicitation, the conceptual model can be extended. The approach differentiates between goals and SG.

Techne: is an abstract language to create languages for modeling proposed requirements by Jureta et al. [29]. Techne is proposed as a core for creating new languages for early elicitation of requirements and does not define a visual modeling syntax. In Techne, soft goals are not refined and maintained as soft goals. Instead, they are represented by a network of functional requirements that are an approximation of the initial soft goals. Techne focuses on early requirements, and it does not define any visual representation.

Adaptive RML: proposed by Qureshi et al. [19], it provides the syntax for modeling the elicitation and representation of requirements for self-adaptive systems. It maintains alignment with the goals of modeling languages and is mapped to Techne. Adaptive RML allows the designer to specify the goals satisfaction in terms of required and optional tasks including relegation of relationships. The relegation establishes an alternative between two or more requirements so that a mandatory requirement can be replaced by an optional one. The concepts in Adaptive RML include goals, soft goals, domain assumptions, quality constraints (QCs) and tasks. A QC relates a task or goal to soft goals, including a text representing the condition. The QC are clear to represent the conditions on the model but we consider difficult the use of plain text to validate alternatives. In our approach, Boolean and Numerical expressions support conditional expressions in claims, making claims ready for automatic reasoning. Moreover, claims in REFAS support different expected satisficing levels over soft goals, not only the complete satisfaction.

Souza et al. [30] propose Awareness Requirements (AwReqs) being a new class of requirements that can monitor the success or failure of other requirements. The monitored requirements can be soft goals, as well as goals, tasks and domain assumptions. AwReqs introduces a new class of conditions to impose restrictions on the behavior of the monitored requirements. At the monitored level, AwReqs represents the contribution relation between goals or tasks to soft goals with a direct relation, including the positive or negative contribution. In REFAS, the relation is represented using claims supporting multiple operationalizations and a conditional expression. REFAS supports different satisficing levels of soft goals but does not consider the control of requirements violation in the way is supported by AwReqs.

Sawyer et al. [20] propose a goal model inspired on KAOS. This proposal uses GridStix as example, the simulation of a physical system monitoring using wireless sensors. This system has been used to ensure security on the banks of the River Ribble and Dee in England. The solution proposed by Sawyer et al. [20] is exemplified in the context of such systems. This proposal, with its case study, is formalized and implemented using a simulation, the execution as being both supported by constraint programming simulations. With constraint programming, it is possible to define optimization functions and find the solution set considering the system variables and the external context.

The approach of Song et al. [31] for adaptation planning of self-adaptive systems includes user preferences and configurations. The user preferences and configurations together with constraints are resolved as a constraint satisfaction problem (CSP) at runtime. The CSP objective is to satisfy as many goals as possible. Song et al. proposed the use of variables for configurations and user decisions [31]. In this sense, REFAS also includes variables sensed from the environment and the target system. REFAS focus on the variation of soft goals satisficing based on variables changes, while Song et al. [31] propose to modify the relative weight assigned to goals, seeking first to meet the goals most relevant to the user.

Rainbow: Garlan et al [32] proposed an architecture based self-adaptation approach with a reusable infrastructure. To be reusable, they provide four layers: system, architecture, translator and adaptation knowledge. The last layer is specific to the target system. They also provide a feedback loop to communicate with the target system. The target system is monitored and adapted with the feedback loop decisions. Rainbow [32] focuses only on architecture and does not propose a representation for top concepts such as goals and soft goals.

8 Conclusions and Future Work

In this paper we presented REFAS, our framework for specifying requirements models, which focus on reducing uncertainty by resolving two challenges. First, REFAS offers a language with concrete syntax and semantics to model adaptation requirements from different viewpoints and concerns (in the broad aspects). Second, REFAS allows to define concern levels from the in-depth analysis of the different viewpoints and concerns identified by the breadth analysis, along with their inter-relationships. The concern levels can be associated to concepts and variables that represent the relevant context.

The REFAS framework is based on a generic meta-model, its core subsystem, which defines the fundamental meta-concepts and meta-associations. Using this meta-model, we define the REFAS language's concrete syntax and semantics. We also describe in detail a modeling process to facilitate the correct and methodical definition of requirements models, by explicitly considering uncertainty. We show the application of REFAS to a running example, and evaluate it by also analyzing its differences with the proposal of Sawyer et al. [20], the approach that inspired our language. We have also analyzed other approaches in the requirements specification for self-adaptive systems, by comparing them to our approach.

