Knowl Inf Syst (2019) 58:83–112
https://doi.org/10.1007/s10115-018-1182-1

REGULAR PAPER

A fuzzy ontology-based approach for tool-supported decision making in architectural design

Tommaso Di Noia[1] · **Marina Mongiello**[1] · **Francesco Nocera**[1] · **Umberto Straccia**[2]

Received: 23 February 2017 / Revised: 17 November 2017 / Accepted: 14 March 2018 /
Published online: 27 March 2018

Abstract In software development, *non-functional requirements* (NFRs) play a crucial role in decision-making procedures for architectural solutions. A strong relation exists between NFRs and design patterns, a powerful method to support the architectural design of software systems, but due to their complexity and abstraction, NFRs are rarely taken into account in software design. In fact, the knowledge on NFRs is usually owned by designers and not formalized in a structured way. We propose to structure the knowledge associated with NFRs via a *Fuzzy Ontology*, which we show is able to model their mutual relations and interactions. The declarative approach makes possible to represent and maintain the above-mentioned knowledge by keeping the flexibility and fuzziness of modeling thanks to the use of fuzzy concepts such as *high*, *low*, *fair*. We present a decision support system based on *(i)* a fuzzy OWL 2 ontology that encodes 109 design patterns, 28 pattern families and 37 NFRs and their mutual relations, *(ii)* a novel reasoning service to retrieve a ranked list of pattern sets able to satisfy the non-functional requirements within a system specification.

Keywords Fuzzy Description Logics · Ontology · Non-Functional Requirements · Architectural design

✉ Francesco Nocera
francesco.nocera@poliba.it

Tommaso Di Noia
tommaso.dinoia@poliba.it

Marina Mongiello
marina.mongiello@poliba.it

Umberto Straccia
umberto.straccia@isti.cnr.it

1 Dipartimento di Ingegneria Elettrica e dell'Informazione (DEI), Politecnico di Bari, Via Orabona, 4, 70125 Bari, Italy

2 ISTI-CNR, G. Moruzzi, 1, 56127 Pisa, Italy

1 Introduction and motivation

In software development, the main goals to accomplish are customer and quality requirements satisfaction, correct execution of the software systems, and cost-effective adaptation to future changes [1]. In order to reach these challenging goals, tools and techniques may help to manage and retrieve the knowledge necessary for decision-making processes. For this reason, recent research trends are focused to strengthen the reasoning and decision-making process to reach these goals [1].

The development of a software system is in fact determined partly by its functionality, i.e., what the system does and partly by requirements about quality attributes or about development [17]. Such requirements are known as non-functional requirements (NFRs). During architectural design, the selection of NFRs is a relevant task, since can be used as selection criteria for choosing the proper design solution among several ones. On the other hand, exploiting design decisions during development emerges as a software architecture [5,22,31,63]. The software architecture is the result of the work of an architect or of a designers team [5]. The architectural design of a software system is made up of a set of selected quality attributes; in this sense, the architecture can be modeled by taking into account the needed quality attributes. To reach this goal, an architect can use primitive design techniques. These primitives are known as tactics. Generally a tactic can be considered as a modeling solution and is related to satisfying a given quality attribute. An architectural strategy is made up of several tactics. The architect takes design decision based on a strategy and hence selects a set of tactics, and hence a set of patterns.

The design process requires a choice of the best combination of tactics to achieve the systems desired goals [5]. For this reason, during architectural design phase a challenging task is the selection of patterns able to satisfy desired requirements [27,53,58]. A main difficulty derives from the relationship between patterns that can be complementary, can be composed, sometimes cooperate to solve a larger problem or are exclusive in a modeling task.[1] Existing classifications or quality models have been defined to categorize quality attributes and requirements [57], anyway a systematic classification of NFRs toward architectural design and a description of their use during system modeling are missing [2,37]. Moreover, only little work has been done in using a knowledge-based approach to support such activities [50].

The main objective of this research is the *realization of a semi-automated tool aimed at supporting decision makers to derive knowledge able to solve architectural problems.*

More precisely we aim to develop a decision support system for supporting designers and software architect—having greater or lesser experience—in the architectural design process.

To achieve this goal, we will define a theoretical framework to model and reason on the knowledge about requirements and reusable schema for design in order to obtain a model of the system satisfying given requirements, quality attributes optimized with reusable and composable design schema.

Toward this goal, the following research questions will be addressed:

Q1. What are the main categories of requirements a software may be asked to fulfill? Q2. What are the main reusable solving schema for given classes (or families) of design problems? Q3. How do categories of requirements, reusable schema and classes of problems are related? Q4. How to facilitate the decision-making process for software design modeling by using relations among these categories of elements? Q5. How to use this knowledge to support modeling during software design?

[1] E.g., [13] addresses this problem by proposing a specific pattern language.

To achieve the main objective of this research and answer the research questions, the following research process has been taken: 1. Propose a theoretical framework for modeling knowledge about NFRs, pattern and define a reasoning algorithm to manipulate the modeled knowledge; 2. build a prototype system to implement the theoretical framework; 3. validate the prototype with use cases; 4. compare solutions provided by the decision support system with human proposed solutions to software design problems.

The framework we propose to represent and reason about NFRs is based on *Fuzzy Description Logics* (FDLs) (we refer the reader to [65] for a description about FDLs), which are the theoretical counterpart of *fuzzy OWL 2* [9], the main formalism to specify fuzzy ontologies. We recall that FDLs are logical languages specifically tailored to model structured information about vague concepts that naturally occur in NFRs. For instance, in our context, FDLs allow one to model that "portability and adaptability are directly proportionate", "stability and adaptability are inversely proportionate" (ontological knowledge) or that "the Adapter pattern has high portability" (factual knowledge). New knowledge about a pattern is obtained by combining existing knowledge, for example, ontological or factual through a reasoning task. For instance, it can be inferred that "the Adapter pattern has high adaptability and low stability". Another type of expression allowed in our framework is "high adaptability implies a medium maintainability". Let us note that in the previous statements, we can use fuzzy sets [71] to characterize concepts like *high*, *medium* and *low*. We would like to stress the point that a formal encoding of the knowledge about patterns, NFRs and family of patterns is particularly useful to automatize the task of selecting a set of patterns that encounters user's needs. Specifically, we will show how the following task can be addressed: *given a desired set of NFRs, perform the task of retrieving the smallest subset of patterns that best match them*. Eventually, in order to make the approach feasible, we collected the knowledge about 109 design patterns, 28 pattern families and 37 NFRs together with their mutual relations and represented them as a fuzzy OWL 2 ontology and show its application in a use-case scenario.[2]

In summary, this paper provides the following contributions:

- we provide a method to model and reason with mutual relations among NFRs in architectural software patterns by relying on fuzzy ontologies;
- the definition of a novel formal reasoning task being able to retrieve a set of patterns that maximally match a user query expressed in terms of desired NFRs;
- the construction of a fuzzy OWL 2 ontology;
- a use-case scenario application to validate the method;
- an implementation of the theoretical framework in a decision support system.

No other method similar to the proposed one was found. This paper considerably extends our previous work [23] where the initial idea and examples were sketched and presented. More specifically, w.r.t. [23], we extend two new sections on Fuzzy Description Logics and Architectural Patterns and NFRs; define the fuzzy sets to be used in the definition of NFRs mutual relations; formally define and implement the *Covering Answer Set* reasoning task to be used for patterns retrieval; extend the use-case scenario by showing a step-by-step retrieval task and add a new use-case; present the implementation of a novel fuzzy OWL 2 ontology, which is publicly available online.

