Softw Syst Model (2018) 17:1339–1364
https://doi.org/10.1007/s10270-016-0562-9

REGULAR PAPER

Efficient parallel reasoning on fuzzy goal models for run time requirements verification

George Chatzikonstantinou[1] · Kostas Kontogiannis[2]

Received: 16 July 2015 / Revised: 17 September 2016 / Accepted: 18 September 2016 / Published online: 30 September 2016

Abstract As software applications become highly interconnected in dynamically provisioned platforms, they form the so-called systems-of-systems. Therefore, a key issue that arises in such environments is whether specific requirements are violated, when these applications interact in new unforeseen ways as new resources or system components are dynamically provisioned. Such environments require the continuous use of frameworks for assessing compliance against specific mission critical system requirements. Such frameworks should be able to (a) handle large requirements models, (b) assess system compliance repeatedly and frequently using events from possibly high velocity and high frequency data streams, and (c) use models that can reflect the vagueness that inherently exists in big data event collection and in modeling dependencies between components of complex and dynamically re-configured systems. In this paper, we introduce a framework for run time reasoning over medium and large-scale fuzzy goal models, and we propose a process which allows for the parallel evaluation of such models. The approach has been evaluated for time and space performance on large goal models, exhibiting that in a simulation environment, the parallel reasoning process offers significant performance improvement over a sequential one.

Communicated by Prof. Marsha Chechik.

✉ George Chatzikonstantinou
gchatzik@cslab.ece.ntua.gr

Kostas Kontogiannis
kostas@csd.uwo.ca

1 Department of Electrical and Computer Engineering, National Technical University of Athens, Athens, Greece

2 Department of Computer Science, Western University, London, Canada

Keywords Goal models reasoning · Fuzzy reasoning · Parallel reasoning · Software systems · Verification

1 Introduction

Over the past few years, we witness a paradigm shift toward the development, deployment, and operation of large-scale software systems. Advances in middleware technologies make also possible the streamlined integration of diverse systems and applications. As such systems are integrated and expanded in scope, they form the so-called ultra-large-scale systems or systems-of-systems [2,39]. Such systems utilize heterogeneous monitoring infrastructures which produce log data at high rates and from various sources [45]. Furthermore, virtualization technologies make possible the dynamic provision of resources in order to meet varying run time operational needs. Such dynamic adaptation may take the form of provisioning new virtual platforms, migrating processes to different virtual machines as well as transferring, splitting, or distributing data and computation logic to different servers [10,15].

Consequently, the functional and non-functional requirements as well as the global constraints, and the invariants of such integrated and virtualized systems, become very complex and interrelated, resulting in models with more that 10000 requirements, a context that Wnuk et al. refer to as Very Large-Scale Requirements Engineering (VLSRE) [48,49]. Examples of such interrelated requirements and constraints include throughput, security, privacy, energy consumption, and optimal utilization of resources, to name a few. It is therefore of no surprise that run time requirements verification and compliance become focal points during the deployment and operation of such systems.

In this respect, the software engineering community has investigated goal models [26,33] as means to represent and denote such diverse and possibly conflicting software requirements. In addition, a number of techniques have also been proposed for analyzing, processing, and reasoning on such models, in a quest to assess the impact one requirement may have on other requirements, how alternative design decisions may affect system goals, or to evaluate whether specific requirements can be satisfied given a set of conditions and constraints [6,23,34]. However, many of these modeling and reasoning techniques have been developed to be applied off-line, during the specification, or the design phase of a software system. On the other hand, techniques that can be applied at run time are mostly focusing on requirements adaptation, or on probabilistic reasoning over goal models [9].

In this paper, we present a complementary to probabilistic reasoning approach that aims to model by a confidence score the strength by which an expert modeler considers that one requirement or system configuration affects another. More specifically, we build on previous work we have conducted [13,14], and we introduce a fuzzy goal model reasoning process which can be parallelized and applied at run time. Such parallelization serves to tackle tractability issues either when large-scale models are involved or when models should be repeatedly and frequently evaluated against high velocity and high frequency logged event streams. We propose the concept of *reasoning units* and a novel framework that allows for the formulation of *reasoning sub-plans* that can be evaluated concurrently. In this work, we adapt and utilize a model transformation engine introduced in [12], in order to transform goal models into fuzzy rules. However, it is important to note that the proposed framework is not bound to the use of fuzzy logic reasoners as it also provides extension points that enable fuzzy reasoning to be replaced by any other form of reasoning under uncertainty.

We consider that this work contributes in two main areas. The first area deals with the parallelization of the goal model reasoning process. As requirements and policy models become more complex and interdependent, the parallelization of the reasoning process addresses tractability issues by providing means to distribute the reasoning load among multiple threads so as to enable the analysis over many different streams of logged events produced during the operation of a system. The second area deals with the investigation of fuzzy reasoners for the run time evaluation of goal models. Even though a number of goal model reasoning frameworks that apply fuzzy logic principles have been proposed in the literature [8,13], our approach has three notable differences. The first difference relates to the objectives and intended use of the framework. In [8], the focus is on determining a certain set of adaptation countermeasures that can be applied so that certain system goals can be fulfilled. Our approach aims to evaluate whether and to what degree (fuzzy truth value) a goal is satisfied when the system is physically altered or modified. The second difference relates to the run time framework itself. In [8], the run time framework depends on a temporal language which will have to execute on a specialized virtual machine or use a special purpose compiler. Our approach utilizes standardized open source fuzzy reasoning engines making it thus readily available as a library or standalone application. The third difference deals with run time performance and scalability. In [13], the focus is on the sequential evaluation of fuzzy rules that are generated from fuzzy goal models, while in the approach presented here is the design and application of a technique that allows for the parallel evaluation of fuzzy goal models. Such parallelization addresses tractability issues either when very large models are considered, or (instance level) models have to be repeatedly and frequently evaluated against high-velocity and high-frequency input data.

This paper is organized as follows. Section 2 presents the motivation for this work and outlines the proposed framework process. Section 3 presents related work, while Sect. 4 discusses fuzzy goal models. Section 5 discusses the generation of the reasoning units. Subsequently, Sect. 6 discusses the identification of goal model dependencies and the extraction of a sequential reasoning plan that respects all dependencies that exist between the reasoning units. Section 7 describes a goal model reasoning parallelization technique. Section 8 presents experimental results and discusses the threats to the validity of the proposed approach. Section 9 describes the application of the proposed reasoning technique in a real-life working example. Section 10 concludes the paper and provides pointers for future research.

2 Motivation and process outline

2.1 Motivation

Over the past few years, we have experienced a significant growth on the deployment of large-scale highly interconnected systems that operate in a number of application domains such as banking, retail, commerce, and government. These systems encompass complex requirements that involve large models [33]. The continuous evaluation of such large models becomes of key factor for ensuring overall functional and non-functional requirements system compliance, especially when many different components, applications, and virtual machines are added or deleted in dynamically provisioned operating environments.

The problem has been recognized in the related research literature as an important one [19,46], both from the requirements verification perspective and from the policy compliance and security perspective. Therefore, it is important to be able to evaluate at *run time*, whether changes in system

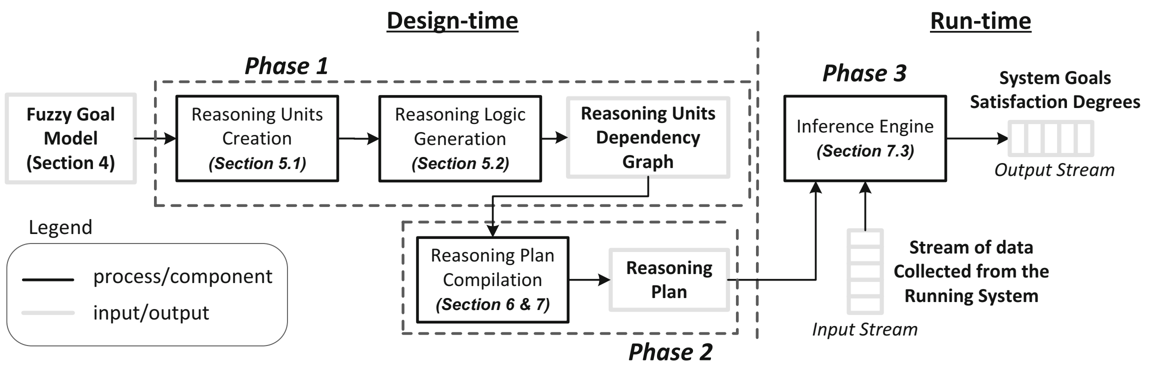

Design-time
Run-time
Phase 1
Fuzzy Goal Model (Section 4)
Reasoning Units Creation (Section 5.1)
Reasoning Logic Generation (Section 5.2)
Reasoning Units Dependency Graph
Phase 3
Inference Engine (Section 7.3)
System Goals Satisfaction Degrees
Output Stream
Legend
process/component
input/output
Reasoning Plan Compilation (Section 6 & 7)
Reasoning Plan
Phase 2
Stream of data Collected from the Running System
Input Stream

Fig. 1 Outline of the proposed process

topology, resources, or operating environments affect specific system requirements or violate system constraints.

Goal models and goal model analysis techniques have been extensively applied in areas related to requirements engineering, software engineering, information systems, and enterprise modeling [26]. In the context of this paper, we focus on reasoning techniques that can be used to perform analysis over goal models in order to answer questions like "To what degree does the running system satisfy certain goals?" or "Which requirements are satisfied below a certain threshold due to dynamic system reconfiguration?". More specifically, given a set of events that are collected from the running system, and a set of goal models, we need to evaluate whether system requirements hold and to what degree as the system is dynamically altered. This problem implies a forward propagation satisfaction analysis for which both quantitative and qualitative approaches have been proposed in the literature [28]. Additionally, as such systems use heterogeneous monitoring infrastructures which produce log data at high rates from various sources, monitoring and reasoning techniques used for run time analysis should be able to cope with high-volume, high-frequency, and high-veracity event streams.

To date, the software engineering community has proposed a number of models, frameworks, and tools that allow for the specification, analysis, and evaluation of system requirements and goal models. However, there is still limited work performed on parallelizing the reasoning process on large requirements models using real-time streaming data, especially when the dependencies and impact of one system requirement to other system requirements cannot be fully and deterministically modeled, due to the high complexity of structural and behavioral inter-dependencies entailed in such complex systems.

In this respect, we define a model transformation process that allows for the generation of *reasoning units* from a given goal model. As it will be discussed in more detail in Sect. 5.2, each such reasoning unit is annotated with a certain *reasoning logic*. Subsequently, this allows for the use of an appropriate inference engine in order to perform deductions on each reasoning unit, by taking into account events collected from the running system. Furthermore, we propose a technique for analyzing the dependencies between these reasoning units, so as to be able to identify collections of reasoning units that can be evaluated in parallel. This parallelization of the reasoning process can assist toward processing large streams of data collected from the running system in a tractable way.

Finally, we investigate the application of fuzzy controllers' principles for reasoning over goal models in which contribution links are annotated with *contribution confidence* values, and goals can hold with a degree of truth, as means for designing models that can reflect the vagueness that inherently exists in reasoning with incomplete or uncertain data or system dependencies.

2.2 Process outline

The proposed framework entails three phases that are associated with a design-time and a run time component. The steps required for each phase to be completed along with the sections in which each step is described are depicted in Fig. 1. The components given in black correspond to processes, while gray boxes denote outputs produced by these processes or inputs required from them.

2.2.1 Design-time component

The design-time component encompasses the first and second phase of the framework. More specifically, in the first phase, the reasoning framework utilizes a transformer to map goal models annotated with a confidence level in their contribution links (i.e., fuzzy goal models), to a corresponding graph we refer to as the *reasoning model*. The reasoning model is composed of nodes which denote *reasoning units*, and edges which denote dependencies between these reasoning units. The transformation of a goal model to a reasoning model entails two steps. In the first step, a reasoning unit is created for each node of the goal model, and each such reasoning unit is annotated with *input* and *output* ports (reasoning units creation step in Fig. 1). Subsequently, in the second step for each reasoning unit, a *reasoning logic* in the form of an appropriate set of rules is generated (reasoning logic generation step in Fig. 1). By analyzing the *input* and *output* port annotations for each reasoning unit, the dependencies between the reasoning units can then be identified, resulting thus in the production of the "Reasoning Model Dependency Graph."

In the second phase of the design-time component, the framework analyzes the Reasoning Model Dependency Graph, in order to produce a *reasoning plan* (reasoning plan compilation step in Fig. 1). A *reasoning plan* is a collection of sub-plans. These *sub-plans* form a partition over all *reasoning units*. The algorithm ensures that each such set of sub-plans in the *reasoning plan* is compliant with the semantics of the original goal model, and each set of sub-plans can be evaluated independently and in parallel with other sets of sub-plans.

2.2.2 Run time component

The run time component encompasses the third phase of the framework where a reasoning control strategy is applied. The reasoning control strategy selects sub-plans, and for each selected sub-plan, evaluates its reasoning units, by executing the corresponding to the unit, *reasoning logic*. For our prototype implementation, we use the jFuzzyLogic library and the *reasoning logic* associated with each *reasoning unit* is either a jFuzzyLogic compliant script or a set of Boolean rules, which are generated according to the process discussed in Sect. 5.2. In this respect, the result of the third phase of the framework is the parallel evaluation of all sub-plans and consequently of all reasoning units in these sub-plans. The evaluation is performed by processing the stream of data collected from the running system (input stream), and assigning truth values to the leaf nodes of the goal model. We then use the proposed mechanism to propagate the truth values to the rest of the nodes, allowing thus the real-time evaluation of the satisfaction degrees for all system goals at run time (output stream).

3 Related work

In this section, we first present existing reasoning techniques that can be applied in the problem under study. An extensive analysis of reasoning techniques for goal models can be found in [26–29]. Second, we discuss the differences between a fuzzy reasoning approach like the one presented in the context of this paper and the probabilistic ones proposed in the literature.

3.1 Qualitative approaches

Examples of qualitative approaches are the ones presented in [7,22]. More specifically, qualitative approaches that perform a forward reasoning over goal models, use labels that correspond to levels of satisfaction (e.g., highly denied, poorly satisfied), and provide rules that can be used in order to propagate these levels from the leafs to the roots of the goal models.

A common problem encountered though to every qualitative approach, is that label propagation becomes rapidly inconclusive as we move up in the soft-goal refinement tree [34]. To alleviate this problem, quantitative approaches have been proposed in the literature, the most indicative of which are discussed below.

3.2 Quantitative approaches

An example of a quantitative analysis over AND/OR goal trees is the one introduced in [21], where the authors conjunction is calculated via the product t-norm [4] as opposed to fuzzy logic. Giorgini et al. in [21] calculate two values for each goal node, one that expresses the probability the node being satisfied and an additional one that corresponds to the probability the node being denied. Those two values are considered independently and are not combined to a single belief value for a given goal node, as it is the case for fuzzy reasoners.

Other examples of approaches that use quantitative reasoning over goal models are the ones presented in [14] and [11]. In [14], goal trees are utilized to denote dependencies between certain software project management metrics and specific goals related to cost, effort, and quality of the software system being built or maintained. Using data from past similar projects, weights are assigned to contribution links, and an MLN reasoner [44] is used to calculate the probabilities of root goal nodes to be satisfied. A probabilistic framework is also introduced in [11]; however, authors are aiming at determining the probability of certain obstacles that will result to some goals to fail. Knowing these probabilities allows for the selection of an appropriate alternative countermeasure. Probabilistic reasoning applied on these cases is quite different from the reasoning considered in the context of this paper, as here we are interested in the degree of confidence a goal node is satisfied, as opposed to the probability the node be satisfied. Quantitative goal models have been studied in [23] in order to evaluate alternative design decisions expressed in the KAOS goal modeling language which is a different problem from the one studied in the context of this paper. A significant difference between that approach and the one proposed here is the annotation of goal nodes with objective functions and quality variables, where the objective functions are being used for the evaluation of the satisfaction degree of each goal node according to the values of the corresponding quality variables. In that approach, by defining specific probability distributions for the quality variables, a stochastic simulation can be used in order to compute the values of the objective functions for each alternative and subsequently select the one with the highest score.