Additionally to systematic breath and depth analysis, REFAS presents several benefits compared to other approaches such as Sawyer et al.[20]. First, REFAS provides enriched associations to express variability in the problem space and the solution space, and also to associate many operationalizations to a claim. Second, REFAS offers conditional expressions in claims and soft dependencies, supporting the combination of variables using logical and numerical expressions in composed expressions. The composed expressions support a better definition of conditions for the addressed problem. Third, expressions for variables support the incorporation of variables controlled by the reasoning process to summarize results and control external actions. Fourth, the approach is based on a common and extensible meta-model. This meta-model definition gives the designer more freedom compared to restrictions imposed by other languages. This freedom is represented in the three extensions supported by REFAS. That is, the support for new association types, new attributes for concepts, and new elements (meta-concepts, meta-associations and meta views).

We developed VariaMos software tool as a vehicle to validate and experiment with the REFAS approach. VariaMos is a partial implementation of REFAS at the syntax and the operational semantics level. A set of expressions defines the operational semantics, according to meta-concepts and meta-associations. Those expressions allow reasoning about the requirements model. We are working to extend the operational semantic in VariaMos to support our second challenge. The benefits of the extension are twofold. First, VariaMos will perform verifications and simulation on a partial specification of the adaptation requirements. Second, VariaMos will aggregate the sub-specification results and obtain general information about the verification and simulation operations.

In addition to the REFAS implementation and validation, we are conducting a systematic mapping study (SMS) of the literature on requirements engineering for self-adaptive systems. This SMS will allow us to identify additional approaches with different contributions. With the analysis of these approaches, we will refine and enrich our models and the adaptability of the target systems.

Finally, we plan to apply the REFAS framework to other case studies to evaluate its concrete syntax and semantics, the modeling process and refine and improve them. We also plan to integrate REFAS with other frameworks for designing, realizing and deploying self-adaptive software systems such as QoS-CARE [33] and others.

References

[1] B. H. C. Cheng, R. de Lemos, H. Giese, P. Inverardi, J. Magee, J. Andersson, B. Becker, N. Bencomo, Y. Brun, B. Cukic, G. D. M. Serugendo, S. Dustdar, A. Finkelstein, C. Gacek, K. Geihs, V. Grassi, G. Karsai, H. M. Kienle, J. Kramer, M. Litoiu, S. Malek, R. Mirandola, H. A. Müller, S. Park, M. Shaw, M. Tichy, M. Tivoli, D. Weyns, and J. Whittle, "Software engineering for self-adaptive systems: A research roadmap," in *Software Engineering for Self-Adaptive Systems*, ser. Lecture Notes in Computer Science, B. H. C. Cheng, R. de Lemos, H. Giese, P. Inverardi, and J. Magee, Eds., vol. 5525. Springer, 2009, pp. 1–26.

[2] N. Esfahani and S. Malek, "Uncertainty in self-adaptive software systems," in *Software Engineering for Self-Adaptive Systems II - International Seminar, Dagstuhl Castle, Germany, October 24-29, 2010 Revised Selected and Invited Papers*, ser. Lecture Notes in Computer Science, R. de Lemos, H. Giese, H. A. Müller, and M. Shaw, Eds., vol. 7475. Springer, 2010, pp. 214–238. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-35813-5_9

[3] S.-W. Cheng and D. Garlan, "Handling uncertainty in autonomic systems," 2007.

[4] B. H. C. Cheng, P. Sawyer, N. Bencomo, and J. Whittle, "A goal-based modeling approach to develop requirements of an adaptive system with environmental uncertainty," in *Model Driven Engineering Languages and Systems, 12th International Conference, MODELS 2009, Denver, CO, USA, October 4-9, 2009. Proceedings*, ser. Lecture Notes in Computer Science, A. Schürr and B. Selic, Eds., vol. 5795. Springer, 2009, pp. 468–483. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-04425-0_36

[5] L. Baresi, L. Pasquale, and P. Spoletini, "Fuzzy goals for requirements-driven adaptation," in *RE 2010, 18th IEEE International Requirements Engineering Conference, Sydney, New South Wales, Australia, September 27 - October 1, 2010*. IEEE Computer Society, 2010, pp. 125–134. [Online]. Available: http://dx.doi.org/10.1109/RE.2010.25

[6] A. M. Elkhodary, N. Esfahani, and S. Malek, "FUSION: a framework for engineering self-tuning self-adaptive software systems," in *Proceedings of the 18th ACM SIGSOFT International Symposium on Foundations of Software Engineering, 2010, Santa Fe, NM, USA, November 7-11, 2010*, G. Roman and K. J. Sullivan, Eds. ACM, 2010, pp. 7–16. [Online]. Available: http://doi.acm.org/10.1145/1882291.1882296

[7] R. Mazo, J. C. Muñoz-Fernández, L. Rincón, S. Camille, and T. Gabriel, "Variamos: an extensible tool for engineering (dynamic) product lines." in *19th International Software Product Line Conference SPLC 2015, Nashville-USA, July 20-24, 2015. (Volume 1)*, 2015.