In the following, we proceed as follows. We start by recalling basic definitions about ontologies and FDLs. Then, we proceed with main properties of design patterns and quality models used to model NFRs. In Sect. 4, we state the addressed problem and the proposed

[2] The ontology is available online at http://sisinflab.poliba.it/semanticweb/ontologies/architecturalpatterns/.

approach. In Sect. 5, we describe a case study. In Sect. 6, the implemented decision support system is described. In Sect. 7, we validate the approach and in Sect. 8 we address related work about existing (ontological) approaches for knowledge representation in software engineering, design patterns and NFRs modeling. Conclusions close the paper.

2 Fuzzy Description Logics

Ontologies play a key role in the success of the *Semantic Web* [7] since they are the recognized knowledge representation formalism for specifying domain knowledge and for sharing and reusing this knowledge. *Description Logics* (DLs) [3] are at the core of the Semantic Web ontology description language OWL 2 [19]. In fact, OWL 2 is based on a specific DL named $\mathcal{SROIQ}(\mathbf{D})$ [44]. In recent times, it has been noted that classical ontologies and its languages are not appropriate to deal with vagueness and imprecise knowledge, which is fundamental to several real-world domains [62]. To handle this problem, the use of fuzzy logics with ontology offers a solution. Fuzzy ontologies and its description logics for the semantic web can handle probabilistic or possibilistic uncertainties and vagueness. Research on fuzzy ontologies began in the early 2000s with the focus on simplistic models used to improve an Information Retrieval System [15]. *Fuzzy DLs* (FDLs) are an extension of DLs to deal with *fuzzy* concepts (see [8,65] for an overview). In these logics, satisfactions of axioms are based on a degree of truth that generally is a value in [0, 1].

2.1 Recap of Fuzzy Description Logics basics

The fuzzy DL we are considering here is called $\mathcal{ALCB}(\mathbf{D})$, whose expressiveness is enough to illustrate our approach. Our framework extends to more expressive fuzzy DLs easily. We recall that $\mathcal{ALCB}(\mathbf{D})$ is a $\mathcal{ALC}$, in which the letter $\mathcal{B}$ represents the *individual value restrictions* that are restricted kind of nominals, and the letter $\mathbf{D}$ indicates the *fuzzy concrete domains* [64].

For making the paper self-contained, we recap here some basic definitions about fuzzy sets and the fuzzy DL $\mathcal{ALCB}(\mathbf{D})$.

2.1.1 Fuzzy sets

Let X be a countable crisp set, i.e., a set in which an element is a either member of the set or not. A *fuzzy set* [71] A over X is characterized by a function $A\colon X \to [0, 1]$, called *fuzzy membership* function. Typical operations on fuzzy sets are defined as: Intersection $(A \cap B)(x) = \min(A(x), B(x))$, Union $(A \cup B)(x) = \max(A(x), B(x))$ and Complement $\bar{A}(x) = 1 - A(x)$. Figure 1 illustrates some frequently used membership functions for specifying fuzzy sets.

2.1.2 The fuzzy DL $\mathcal{ALCB}(\mathbf{D})$

At first, let us recap the notion of *fuzzy concrete domain* [64] that is a tuple $\mathbf{D} = \langle \Delta^{\mathbf{D}}, \cdot^{\mathbf{D}} \rangle$ where $\Delta^{\mathbf{D}}$ is the datatype domain and $\cdot^{\mathbf{D}}$ is a mapping that assigns an element of $\Delta^{\mathbf{D}}$ to each data value, and a 1-ary fuzzy relation over $\Delta^{\mathbf{D}}$ to every 1-ary datatype predicate $\mathbf{d}$. Hence the mapping $\cdot^{\mathbf{D}}$ relates each datatype predicate to a function from the domain $\Delta^{\mathbf{D}}$ to [0, 1]. The datatype predicates $\mathbf{d}$ we are considering here are the membership functions shown in Fig. 1:

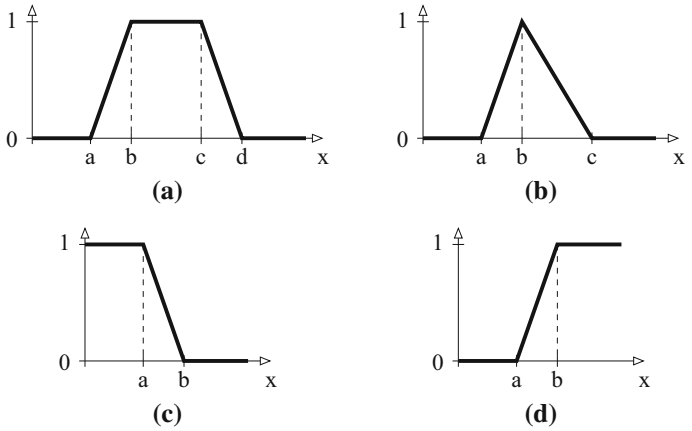

Fig. 1 **a** Trapezoidal function $trz(a, b, c, d)$, **b** triangular function $tri(a, b, c)$, **c** left-shoulder function $ls(a, b)$, and **d** right-shoulder function $rs(a, b)$

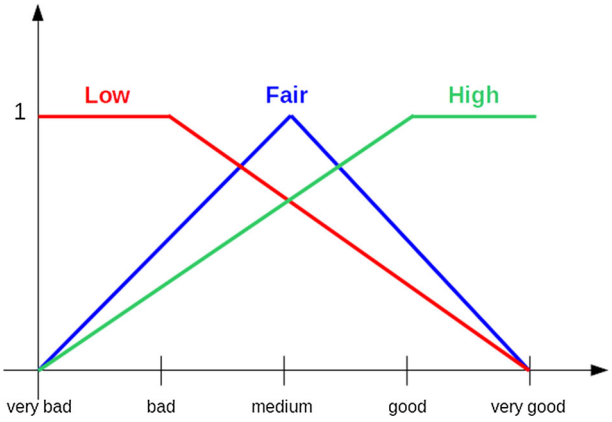

Fig. 2 The fuzzy sets we use to deal with non-functional requirements

trapezoidal $trz(a, b, c, d)$, triangular $tri(a, b, c)$, left-shoulder $ls(a, b)$ and right-shoulder $rs(a, b)$:

$$\mathbf{d} \rightarrow ls(x, y) \mid rs(x, y) \mid tri(x, y, z) \mid trz(x, y, z, t)$$
$$\mid \geq_v \mid \leq_v \mid =_v \mid \in_V ,$$

and $v \in \Delta^{\mathbf{D}}$ and $V \subseteq \Delta^{\mathbf{D}}$. To what concerns our framework, we will consider the following fuzzy concrete domain as illustrated in Fig. 2. The values/ratings in

$$\{\texttt{verybad}, \texttt{bad}, \texttt{medium}, \texttt{good}, \texttt{verygood}\}$$

are macros for some predefined numerical values.

Now, let us consider the following notations: the set of atomic concepts or *concept names* is denoted as **A**, the set of *role names* denoted as **R**, the set of *individual names* denoted as **I**. We assume that a role can be a *datatype property* or an *object property*. Using DL constructors, *concepts* are built from concept names A, object properties R and datatype properties S. The syntactic rule used for concept construction is the following:

$$C \rightarrow \top \mid \bot \mid A \mid C_1 \sqcap C_2 \mid C_1 \sqcup C_2 \mid \neg C \mid C_1 \rightarrow C_2$$
$$\mid \exists R.C \mid \forall R.C \mid \exists S.\mathbf{d} \mid \forall S.\mathbf{d} \mid \exists R.\{a\} \ .$$

Concepts of the form $\{a\}$, with $a \in \mathbf{I}$, are called *nominals*. In $\mathcal{ALCB}(\mathbf{D})$, they can only appear in concepts of the form $\exists R.\{a\}$.