Furthermore, another example of a quantitative analysis that is applied to Business Intelligence Models is introduced in [25]. In that paper, the authors propose techniques that can be used to reason with indicators linked to business data using interpolation, unit conversion or specialized mathematical formulas to combine values. At the same time, the authors in [25] introduce a hybrid reasoning technique as means to reason over incomplete indicators. In this respect, our approach could be adapted and used to model composite dependencies between such indicators and business data, bringing thus the operationalization aspects of these two frameworks, closer.

3.3 Fuzzy versus probabilistic approaches

While the use of fuzzy logic in goal models has been presented before in [8], our approach is rather different. The key difference is the fact that here we apply a pure fuzzy reasoning process in order to propagate the truth values of the leaf nodes to the roots of the given goal models. In contrast, the authors in [8] utilize fuzzy temporal logic as means to formally express adaptation capabilities for goal models. Additionally, through the proposed analysis, we bring closer, methods and techniques from the field of fuzzy control theory, with methods and techniques for goal model reasoning.

With respect to probabilistic approaches such as the one presented in [21], these are not equivalent to fuzzy ones but rather complementary. The probabilistic approaches serve the purpose of the elicitation of the probability two modeling elements being dependent, while the fuzzy approaches serve the purpose of the estimation of a degree a modeling element is satisfied [50]. In our opinion, there are some key aspects to fuzzy logic, mainly related to expressiveness and interpretation of the results [42,50], that make fuzzy approaches more appropriate for the problem under study as: *a)* we need to model vagueness in human reasoning and observations as opposed to uncertainty; *b)* we should reason over known observations (i.e., we know that an event has occurred as opposed the probability of the event to occur); and consequently *c)* we should evaluate how this impacts the satisfaction of higher-level goals.

4 Fuzzy goal models

AND/OR goal models have been proposed as a modeling formalism in requirements engineering. The basic concept of goal models is a top-down, AND/OR decomposition of goals into sub-goals, where an AND-decomposed goal is satisfied if all of its sub-goals hold, while an OR-decomposed goal is satisfied, if at least one of its sub-goals holds.

In this section, we introduce a domain model for the definition of goal models that can be used for run time reasoning verification. The proposed domain model is depicted in Fig. 2 and consists of two main parts, namely the *generic goal model* part and the *fuzzy goal model* part. The former is the core part of the domain model. It supports instances that contain the most commonly used goal model components and has been defined in such a way that can be extended in order to enable the utilization of various reasoning techniques (e.g., qualitative, probabilistic, fuzzy) depending on the needs of the analysis. In contrast, the fuzzy goal model part contains the additional components (given in light gray) required to support the fuzzy reasoning approach proposed in the context of this paper. More specifically, the fuzzy goal model part provides concrete implementations of the interfaces defined in the generic goal model part given in dark gray.

As depicted in Fig. 2 a fuzzy goal model is composed of nodes (*GraphNode* class), decomposition links (*Decomposition* class), and contribution links (*ContributionLink* class). To facilitate the description of the various components of the domain model, we also provide an example FGM illustrated in Fig. 3, which is partially based on examples used in [5,20,30] and describes a fraction of goals of a simplified, yet realistic, data management system.

4.1 Goals and goal model nodes

Goals in goal models can be classified into two types, namely *hard goals* and *soft goals* [40]. Hard goals denote goals for which their Boolean truth value can be inferred utilizing first-order logic. In contrast, soft goals are goals for which their *degree* of satisfaction is determined according to a reasoning process that takes into account positive (supporting) and negative (denying) evidence, allowing thus the existence of nodes that can be satisfied to a certain degree. In this paper, we adopt the terminology introduced in [8], where hard goals referred to as *crisp goals* and soft goals as *fuzzy goals*.

Furthermore, each node in the goal model is represented by an instance of *GraphNode* class and has a *goal* attribute parameterized as *IGoal* type, which in the case of fuzzy goal models is either a fuzzy goal (*FuzzyGoal* class) or a crisp goal (*CrispGoal* class). Additionally, a goal (*IGoal interface class*) is related to a validation strategy (*IValidationStrategy* class), which encapsulates the reasoning logic required to validate whether the goal is true, false, or holds with a satis-

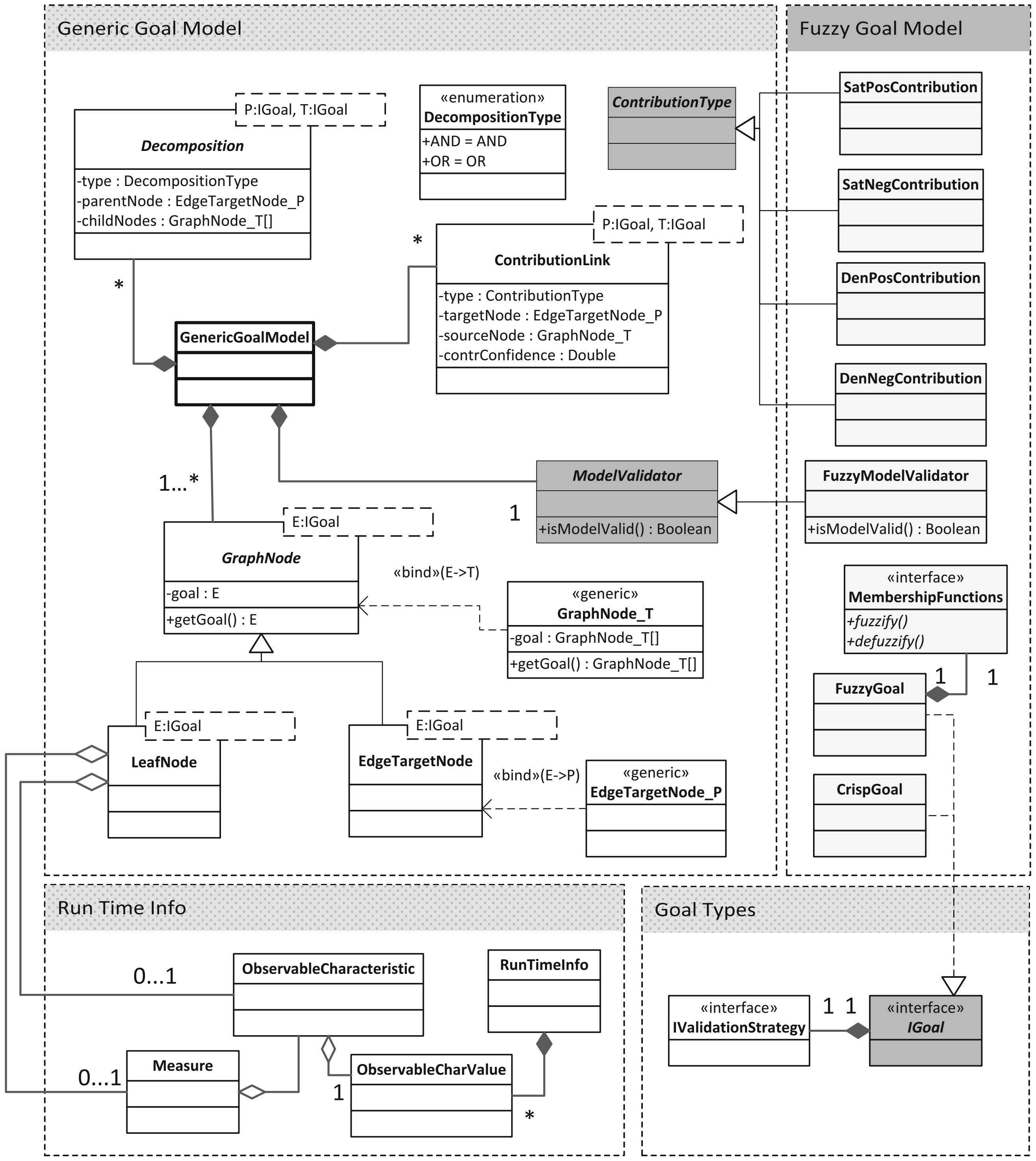


Fig. 2 FGM domain model

faction percentage within the (0,100) interval depending on the type of analysis applied.

Depending on the factors that affect the truth value of a goal, a node can be classified either as an *EdgeTargetNode* or as a *LeafNode*. The *EdgeTargetNode* class denotes nodes, the truth value of which depends on the truth value of other nodes in the graph, and hence, there are edges targeting these nodes. These nodes can represent composite goals that can be further decomposed to simpler goal nodes (e.g., node *A*1 in Fig. 3) or/and can be the target node of one or more contribu-

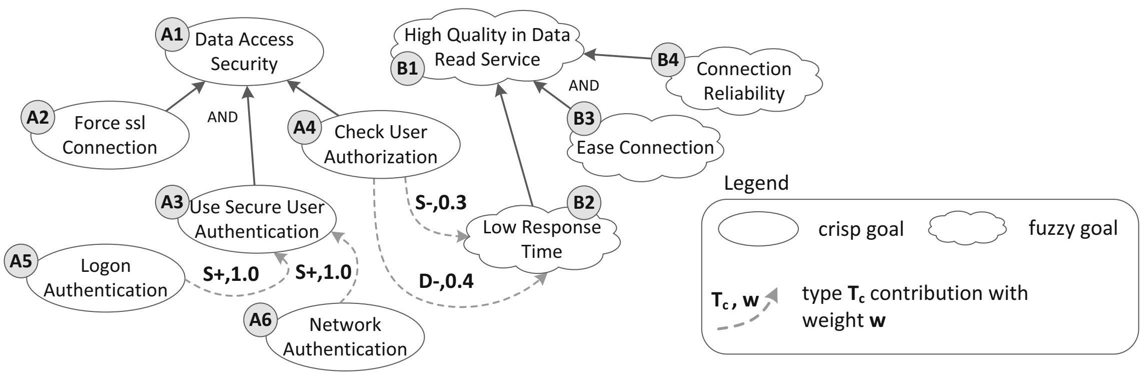

A1 Data Access Security
A2 Force ssl Connection
AND
A4 Check User Authorization
A3 Use Secure User Authentication
A5 Logon Authentication
A6 Network Authentication
S+,1.0
S+,1.0
S-,0.3
D-,0.4
B1 High Quality in Data Read Service
B4 Connection Reliability
AND
B3 Ease Connection
B2 Low Response Time
Legend
crisp goal
fuzzy goal
Tc, w
type Tc contribution with weight w

Fig. 3 Fuzzy goal model (FGM) example

tion edges (e.g., node $B2$ in Fig. 3). Similarly, the *LeafNode* class denotes goals the truth value of which is evaluated by analyzing run time monitored data and are neither AND/OR decomposed nor the target node of a contribution link (e.g., node A_5 in Fig. 3).

4.2 Decomposition

AND/OR decomposition of composite goal nodes is represented as instances of the *Decomposition* class. Decomposition is defined as a parameterized type, in which the parent and the child nodes may correspond to either fuzzy or crisp goals depending on the actual type selected for the parameters P and T. The only restriction is that the parent node is of type *EdgeTargetNode*, as the parent node of a decomposition cannot be a leaf node.

More formally, the decomposition of a goal parent node p to a set $\{c_1, c_2, \ldots c_n\}$ of *child* goal nodes is denoted as a rule r of the form:

$$r = \langle T_d, p, \{c_1, c_2, \ldots, c_n\}\rangle \tag{1}$$

where $T_d \in \{\text{AND, OR}\}$. For example, in the model given in Fig. 3, the AND decomposition of the fuzzy goal node $B1$ to nodes $B2$, $B3$ and $B4$ is formally written as:

$$\langle AND, B1, \{B2, B3, B4\}\rangle$$

4.3 Contribution links

Two goal model nodes may be connected by a directed contribution link that denotes whether a source node contributes positively or negatively to other target nodes, and practically, is a link that propagates the truth value of the source node to the target node of the link. In the context of this paper, we utilize four types of contribution links namely, S^+ (*SatPosContribution* class), S^- (*SatNegContribution* class), D^+ (*DenPosContribution* class), and D^- (*DebNegContribution* class) and adopt the semantics of [16] regarding the way values are propagated from the source to the target node. For example, the S^+ contribution link from node $A6$ to node $A3$ in the example depicted in Fig. 3 denotes the fact that when $A6$ is true (i.e., the node is satisfied) then also the target node (i.e., node $A3$) will be satisfied as a consequence of the satisfaction of the source node.

Additionally, we consider that each contribution link is also associated with a real number, we refer to as *contribution confidence* (w). However, we utilize a quite different interpretation of the one used for weights in [7,25,34] where a probabilistic reasoning is applied. More specifically, contribution confidence values in the proposed fuzzy analysis denote the subjective degree of belief a domain expert has in the corresponding rule. Hence, for a contribution link from a source node (g_s) to a target node (g_t), the following interpretations apply assuming that these links are annotated with a contribution confidence w:

$$\begin{aligned} S^+/D^+ &: \text{the degree of belief that } g_s \text{ satisfaction/denial} \\ &\quad\ \text{implies } g_t \text{ satisfaction/denial is } w \\ S^-/D^- &: \text{the degree of belief that } g_s \text{ satisfaction/denial} \\ &\quad\ \text{implies } g_t \text{ denial/satisfaction is degree } w \end{aligned} \tag{2}$$

Contribution links between the goal nodes of an FGM are modeled as instances of the abstract parameterized type *ContributionLink*. Formally, we denote a contribution link from node s to node t with contribution confidence w as a rule r of the form:

$$r = \langle T_c, s, t, w\rangle \tag{3}$$

where $T_c \in \{S^+, D^+, S^-, D^-\}$. For example, the S^- contribution from node $A4$ to node $B2$ is formally written as:

$$\langle S^-, A4, B2, 0.3\rangle$$

Furthermore, as multiple contribution links of the same type may be targeting a single goal node b, we define the following sets for each type T_c of a contribution link:

$$\mathcal{N}(T_c, b) = \{s | \exists r = \langle T_c, s, b, w\rangle\} \qquad (4)$$

where $T_c \in \{S^+, D^+, S^-, D^-\}$. which includes the source nodes s of all contribution links of type T_c for which, the source node is s and the target node is b. For example, in Fig. 3, $\mathcal{N}(S^+, A3) = \{A5, A6\}$.

4.4 Leaf goal model nodes

A graph node that appears neither as the target node of any contribution link nor as the parent node of a decomposition is called *leaf node*. Such a node is linked either to a measure, as denoted by the *Measure* class, or to a specific observable logged event, as denoted by the *ObservableCharacteristic* class in the domain model. The measure or the logged events can then be used to evaluate the leaf node's truth value, or determine the degree of satisfaction for this node, so that reasoning can commence. More specifically, regarding the observable characteristics, as the system operates, values are assigned to them (*ObservableCharValue* class) via the system's monitoring infrastructure, and if the leaf node is a fuzzy one, suitable membership functions are used in order to transform the monitored events and their values to fuzzy truth values, for the corresponding leaf nodes. In the example, fuzzy goal model (FGM) of Fig. 3, goal nodes $A2$, $A5$, $A6$, $B3$, and $B4$ are the leaf goal nodes of the model, while $B2$ is not a leaf node as it is the target of two contribution links.

4.5 Constraints

There are some constraints regarding the structure of an FGM, which are imposed by the type of analysis applied to the model. An instance of the abstract type *ModelValidator* is used to evaluate whether the given model violates any of these constraints, which for the case of FGMs are as follows:

1. Goals can only be decomposed to goals of the same type (i.e., the parent node and the child nodes of a decomposition are all either crisp or fuzzy);
2. Only crisp nodes can contribute to crisp nodes and for those contribution rules the contribution confidence is always equal to 1 (i.e., if for a contribution link the target node is crisp, then the source node should be crisp and the contribution confidence should be 1);
3. The contribution links that exist in an FGM should not result in circular dependencies as this will be explained later in Sect. 6.