[8] D. Hughes, P. Greenwood, G. Blair, G. Coulson, P. Smith, and K. Beven, "An intelligent and adaptable grid-based flood monitoring and warning system," 2006.

[9] P. Grace, D. Hughes, B. Porter, G. S. Blair, G. Coulson, and F. Taiani, "Experiences with open overlays: A middleware approach to network heterogeneity," in *Proceedings of the 3rd ACM SIGOPS/EuroSys European Conference on Computer Systems 2008*, ser. Eurosys '08. New York, NY, USA: ACM, 2008, pp. 123–136. [Online]. Available: http://doi.acm.org/10.1145/1352592.1352606

[10] N. Bencomo, P. Grace, C. Flores, D. Hughes, and G. Blair, "Genie: Supporting the model driven development of reflective, component-based adaptive systems," in *Proceedings of the 30th International Conference on Software Engineering*, ser. ICSE '08. New York, NY, USA: ACM, 2008, pp. 811–814. [Online]. Available: http://doi.acm.org/10.1145/1368088.1368207

[11] D. Hughes, P. Greenwood, G. Coulson, and G. Blair, "Gridstix: supporting flood prediction using embedded hardware and next generation grid middleware," in *World of Wireless, Mobile and Multimedia Networks, 2006. WoWMoM 2006. International Symposium on a*, 2006, pp. 6 pp.–626.

[12] H. McMillan, J. Freer, F. Pappenberger, T. Krueger, and M. Clark, "Impacts of uncertain river flow data on rainfall-runoff model calibration and discharge predictions," *Hydrological Processes*, vol. 24, no. 10, pp. 1270–1284, 2010. [Online]. Available: http://dx.doi.org/10.1002/hyp.7587

[13] A. Dardenne, A. van Lamsweerde, and S. Fickas, "Goal-directed requirements acquisition," *Sci. Comput. Program.*, vol. 20, no. 1-2, pp. 3–50, 1993.

[14] A. Lapouchnian, "Goal-Oriented Requirements Engineering: An Overview of the Current Research," Department of Computer Science, University of Toronto, Toronto, Canada, Tech. Rep., Juni 2005.

[15] L. Chung, B. A. Nixon, E. Yu, and J. Mylopoulos, *Non-Functional Requirements in Software Engineering*, ser. The Kluwer international series in software engineering. Kluwer Academic Publishers Group, Dordrecht, Netherlands, 1999.

[16] R. Darimont, E. Delor, P. Massonet, and A. van Lamsweerde, "GRAIL/KAOS: an environment for goal-driven requirements engineering," in *Pulling Together, Proceedings of the 19th International Conference on Software Engineering, Boston, Massachusetts, USA, May 17-23, 1997.*, W. R. Adrion, A. Fuggetta, R. N. Taylor, and A. I. Wasserman, Eds. ACM, 1997, pp. 612–613. [Online]. Available: http://doi.acm.org/10.1145/253228.253499

[17] F. Giunchiglia, J. Mylopoulos, and A. Perini, "The tropos software development methodology: Processes, models and diagrams," in *AOSE*, ser. Lecture Notes in Computer Science, F. Giunchiglia, J. Odell, and G. Weiß, Eds., vol. 2585. Springer, 2002, pp. 162–173.

[18] P. Bresciani, A. Perini, P. Giorgini, F. Giunchiglia, and J. Mylopoulos, "Tropos: An agent-oriented software development methodology," *Autonomous Agents and Multi-Agent Systems*, vol. 8, no. 3, pp. 203–236, 2004.

[19] N. A. Qureshi, I. Jureta, and A. Perini, "Towards a requirements modeling language for self-adaptive systems," in *REFSQ*, ser. Lecture Notes in Computer Science, B. Regnell and D. E. Damian, Eds., vol. 7195. Springer, 2012, pp. 263–279.

[20] P. Sawyer, R. Mazo, D. Diaz, C. Salinesi, and D. Hughes, "Using constraint programming to manage configurations in self-adaptive systems," *IEEE Computer*, vol. 45, no. 10, pp. 56–63, 2012.