A *Fuzzy Knowledge Base* (or *fuzzy Ontology*) $\mathcal{K} = \langle \mathcal{A}, \mathcal{T} \rangle$ contains a fuzzy ABox $\mathcal{A}$ with axioms about individuals and a fuzzy TBox $\mathcal{T}$ with axioms about concepts.

A *fuzzy ABox* contains a finite set of *fuzzy assertions* of the following types: (i) *Concept assertions* of the form $\langle a{:}C, \alpha \rangle$, with $\alpha \in (0, 1]$ and stating that individual a is an instance of concept C with degree greater than or equal to α; and (ii) *Role assertions* of the form $\langle (a, b){:}R, \alpha \rangle$, $\alpha \in (0, 1]$, meaning that the pair of individuals (a, b) is an instance of role R with degree greater than or equal to α. A *fuzzy TBox* consists of a finite set of *fuzzy General Concept Inclusions (fuzzy GCIs)*, which are expressions of the form $\langle C_1 \sqsubseteq C_2, \alpha \rangle, \alpha \in (0, 1]$, meaning that the degree of concept C_1 being subsumed by C_2 is greater than or equal to α.

For the rest of the paper, we also make the following assumptions: (i) in fuzzy assertions and GCIs, we may omit the degree α and in that case the value 1 is assumed; (ii) we may write $\exists R$ in place of $\exists R.\top$; and (iii) we use the GCI $C \equiv D$ as a macro of both expressions $C \sqsubseteq D$ and $D \sqsubseteq C$, dictating that C and D are "equivalent" concept expressions.

Concerning the semantics, let us fix a fuzzy logic. Unlike classical DLs, in fuzzy DLs, an interpretation $\mathcal{I}$ maps a concept C into a function $C^{\mathcal{I}} : \Delta^{\mathcal{I}} \rightarrow [0, 1]$ and, thus, an individual belongs to the extension of C to some degree in $[0, 1]$, i.e., $C^{\mathcal{I}}$ is a fuzzy set.

Specifically, a *fuzzy interpretation* is a pair $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ consisting of a nonempty (crisp) set $\Delta^{\mathcal{I}}$ (the *domain*) and of a *fuzzy interpretation function* $\cdot^{\mathcal{I}}$ that assigns: *(i)* to each atomic concept A a function $A^{\mathcal{I}} : \Delta^{\mathcal{I}} \rightarrow [0, 1]$; *(ii)* to each object property R a function $R^{\mathcal{I}} : \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}} \rightarrow [0, 1]$; *(iii)* to each datatype property S a function $S^{\mathcal{I}} : \Delta^{\mathcal{I}} \times \Delta^{\mathbf{D}} \rightarrow [0, 1]$; *(iv)* to each individual a an element $a^{\mathcal{I}} \in \Delta^{\mathcal{I}}$ such that $a^{\mathcal{I}} \neq b^{\mathcal{I}}$ if $a \neq b$ (unique name assumption); and *(v)* to each data value v an element $v^{\mathcal{I}} \in \Delta^{\mathbf{D}}$. Now, a fuzzy interpretation function is extended to concepts as specified below (where $x \in \Delta^{\mathcal{I}}$):

$$\bot^{\mathcal{I}}(x) = 0 \ , \ \top^{\mathcal{I}}(x) = 1 \ , (C \sqcap D)^{\mathcal{I}}(x) = C^{\mathcal{I}}(x) \otimes D^{\mathcal{I}}(x) \ , (C \sqcup D)^{\mathcal{I}}(x) = C^{\mathcal{I}}(x) \oplus D^{\mathcal{I}}(x) \ ,$$
$$(\neg C)^{\mathcal{I}}(x) = \ominus C^{\mathcal{I}}(x) \ , (C \rightarrow D)^{\mathcal{I}}(x) = C^{\mathcal{I}}(x) \Rightarrow D^{\mathcal{I}}(x) \ , (\forall R.C)^{\mathcal{I}}(x) = \inf_{y \in \Delta^{\mathcal{I}}} \{R^{\mathcal{I}}(x, y) \Rightarrow C^{\mathcal{I}}(y)\} \ ,$$
$$(\exists R.C)^{\mathcal{I}}(x) = \sup_{y \in \Delta^{\mathcal{I}}} \{R^{\mathcal{I}}(x, y) \otimes C^{\mathcal{I}}(y)\} \ , (\exists R.\{a\})^{\mathcal{I}}(x) = R^{\mathcal{I}}(x, a^{\mathcal{I}}) \ ,$$
$$(\forall S.\mathbf{d})^{\mathcal{I}}(x) = \inf_{y \in \Delta^{\mathbf{D}}} \{S^{\mathcal{I}}(x, y) \Rightarrow \mathbf{d}^{\mathbf{D}}(y)\} \ , (\exists S.\mathbf{d})^{\mathcal{I}}(x) = \sup_{y \in \Delta^{\mathbf{D}}} \{S^{\mathcal{I}}(x, y) \otimes \mathbf{d}^{\mathbf{D}}(y)\} \ .$$

The *satisfiability of axioms* is then defined by the following conditions: *(i)* $\mathcal{I}$ satisfies an axiom $\langle a{:}C, \alpha \rangle$ if $C^{\mathcal{I}}(a^{\mathcal{I}}) \geq \alpha$; *(ii)* $\mathcal{I}$ satisfies an axiom $\langle (a, b){:}R, \alpha \rangle$ if $R^{\mathcal{I}}(a^{\mathcal{I}}, b^{\mathcal{I}}) \geq \alpha$; *(iii)* $\mathcal{I}$ satisfies an axiom $\langle C \sqsubseteq D, \alpha \rangle$ if $(C \sqsubseteq D)^{\mathcal{I}} \geq \alpha$ where[3] $(C \sqsubseteq D)^{\mathcal{I}} = \inf_{x \in \Delta^{\mathcal{I}}} \{C^{\mathcal{I}}(x) \Rightarrow D^{\mathcal{I}}(x)\}$. $\mathcal{I}$ is a *model* of $\mathcal{K} = \langle \mathcal{A}, \mathcal{T} \rangle$ iff $\mathcal{I}$ satisfies each axiom in $\mathcal{K}$.

The most common reasoning tasks on fuzzy DLs, which are usually inter-definable [65], are the following ones ($\mathcal{K}$ is a fuzzy KB): (i) *Consistency* (or KB satisfiability). Check if $\mathcal{K}$ has a *model*; (ii) *Fuzzy concept satisfiability*. A fuzzy concept C is α-satisfiable w.r.t. $\mathcal{K}$ iff there is a model $\mathcal{I}$ of $\mathcal{K}$ such that $C(x)^{\mathcal{I}} \geq \alpha$ for some element $x \in \Delta^{\mathcal{I}}$; (iii) *Entailment*. A fuzzy axiom τ is a *logical consequence* of $\mathcal{K}$ (or $\mathcal{K}$ entails τ), denoted $\mathcal{K} \models \tau$, iff every model of $\mathcal{K}$ is a model of τ. (iv) *Fuzzy concept subsumption*. C_2 α-subsumes C_1 w.r.t. $\mathcal{K}$ iff every model $\mathcal{I}$ of $\mathcal{K}$ satisfies $\forall x \in \Delta^{\mathcal{I}}, C_1^{\mathcal{I}}(x) \Rightarrow C_2^{\mathcal{I}}(x) \geq \alpha$; (v) *Best Entailment Degree* (BED). The BED of an axiom $\phi \in \{a{:}C, (a, b){:}R, C \sqsubseteq D\}$ w.r.t. $\mathcal{K}$ is defined as $\mathtt{bed}(\mathcal{K}, \phi) = \sup \{\alpha \mid \mathcal{K} \models \langle \phi, \alpha \rangle\}$; (vi) *Best Satisfiability Degree* (BSD). The BSD of a concept C w.r.t. $\mathcal{K}$ is defined as $\mathtt{bsd}(\mathcal{K}, C) = \sup_{\mathcal{I} \models \mathcal{K}} \sup_{x \in \Delta^{\mathcal{I}}} C^{\mathcal{I}}(x)$; (vii) *answer set*.