5 Reasoning model generation

As already discussed in the previous section, an FGM can be fully denoted as a set of decomposition and contribution rules, presented in the form of Eqs. (1) and (3). For example, the following 6 rules correspond to the FGM illustrated in Fig. 3:

$$\langle \text{AND}, A1, \{A2, A3, A4\}\rangle, \langle S^-, A4, B2, 0.3\rangle,$$
$$\langle \text{AND}, B1, \{B2, B3, B4\}\rangle, \langle S^+, A5, A3, 1.0\rangle,$$
$$\langle D^-, A4, B2, 0.4\rangle, \langle S^+, A6, A3, 1.0\rangle$$

Given this set of rules, we can group rules in such a way that rules which define the value of the same node belong to the same group. For example, rules $\langle S^-, A4, B2, 0.3\rangle$ and $\langle D^-, A4, B2, 0.4\rangle$ are the only rules that can be used to extract some conclusions for the value of goal node $B2$ according to the given FGM.

Additionally, by knowing the complete set of rules that define the value of a node p, we also know the nodes the values of which are required in order to calculate the value of p. In the case of node $B2$ of the FGM illustrated in Fig. 3, we only need to know the value of $A4$ in order to calculate the value of $B2$.

Furthermore, the way the value is calculated is dictated by the corresponding rules. Hence, by grouping the rules as described above, we can create for each node p, a unit that combines the values of all nodes on which p depends, and uses the rules to extract a value for p. We call such units *reasoning units* and the mechanism used to extract the value of the corresponding node *reasoning logic*. The set of the generated reasoning units constitutes the *reasoning model* of the FGM.

In the rest of this section, we introduce a two-step process to systematically generate the reasoning model for a given FGM. The first step deals with the creation of the *reasoning units*, while the second step deals with the processing of the rules of each unit in order to generate the corresponding *reasoning logic*.

5.1 Reasoning units creation

In order to generate the reasoning units, we iterate over each goal node p in the initial FGM, and if the node is not a leaf node, we create a *reasoning unit* U_p for it. For example, for the FGM in Fig. 3, four reasoning units U_{A1}, U_{A3}, U_{B1}, and U_{B2} will be created, one for each non-leaf goal node $A1$, $A3$, $B1$, and $B2$, respectively. To better illustrate the dependency that exists between p and its reasoning unit U_p, we call goal node p the *output* of the reasoning unit U_p. Additionally, the collection of all source goal nodes for the contribution links targeting p, and the child nodes of p, in case p is a decomposition node, is referred to as the *input* of the reasoning unit. In the context of this paper, we are going to use the notations $In[U_p]$ and $Out[U_p]$ to denote the input and output goal nodes of a reasoning unit U_p, respectively. For example, for the reasoning units generated for the FGM in Fig. 3, the input and output goal nodes are:

$$\begin{aligned} &Out[U_{A1}] = A1\,,\; In[U_{A1}] = \{A2, A3, A4\}\\ &Out[U_{A3}] = A3\,,\; In[U_{A3}] = \{A5, A6\}\\ &Out[U_{B1}] = B1\,,\; In[U_{B1}] = \{B2, B3, B4\}\\ &Out[U_{B2}] = B2\,,\; In[U_{B2}] = \{A4\} \qquad (5)\end{aligned}$$

Practically, identifying the input goal nodes of a given node p is a special case of taking a slice in a graph. More specifically, we collect all nodes that are connected to node p through a path of length 1, as opposed to paths of length greater than 1, as this is determined by the goal model's topology.

The dependency that exists between the reasoning units and the goal nodes is also depicted in the domain model of Fig. 4. According to this domain model, the set of all reasoning units generated for a given FGM constitutes the *FGM reasoning configuration*. Additionally, each reasoning unit is associated with a *UnitMetrics* class which refers to metrics related to the reasoning unit. The usage of these metrics will be illustrated in Sect. 7.

At this point, we know for each node what are its dependencies, i.e., which are the input nodes of the corresponding reasoning unit. However, the way the value of the output node will be calculated, by combining the values of all input nodes, is defined by the set of decomposition and contribution rules that have as target node the output of the reasoning unit. In the following subsection, we describe in detail how this set of rules can be used in order to generate the reasoning logic of the unit.

5.2 Generate the reasoning logic of each unit

Once all reasoning units have been created, we collect for each unit U_p the set of rules generated from the contribution links for which p is the target node, and also the decomposition rule of p, if one such decomposition exists (see Eq. (1) and Eq. (3)). Subsequently, depending on whether p is a crisp or a fuzzy node, we create for each *reasoning unit* either a *CrispReasoningLogic* or a *FuzzyReasoningLogic*, which are implementations of the abstract type *IReasoningLogic* as this is depicted in Fig. 4, and practically contain all rules required to calculate the truth value of p. These two reasoning logic sub-types implement the abstract method *"evaluate"* which given the truth values for all input nodes of the reasoning unit will calculate the truth value of its output goal node. To perform this evaluation, a Reasoning Logic uses either a set of Boolean rules (when the output goal node is a crisp one), or a Fuzzy Control Language module [1] (when the output goal node is a fuzzy one). More details on how this works will be presented later on Sects. 5.2.1 and 5.2.2.

However, in addition to this evaluation, a run time validation strategy related to the output goal node should be also invoked. The validation strategy must collect data from the logging and monitoring infrastructure of the running system and apply a validation logic to ensure that the goal model node's inferred value (e.g., "true" or 100 % satisfied, "false" or 0 % satisfied, "30 % satisfied") does not conflict with any hard evidence obtained from the running system (e.g., a "*low response time warning*" event). For example, consider the scenario in which the goal model reasoning process evaluates

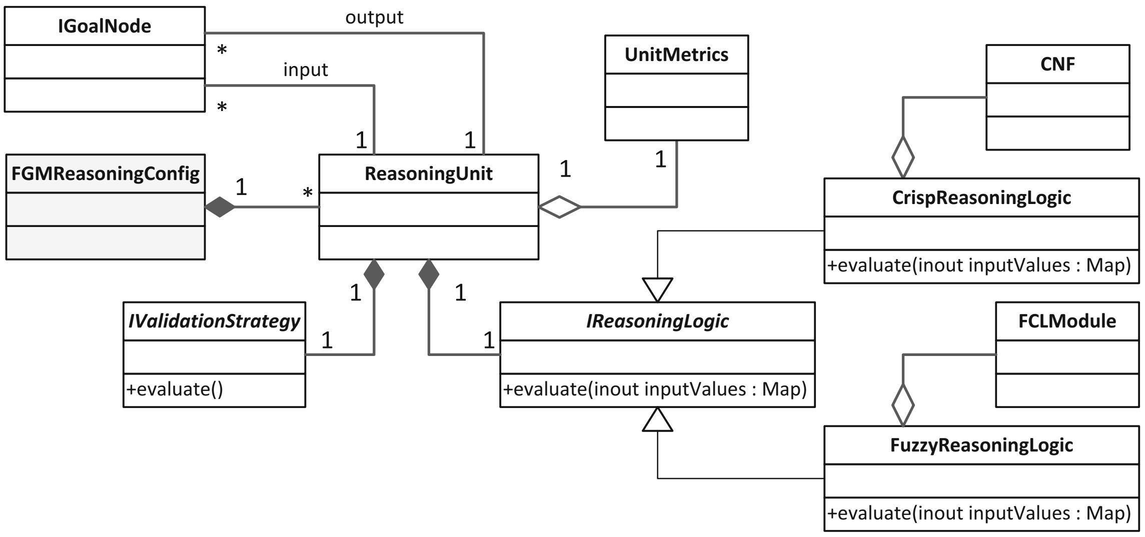


Fig. 4 Reasoning units metamodel

Table 1 AND/OR rules generation for a crisp goal node p ($r_d = \langle T, p, \{c_1, c_2, \ldots, c_n\}\rangle$)

$N^{pos}(p)$	$N^{neg}(p)$	Generated AND/OR rules	
		r_d exists	NO r_d exists
$= \emptyset$	$= \emptyset$	$p \leftarrow \mathrm{T}(c_1, c_2, \ldots, c_n)$	-
$= \emptyset$	$\neq \emptyset$	$p \leftarrow$ AND(p_d, ¬p_c_neg)	$p \leftarrow$ AND(True, ¬p_c_neg)
$\neq \emptyset$	$= \emptyset$	$p \leftarrow$ OR(p_d, p_c_pos)	$p \leftarrow$ OR(False, p_c_pos)
$\neq \emptyset$	$\neq \emptyset$	$p \leftarrow$ OR(p_d, p_c) p_c $\leftarrow$ AND(p_c_pos, ¬p_c_neg)	$p \leftarrow$ AND(p_c_pos, ¬p_c_neg)

the goal model node p: *"Low Response Time"* as *"20 % satisfied,"* because p's contribution links support this value but at the same time, the performance monitor indicates that there is a positive low response time system behavior as observed by the warning "*low response time warning*," in the emitted system logs. Such a failed validation of the node's inferred value, will signify a discrepancy between what the system administrators think should be happening and what is actually happening, as a result of possible violation of a requirement (i.e., the requirement related to the root goal).

For practical applications, this introduces a validation overhead in the run time engine, which would be required in order to validate with hard evidence (e.g., unambiguous data collected from system loggers) that the outcome of the Boolean rules or FCL module is consistent with what is observed in the running system. While the validation process is outside the scope of this paper, in the experimentation section, we consider a random validation overhead of 10–100 ms for each node, assuming that there is a monitoring and logging infrastructure in place that feeds this evaluation process with logged events.

5.2.1 Boolean rules generation

In this section, we discuss the generation process for the Boolean rules as these will be attached as *reasoning logic* to *reasoning units* stemming from crisp goal model nodes. The Boolean Rules generation process involves the computation of two sets for each non-leaf goal node p in the model. These sets are $N^{\mathrm{pos}}(p)$ and $N^{\mathrm{neg}}(p)$ with the interpretation that $N^{\mathrm{pos}}(p)$ is the set of all goal nodes that positively contribute to node p, while in contrast $N^{\mathrm{neg}}(p)$ is the set of all goal nodes that negatively contribute to node p. These are formally defined as:

$$\begin{aligned}\mathcal{N}(T_c, b) N^{\mathrm{pos}}(p) &= \mathcal{N}(S^+, p) \cup \mathcal{N}(D^-, p) \\ &= \{e_1, \ldots, e_k\} \cup \{f_1, \ldots, f_l\} \qquad (6)\\ N^{\mathrm{neg}}(p) &= \mathcal{N}(S^-, p) \cup \mathcal{N}(D^+, p) \\ &= \{g_1, \ldots, g_m\} \cup \{h_1, \ldots, h_o\} \qquad (7)\end{aligned}$$

where $e_1, \ldots, e_k$ are the source nodes of all S^+-type contributions to node p, $f_1, \ldots, f_l$ are the source nodes of all D^--type contributions to node p, $g_1, \ldots, g_m$ are the source nodes of all S^--type contributions to node p, and $h_1, \ldots, h_o$ are the source nodes of all D^+-type contributions to node p. For example, for goal node $A3$ $N^{\mathrm{pos}}(A3) = \{A5, A6\}$.

These sets along with the decomposition rule $r_d = \langle T_d, p, \{c_1, c_2, \ldots, c_n\}\rangle$ as depicted in Eq. (1) (refers to the case that p is a composite goal node that is decomposed to child nodes $\{c_1, c_2, \ldots, c_n\}$), determine the set of Boolean Rules that should be generated. The resulting set of Boolean rules will be used to calculate the truth value of p, denoted as $\mathcal{V}[p]$ from the truth values of the nodes it depends on. In order to simplify the equations given in the rest of this section, we are going to use the name of a node p to refer either to the node itself or to its truth value. We only use the notation $\mathcal{V}[p]$ when it is not clear from the context whether p is referred to the node or to its truth value.

According to whether a decomposition rule exists and to whether $N^{\mathrm{pos}}(p)$ and $N^{\mathrm{neg}}(p)$ are empty, a different set of rules is generated as illustrated in Table 1. Additionally, for the pseudo-variable p_c_pos (which becomes true when at least one node contributes positively to node p), p_c_neg (which becomes true when at least one node contributes negatively to node p), and p_d (which becomes true when node p is supported by its child nodes through goal node decomposition) that appear in Table 1, the following formulas apply:

$$\text{p\_c\_pos} = \mathrm{OR}(e_1, \ldots, e_k, \neg f_1, \ldots, \neg f_l) \qquad (8)$$
$$\text{p\_c\_neg} = \mathrm{OR}(g_1, \ldots, g_m, \neg h_1, \ldots, \neg h_o) \qquad (9)$$
$$\text{p\_d} = \mathrm{T}(c_1, c_2, \ldots, c_n) \qquad (10)$$

where T is either AND or OR type of decomposition, $c_1, c_2, \ldots, c_n$ are the child nodes of p, and nodes e_i, f_i, g_i and h_i correspond to the ones defined in Eqs. (6) and (7). For example, for the goal model in Fig. 3 and for the crisp goal node $A3$ the pseudo-variable $A3\_c\_pos$ is equal to OR($A5$, $A6$) as $N^{\mathrm{pos}}(A3) = \{A5, A6\}$.

It is important to note that if both indexes k and l (alt. m and o) are equal to zero, the set N^{pos} (alt. N^{neg}) is empty, and the pseudo-variable p_c_pos (alt. p_c_neg) does not appear in the rules.

A graphical representation of the dependencies between p, p_c_pos, p_c_neg, and p_d in case $N^{pos}(p) \neq \emptyset$,

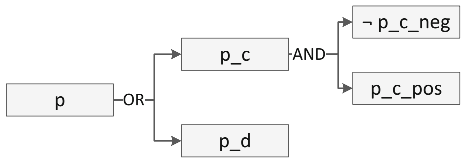


Fig. 5 AND/OR rules graphical representation in case $N^{pos}(p) \neq \emptyset$, $N^{neg}(p) \neq \emptyset$, and a decomposition rule r_d exists

$N^{neg}(p) \neq \emptyset$ (see last row of Table 1), and a decomposition rule r_d exists is depicted in Fig. 5. Fig. 5, depicts that a node p can be evaluated by its decomposition logic, or by its contribution links, or both. Furthermore, for the contribution links to be used for evaluation purposes, the negative and positive factors (in the form of pseudo nodes) have to be considered.

The basic assumption when building the transformation rules summarized in Table 1, is that a node p can be satisfied either via its decomposition or because some node contributes positively to it. In case some node contributes negative to p through an S^- or a D^+ contribution link, then p is denied. Thus, we consider that a negative contribution has higher priority that a positive contribution or a decomposition. For example, in the transformation depicted in the second row of Table 1, there should be no negative contribution to p (i.e., p_c_neg should be false) for p to be satisfied even if p is satisfied through its decomposition.

5.2.2 FCL module generation

In this section, we describe the process used in order to generate the reasoning logic for a unit that corresponds to a fuzzy node. The reasoning logic in this case is implemented in the form of a fuzzy controller. As this is also described in [43], a fuzzy controller is composed of the four following elements: a) *a fuzzification process* for the transformation of the input in a processable form by using an appropriate set of membership functions; b) *a rule-base*, that consist of rules that describe the experts' knowledge of the domain; c) *an inference mechanism* which combines the rule-base with the input to deduce membership degrees for a set of output variables; and d) *a defuzzification process* which converts the inference outcome (i.e., the membership degrees) for each output variable into a quantifiable result by combining the membership degrees.