[21] K. Welsh and P. Sawyer, "Requirements tracing to support change in dynamically adaptive systems," in *Requirements Engineering: Foundation for Software Quality, 15th International Working Conference, REFSQ 2009, Amsterdam, The Netherlands, June 8-9, 2009, Proceedings*, ser. Lecture Notes in Computer Science, M. Glinz and P. Heymans, Eds., vol. 5512. Springer, 2009, pp. 59–73. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-02050-6_6

[22] K. Welsh, N. Bencomo, and P. Sawyer, "Tracing requirements for adaptive systems using claims," in *TEFSE'11, Proceedings of the 6th International Workshop on Traceability in Emerging Forms of Software Engineering, May 23, 2011, Waikiki, Honolulu, HI, USA*, D. Poshyvanyk, M. D. Penta, and H. H. Kagdi, Eds. ACM, 2011, pp. 38–41. [Online]. Available: http://doi.acm.org/10.1145/1987856.1987865

[23] E. Guerra, J. de Lara, A. Malizia, and P. Díaz, "Supporting user-oriented analysis for multi-view domain-specific visual languages," *Information & Software Technology*, vol. 51, no. 4, pp. 769–784, 2009.

[24] J. C. Muñoz-Fernández, G. Tamura, R. Mazo, and C. Salinesi, "Towards a requirements specification multi-view framework for self-adaptive systems," in *2014 XL Latin American Computing Conference (CLEI), Montevideo, Uruguay, Septiembre 15-19, 2014*. IEEE, 2014.

[25] D. L. Moody, "The "physics" of notations: a scientific approach to designing visual notations in software engineering," in *ICSE (2)*, J. Kramer, J. Bishop, P. T. Devanbu, and S. Uchitel, Eds. ACM, 2010, pp. 485–486.

[26] E. S. K. Yu, "Towards modelling and reasoning support for early-phase requirements engineering," in *Proceedings of the 3rd IEEE International Symposium on Requirements Engineering*, ser. RE '97. Washington, DC, USA: IEEE Computer Society, 1997, pp. 226–. [Online]. Available: http://dl.acm.org/citation.cfm?id=827255.827807

[27] J. Whittle, P. Sawyer, N. Bencomo, B. H. C. Cheng, and J. Bruel, "RELAX: incorporating uncertainty into the specification of self-adaptive systems," in *RE 2009, 17th IEEE International Requirements Engineering Conference, Atlanta, Georgia, USA, August 31 - September 4, 2009*. IEEE Computer Society, 2009, pp. 79–88. [Online]. Available: http://dx.doi.org/10.1109/RE.2009.36

[28] J. Whittle, P. Sawyer, N. Bencomo, B. H. C. Cheng, and J.-M. Bruel, "RELAX: a language to address uncertainty in self-adaptive systems requirement," *Requir. Eng.*, vol. 15, no. 2, pp. 177–196, 2010.

[29] I. Jureta, A. Borgida, N. A. Ernst, and J. Mylopoulos, "Techne: Towards a new generation of requirements modeling languages with goals, preferences, and inconsistency handling," in *RE*. IEEE Computer Society, 2010, pp. 115–124.

[30] V. E. S. Souza, A. Lapouchnian, W. N. Robinson, and J. Mylopoulos, "Awareness Requirements for Adaptive Systems," in *Proc. of the 6$^{\text{th}}$ International Symposium on Software Engineering for Adaptive and Self-Managing Systems*. ACM, 2011, pp. 60–69.

[31] H. Song, S. Barrett, A. Clarke, and S. Clarke, "Self-adaptation with end-user preferences: Using runtime models and constraint solving," in *MoDELS*, ser. Lecture Notes in Computer Science, A. Moreira, B. Schätz, J. Gray, A. Vallecillo, and P. J. Clarke, Eds., vol. 8107. Springer, 2013, pp. 555–571.

[32] D. Garlan, S.-W. Cheng, A.-C. Huang, B. R. Schmerl, and P. Steenkiste, "Rainbow: Architecture-based self-adaptation with reusable infrastructure," *IEEE Computer*, vol. 37, no. 10, pp. 46–54, 2004.

[33] G. Tamura, R. Casallas, A. Cleve, and L. Duchien, "QoS Contract Preservation through Dynamic Reconfiguration: A Formal Semantics Approach," *Science of Computer Programming (SCP)*, vol. 94, no. 3, pp. 307–332, 2014. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0167642313003390