[3] However, note that under standard logic $\sqsubseteq$ is interpreted as $\Rightarrow_z$ and not as $\Rightarrow_{kd}$.

The *answer set* of a concept C w.r.t. $\mathcal{K}$ is the set $\mathtt{ans}(\mathcal{K}, C) = \{\langle a, \alpha\rangle \mid \alpha = \mathtt{bed}(\mathcal{K}, a{:}C)\}$. Each element in $\mathtt{ans}(\mathcal{K}, C)$ is called an *answer* to C w.r.t. $\mathcal{K}$; (viii) *top-k Answer Set*. The *top-k answer set* of C w.r.t. $\mathcal{K}$, denoted $\mathtt{ans}_k(\mathcal{K}, C)$, is the set of top-$k$ ranked answers of to C w.r.t. $\mathcal{K}$. Formally, $\mathtt{ans}_k(\mathcal{K}, C) \subseteq \mathtt{ans}(\mathcal{K}, C)$ is maximal under set inclusion, $|\mathtt{ans}_k(\mathcal{K}, C)| \leq k$, there is no $\langle a, 0\rangle \in \mathtt{ans}_k(\mathcal{K}, C)$, and there is no $\langle b, \beta\rangle \in \mathtt{ans}(\mathcal{K}, C) \setminus \mathtt{ans}_k(\mathcal{K}, C)$ with $\beta > \alpha$ for some $\langle a, \alpha\rangle \in \mathtt{ans}_k(\mathcal{K}, C)$. Each element in $\mathtt{ans}_k(\mathcal{K}, C)$ is called a *top-k answer* to C w.r.t. $\mathcal{K}$.

Eventually, fuzzy DL reasoners enable to manage fuzzy ontologies in practice (see [10] for an overview). *fuzzyDL* [10] is an expressive fuzzy ontology reasoner (the supported language is more expressive that the one presented here) that supports the three main semantics of fuzzy logics: Zadeh, Łukasiewicz, and classical DLs, for ensuring compatibility with crisp ontologies.

3 NFRs, design and architectural patterns

Non-functional requirements. A software system design is obtained as the results of both functional requirements implementation, i.e., what the system does—and requirements about development specifications or quality attributes. Such requirements concern development features, operational costs, but also quality attributes [17]. Non-functional requirements (NFRs) are crucial in software design since help designers in selecting the supposed best design solution among several alternatives. For this reason, best practices in taking into account these requirements are the basis for ensuring successful development. Also taking properly into account requirements can prevent errors which may have negative impact on management and costs of the whole software project [12,17,21]. Non-functional requirements lend to a qualitative assessment more than an objective definition, and for this reason identifying a particular NFR is a challenging task. Besides, identifying NFRs is a architectural task, so the availability of reusable solutions that ensure satisfaction of a given NFR or of a set of NFR gives a valid support to performing architectural modeling. Description of knowledge about NFRs is proposed in NFR catalogues [52],[20] those enabling reuse and catalogation and a useful way of addressing the need for help on NFR elicitation. In the literature, a plethora of definitions can be found of non-functional requirements (NFRs). A series of such definitions is summarized in the work by Glinz [33]:

(a) "Describe the non-behavioral aspects of a system, capturing the properties and constraints under which a system must operate".
(b) "The required overall attributes of the system, including portability, reliability, efficiency, human engineering, testability, understandability, and modifiability".
(c) "Requirements which are not specifically concerned with the functionality of a system. They place restrictions on the product being developed and the development process, and they specify external constraints that the product must meet".
(d) "…global requirements on its development or operational cost, performance, reliability, maintainability, portability, robustness, and the like. …There is not a formal definition or a complete list of nonfunctional requirements".
(e) "The behavioral properties that the specified functions must have, such as performance, usability".
(f) "A property, or quality, that the product must have, such as an appearance, or a speed or accuracy property".

(g) "A description of a property or characteristic that a software system must exhibit or a constraint that it must respect, other than an observable system behavior".

Design and architectural pattern Reusable solutions of design models are available mainly realized using patterns. These approaches are important vehicles for constructing high-quality software architectures since provide already tested solutions [14,30].

Design patterns proposed during the last decades are reusable solutions to modeling recurrent problems. They are mainly based on the expert's experience that use solution proposed for similar problems [14,30].

Therefore, the use of patterns or tactics [6,34,37] for architectural modeling constitutes an effective solution for addressing design decisions [39]. They also support the construction and documentation of software architectures. In summary, patterns provide a set of predefined design schemes for software systems organization and provide an abstract formalization of the design solution [14,43,67].

According to [13], design patterns can be classified into families that identify a problem area addressing a specific topic. The pattern language proposed in [13] includes 114 patterns, which are grouped into thirteen problem areas addressing a specific technical topic.

The main purpose of families and problem areas is to make the language and its patterns more tangible and comprehensible. In Table 1, we briefly describe and summarize some

Table 1 Some of the most relevant families of patterns

Family	Description
Distribution Infrastructure	Patterns concerning middleware's issues, i.e., distribution infrastructure software that help to simplify applications in distributed systems
Interface Partitioning	Patterns that specify usable and meaningful component interfaces. Interfaces inform clients about the component's responsibilities and usage protocols
Object Interaction	Patterns for make object interact in standalone programs
Application Control	These patterns separate interface from application's main functionality. Transforming user input for an application into concrete service requests, executing these requests, and transforming results back into an output that is meaningful for users
Event Demultiplexing and Dispatching	It concerns patterns that describe different approaches for managing events in distributed and networked systems
Synchronization	Patterns addressing the problem of synchronous access to shared components, objects and resources
Concurrency	Patterns that deal with concurrency control
Adaptation and Extension	Patterns for making applications and components adaptive to specific environments
Modal Behavior	Patterns that support the implementation of state-machines
Resource Management	These patterns for managing resource's lifecycle and availability to clients
Database Access	Patterns belonging to this family manage the access to relational databases and the mapping with other data models
Cloud	Patterns belonging to this family describe different types of clouds, the services they provide, how to build applications with them and the interconnections between patterns

relevant families of patterns as described in [13] to which we also add the family of cloud patterns [28].

4 Problem statement and approach

A typical software design problem can be stated as follows:

> Given a set of requirements define the software design that best matches them.

Software design and software architecture design model, requirements and specifications are composed of *functional requirements* (FRs) and *non-functional requirements* (NRFs). Generally, modeling and definition of FRs is a strong and wide phase of software development process. On the other hand, enterprise architects could take advantage also from formalization and validation of NFRs. In fact, NFRs compliance impacts on the long-term benefit of the system. So being able to predict and evaluate NFRs from the first stages of the development process would ensure a high-quality level of the software systems. Anyway this remains still a challenging task.