A domain-specific programming language that can be used to define and implement fuzzy controllers is the Fuzzy Control Language (FCL), which has been standardized and published by the International Electrotechnical Commission (IEC 61131-7) [1]. In the context of this paper, we utilize jFuzzyLogic [17], which is an open source fuzzy logic library that implements the IEC 61131-7 standard.

```
FUNCTION_BLOCK ${goal_name}

    VAR_INPUT
          ${Vin_1} : REAL;
          ...
          ${Vin_n} : REAL;
    END_VAR

    VAR_OUTPUT
          ${goal_name} : REAL;
    END_VAR

    FUZZIFY ${Vin_1}
          TERM low := (0,1) (10,1) (60,0);
          TERM high := (40,0) (90,1) (100,1);
    END_FUZZIFY
    ...
    FUZZIFY ${Vin_n}
          TERM low := (0,1) (10,1) (60,0);
          TERM high := (40,0) (90,1) (100,1);
    END_FUZZIFY

    DEFUZZIFY ${goal_name}
          TERM low := (0,1) (10,1) (60,0);
          TERM high := (40,0) (90,1) (100,1);
          METHOD : COG;
          DEFAULT := 50;
    END_DEFUZZIFY

    RULEBLOCK No1
          AND : PROD;
          ACT : PROD;
          ACCU : NSUM;

          RULE 1 : ${rule_1};
          ...
          RULE k : ${rule_k};
    END_RULEBLOCK

END_FUNCTION_BLOCK
```

Fig. 6 FCL module template

For each reasoning unit that corresponds to a fuzzy goal, we compile an FCL module, which is a function block generated by utilizing the template depicted in Fig. 6. The resulting FCL module contains contracts that are going to be used by the underline reasoner in order to implement the fuzzification (FUZZIFY blocks in the template) and defuzzification (DEFUZZIFY block in the template) processes, as well as a set of fuzzy rules that constitute the rule-base of the module. All terms printed in italics and starting with the special character \$ in the template of Fig. 6, are terms that must be substituted according to the values that correspond to the given node p. More specifically:

- *goal_name* is the name of the output goal which should be unique
- *Vin_1* ... *Vin_n* are the names of the input goal nodes, and

Table 2 Transformation of FGM rules (i.e., decomposition and contribution rules) to FCL rules

FGM rule	FCL rules
$\langle$AND, p, $\{c_1, c_2, \dots c_n\}\rangle$	**IF** c_1 **IS** *high* **AND** . . . **AND** c_n **IS** *high* **THEN** p **IS** *high*
	IF c_1 **IS** *low* **THEN** p **IS** *low*
	⋮
	IF c_n **IS** *low* **THEN** p **IS** *low*
$\langle$OR, p, $\{c_1, c_2, \dots c_n\}\rangle$	**IF** c_1 **IS** *low* **AND** . . . **AND** c_n **IS** *low* **THEN** p **IS** *low*
	IF c_1 **IS** *high* **THEN** p **IS** *high*
	⋮
	IF c_n **IS** *high* **THEN** p **IS** *high*
$\langle S^+, g_s, g_t, w\rangle$	**IF** g_s **IS** *high* **THEN** g_t **IS** *high* **WITH** w
$\langle S^-, g_s, g_t, w\rangle$	**IF** g_s **IS** *high* **THEN** g_t **IS** *low* **WITH** w
$\langle D^-, g_s, g_t, w\rangle$	**IF** g_s **IS** *low* **THEN** g_t **IS** *high* **WITH** w
$\langle D^+, g_s, g_t, w\rangle$	**IF** g_s **IS** *low* **THEN** g_t **IS** *low* **WITH** w

– *rule_1* . . . *rule_k* are FCL rules generated for goal node p as this is described in Table 2.

In the rest of this section, we are going to describe (a) how FUZZIFY blocks are used during the fuzzification process, (b) how the required rules are generated for each reasoning unit, and (c) how the defuzzification process is applied utilizing the DEFUZZIFY block defined in the FCL module. To better illustrate the various artifacts of the FCL module, we consider the module that will be generated for the fuzzy node $B2$ of the example of Fig. 3. In this case, for the template of Fig. 6, \$*goal_name* will be substituted by $B2$, and there will be only one input variable, namely $A4$. Hence, there will be only one FUZZIFY block. Furthermore, the following two rules, one for each contribution link targeting $B2$ in Fig. 3, will be generated:

RULE 1: **IF** $A4$ **IS** *high* **THEN** $B2$ **IS** *low* **WITH** 0.3
RULE 2: **IF** $A4$ **IS** *low* **THEN** $B2$ **IS** *high* **WITH** 0.4

The *high* and *low* used in the rules are fuzzy predicated that correspond to the truth value of the statement "The goal is highly satisfied" and "The goal is poorly satisfied," respectively.

Fuzzification: Regarding the fuzzification process, the objective is that given values (e.g., numeric values) for the input variable $A4$, to be able to define the degree in which this input variable belongs at the same time to the fuzzy sets "low" and the fuzzy set "high."

This is accomplished by defining a set of membership functions for the input variable by a *FUZZIFY* block as follows:

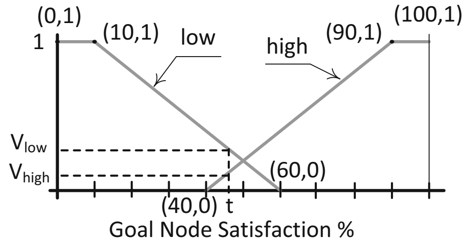


Fig. 7 Use of the membership functions in fuzzification

FUZZIFY a_1
  TERM low := (0, 1) (10, 1) (60, 0);
  TERM high := (40, 0) (90, 1) (100, 1);
END_FUZZIFY.

The two membership functions and the fuzzification process are depicted in Fig. 7. Please note that in the *FUZZIFY* block listed above, the values (0,1) (10,1) (60,0) refer to the coordinates defining the "low" membership function as depicted in Fig. 7 and that different values can be used for each input variable.

According to this example, given that the value of $A1$ is equal to t, the fuzzification process will return V_{low} and V_{high} as the degree in which the variable belongs to the fuzzy sets "low" and "high," respectively. This values in combination with the rules will be used in order to calculate the degrees for "low" and "high" for the output variable, namely $B2$.

Rules Generation: Given the reasoning unit that corresponds to a fuzzy node p, we can extract a set of FCL rules for each decomposition and/or contribution related to node p (i.e., p is the parent node of the decomposition or the target node of the contribution), using the transformations summarized in Table 2. Hence, for the $S-$ contribution link from node $A4$ to the fuzzy node $B2$, which implies that $A4$ satisfaction results in the denial of node $B2$, the following FCL rule will be generated as this is defined by the transformation depicted in the 4th row of Table 2:

IF $A4$ **IS** *high* **THEN** $B2$ **IS** *low* **WITH** 0.3

Defuzzification: Once the input variables are fuzzified, the inference engine can extract conclusions about the output variables, such as the variable $B2$, using fuzzy rules, such as the ones presented above. For example, if $A4$ is determined to hold by $V_{low} = 0.2$ and $V_{high} = 0.1$, then both of the above rules will be triggered with the appropriate values (0.2, and 0.1) for "low" and "high." These values are then combined by a proper defuzzification process which calculates the final satisfaction percentage of the output variable $B2$.

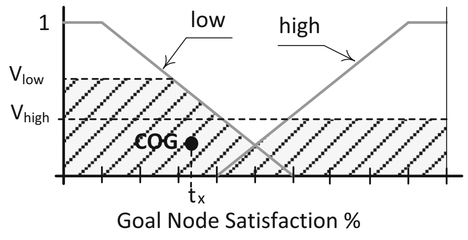


Fig. 8 Use of the membership functions in defuzzification

The details of the defuzzification process are defined by a *DEFUZZIFY* block given in the following form:

DEFUZZIFY *c*

TERM low := (0, 1) (10, 1) (60, 0);

TERM high := (40, 0) (90, 1) (100, 1);

METHOD := **COG**;

END_DEFUZZIFY

where except from the membership functions, we also denote the method that should be utilized, which in this case is the centroid, or Center Of Gravity (COG) defuzzification method [3]. The two membership functions and the defuzzification process are depicted in Fig. 8. According to this example, given that V_{low} and V_{high} are the degrees in which variable $B2$ has been inferred to belong to the fuzzy sets "low" and "high," respectively, the COG for the gray area is then calculated. The gray area is produced by combining the areas that correspond below the V_{low}, and V_{high} values of $B2$, using the low and high membership function graph, respectively. The x-coordinate of the COG is then the defuzzified value of variable $B2$.

6 Sequential reasoning plan compilation

Each *reasoning unit* created through the process presented in the previous section encompasses *reasoning logic* for determining the truth value of its output node. However, in order to evaluate the truth value of the output node, the truth values for all input nodes are required. In turn, these values should be calculated from other reasoning units which also require a proper input to perform their evaluation. Hence, there are dependencies between the reasoning units generated from a given FGM, which implies that we have to evaluate them in a specific order that respects these dependencies. In this section, we present the mechanics to formulate sequential reasoning plans.

Definition 1 We say that a reasoning unit U_a *directly requires* reasoning unit U_b, denoted as $U_a \xrightarrow{req} U_b$ iff $Out[U_b] \in In[U_a]$.

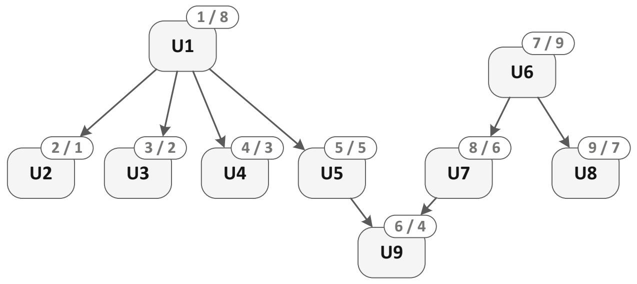


Fig. 9 Reasoning unit dependency graph example

Definition 2 We say that a reasoning unit U_a *requires* reasoning unit U_b, denoted as $U_a \xrightarrow{req*} U_b$ iff $U_a \xrightarrow{req} U_b$ or there exists a reasoning unit U_k such that $U_a \xrightarrow{req} U_k$ and $U_k \xrightarrow{req*} U_b$.

In this respect, the $\xrightarrow{req*}$ operator provides a formalism to denote a directed graph of reasoning units in which, every unit depends *only* on units that appear earlier on paths the unit participates in. We call this directed graph *Reasoning Unit Dependency Graph* and an example is depicted in Fig. 9, for which the following dependencies apply:

$$U_1 \xrightarrow{req} U_2 \text{ , } U_1 \xrightarrow{req} U_3 \text{ , } U_1 \xrightarrow{req} U_4 \text{ , } U_1 \xrightarrow{req} U_5$$
$$U_6 \xrightarrow{req} U_7 \text{ , } U_6 \xrightarrow{req} U_8 \text{ , } U_5 \xrightarrow{req} U_9 \text{ , } U_7 \xrightarrow{req} U_9$$

The two numbers displayed over each reasoning unit (Fig. 9) are the sequence numbers assigned to each unit when traversing the graph utilizing *a)* pre-order algorithm and *b)* a topological sorting algorithm. For example, for node U_1, the first number (1) denotes the order of the node using pre-order traversal, while the second number (8) denotes the order of the node using topological sorting.

These two values are going to be used by the algorithm introduced in the next section in order to compile a proper parallel execution plan.

Definition 3 We say that a path of reasoning units U_1, U_2, ..., U_n is a *proper* one, if for every pair U_i, U_j of reasoning units in the sequence, U_i appears earlier than U_j in case $U_j \xrightarrow{req*} U_i$.

If circular dependencies exist in a set of reasoning units, no proper path of these units exists in the graph. However, in the context of this paper, we assume that FGMs do not have any circular dependencies, due to contribution links. Hence, given the directed graph of all *reasoning units*, we can extract a sequence, which ensures that every reasoning unit appears in the sequence after all reasoning units on which it depends. This sequential execution plan can be computed at definition time through a typical topological sorting algorithm (e.g., Tarjan's algorithm [47]). For example, for the graph depicted in Fig. 9, a sequential plan could be the following:

$$U_2\,,\ U_3\,,\ U_4\,,\ U_9\,,\ U_5\,,\ U_7\,,\ U_8\,,\ U_1\,,\ U_6 \qquad (11)$$

Additionally, given a set $A = \{U_1, U_2, \ldots, U_n\}$ of reasoning units we can extract the variables that appear only as input parameters in the reasoning units in set A. These variables can be formally defined as a set using the following formula:

$$L[A] = \bigcup_{i=1}^{n} In[U_i] - \bigcup_{i=1}^{n} Out[U_i] \qquad (12)$$

and for the reasoning units of Eq. (5), this set is:

$$\begin{aligned} &L[\{U_{A1}, U_{A3}, U_{B1}, U_{B2}\}] \\ &\quad = \{A2, A3, A4, A5, A6, B2, B3, B4\} \\ &\quad - \{A1, A3, B1, B2\} = \{A2, A4, A5, A6, B3, B4\} \end{aligned}$$

which correspond to the leaf nodes of the initial fuzzy goal model, and practically are the variables the values of which are required to initiate the sequential evaluation of the reasoning units.

Subsequently, by sequentially evaluating all reasoning units, we manage to calculate the values of all the output variables of the reasoning units which also include the root goals of the initial fuzzy goal model.

7 Parallel reasoning plan compilation

In addition to defining a sequence of reasoning units, the sequential execution of which guarantees the calculation of the root goals, we propose now an algorithm which allows for the compilation of a reasoning plan that enables the parallel execution of reasoning units.

7.1 Reasoning units metrics

Here, we define a set of metrics for the reasoning units that will be used to determine *reasoning sub-plans* that can be executed in parallel. The following metrics annotate each *reasoning unit* in the *reasoning model*:

- DFS Label (dfs-label) is the sequence number assigned to a reasoning unit when the dependency graph is traversed using the pre-order algorithm, and we denote is as DFS as we use a depth first traversal to visit the nodes of the graphs;
- topological sorting label (to-label) is the sequence number assigned to a reasoning unit when the dependency graph is traversed in topological order;
- total number of child nodes (child-all) is the number of reasoning units; this reasoning unit depends on which are connected by a path of size 1 (hereinafter denoted as *immediate dependencies*);
- number of processed child nodes (child-proc) is the number of immediate dependencies that have been already visited by the algorithm, which is initially set to 0;
- first dependency node for current parallel level (level-first-dep) is the name of the first visited node this unit depends on which is initialized to null;
- parallel level dependency (dep-level) initially set to -1.

The above metrics are used by Algorithm 1 which will be introduced in the next section. In particular, the values of the last 3 metrics are updated as the reasoning units in the graph are visited by the algorithm. As this will be also explained in the next section, Algorithm 1 utilizes both a DFS and a topological sorting sequence number in order to ensure that the parallel sub-plans are created in such a way that the dependencies will not be violated when the plan is executed at run time. The plan compilation algorithm also uses the *SubPlan length* parameter M that denotes the maximum size of each sub-plan that is to be formulated.

Algorithm 1 Plan Compilation Algorithm

Input: L: leaf nodes, M sub-pal size
Output: $plan$: List[Set[Node]]

```
1: $plan \leftarrow$ emptyList
2: $freeNodesSet \leftarrow$ emptySet
3: $currentNodesList \leftarrow$ emptyList[dfs-label-comparator]
4: $currentNodesList.addAll(L)$
5: $currentParLevel \leftarrow 0$
6: while $currentNodesList$ IS NOT EMPTY do
7:    $currentParLevel++$
8:    $subPlanSet \leftarrow emptySet$
9:    $loopNodesList \leftarrow$ emptyList[to-label-comparator]
10:   for all $currentNode \in$ currentNodesList do
11:      $loopNodesList.add(currentNode)$
12:      processParentNodes(currentNode, freeNodesSet,
             loopNodesList, M, currentParLevel)
13:      if loopNodesList.size == M then
14:         $subPlanSet.add(loopNodesList)$
15:         $loopNodesList.removeAll()$
16:      end if
17:   end for
18:   if loopNodesList IS NOT EMPTY then
19:      $subPlanSet.add(loopNodesList)$
20:      $loopNodesList.removeAll()$
21:   end if
22:   $currentNodesList.removeAll()$
23:   $currentNodesList.addAll(freeNodesSet)$
24:   $freeNodesSet \leftarrow$ emptySet
25:   $plan.add(subPlan)$
26: end while
27: return $plan$
```

7.2 Parallel plan compilation algorithm

Algorithm 1 aims to create a set of independent *reasoning sub-plans* from a *Reasoning Model Dependency Graph*. Each sub-plan is a sequence of maximum of M many rea-

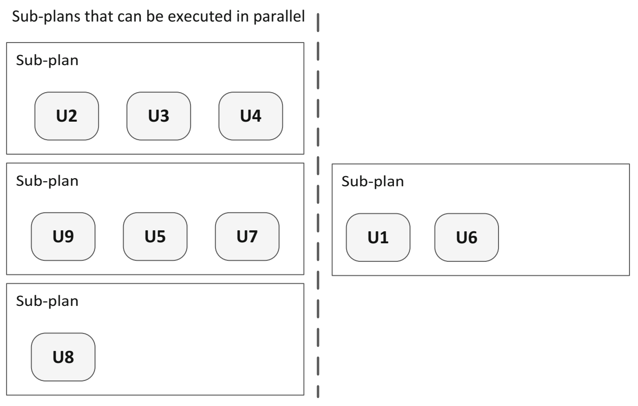


Fig. 10 Execution plan returned by the proposed algorithm for $M = 3$

soning units, where the value of M can be set by the user. More specifically, each sub-plan consists of sequences that do no depend on one another and hence each sub-plan can be executed in parallel. For example, Fig. 10 illustrates the execution plan returned by the proposed algorithm for $M = 3$ for the dependency graph of Fig. 9. According to this plan, the evaluation will be completed in two phases, where during the first phase there will be 3 sub-plans that can be computed in parallel, and in the second phase the overall evaluation completes by computing the remaining sub-plan.

The related process is depicted in Algorithm 1 (*Plan Compilation* algorithm). The algorithm iterates over the nodes in "currentNodesList" (line 10), which initially is the set of all leaf nodes. Gradually nodes are added in, or removed from this list, and the process completes when this list becomes empty (line 6). Initially this list contains all leaf nodes (line 4), i.e., the units that do not depend on other reasoning units. Each time a node is visited, the algorithm checks whether any of the nodes that depend on it, is now *free*. A node is considered to be *free* when *all* of its dependencies have already been placed on a sub-plan. For example, for the dependency graph depicted in Fig. 9, reasoning unit $U6$ will be free only when both of its child units, i.e., $U7$ and $U8$, have been assigned to a sub-plan. This is performed by calling the "processParentNodes" process (line 12) as presented in Algorithm 2. It is though worth noting that in this process all units that depend on the current unit are checked, and if they are free they are either added to "freeNodesSet" in order to be processed in the next iteration, or they are added in "loopNodesList" in order to be part of the current sub-plan set. Once the size of "loopNodesList" is equal to the given parameter M (line 13), a new sub-plan is added to the sub-plan set (line 14) and "loopNodesList" is re-initialized to an empty list (line 15). It is important to note that all units in "loopNodesList" are sorted using the to-label, and hence, all dependencies between the units in "loopNodesList" are respected.

If all nodes in "currentNodesList" have been visited, a new sub-plan is created that contains all units in "loopNodesList"

Algorithm 2 processParentNodes

```
Input: $cn$: unit node, $freeNodesSet$: set of units
$loopNodeList$: List of units, $M$: sub-plan size
$currentParLevel$
1:  $parentNodes \leftarrow cn.parentNodes$
2:  for all $p \in$ parentNodes do
3:    if p.dev-level $\neq$ currentParLevel then
4:      p.level-first-dep = cn
5:      p.dev-level = currentParLevel
6:    end if
7:    p.child-proc++
8:    if p.child-proc == p.child-all then
9:      if p.level-first-dep in loopNodesList AND
           loopNodesList.size < M then
10:       loopNodesList.add(p)
11:       processParentNodes(p, freeNodesSet,
            loopNodesList, M, currentParLevel)
12:     else
13:       freeNodesSet.add(p)
14:     end if
15:   end if
16: end for
17: return
```

(line 18 -20) and then "currentNodesList," that is, now empty, is filled in with all free units in "freeNodesSet" (line 23). The process terminates when all nodes in the reasoning units dependency graph have been visited.

Having outlined the operation of Algorithm 1, we proceed outlining the steps of Algorithm 2, which performs the detection of which nodes are free in each iteration of the main Algorithm 1. Algorithm 2 proceeds by iterating over all unit nodes that are connected with unit node cn with a path of length 1, i.e., the parent nodes (line 1). For each one of them, it initially updates the values for metrics "level-first-dep" and "dev-level" (lines 3 -6), if this is the first immediate dependency visited by the algorithm. It then increases the number of visited immediate dependencies by one (line 7). Subsequently, it checks whether *a)* all child nodes are already free (line 8); *b)* the first free child unit belongs to the current loop list ("loopNodesList") (line 9); *and c)* the sub-plan has not exceeded the maximum given size (line 9), and if yes it recursively calls "processParentNodes" for node p (line 11). Otherwise, if the current parent node is free but, either the size of the sub-plan is equal to M, or the first free child node belongs to another sub-plan it just adds node p to the set of already free nodes (line 13).

7.2.1 Sub-plan compilation example trace

Below, we present a sample trace as a sequence of steps, when the algorithm is applied to the reasoning unit dependency graph depicted in Fig. 9. The sequence of steps is depicted in Figs. 11–15.

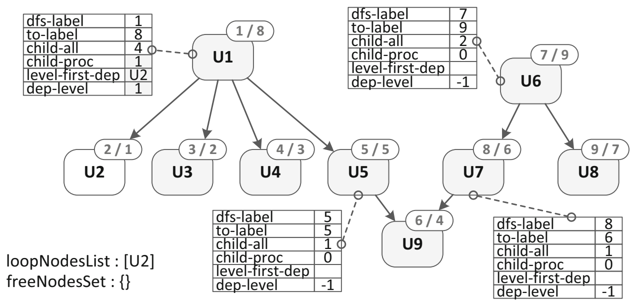

dfs-label 1 | to-label 8 | child-all 4 | child-proc 1 | level-first-dep U2 | dep-level 1
dfs-label 7 | to-label 9 | child-all 2 | child-proc 0 | level-first-dep | dep-level -1
1 / 8 U1
7 / 9 U6
2 / 1 U2
3 / 2 U3
4 / 3 U4
5 / 5 U5
8 / 6 U7
9 / 7 U8
6 / 4 U9
dfs-label 5 | to-label 5 | child-all 1 | child-proc 0 | level-first-dep | dep-level -1
dfs-label 8 | to-label 6 | child-all 1 | child-proc 0 | level-first-dep | dep-level -1
loopNodesList : [U2]
freeNodesSet : {}

Fig. 11 Step 1—visiting node U_2

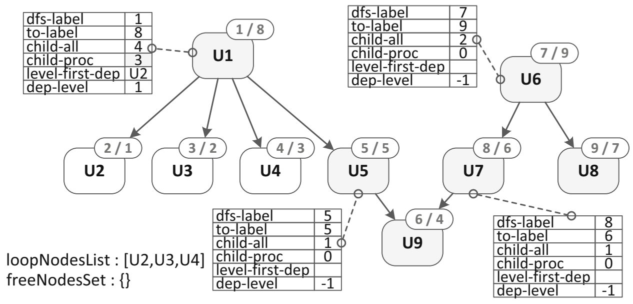

dfs-label 1 | to-label 8 | child-all 4 | child-proc 3 | level-first-dep U2 | dep-level 1
dfs-label 7 | to-label 9 | child-all 2 | child-proc 0 | level-first-dep | dep-level -1
1 / 8 U1
7 / 9 U6
2 / 1 U2
3 / 2 U3
4 / 3 U4
5 / 5 U5
8 / 6 U7
9 / 7 U8
6 / 4 U9
dfs-label 5 | to-label 5 | child-all 1 | child-proc 0 | level-first-dep | dep-level -1
dfs-label 8 | to-label 6 | child-all 1 | child-proc 0 | level-first-dep | dep-level -1
loopNodesList : [U2,U3,U4]
freeNodesSet : {}

Fig. 12 Step 2—visiting nodes U_3 and U_4

Step 1 Initially, when the loop at line 6 (Algorithm 1) executes for the first time, "currentNodesList" contains nodes U_2, U_3, U_4, U_9, U_8 which are leaf nodes, and the "currentParLevel" variable is increased from 0 to 1 (line 7 of Algorithm 1). Then (see Fig. 11), node "U2" is visited and added in the $loopNodesList$, while the values for "child-proc," "level-first-dep," and "dep-level" of its parent node are set to "1," "U_2," and "1," respectively.

Step 2 Subsequently (Fig. 12) and by keeping on processing, the nodes in the *currentNodesList*, nodes U_3 and U_4 are visited and added to *loopNodesList*, increasing the value of "child-proc" for node U_1 to "2" and then to "3." At this point, as the size of *loopNodesList* is equal to max size determined by the parameter M which is 3 in our example, the first sub-plan is created with nodes $[U_2, U_3, U_4]$, and the list *loopNodesList* is emptied.

Step 3 The algorithm proceeds by visiting node U_9 (Fig. 13) and adding it to the $loopNodesList$, setting the values "child-proc," "level-first-dep," and "dep-level" for its parent node U_5 to "1," "U_9" and "1," respectively. As now "child-proc" is equal to "child-all" for node U_5, and its "level-first-dep" is equal to U_9 which is included in the "loopNodesList," U_5 is also added in the "loopNodesList." Additionally, U_1 which is the parent node of U_5 is added in the "freeNodesSet" as now all of its immediate dependencies have been visited and added to a sub-plan.

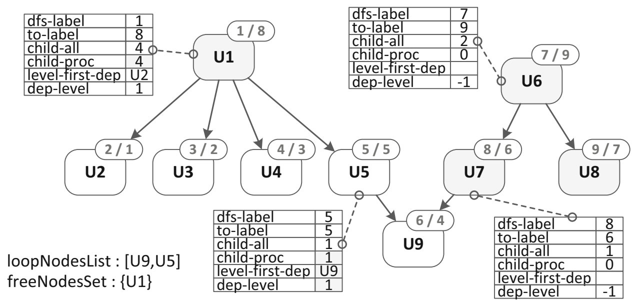

dfs-label 1 | to-label 8 | child-all 4 | child-proc 4 | level-first-dep U2 | dep-level 1
dfs-label 7 | to-label 9 | child-all 2 | child-proc 0 | level-first-dep | dep-level -1
1 / 8 U1
7 / 9 U6
2 / 1 U2
3 / 2 U3
4 / 3 U4
5 / 5 U5
8 / 6 U7
9 / 7 U8
6 / 4 U9
dfs-label 5 | to-label 5 | child-all 1 | child-proc 1 | level-first-dep U9 | dep-level 1
dfs-label 8 | to-label 6 | child-all 1 | child-proc 0 | level-first-dep | dep-level -1
loopNodesList : [U9,U5]
freeNodesSet : {U1}

Fig. 13 Step 3—visiting nodes U_9 and U_5

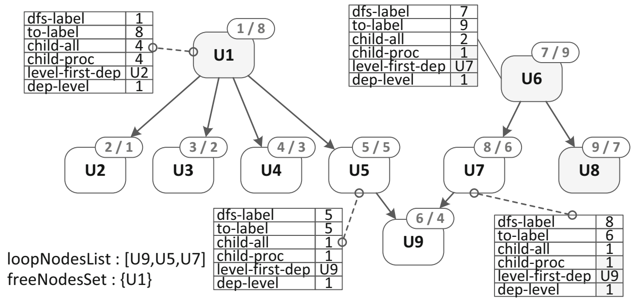

dfs-label 1 | to-label 8 | child-all 4 | child-proc 4 | level-first-dep U2 | dep-level 1
dfs-label 7 | to-label 9 | child-all 2 | child-proc 1 | level-first-dep U7 | dep-level 1
1 / 8 U1
7 / 9 U6
2 / 1 U2
3 / 2 U3
4 / 3 U4
5 / 5 U5
8 / 6 U7
9 / 7 U8
6 / 4 U9
dfs-label 5 | to-label 5 | child-all 1 | child-proc 1 | level-first-dep U9 | dep-level 1
dfs-label 8 | to-label 6 | child-all 1 | child-proc 1 | level-first-dep U9 | dep-level 1
loopNodesList : [U9,U5,U7]
freeNodesSet : {U1}

Fig. 14 Step 4—visiting node U_7

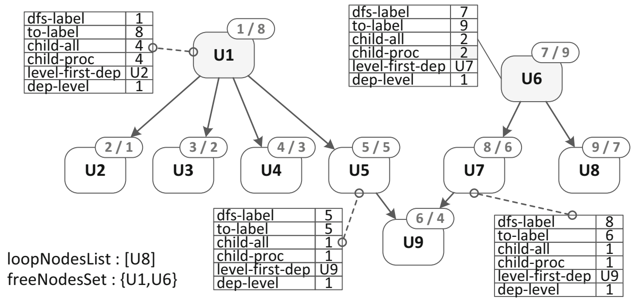

dfs-label 1 | to-label 8 | child-all 4 | child-proc 4 | level-first-dep U2 | dep-level 1
dfs-label 7 | to-label 9 | child-all 2 | child-proc 2 | level-first-dep U7 | dep-level 1
1 / 8 U1
7 / 9 U6
2 / 1 U2
3 / 2 U3
4 / 3 U4
5 / 5 U5
8 / 6 U7
9 / 7 U8
6 / 4 U9
dfs-label 5 | to-label 5 | child-all 1 | child-proc 1 | level-first-dep U9 | dep-level 1
dfs-label 8 | to-label 6 | child-all 1 | child-proc 1 | level-first-dep U9 | dep-level 1
loopNodesList : [U8]
freeNodesSet : {U1,U6}

Fig. 15 Step 5—visiting node U_8

Step 4 In a similar manner as before, the values "child-proc," "level-first-dep," and "dep-level" for node U_7 are set to 1, U_9 and 1, respectively (Fig. 14). However, as "child-proc" is equal to "child-all" for node U_7, the node is also added in the "loopNodesList." Once more, as the size of "loopNodesList" is equal to M which is 3 containing the nodes U_9, U_5, U_7, the second sub-plan $[U_9, U_5, U_7]$ is created, and the $loopNodesList$, once more, is emptied.

Step 5 In step 5, the metrics of node U_6 are updated (Fig. 15), node U_8 is visited and added in the $loopNodesList$, setting the value of "child-proc" to "2" for node U_6. Since all child nodes of U_6 have been processed, U_6 is added to "freeNodesSet."