With respect to the general problem stated before, we can reformulate its statement by restricting our attention to the relation among patterns, NFRs and families. Specifically, with reference to the design problem at the beginning of this section, we address the following problem:

> Given a software design to model, a set of NFRs and the problem areas the software refers to, which are the components/patterns that best fit them?

The task poses some difficulties that we are going to highlight:

- some NFRs cannot be satisfied together and some others may be disjoint;
- patterns satisfying the required NFRs may not be contained in families;
- not all the patterns providing a given NFR or belonging to a family may be known by the designer.

4.1 Representing and reasoning about NFRs via fuzzy DLs

We propose the use of fuzzy DLs and fuzzy reasoning services (i) to define a formal model of the relations among problem areas, design patterns and satisfied NFRs; and (ii) to provide the designer a reasoning mechanism as a support for selecting a set of patterns that ensure the satisfaction of a set of desired requirements. More specifically:

- we use fuzzy DL statements to model the domain of architectural modeling at a high level and to model relations among non-functional requirements;
- we propose a fuzzy DL reasoning service to deduce new knowledge about mutual relation between NFRs and to answer the retrieval problem of finding sets of patterns that satisfy desired requirements.

In the following, detail about addressing the two above-mentioned steps is provided.

The (upper) ontology (TBox) is used to formalize the knowledge about the NFRs, about patterns and families that pattern belong to.

The upper ontology is composed by three main classes `SoftwareDesignPattern`, `Families` and `NonFunctionalRequirement`. Its formal definition can be encoded in (classical) DLs as:

$$\exists \texttt{isInFamily} \sqsubseteq \texttt{SoftwareDesignPattern} \quad (1)$$

$$\exists \texttt{nFR} \sqsubseteq \texttt{SoftwareDesignPattern} \quad (2)$$

$$\top \sqsubseteq \forall \texttt{isInFamily.Families} \quad (3)$$

$$\top \sqsubseteq \forall \texttt{nFR.NonFunctionalRequirement} \quad (4)$$

In the first two statements are defined: the *Domain* restrictions to check the first two statements, while *range* restrictions are defined in the last two statements. To better explain the meaning of restriction, let us consider that `isInFamily` is a role that the concept `SoftwareDesignPattern` to the concept `Families` by connecting their instances. Instead, the role `nFR` relates the class `SoftwareDesignPattern` to the class `NonFunctionalRequirement` through their instances. Of course, we may easily extend the ontology to deal also with other elements, such as FRs, which, however, we are not going to address here.

In our ontology, we make statements about the description of a pattern, related the pattern to the family it belongs to and make statements about the NFRs it satisfies. Such statements make up the ABox of the knowledge base we defined. To better explain the knowledge modeling in the ontology, let us consider the following statements using the classical DL syntax:

$$\texttt{proxy:SoftwareDesignPattern} \quad (5)$$

$$\texttt{resourceManagement:Families} \quad (6)$$

$$\texttt{reliability:NonFunctionalRequirement} \quad (7)$$

$$\texttt{loadBalancing:NonFunctionalRequirement} \quad (8)$$

$$\texttt{reusability:NonFunctionalRequirement} \quad (9)$$

$$(\texttt{proxy}, \texttt{resourceManagement}):\texttt{isInFamily} \quad (10)$$

In the example, the pattern `proxy` is defined as an instance of the class `SoftwareDesignPattern`; the family `resourceManagement` of the class `Families` and the NFRs `reliability`, `loadBalancing`, `reusability` as individuals of the class, `NonFunctionalRequirement`; definition are provided, respectively, in the statements 5–9. Moreover, statement 10 asserts that `proxy` belongs to the `resourceManagement` family.

Additionally, the pattern *proxy* has a level of NFRs *Load Balancing* and *Reliability*. To specify such properties, we define new properties, that are related to NFRs. The fuzzy sets used by our framework are the ones represented in Fig. 2. Such datatype properties are formally represented using **R** that stands for rating. The rating models a ordered set {`verybad`, `bad`, `medium`, `good`, `verygood`}. We refer to the following axioms:

$$\exists \texttt{nFR.}\{\texttt{reliability}\} \equiv \exists \texttt{reliabilityRate.} \in_{\mathbf{R}} \quad (11)$$

$$\exists \texttt{nFR.}\{\texttt{loadBalancing}\} \equiv \exists \texttt{loadBalancingRate.} \in_{\mathbf{R}} \quad (12)$$

$$\exists \texttt{nFR.}\{\texttt{reusability}\} \equiv \exists \texttt{reusabilityRate.} \in_{\mathbf{R}} \quad (13)$$

$$\exists \texttt{nFR.}\{\texttt{adaptability}\} \equiv \exists \texttt{adaptabilityRate.} \in_{\mathbf{R}} \quad (14)$$

$$\exists \texttt{nFR.}\{\texttt{maintainability}\} \equiv \exists \texttt{maintainabilityRate.} \in_{\mathbf{R}} \quad (15)$$

These axioms states that a pattern that has associated a NFR will have a degree. So we can state for example:

$$\texttt{proxy}{:}\exists \texttt{loadBalancingRate}.\ =_{\texttt{good}} \tag{16}$$

$$\texttt{proxy}{:}\exists \texttt{reliabilityRate}.\ =_{\texttt{good}} \tag{17}$$

Besides the modeling of the ABox relations, we use FDLs also to explicitly model relations among NFRs. Consider the NFRs *reliability*, *load balancing*, *reusability* as previously defined statement (7)–(9). An example of mutual relation between NFRs is the one between *load balancing* and *reliability*. Indeed, they are directly proportionate, i.e., if the *loadBalancing* increases (decreases) the same happens for the *reliability*. Such a relation can be written in fuzzy DLs as

$$\exists \texttt{loadBalancingRate}.\texttt{High} \sqsubseteq \exists \texttt{reliabilityRate}.\texttt{High} \tag{18}$$

$$\exists \texttt{loadBalancingRate}.\texttt{Fair} \sqsubseteq \exists \texttt{reliabilityRate}.\texttt{Fair} \tag{19}$$

$$\exists \texttt{loadBalancingRate}.\texttt{Low} \sqsubseteq \exists \texttt{reliabilityRate}.\texttt{Low} \tag{20}$$

We also know that a system cannot be *reliable* and *reusable* at the same time. That is, the two NFRs are inversely proportionate. Hence, if a pattern guarantees *reliability* it cannot guarantee also *reusability*. We may encode such disjoint relations with the following statement:

$$\exists \texttt{reusabilityRate}.\texttt{High} \sqsubseteq \exists \texttt{reliabilityRate}.\texttt{Low} \tag{21}$$

In the same manner, we may also represent statements like

$$\exists \texttt{adaptabilityRate}.\texttt{High} \sqsubseteq \exists \texttt{maintainabilityRate}.\texttt{Fair} \tag{22}$$

stating that "a system with a high adaptability has a fair maintainability", in fact if a system has an adaptable behavior to different requirements or to changes in the context, it is poorly maintainable.

Note that from statements (18)–(21), it can be inferred that the pattern *proxy* cannot guarantee a high degree of *reusability*, since it has a high degree of *load balancing*. Of course, many other kinds of implicit relations can be automatically inferred that support the search for better results during the design phase.

4.2 Proposed reasoning task

We conclude this section by proposing a novel reasoning task called *Covering Answer Set*, which will result to be fundamental for our modeling method to solve the defined problem statement.

Let $C_1, \ldots, C_n$ be concepts and let $@$ be an *aggregation operator* [69]. We recall the *aggregation operators* (AOs). They are mathematical functions that combine real values. Specifically, an AO has a dimension n and is a mapping $@ : [0, 1]^n \rightarrow [0, 1]$ such that[4] $@(\mathbf{0}) = 0, @(\mathbf{1}) = 1$. Besides, the aggregation operator $@$ is monotone in its arguments. Typical examples of aggregation operators are *maximum* ($@^{MAX}$), *weighted maximum* $@_W^{\text{wmax}}$, *minimum* ($@^{min}$), *weighted minimum* $@_W^{\text{wmin}}$, *median*, *arithmetic mean* ($@^{AM}$), *weighted sum* $@_W^{\text{WS}}$, strict weighted sum ($@_W^{\text{WSZ}}$).

Now, the *covering answer set* of $C_1, \ldots, C_n$ w.r.t. $@$ and $\mathcal{K}$,

$$\texttt{ans}(@, \mathcal{K}, C_1, \ldots, C_n) \ ,$$

is defined as follows:

[4] With $\mathbf{0}$ and $\mathbf{1}$ we identify a vector whose elements are all 0 or 1, respectively.

1. determine $\texttt{ans}(\mathcal{K}, C_1), \ldots, \texttt{ans}(\mathcal{K}, C_n)$;
2. consider the set of tuples

$$A^{@}_{C_1,\ldots,C_n,\mathcal{K}} = \{\langle\{a_1, \ldots, a_n\}, \beta\rangle \mid \langle a_1, \alpha_1\rangle \in \texttt{ans}(\mathcal{K}, C_1), \ldots, \langle a_n, \alpha_n\rangle \in \texttt{ans}(\mathcal{K}, C_n), \beta = @(\alpha_1, \ldots, \alpha_n)\} .$$

   That is, a tuple in $A^{@}_{C_1,\ldots,C_n,\mathcal{K}}$ is built by picking up an element from each answer set and then by aggregating the individual scores.
3. Eventually, $\texttt{ans}(@, \mathcal{K}, C_1, \ldots, C_n)$ is obtained from $A^{@}_{C_1,\ldots,C_n,\mathcal{K}}$ by removing from $A^{@}_{C_1,\ldots,C_n,\mathcal{K}}$ all non-maximal scores and non-minimal subsets, i.e.,

$$\texttt{ans}(@, \mathcal{K}, C_1, \ldots, C_n) = \{\langle S, \beta\rangle \in A^{@}_{C_1,\ldots,C_n,\mathcal{K}} \mid \beta > 0, \nexists\langle S', \beta'\rangle \in A^{@}_{C_1,\ldots,C_n,\mathcal{K}} \text{ s.t. } \beta' > \beta, S \subseteq S' \text{ and } \nexists\langle S', \beta'\rangle \in A^{@}_{C_1,\ldots,C_n,\mathcal{K}} \text{ s.t. } S' \subset S\} .$$

Each element in $\texttt{ans}(@, \mathcal{K}, C_1, \ldots, C_n)$ is a *covering answer* to $C_1, \ldots, C_n$ w.r.t. @ and $\mathcal{K}$. Eventually, the *top-k covering answer set* of $C_1, \ldots, C_n$ w.r.t. @ and $\mathcal{K}$, denoted $\texttt{ans}_k(@, \mathcal{K}, C_1, \ldots, C_n)$, is the set top-$k$ ranked covering answers to $C_1, \ldots, C_n$ w.r.t. @ and $\mathcal{K}$. We say that each element in $\texttt{ans}_k(@, \mathcal{K}, C_1, \ldots, C_n)$ is a *top-k covering answer* to $C_1, \ldots, C_n$ w.r.t. @ and $\mathcal{K}$.

5 Use-case scenario

In the following, we describe a Cloud-Social-Adaptable System use case to illustrate how to apply the proposed modeling.
We consider a social domain in which user's data are stored on cloud platforms and distributed on different clusters or data centers; data are managed by interacting distributed applications. A similar context would require a software architecture based on an extensible model consisting of loosely coupled components. Let us think of web applications for mobile devices, client applications for web-based systems and let us assume that there is a main component—a sort of manager—that performs coordination activities, and other two key components: a manager to gather and manage multimedia data and a location-based application that records all movements made by the user.

On the basis of variations in the user's information needs or changes in the external environment, the system is able to dynamically and extensively change applications to be loaded.

Suppose a user is traveling on a weekend or holiday, the idea is that the system automatically launches an app that organizes the archived multimedia material (photo, movies, etc.) relating to the travel destination by creating albums, photo collections with captions, stories, etc.

In addition to user's localization, other conditions dependent on the context set in the application may allow the system to dynamically load different applications. The dynamically loaded applications may compromise the system properties; therefore, runtime mechanisms to monitor and guarantee the preservation of the properties of interest are needed.

This solution provides flexibility in the architecture as well as access to a public service which is made possible by exploiting the resources available and preserving costs.

Furthermore, virtual machines are loosely coupled, so a possible failure in one of them does not impair the operation of the other guaranteeing an acceptable level of fault tolerance. The virtual machines are located on the middleware and are launched directly from it only if requested by the consumer.

In case of failure, the remaining virtual machines would not be affected and the content is made available by the presence of a network.

In order to model the scenarios just described, we have to search for patterns that belong to the family *Cloud* as regards the management of features related to the cloud; but it also requires the solution of problems relating to the communication of process and the middleware so belong to the family *Distribution Infrastructure*. Furthermore, the model also requires the use of patterns able to make the system adaptable to changes in the context. So another family to consider is that related to pattern ensuring adaptability.

So, given the application context of the system to be modeled, it is necessary to ensure *adaptability*. Also, since the software must operate in the cloud, the model will have to contemplate *elasticity* requirements. Another fundamental requirement to ensure *fault tolerance*, and hence reliability. A low level of *coupling* is also needed since the system implements features of a cloud environment the coupling should be low. Formally, the previously described analysis can be posed as a query as follows:

– retrieve pattern belonging to the following families: *Distribution Infrastructure*, *Application Control*, *Adaptation and Extension*, *Cloud Patterns*;
– and satisfying the following NFRs: a low degree of *coupling*, a high level of *adaptability*, a high *fault tolerance*, a high level of *elasticity*.

To show how requirements and families necessary to describe the proposed scenario can be modeled using fuzzy DLs, we are going to define the related knowledge base $\mathcal{K}$. The Abox $\mathcal{A}$ contains the following statements. We initially introduce families, patterns and their relations:

$$\begin{array}{l}
\texttt{applicationControl:Familiesadaptation AndExtension:Families} \\
\texttt{distributionInfrastructure:Families} \\
\texttt{cloud:Families}
\end{array}$$

$$\begin{array}{l}
\texttt{observer:SoftwareDesignPattern} \\
\texttt{broker:SoftwareDesignPatternreflection:SoftwareDesignPattern} \\
\texttt{hypervisor:SoftwareDesignPattern} \\
\texttt{strictConsistency:SoftwareDesignPattern}
\end{array}$$

$$\begin{array}{l}
\texttt{(hypervisor, cloud):isInFamily} \\
\texttt{(broker, distributionInfrastructure):isInFamily .(strictConsistency, cloud):isInFamily} \\
\texttt{(observer, applicationControl):isInFamily} \\
\texttt{(reflection, adaptationAndExtension):isInFamily}
\end{array}$$