Step 6 Finally, as all nodes in "currentNodesList" have been visited, the first execution of the loop starting at line 6 of

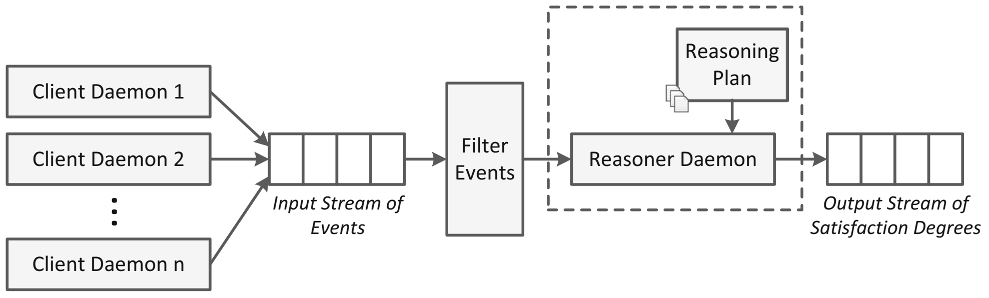


Fig. 16 Architecture of the inference engine used in the proposed framework

Algorithm 1 completes, and the "currentNodesList" is emptied. Nodes in "loopNodesList," e.g., node U_8, are used to create a new sub-plan (lines 18–21 of Algorithm 1), and nodes in "freeNodesSet" are now added in "currentNodesList" which is empty at this point (lines 22–23 of Algorithm 1). At this point, if the graph was larger, the algorithm could be re-applied with "currentNodesList" be the "freeNodesSet." The algorithm will terminate when all nodes in the graph are visited.

7.3 Inference engine

Having created a collection of *sub-plans* that can be evaluated in parallel, we outline the inference engine architecture and the inference control strategy.

7.3.1 Inference engine architecture

An overview of the inference engine architecture is illustrated in Fig. 16. As it is depicted in Fig. 16, the required events that correspond to the values of the system's observable characteristics are collected at run time by a set of "Client Daemons" acting as loggers and monitors, each one of which is responsible to periodically collect the values of the observable characteristics from the various components of the system (e.g., databases, application servers). This process results in the creation of a continuous input stream which can be filtered using a technique like the one introduced in [32] which is not in the scope of this paper.

Subsequently, the filtered stream of events should be processed by the "Reasoner Daemon" using the parallel plan that has been compiled off-line, in order to provide a continuous output stream of satisfaction degrees for the required system goals. In short, it is responsible to *periodically* run the reasoning process and provide the calculated results. At this point, it is important to note that there are multiple criteria that can be used to periodically trigger the evaluation of the provided model. One possible way to trigger the reasoning could be the number of events received. In this case, reasoning is triggered at fix number of events, e.g., every 1000 events. Another possibility would be to detect specific events in the input stream that should trigger the evaluation of the model, or even trigger reasoning at regular time intervals no matter what the number of events received is. Finally, a composite criterion that uses combinations of the above criteria can also be used. However, regardless of the triggering mechanism, the framework should be able to provide an output stream of values that reflect in real time, the degree in which the system complies with the predefined requirements.

7.3.2 Reasoning process strategy

As it has been already mentioned, the execution plan returned by the proposed algorithm consists of a sequence of sets of sub-plans (e.g., Fig. 10). A set of sub-plans can be executed only after all sets preceding in the sequence of the execution plan have been already applied. In contrast, sub-plans that belong to the same set can be executed in any order, or in parallel, as there are no dependencies between them. Hence, we apply the following strategy for the reasoning engine. First, we initialize a pool of threads (or processes) where each thread can be used to complete the execution of one sub-plan at a time. Subsequently, we get the first set of sub-plans in the execution plan and assign one sub-plan to each one of the threads in the pool. When a thread completes the execution of a sub-plan, a new sub-plan from the set is assigned to it, or if there is no other sub-plan in the set, it is returned to the pool and remains there until all sub-plans in the set are executed. Once all sub-plans in the set have been executed, we repeat the same process for the next set of sub-plans in the execution plan. Through this strategy, we manage to execute the whole plan in a way that respects all dependencies that exist between the reasoning units, and at the same time execute in parallel the appropriate reasoning units.

8 Evaluation

In this section, we present experimental results regarding the performance of the proposed framework. The performance of the framework was evaluated in randomly generated FGMs, by adjusting a number of structural parameters as follows:

– total goal nodes in the model: 300–4500, with an interval of 200 nodes
– maximum number of child nodes per node: 5, 10, 20
– percentage of contribution links in the model: 0, 30 %

In all generated FGMs, the probability of a decomposition to be either and AND or an OR decomposition was set to 50 %, while in a similar manner the probability of each one of the 4 contribution types, i.e., S^+, S^-, D^+, D^- was set to 25 % for each type. We have also produced initial random assignments of values for the leaf nodes of each model, in order to provide the initial input to the reasoner. We then used the generated models to evaluate the truth values for their root nodes and recorded the time and space resources as a function of the model's size and complexity (i.e., number of nodes and contribution links). Initially, we used a reasoning plan generated by a topological sorting of the Reasoning Units Dependency Graph, for example, the plan given in Eq. (11) for the graph of Fig. 9, which is used for the sequential execution of the reasoning process. Subsequently, the reasoning is reapplied using a parallel reasoning plan produced by the proposed algorithm with $M = 1$.

We conducted two series of experiments, one to investigate how the parallel and the sequential execution scales depending on the number of events that should be processed in unit time, and a second one to investigate how the reasoning time and the required amount of memory are affected for models of varying size and complexity.

8.1 Processing large volume of events

Initially, we investigate how the two approaches (i.e., the parallel and the sequential one) scale in terms of execution time when a large volume of incoming events need to be processed. More specifically, we consider the use case where for every 1000 events of logged data the reasoning process is triggered by an alarm. This implies that when, for example, the rate of incoming events is 5000/s, then the reasoning should be applied 5 times within one second. For this experiment, we utilized 3 randomly generated models of 200 nodes and applied the reasoning when the incoming data rate increases from 2000 events/sec to 50 000 events/s. We then recorded the time required for the reasoning to complete when either a sequential or a parallel execution plan is used. The results are depicted in Fig. 17. According to these results, while the time required increases almost linearly to the rate of incoming events for both approaches, the growth rate in the case of the parallel plan is smaller than the one in the case of the sequential plan.

8.2 Memory and time requirements

In this series of experiments, our aim is to investigate how the reasoning time and the required amount of memory are

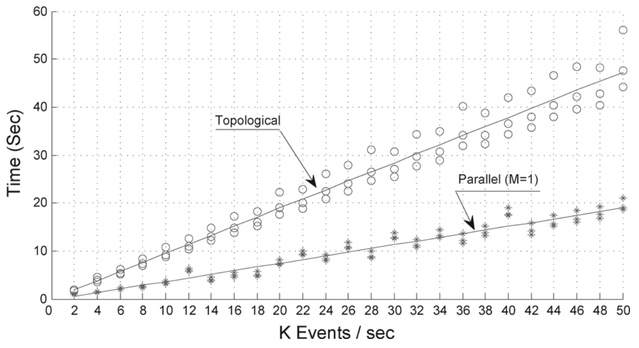


Fig. 17 Time required for reasoning for models of 200 nodes when a reasoning session is triggered for each 1000 events received

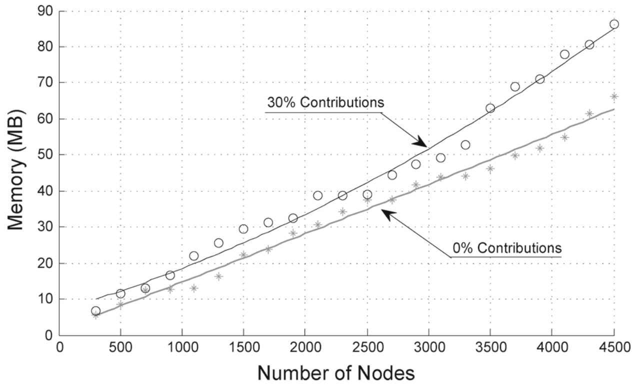


Fig. 18 Memory required for reasoning when either no contributions exist in the model or contribution links percentage is equal to 30 %

affected, when either the complexity of the model increases by adding more contribution links in the models, or a different strategy (sequential or parallel) is used to build and evaluate the reasoning plan.

Regarding memory requirements, the amount of memory is almost the same no matter whether a sequential or parallel execution plan is used. This is because memory depends mainly on the size and the complexity of the model rather than on the type of reasoning applied. As depicted in Fig. 18, the amount of memory required increases almost linearly to the size of the models, with a value equal to 5.5 MB for models with a mean number of nodes equal to 500, and a value equal to 66 MB for models with a mean number of nodes equal to 4500 when no contribution links exist. Similarly, when the contribution links-to-node ratio is equal to 30 %, the required memory also increases linearly to the size of the models; however, the memory required in this case starts from 6.5 MB for models with a mean number of nodes equal to 500 and reaches a value equal to 88 MB for models with a mean number of nodes equal to 4500.

As discussed earlier in the paper, in real-life deployments, goal models provide the requirements specifications that are to be verified, while the reasoner assigns truth values to the specification, given initial truth values to the leafs. In this

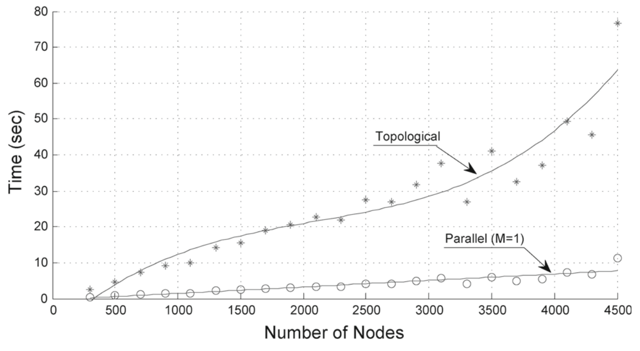


Fig. 19 Time required for reasoning when no contributions exist in the model and a validation overhead of 10–100 ms is assumed

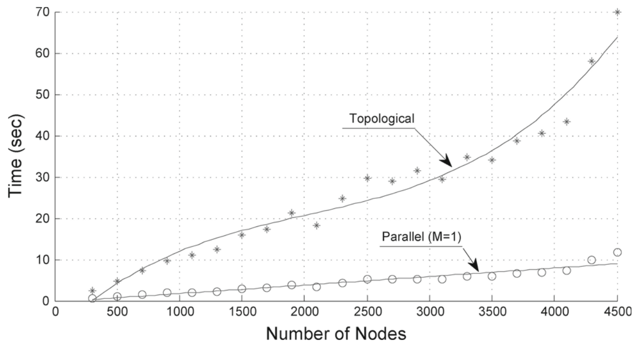


Fig. 20 Time required for reasoning when contribution links percentage is equal to 30 % and a validation overhead of 10–100 ms is assumed

respect, every truth value evaluated by the reasoner for every non-leaf goal model node has to be further validated for consistency to ensure that there are no evidence in the system logs that contradicts the node's evaluated truth value, which otherwise would signify a possible violation of the requirement. For our experiments, we assumed a random validation overhead of 10–100 ms for each node and performed experiments to evaluate the time required to evaluate and validate large goal models utilizing the sequential and parallel reasoning versions.

As depicted in Figs. 19 and 20, the time required for the reasoning process to complete as a function of model size is shown to be suitable for run time use. The experiments indicate that the parallel version of the execution of the reasoning plan has a significant performance increase over the sequential version. More specifically, for models with 4500 nodes, the time required for the parallel execution is almost 85 % lower than the time required when the sequential execution produced using topological sorting. For the parallel version, the experimental results indicate that in order to evaluate and validate goal models of 4500 nodes, a time of approximately 10 s is required in a single computer running 8 threads, in 16 CPU cores. Additionally, as the number of contribution links increases (Fig. 20), and thus, the complexity of the models increases too, we do not observe a significant increase in time performance, as in our approach the number of reasoning units is not affected by the number of contribution links in the model. Finally, another benefit of the parallel version is that different sub-plans can be evaluated on servers or processes that are physically closer to where log data required for a node's post-validation, reside. In this respect, instead of fetching log data to the reasoning engine, sub-plans are moved to where data reside, increasing thus the potential throughput of the reasoning system. Overall, our experimental results indicate that the proposed approach can scale up, so that it can be used at run time for sizeable models.

8.3 Discussion and threats to validity

In this section, we are going to describe some important details of the proposed method and also discuss the threats to its validity.

As this has been already mentioned, for the proposed method to be applied, the given fuzzy goal model should not contain cycles. This is mainly because of the way the reasoning plan is being created and from the fact that there is no topological sorting for a directed graph with cycles, hence if cycles exist in the Reasoning Unit Dependency Graph no reasoning plan could be created. Given a graph with cycles, it is most probable that cycles have been created duo to specific contribution link as decomposition links describe how goals are decomposed to simpler sub-goals and most of the times are added to the model by applying a top-down analysis. Consider now the model of Fig. 21 which contains a cycle. Taking into account the semantics of contribution links, as these have been described in Sect. 4, we can decompose the initial model to the two acyclic sub-models depicted in Fig. 21. This can be done because the D^+ contribution link from node $B2$ to node A is only "triggered" (i.e., takes part in the reasoning process) when $B2$ is false. However, in this case A becomes also false, because of the semantics of D^+ contribution, which means that the path from A to $B2$ will not be "triggered" as the S^+ contribution links is only "triggered" when A is true. While this strategy of decomposing a graph which contains cycles to a set of acyclic graphs is valid when the nodes are crisp, it cannot be applied to fuzzy goal models. This is because fuzzy nodes may have a truth value in the interval [0,1] rather than a value in the set {0,1}, and the rules that correspond to contribution links are always "triggered" no mater what the value of the source node is.

Regarding the use of fuzzy reasoning instead of a probabilistic one, probabilistic reasoning approaches have been extensively proposed in the literature as an efficient and sound method to reason under uncertainty over goal models. In these cases, the analysis starts by defining which are the probabilities of certain events to be occurred and the final result corresponds to the probability of the requirements to hold, enabling thus the comparison between alternatives in the

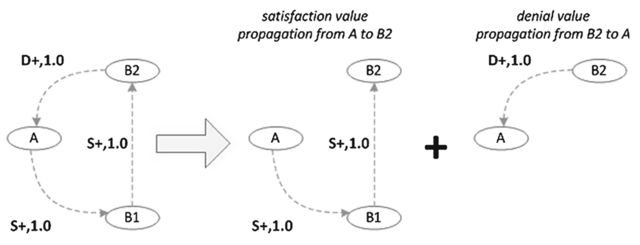

satisfaction value propagation from A to B2
denial value propagation from B2 to A
D+,1.0
B2
A
S+,1.0
B1
S+,1.0
B2
A
S+,1.0
B1
S+,1.0
+
D+,1.0
B2
A

Fig. 21 Decomposition of a goal model with cycles to two acyclic sub-models

design phase. Fuzzy logic, on the other hand, can very well deal with vagueness and also eases the encoding of human expertise and reasoning through the use of linguistic variables (e.g., is highly/poorly satisfied). In the problem under study, we consider that we know exactly which events hold (as they are collected from the system's monitoring infrastructure rather than the probabilities these events may have occurred), and so the primary aim is calculating the degree of satisfaction of a property or a requirement, given the vagueness of how an observed event contributes toward or against it, as opposed calculating the conditional probability, a property or requirement holds, given the probabilities of the evidence supporting it. In this context, the probabilistic approach could serve as an initial step toward modeling uncertainty on collecting or observing system events, followed by the fuzzy approach which could serve toward modeling vagueness on the impact an event has on its parent goals.

Another issue that requires attention is instance versus class level reasoning in goal models. The problem has been identified in [18] and relates to how a run time environment can select the way by which "generic" class level goal model nodes can be evaluated at run time along with "specific" instance level goal model nodes. In order to address this problem efficiently, novel run time architectures have to be devised. Such architectures should be able to recognize and prioritize the evaluation of different types of goal model nodes as to maximize reasoning throughput.

The main threats to the validity of the proposed method are a) the elicitation of the contribution confidence values for the contribution links, b) the maximum size of goal models that can be used in practice, and c) the selection of the appropriate set of the membership functions for the fuzzy nodes.