We then define the NFRs and we state how the patterns previously introduced satisfy them:

$$\begin{array}{l}
\texttt{adaptability:NonFunctionalRequirement} \\
\texttt{dependability:NonFunctionalRequirement} \\
\texttt{reliability:NonFunctionalRequirement} \\
\texttt{elasticity:NonFunctionalRequirement} \\
\texttt{faultTolerance:NonFunctionalRequirement} \\
\texttt{loadBalancing:NonFunctionalRequirement} \\
\texttt{flexibility:NonFunctionalRequirement} \\
\texttt{coupling:NonFunctionalRequirement} \\
\texttt{robustness:NonFunctionalRequirement} \\
\texttt{reflection:}\exists \texttt{adaptabilityRate.} =_{\texttt{verygood}}
\end{array}$$

$$\texttt{strictConsistency}{:}\exists \texttt{dependabilityRate}.=_{\texttt{verygood}}$$
$$\texttt{strictConsistency}{:}\exists \texttt{reliabilityRate}.=_{\texttt{good}}$$

$$\texttt{hypervisor}{:}\exists \texttt{elasticityRate}.=_{\texttt{verygood}}$$
$$\texttt{hypervisor}{:}\exists \texttt{faultToleranceRate}.=_{\texttt{good}}$$
$$\texttt{hypervisor}{:}\exists \texttt{loadBalancingRate}.=_{\texttt{good}}$$

$$\texttt{observer}{:}\exists \texttt{flexibilityRate}.=_{\texttt{good}}\ \texttt{observer}{:}\exists \texttt{adaptabilityRate}.=_{\texttt{medium}}$$

$$\texttt{broker}{:}\exists \texttt{loadBalancingRate}.=_{\texttt{verygood}}\ \texttt{broker}{:}\exists \texttt{robustnessRate}.=_{\texttt{medium}}$$
$$\texttt{broker}{:}\exists \texttt{faultToleranceRate}.=_{\texttt{bad}}\ \texttt{broker}{:}\exists \texttt{reliabilityRate}.=_{\texttt{good}}.$$

The TBox contains axioms defining inverse and direct relations between pairs of NFRs. Namely:

$$\exists \texttt{flexibilityRate.High} \sqsubseteq \exists \texttt{couplingRate.Low}$$

$$\exists \texttt{elasticityRate.High} \sqsubseteq \exists \texttt{adaptabilityRate.High}$$
$$\exists \texttt{elasticityRate.Fair} \sqsubseteq \exists \texttt{adaptabilityRate.Fair}$$
$$\exists \texttt{elasticityRate.Low} \sqsubseteq \exists \texttt{adaptabilityRate.Low}$$

$$\exists \texttt{robustnessRate.High} \sqsubseteq \exists \texttt{faultToleranceRate.High}$$
$$\exists \texttt{robustnessRate.Medium} \sqsubseteq \exists \texttt{faultToleranceRate.Medium}$$
$$\exists \texttt{robustnessRate.Low} \sqsubseteq \exists \texttt{faultToleranceRate.Low}$$

$$\exists \texttt{faultToleranceRate.High} \sqsubseteq \exists \texttt{reliabilityRate.High}$$
$$\exists \texttt{faultToleranceRate.Medium} \sqsubseteq \exists \texttt{reliabilityRate.Medium}$$
$$\exists \texttt{faultToleranceRate.Low} \sqsubseteq \exists \texttt{reliabilityRate.Low}$$

$$\exists \texttt{reliabilityRate.High} \sqsubseteq \exists \texttt{dependabilityRate.High}$$
$$\exists \texttt{reliabilityRate.Medium} \sqsubseteq \exists \texttt{dependabilityRate.Medium}$$
$$\exists \texttt{reliabilityRate.Low} \sqsubseteq \exists \texttt{dependabilityRate.Low}$$

$$\exists \texttt{loadBalancingRate.High} \sqsubseteq \exists \texttt{elasticityRate.High}$$
$$\exists \texttt{loadBalancingRate.Medium} \sqsubseteq \exists \texttt{elasticityRate.Medium}$$
$$\exists \texttt{loadBalancingRate.Low} \sqsubseteq \exists \texttt{elasticityRate.Low}.$$

Retrieval of the set of patterns that best satisfies the needed requirements will be obtained using the novel *Covering Answer Set* reasoning task we have introduced in Sect. 4.2. That is, we define a covering answer set query of the form

$$Q = \texttt{ans}_3(@^{AM}, \mathcal{K}, C_1, C_2, C_3, C_4)\ , \tag{23}$$

which we are going now to build incrementally. First of all let us now consider the query:

- families to consider are the following: *Adaptation and Extension*;
- needed NFRs is *adaptability* with a high degree.

The requirements are modeled as:

$$C' = \exists \texttt{isInFamily.}\{\texttt{adaptationAndExtension}\} \sqcap \exists \texttt{adaptabilityRate.High}.$$

Under standard fuzzy logic, it can be shown that $\texttt{ans}(\mathcal{K}, C')$ contains the following statements:

$$\langle \texttt{reflection}{:}\exists \texttt{isInFamily.}\{\texttt{adaptationAndExtension}\}, 1\rangle$$
$$\langle \texttt{reflection}{:}\exists \texttt{adaptability.High}, 1\rangle$$
$$\langle \texttt{reflection}{:}C', 1\rangle$$

$$\langle \texttt{strictConsistency}{:}\exists \texttt{isInFamily.}\{\texttt{adaptationAndExtension}\}, 0\rangle$$
$$\langle \texttt{strictConsistency}{:}\exists \texttt{adaptability.High}, 0\rangle$$
$$\langle \texttt{strictConsistency}{:}C', 0\rangle$$

$$\langle \texttt{hypervisor}{:}\exists \texttt{isInFamily.}\{\texttt{adaptationAndExtension}\}, 0\rangle$$
$$\langle \texttt{hypervisor}{:}\exists \texttt{adaptability.High}, 1\rangle$$
$$\langle \texttt{hypervisor}{:}C', 0\rangle$$

$$\langle \texttt{observer}{:}\exists \texttt{isInFamily.}\{\texttt{adaptationAndExtension}\}, 0\rangle$$
$$\langle \texttt{observer}{:}\exists \texttt{adaptability.High}, 0.66\rangle$$
$$\langle \texttt{observer}{:}C', 0\rangle$$

$\langle$broker:$\exists$isInFamily.{adaptationAndExtension}, 0$\rangle$
$\langle$broker:$\exists$adaptability.High, 0.66$\rangle$
$\langle$broker:C', 0$\rangle$.

$\exists$couplingRate.Low $\sqsubseteq$ $\exists$scalabilityRate.High

$\exists$adapatabilityRate.High $\sqsubseteq$ $\exists$interoperabilityRate.High
$\exists$adapatabilityRate.Fair $\sqsubseteq$ $\exists$interoperabilityRate.Faie
$\exists$adapatabilityRate.Low $\sqsubseteq$ $\exists$interoperabilityRate.Low

$\exists$responseTimeRate.High $\sqsubseteq$ $\exists$performanceRate.High
$\exists$responseTimeRate.Medium $\sqsubseteq$ $\exists$performanceRate.Medium
$\exists$responseTimeRate.Low $\sqsubseteq$ $\exists$performanceRate.Low .