Regarding the contribution confidence values, while when applying *qualitative* reasoning, the propagation of labels becomes rapidly inconclusive as we move up in the goal refinement tree [34], a common problem encountered to almost every *quantitative* reasoning approach is how contribution confidence values are assigned to contribution links. This problem, and whether it is easier to comprehend qualitative rather than quantitative labels is the subject of [35,36], where authors present results which verify that the use of numerical values increases the comprehension of the models even for non-experts, and they also propose an elicitation method for these values.

Furthermore, the existence of complex software systems can result in models with more than 10 000 requirements, a context that Wnuk et al. refer to as Very Large-Scale Requirements Engineering (VLSRE) [48,49]. Additionally, in the context of this paper, we take the view that stakeholders can not only define custom goal trees, but can also use predefined ones from repositories, where requirements are created from experts or extracted utilizing text mining techniques from specifications and policy documents [41]. Using such repositories, composite goal models of varying size and complexity can be created.

Finally, selecting the most appropriate membership functions is one of the most important problems of fuzzy controllers development. While there are some general guidelines proposed in the literature, and methods have been proposed for the elicitation of membership functions using genetic algorithms [24], membership function selection is mainly based on domain expertise and results interpretation needs [43]. An analysis of the impact different membership functions have on the quality of the final results can be found in [13], where we compare the satisfaction degrees calculated for a certain node as the parameters of the membership functions for the two linguistic variables "low" and "high" are modified and give some general guidelines regarding the membership function parameters.

9 Working example

While in the previous section we used random goal models of varying size and complexity in order to evaluate the time and space performance of the proposed method, in this section we provide a working concrete example of applying the method to a middle sized real-life model. As there is no publicly available set of goal models that can be used as a benchmark for goal model reasoning techniques, we have tried to contact fellow academic groups in order to acquire large real-life goal models which can be utilized to run additional real-life experiments.

The model used in this section is an i* Strategic Rational model that has been provided to us by Dr Jennifer Horkoff.[1] This model is depicted in Fig. 22. It contains almost 400 nodes that correspond to *Tasks*, *Goals*, or *Softgoals* and 500 links (e.g., "Break," "Hurt"), and describes the requirements of a counseling service for kids, as this has been modeled during the first phases of a project aiming at capturing the goals and the interactions within the organization.

More specifically, the model depicts the requirements of the phone and Web services of the organization related to kids

[1] http://www.cs.toronto.edu/~jenhork/.

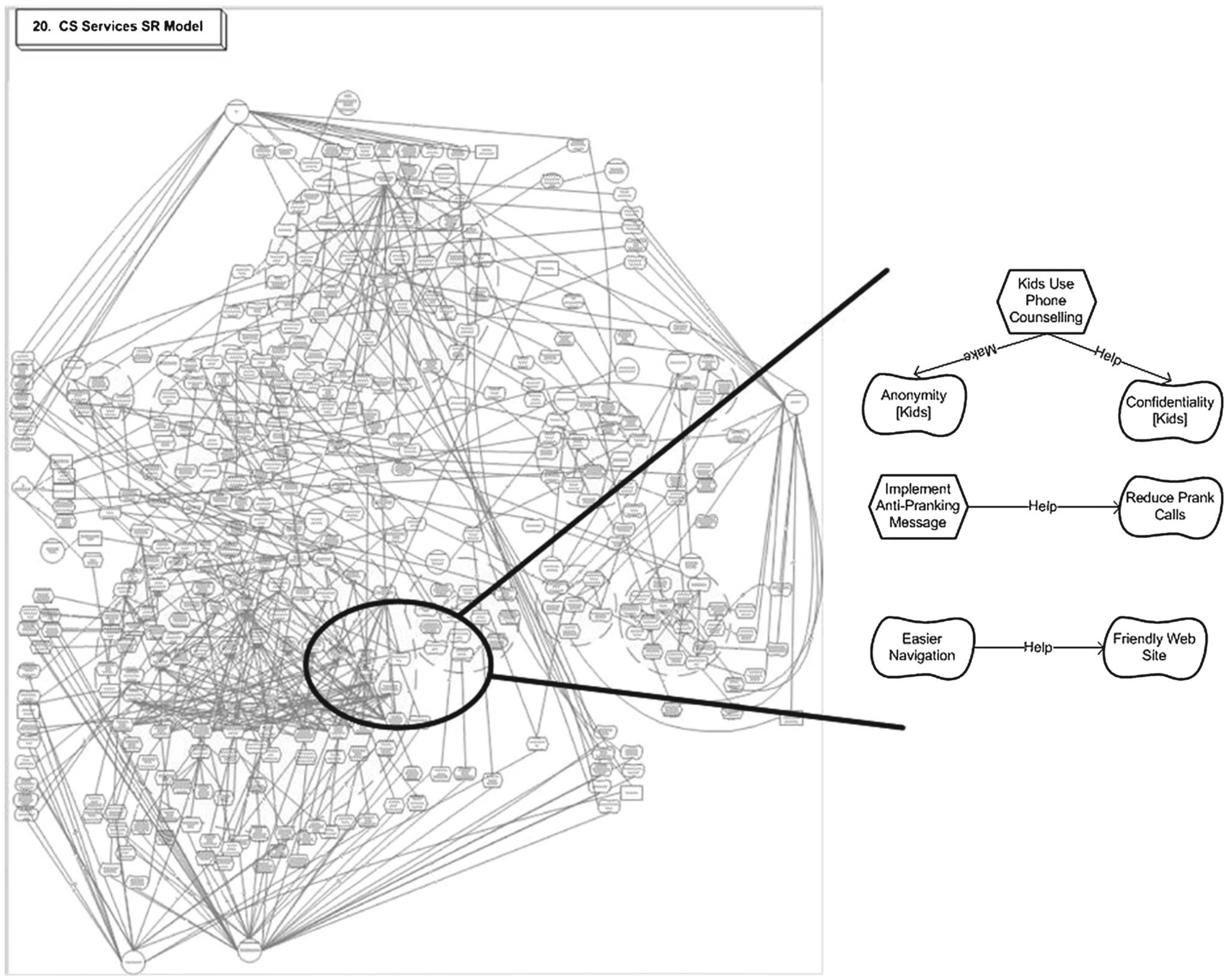


Fig. 22 Overview of the model used in the working example

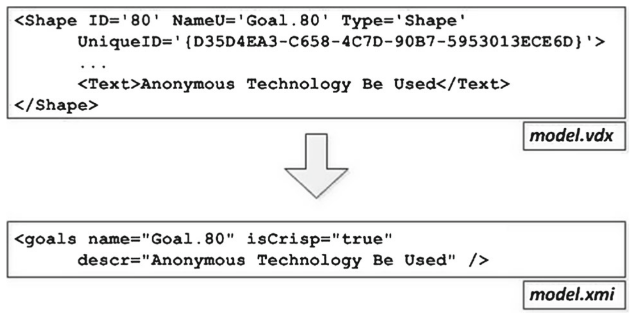
counseling. Most of the requirements are related to preserving the "Anonymity" and the "Confidentiality" of the kids or of their parents when they use the services, which are *Soft-goals*, or in terms of the analysis applied in the context of this paper *fuzzy goals*. Also, depending on the type of service, there is, for example, the requirement to "reduce prank calls" by "implementing a proper anti-pranking message", in the case of phone services, or to "increase the ease of use" in the case of web services.

Table 3 Transformation of i* links to contribution and decomposition links

i* model link	Fuzzy goal model link
Make	S^+ with contribution confidence 1.0
Break	S^- with contribution confidence 1.0
Help	S^+ with contribution confidence w
Hurt	S^- with contribution confidence w
Means-ends	OR decomposition
Decomposition	AND decomposition

9.1 Transformation process

The model was originally created using the Microsoft Visio application and exported in *vdx* format, which is practically an *xml* file, allowing thus the model to be programmatically manipulated. We have created a transformation process which given a *vdx* file creates and populates a fuzzy goal model which is an *ecore* instance given in the form of an *xmi* file, that can be used by the reasoner and the domain model introduced in the context of this paper. More specifically, as the given model is an i* instance, we transform each one of the i* links (e.g., "Help," "Make") to an equivalent contribution or decomposition link in the target model. The transformation of the i* links to contribution and decomposition links of a fuzzy goal model is summarized in Table 3.



```
<Shape ID='80' NameU='Goal.80' Type='Shape'
      UniqueID='{D35D4EA3-C658-4C7D-90B7-5953013ECE6D}'>
      ...
      <Text>Anonymous Technology Be Used</Text>
</Shape>
```

```
<goals name="Goal.80" isCrisp="true"
      descr="Anonymous Technology Be Used" />
```

Fig. 23 Example transformation of a vdx file to an ecore instance given in the form of an xmi file

Regarding the nodes of the given model, *Task* and *Goal* nodes were transformed to crisp goal nodes, while for each *Softgoal* in the initial model a corresponding fuzzy goal node was created in the generated fuzzy goal model. An overview of the transformation process of a *vdx* file to an ecore instance is depicted in Fig. 23.

9.2 Running cases

Once the transformation has been completed, we can use the resulting fuzzy goal model in order to apply the reasoning technique introduced in this paper. In a more descriptive manner, we initially generate the corresponding reasoning model using the transformation process described in Sect. 5, and we then compile a parallel reasoning plan using Algorithm 1. This allows us to compute how the satisfaction degrees of certain root nodes are modified for given sets of initial values of the leaf nodes of the model.

More specifically, we consider a scenario in which the quality of the services provided by the system is poor, i.e., the satisfaction degree calculated for the fuzzy root node *"High Quality Services"* is low, and hence, the stakeholders of the system are interested in applying changes that will result in the increase in the satisfaction degree of the *"High Quality Services"* goal. Subsequently, we examine how the satisfaction degree calculated for the *"High Quality Services"* fuzzy root node is gradually increased as the satisfaction degrees of certain leaf nodes which are connected to the root node through a path of decomposition and contribution links are altered. Note that nodes that do not affect the value of the *"High Quality Services"* root node remain unchanged between the various cases so as to ensure that the alterations observed are due to the changes in these specific leafs. The set of leaf nodes and the values assigned to these nodes for each case are summarized in Table 4. For each one of the three cases presented in Table 4, we run the reasoning process and deduct the value of the root node by applying the parallel reasoning process presented in this paper.

9.2.1 Case 1—Service quality goal denial

The first case depicts a scenario in which leaf nodes which have overall positive contributions toward the goal node *"High Quality Services"*, either are denied or have a low satisfaction value (i.e., all crisp leaf nodes in Table 4 are false, while all fuzzy leaf nodes have low satisfaction degrees). More specifically, considering that all the leaf nodes in Table 4 contribute directly or indirectly to the top goal, and by considering a fuzzy membership function like the one depicted in Fig. 7, the first scenario assumes a 10 % satisfaction degree for leaf #05, 10 % satisfaction degree for leaf

Table 4 Initial values assigned to the leaf nodes for the cases of the working example

Leaf node	Crisp/fuzzy	Case 1	Case 2	Case 3
01. Implement Voice Counselling	Crisp	false	false	true
02. Sufficient Counselling Resources	Crisp	false	false	false
03. Implement Phone Feedback	Crisp	false	false	false
04. Trace Calls	Crisp	false	true	true
05. Increase Emphasis on Online Feedback Form	Fuzzy	10 % (1.0/0.0)	60 % (0.0/0.4)	90 % (0.0/1.0)
06. Block Kids who Display Inappropriate Behavior	Crisp	false	false	true
07. Web Site Content Be Updated Daily	Crisp	false	false	true
08. Implement Video Counselling	Crisp	false	false	true
09. Decrease [Phone Waiting Time]	Fuzzy	10 % (1.0/0.0)	10 % (1.0/0.0)	70 % (0.0/0.6)
10. Implement Anti-Pranking Message	Crisp	false	false	true
11. Confidentiality [Services]	Fuzzy	20 % (0.8/0.0)	40 % (0.4/0.0)	85 % (0.0/0.9)
12. Inform Kids about Anonymity of Web Services	Crisp	false	false	true
13. Counsellors Be Professionally Trained	Crisp	false	false	true

For fuzzy goal nodes' satisfaction degrees, we also provide the values extracted using the fuzzification process described in Sect. 5.2.2 in the format (V_{low}/V_{high})

#09, and a 20 % satisfaction degree for leaf #11. These fuzzy values correspond to a user or the system assigning a LowSat value 1.0 and a HighSat value 0.0 that collectively yield a total 10 % satisfaction degree. Similarly the LowSat and HighSat values of 0.0 and 0.8 yield a total 20 % satisfaction degree. By evaluating the model, the satisfaction degree for the root goal node *"High Quality Services"* is then computed to 49 %, as all leaf nodes that are connected to the root trough a path that consists of S^+ contributions are false or have a very low satisfaction degree as presented above.

9.2.2 Case 2—Service quality goal partial satisfaction

The second case depicts the impact the increase in values of two fuzzy leaf nodes have on the satisfaction degree of the root node. In this respect, the fuzzy leaf nodes #05 and #11 increase to 60 and 40 %, respectively. These fuzzy values correspond to a (0.0, 0.4) LowSat and HighSat pair that yields an overall 60 % satisfaction degree and the (0.4, 0.0) LowSat and HighSat pair that yields an overall 40 % satisfaction degree.

In turn, the increase in the satisfaction degrees of the leaf nodes results in the partial satisfaction of the fuzzy root which now has a satisfaction degree equal to 57 % as opposed to 49 % in the first scenario.

9.2.3 Case 3—Service quality goal satisfaction

The third case depicts the scenario in which both the crisp nodes are satisfied and the fuzzy nodes have high satisfaction scores. This situation is presented in Table 4 where the crisp nodes are true, and the fuzzy nodes have a satisfaction degree greater than 70 %. By evaluating the model in this scenario, the overall satisfaction degree of the root node increases to 77 % from 57 %, which implies that the top goal *"High Quality Services"* is satisfied to a greater extent than in the previous cases.

10 Conclusion and future work

As software systems become interconnected and offered via virtualized and dynamically provisioned platforms, forming thus ultra-large-scale systems, a key question that emerges is whether specific requirements still hold when systems interact in new and unforeseen ways, or when new resources are dynamically provisioned to meet varying load and stakeholder needs. This question encompasses two main issues. The first issue deals with being able to reason at run time, and in the presence of vague evidence, as to whether specific system requirements are affected as a result of dynamic alterations in the system's operating environment and resources. The second issue deals with being able to deal with tractability issues, so that real time, or near real time, performance can be achieved. In this respect, it is important to be able to parallelize the computation and distribute it to different servers, so that performance can stay within acceptable and tractable limits as load, systems, or logged data increase.

In this paper, we propose fuzzy goal models as a way of dealing with modeling uncertainty and incomplete knowledge about requirements inter-dependencies, and we introduce a reasoning framework for fuzzy goal models that can be used to analyze and evaluate system requirements at run time. The system takes as input, data collected as the system operates. Fuzzy goal models denote dependencies between various requirements as well as, relationships that exist between monitored events and the system requirements. As these models may grow in size when more systems become interconnected, we propose a technique that allows for the analysis of goal model node dependencies, so that independent areas (or subgraphs) can be identified, and fuzzy reasoning can be parallelized. In this respect, given a goal model with fuzzy and crisp goal nodes, we introduce first a process that allows for the generation of fuzzy rules from fuzzy goal models and second, a modeling transformation process that allows for the generation of a dependency graph, where each node is an abstraction (i.e., reasoning unit) of the evaluations that should be performed in order to calculate the degree a single goal node is satisfied or not. Subsequently, we create a sequence of parallel evaluations, we refer to as the *reasoning plan*, that ensures that all reasoning units are executed in an order that respects the dependencies that exist between the nodes in the original fuzzy goal model. Hence, as events are collected from the running system forming a fact base, the reasoning plan and the rules generated at design time, can be used to infer deductions as to whether and to what degree, specific requirements or goals may still hold given the altered state of the system and the events been logged.

In this context, we evaluated the performance of the proposed framework by conducting a series of experiments with randomly generated models of varying size and complexity. More specifically, we evaluated the application of the proposed method with respect to execution time and amount of memory required for models of different sizes. The experimental results indicate that assuming that there is a computational overhead to validate the logged data against individual goal nodes, the amount of time required for the parallel reasoning to complete, is significantly lower compared to sequential one.

The work presented in this paper can be extended in a number of possible directions. One possible direction is to investigate how reasoning can be applied to goal models with temporal dependencies like the ones introduced in [37,38], where pre- and post-condition dependency links are used to denote the temporal dependencies between goal model nodes. These extensions may be proven useful for enhancing the expressiveness of the modeling notation.

A second direction is to consider adapting and applying the work presented in [12]. More specifically, the genetic algorithm introduced in [12] for the top-down reasoning of crisp goal models could also be adapted and applied to goal models with fuzzy goal nodes, where the evaluation of the possible solutions would be performed by the parallel algorithm introduced in this paper.

A third direction is to apply the method on fuzzy goal models that denote requirements at the instance level rather than at the class level. In this case, the models in which the reasoning should be applied at run time would have been larger and more complex, making the utilization of the parallel reasoning approach even more useful.

Finally, a fourth direction is to investigate the use of a feedback loop strategy, in order to allow self-adaptation of the system under study. Once the satisfaction degrees for all goal nodes are evaluated, the feedback loop could be used to identify the actions required and to propose a plan of changes in order to satisfy currently failed system goals. This work can be used to devise controllers for autonomic and self-adaptive systems, an area that attracts significant attention from researchers and practitioners alike.

Acknowledgements This research has been co-financed by the European Union (European Social Fund ESF) and Greek national funds through the Operational Program "Education and Lifelong Learning" of the National Strategic Reference Framework (NSRF)—Research Funding Program: Heracleitus II. Investing in knowledge society through the European Social Fund. We would also like to thank Dr. Jennifer Horkoff for providing us with a set of real-life requirement's models.

References

1. Programmable controllers—part 7: Fuzzy control programming. Tech. Rep. IEC 61131-7:2000, International Electrotechnical Commission (2000)
2. Ultra-Large-Scale Systems The Software Challenge of the Future: 1st edn. Carnegie Mellon University, Pittsburgh, PA, USA, Software Engineering Institute (2006)
3. Centroid (2016). http://en.wikipedia.org/wiki/Centroid
4. T-norm (2016). https://en.wikipedia.org/wiki/T-norm
5. Ali, R., Dalpiaz, F., Giorgini, P.: Location-based software modeling and analysis: tropos-based approach. In: Li, Q., Spaccapietra, S., Yu, E.S.K., Olivé, A. (eds.) Conceptual Modeling—ER 2008, 27th International Conference on Conceptual Modeling, Barcelona, Spain, October 20–24, 2008. Proceedings, Lecture Notes in Computer Science, vol. 5231, pp. 169–182. Springer (2008)
6. Ali, R., Dalpiaz, F., Giorgini, P.: A goal-based framework for contextual requirements modeling and analysis. Requir. Eng. **15**(4), 439–458 (2010)
7. Amyot, D., Ghanavati, S., Horkoff, J., Mussbacher, G., Peyton, L., Yu, E.S.K.: Evaluating goal models within the goal-oriented requirement language. Int. J. Intell. Syst. **25**(8), 841–877 (2010)
8. Baresi, L., Pasquale, L., Spoletini, P.: Fuzzy goals for requirements-driven adaptation. In: RE, pp. 125–134. IEEE Computer Society (2010)
9. Bencomo, N., Belaggoun, A.: Supporting decision-making for self-adaptive systems: From goal models to dynamic decision networks. In: Doerr, J., Opdahl, A.L. (eds.) Requirements engineering: foundation for software quality—19th International working conference, REFSQ 2013, Essen, Germany, April 8–11, 2013. Proceedings, Lecture Notes in Computer Science, vol. 7830, pp. 221–236. Springer (2013)
10. Bi, J., Zhu, Z., Tian, R., Wang, Q.: Dynamic provisioning modeling for virtualized multi-tier applications in cloud data center. In: Cloud Computing (CLOUD), 2010 IEEE 3rd international conference on, pp. 370–377 (2010)
11. Cailliau, A., van Lamsweerde, A.: Assessing requirements-related risks through probabilistic goals and obstacles. Requir. Eng. **18**(2), 129–146 (2013)
12. Chatzikonstantinou, G., Athanasopoulos, M., Kontogiannis, K.: Task specification and reasoning in dynamically altered contexts. In: Jarke et al. [31], pp. 625–639
13. Chatzikonstantinou, G., Kontogiannis, K.: Run-time requirements verification for reconfigurable systems. Inf. Softw. Technol. **75**, 105–121 (2016)
14. Chatzikonstantinou, G., Kontogiannis, K., Attarian, I.: A goal driven framework for software project data analytics. In:Salinesi, C., Norrie, M.C., Pastor, O. (eds.) Advanced information systems engineering—25th International conference, CAiSE 2013, Valencia, Spain, June 17–21, 2013. Proceedings, Lecture Notes in Computer Science, vol. 7908, pp. 546–561. Springer (2013)
15. Chieu, T., Mohindra, A., Karve, A., Segal, A.: Dynamic scaling of web applications in a virtualized cloud computing environment. In: e-Business Engineering, 2009. ICEBE '09. IEEE international conference on, pp. 281–286 (2009)
16. Chopra, A.K., Dalpiaz, F., Giorgini, P., Mylopoulos, J.: Reasoning about agents and protocols via goals and commitments. In: van der Hoek, W., Kaminka, G.A., Lespérance, Y., Luck, M., Sen, S. (eds.) AAMAS, pp. 457–464. IFAAMAS (2010)
17. Cingolani, P., Alcalá-Fdez, J.: jfuzzylogic: a robust and flexible fuzzy-logic inference system language implementation. In: FUZZ-IEEE 2012, IEEE international conference on fuzzy systems, Brisbane, Australia, June 10–15, 2012, Proceedings., pp. 1–8. IEEE (2012)
18. Dalpiaz, F., Borgida, A., Horkoff, J., Mylopoulos, J.: Runtime goal models: Keynote. In: Wieringa, R., Nurcan, S., Rolland, C., Cavarero, J. (eds.) IEEE 7th international conference on research challenges in information science, RCIS 2013, Paris, France, May 29–31, 2013, pp. 1–11. IEEE (2013)
19. Falcone, Y., Jaber, M., Nguyen, T., Bozga, M., Bensalem, S.: Runtime verification of component-based systems. In: Barthe, G., Pardo, A., Schneider, G. (eds.) Software engineering and formal methods—9th International conference, SEFM 2011, Montevideo, Uruguay, November 14–18, 2011. Proceedings, Lecture Notes in Computer Science, vol. 7041, pp. 204–220. Springer (2011)
20. Giorgini, P., Massacci, F., Mylopoulos, J., Zannone, N.: Modeling security requirements through ownership, permission and delegation. In: Requirements Engineering, 2005. Proceedings. 13th IEEE international conference on, pp. 167–176 (2005)
21. Giorgini, P., Mylopoulos, J., Nicchiarelli, E., Sebastiani, R.: Reasoning with goal models. In: Spaccapietra, S., March, S.T., Kambayashi, Y. (eds.) ER, Lecture Notes in Computer Science, vol. 2503, pp. 167–181. Springer (2002)
22. Giorgini, P., Mylopoulos, J., Sebastiani, R.: Goal-oriented requirements analysis and reasoning in the tropos methodology. Eng. Appl. AI **18**(2), 159–171 (2005)
23. Heaven, W., Letier, E.: Simulating and optimising design decisions in quantitative goal models. In: RE, pp. 79–88. IEEE (2011)
24. Homaifar, A., McCormick, E.: Simultaneous design of membership functions and rule sets for fuzzy controllers using genetic algorithms. IEEE T. Fuzzy Syst. **3**(2), 129–139 (1995)

25. Horkoff, J., Barone, D., Jiang, L., Yu, E.S.K., Amyot, D., Borgida, A., Mylopoulos, J.: Strategic business modeling: representation and reasoning. Softw. Syst. Model. **13**(3), 1015–1041 (2014)
26. Horkoff, J., Li, T., Li, F., Salnitri, M., Cardoso, E., Giorgini, P., Mylopoulos, J.: Using goal models downstream: a systematic roadmap and literature review. IJISMD **6**(2), 1–42 (2015)
27. Horkoff, J., Li, T., Li, F., Salnitri, M., Cardoso, E., Giorgini, P., Mylopoulos, J., Pimentel, J.: Taking goal models downstream: a systematic roadmap. In: Bajec, M., Collard, M., Deneckère, R. (eds.) IEEE 8th international conference on research challenges in information science, RCIS 2014, Marrakech, Morocco, May 28–30, 2014, pp. 1–12. IEEE (2014)
28. Horkoff, J., Yu, E.S.K.: Analyzing goal models: different approaches and how to choose among them. In: Chu, W.C., Wong, W.E., Palakal, M.J., Hung, C. (eds.) Proceedings of the 2011 ACM symposium on applied computing (SAC), TaiChung, Taiwan, March 21–24, 2011, pp. 675–682. ACM (2011)
29. Horkoff, J., Yu, E.S.K.: Comparison and evaluation of goal-oriented satisfaction analysis techniques. Requir. Eng. **18**(3), 199–222 (2013)
30. Ingolfo, S., Siena, A., Mylopoulos, J.: Establishing regulatory compliance for software requirements. In: Jeusfeld, M.A., Delcambre, L.M.L., Ling, T.W. (eds.) Conceptual modeling—ER 2011, 30th International conference, ER 2011, Brussels, Belgium, October 31–November 3, 2011. Proceedings, Lecture Notes in Computer Science, vol. 6998, pp. 47–61. Springer (2011)
31. Jarke, M., Mylopoulos, J., Quix, C., Rolland, C., Manolopoulos, Y., Mouratidis, H., Horkoff, J. (eds.): Advanced Information Systems Engineering—26th International conference, CAiSE 2014, Thessaloniki, Greece, June 169–20, 2014. Proceedings, Lecture Notes in Computer Science, vol. 8484. Springer (2014)
32. Kalamatianos, T., Kontogiannis, K.: Schema independent reduction of streaming log data. In: Jarke et al. [31], pp. 394–408
33. van Lamsweerde, A.: Goal-oriented requirements enginering: a roundtrip from research to practice. In: 12th IEEE international conference on requirements engineering (RE 2004), 6–10 September 2004, Kyoto, Japan, pp. 4–7. IEEE Computer Society (2004)
34. van Lamsweerde, A.: Reasoning about alternative requirements options. In: Borgida, A., Chaudhri, V.K., Giorgini, P., Yu, E.S.K. (eds.) Conceptual modeling: foundations and applications - Essays in Honor of John Mylopoulos, Lecture Notes in Computer Science, vol. 5600, pp. 380–397. Springer (2009)
35. Liaskos, S., Hamidi, S., Jalman, R.: Qualitative vs. quantitative contribution labels in goal models: Setting an experimental agenda. In: Castro, J., Horkoff, J., Maiden, N.A.M., Yu, E.S.K. (eds.) Proceedings of the 6th international *i** workshop 2013, Valencia, Spain, June 17–18, 2013, CEUR Workshop Proceedings, vol. 978, pp. 37–42. CEUR-WS.org (2013)
36. Liaskos, S., Jalman, R., Aranda, J.: On eliciting contribution measures in goal models. In: Heimdahl, M.P.E., Sawyer, P. (eds.) 2012 20th IEEE international requirements engineering conference (RE), Chicago, IL, USA, September 24–28, 2012, pp. 221–230. IEEE Computer Society (2012)
37. Liaskos, S., Khan, S.M., Litoiu, M., Jungblut, M.D., Rogozhkin, V., Mylopoulos, J.: Behavioral adaptation of information systems through goal models. Inf. Syst. **37**(8), 767–783 (2012)
38. Liaskos, S., Mylopoulos, J.: On temporally annotating goal models. In: iStar, pp. 62–66 (2010)
39. Luzeaux, D., Ruault, J.R.: Systems of Systems, 1st edn. John Willey & Sons, Hobiken, NJ, USA (2010)
40. Mylopoulos, J., Chung, L., Yu, E.S.K.: From object-oriented to goal-oriented requirements analysis. Commun. ACM **42**(1), 31–37 (1999)
41. Nekvi, M.R.I., Madhavji, N.H.: Impediments to regulatory compliance of requirements in contractual systems engineering projects: A case study. ACM Trans. Manag. Inf. Syst. **5**(3), 15:1–15:35 (2015)
42. Pan, H., McMichael, D.: Fuzzy causal probabilistic networks—a new ideal and practical inference engine (1998)
43. Passino, K.M., Yurkovich, S.: Fuzzy Control, 1st edn. Addison-Wesley Longman Publishing Co., Inc, Boston, MA, USA (1997)
44. Richardson, M., Domingos, P.: Markov logic networks. Mach. Learn. **62**(1–2), 107–136 (2006)
45. Shao, J., Wei, H., Wang, Q., Mei, H.: A runtime model based monitoring approach for cloud. In: Cloud Computing (CLOUD), 2010 IEEE 3rd international conference on, pp. 313–320 (2010)
46. Sharifloo, A.M., Spoletini, P.: LOVER: light-weight formal verification of adaptive systems at run time. In: Pasareanu, C.S., Salaün, G. (eds.) Formal Aspects of Component Software, 9th International symposium, FACS 2012, Mountain View, CA, USA, September 12–14, 2012. Revised selected papers, Lecture Notes in Computer Science, vol. 7684, pp. 170–187. Springer (2012)
47. Tarjan, R.E.: Depth-first search and linear graph algorithms. SIAM J. Comput. **1**(2), 146–160 (1972)
48. Wnuk, K., Borg, M., Assar, S.: Towards scalable information modeling of requirements architectures. In: Castano, S.,Vassiliadis, P., Lakshmanan, L.V.S., Lee, M. (eds.) Advances in conceptual modeling—ER 2012 workshops CMS, ECDM-NoCoDA, MoDIC, MORE-BI, RIGiM, SeCoGIS, WISM, Florence, Italy, October 15–18, 2012. Proceedings, Lecture Notes in Computer Science, vol. 7518, pp. 141–150. Springer (2012)
49. Wnuk, K., Regnell, B., Schrewelius, C.: Architecting and coordinating thousands of requirements—An industrial case study. In: Glinz, M., Heymans, P. (eds.) Requirements engineering: foundation for software quality, 15th International working conference, REFSQ 2009, Amsterdam, The Netherlands, June 8–9, 2009, Proceedings, Lecture Notes in Computer Science, vol. 5512, pp. 118–123. Springer (2009)
50. Zadeh, L.: Discussion: probability theory and fuzzy logic are complementary rather than competitive. Technometrics **37**(3), 271–276 (1995)

George Chatzikonstantinou holds a Diploma and a Ph.D. from the School of Electrical and Computer Engineering of the National Technical University of Athens, Greece. For his Ph.D. program, George has received a three-year scholarship from EU and the Greek government. He has worked for several years as a software engineer in the design, development, and maintenance of enterprise systems. His research interests include software engineering, requirements engineering and model-driven software development.

Kostas Kontogiannis is a Professor of Computer Science at Western University where he holds a Research Chair in Software Engineering for Cyber-Physical Systems. He is a Faculty Fellow of the IBM Center for Advanced Studies, and a registered Professional Engineer in Ontario in the area of Software Engineering. He has received a B.Sc. degree in Mathematics from the University of Patras, Greece, a M.Sc. degree in Computer Science from Katholieke Universiteit Leuven, Belgium, and a Ph.D. degree in Computer Science from McGill University, Canada. Kostas is working in the areas of software analysis, software architecture, software evolution, and services computing.