C' is satisfied by the list of pattern in:

$$\begin{aligned}\text{ans}(\mathcal{K}, C') = \{&\langle\text{reflection}, 1\rangle, \langle\text{strictConsistency}, 0\rangle,\\ &\langle\text{hypervisor}, 0\rangle, \langle\text{observer}, 0\rangle, \langle\text{broker}, 0\rangle\}\ .\end{aligned}$$

Please note that, although there is no explicit statements about the adaptability value of *Hypervisor* patterns, thanks to the interaction between elasticityRate and adaptabilityRate stated in the TBox, we infer that *Hypervisor* has a high level of adaptability.

We now add all the other families required to solve our task. Therefore, we modify C' in C_1 as follows:

$$\begin{aligned}C_1 = (&\exists\text{isInFamily.}\{\text{adaptationAndExtension}\}\\ &\sqcup\exists\text{isInFamily.}\{\text{cloud}\}\\ &\sqcup\exists\text{isInFamily.}\{\text{applicationControl}\}\\ &\sqcup\exists\text{isInFamily.}\{\text{distributionInfrastructure}\})\\ \sqcap&\exists\text{adaptabilityRate.High.}\end{aligned}$$

It can be shown that ans($\mathcal{K}$, C_1) under standard fuzzy logic contains the following statements:

$\langle$reflection:$\exists$isInFamily.{adaptationAndExtension}, 1$\rangle$
$\langle$reflection:$\exists$isInFamily.{cloud}, 0$\rangle$
$\langle$reflection:$\exists$isInFamily.{applicationControl}, 0$\rangle$
$\langle$reflection:$\exists$isInFamily.{adaptationAndExtension}, 0$\rangle$
$\langle$reflection:$\exists$adaptability.High, 1$\rangle$
$\langle$reflection:C_1, 1$\rangle$

$\langle$strictConsistency:$\exists$isInFamily.{adaptationAndExtension}, 0$\rangle$
$\langle$strictConsistency:$\exists$isInFamily.{cloud}, 1$\rangle$
$\langle$strictConsistency:$\exists$isInFamily.{applicationControl}, 0$\rangle$
$\langle$strictConsistency:$\exists$isInFamily.{distributionInfrastructure}, 0$\rangle$
$\langle$strictConsistency:$\exists$adaptability.High, 0$\rangle$
$\langle$strictConsistency:C_1, 0$\rangle$

$\langle$hypervisor:$\exists$isInFamily.{adaptationAndExtension}, 0$\rangle$
$\langle$hypervisor:$\exists$isInFamily.{cloud}, 1$\rangle$
$\langle$hypervisor:$\exists$isInFamily.{applicationControl}, 0$\rangle$
$\langle$hypervisor:$\exists$isInFamily.{distributionInfrastructure}, 0$\rangle$
$\langle$hypervisor:$\exists$adaptability.High, 1$\rangle$
$\langle$hypervisor:C_1, 1$\rangle$

$\langle$observer:$\exists$isInFamily.{adaptationAndExtension}, 0$\rangle$
$\langle$observer:$\exists$isInFamily.{cloud}, 0$\rangle$
$\langle$observer:$\exists$isInFamily.{applicationControl}, 1$\rangle$
$\langle$observer:$\exists$isInFamily.{distributionInfrastructure}, 0$\rangle$
$\langle$observer:$\exists$adaptability.High, 0.66$\rangle$
$\langle$observer:C_1, 0.66$\rangle$

$\langle$broker:$\exists$isInFamily.{adaptationAndExtension}, 0$\rangle$
$\langle$broker:$\exists$isInFamily.{cloud}, 0$\rangle$
$\langle$broker:$\exists$isInFamily.{applicationControl}, 0$\rangle$
$\langle$broker:$\exists$isInFamily.{distributionInfrastructure}, 1$\rangle$
$\langle$broker:$\exists$adaptability.High, 0.66$\rangle$
$\langle$broker:C_1, 0.66$\rangle$.

Table 2 Answer sets use case I

$\texttt{ans}(\mathcal{K}, C_2)$	$\texttt{ans}(\mathcal{K}, C_3)$	$\texttt{ans}(\mathcal{K}, C_4)$
⟨reflection, 0⟩	⟨reflection, 0⟩	⟨reflection, 0⟩
⟨strictConsistency, 0⟩	⟨strictConsistency, 0⟩	⟨strictConsistency, 0⟩
⟨hypervisor, 1⟩	⟨hypervisor, 1⟩	⟨hypervisor, 0⟩
⟨observer, 0⟩	⟨observer, 0⟩	⟨observer, 0⟩
⟨broker, 1⟩	⟨broker, 0.33⟩	⟨broker, 1⟩

Based on the previous results, it follows that

$$\texttt{ans}(\mathcal{K}, C_1) = \{\ \langle \texttt{reflection}, 1\rangle, \langle \texttt{strictConsistency}, 0\rangle, \\ \langle \texttt{hypervisor}, 1\rangle, \langle \texttt{observer}, 0.66\rangle, \langle \texttt{broker}, 0.66\rangle\}\ .$$

We are now ready to define also C_2, C_3 and C_4 in Eq. (23).

$$\begin{aligned} C_2 = (&\exists \texttt{isInFamily}.\{\texttt{adaptationAndExtension}\} \\ &\sqcup \exists \texttt{isInFamily}.\{\texttt{cloud}\} \\ &\sqcup \exists \texttt{isInFamily}.\{\texttt{applicationControl}\} \\ &\sqcup \exists \texttt{isInFamily}.\{\texttt{distributionInfrastructure}\}) \\ &\sqcap \exists \texttt{elasticityRate}.\texttt{High}. \end{aligned}$$

$$\begin{aligned} C_3 = (&\exists \texttt{isInFamily}.\{\texttt{adaptationAndExtension}\} \\ &\sqcup \exists \texttt{isInFamily}.\{\texttt{cloud}\} \\ &\sqcup \exists \texttt{isInFamily}.\{\texttt{applicationControl}\} \\ &\sqcup \exists \texttt{isInFamily}.\{\texttt{distributionInfrastructure}\}) \\ &\sqcap \exists \texttt{faultToleranceRate}.\texttt{High}. \end{aligned}$$

$$\begin{aligned} C_4 = (&\exists \texttt{isInFamily}.\{\texttt{adaptationAndExtension}\} \\ &\sqcup \exists \texttt{isInFamily}.\{\texttt{cloud}\} \\ &\sqcup \exists \texttt{isInFamily}.\{\texttt{applicationControl}\} \\ &\sqcup \exists \texttt{isInFamily}.\{\texttt{distributionInfrastructure}\}) \\ &\sqcap \exists \texttt{couplingRate}.\texttt{Low}. \end{aligned}$$

With reference to these concepts, the results for $\texttt{ans}(\mathcal{K}, C_i)$ with $i = 2 \ldots 4$ are illustrated in Table 2.

We can finally compute the result for Eq. (23):

$$\begin{aligned} Q = \{\ &\langle \{\texttt{hypervisor}, \texttt{broker}\}, 1\rangle, \\ &\langle \{\texttt{reflection}, \texttt{broker}\}, 0.8325\rangle, \\ &\langle \{\texttt{observer}, \texttt{broker}\}, 0.75\rangle\}\ . \end{aligned}$$

The pair of pattern `hypervisor` and `broker` is the best solution; the second rank is the pair `broker` and `reflection`; the third is the pair `broker` and `observer`.

Please note that e.g. $\langle\{\texttt{broker}\}, 0.75\rangle$ is ruled out from Q as there exists a superset with strictly higher score (e.g., $\{\langle\texttt{hypervisor}, \texttt{broker}\}, 1\rangle$).

6 Implementation

In this section, we present the implemented fuzzy ontology-driven decision support system. The overall architecture is depicted in Fig. 3.

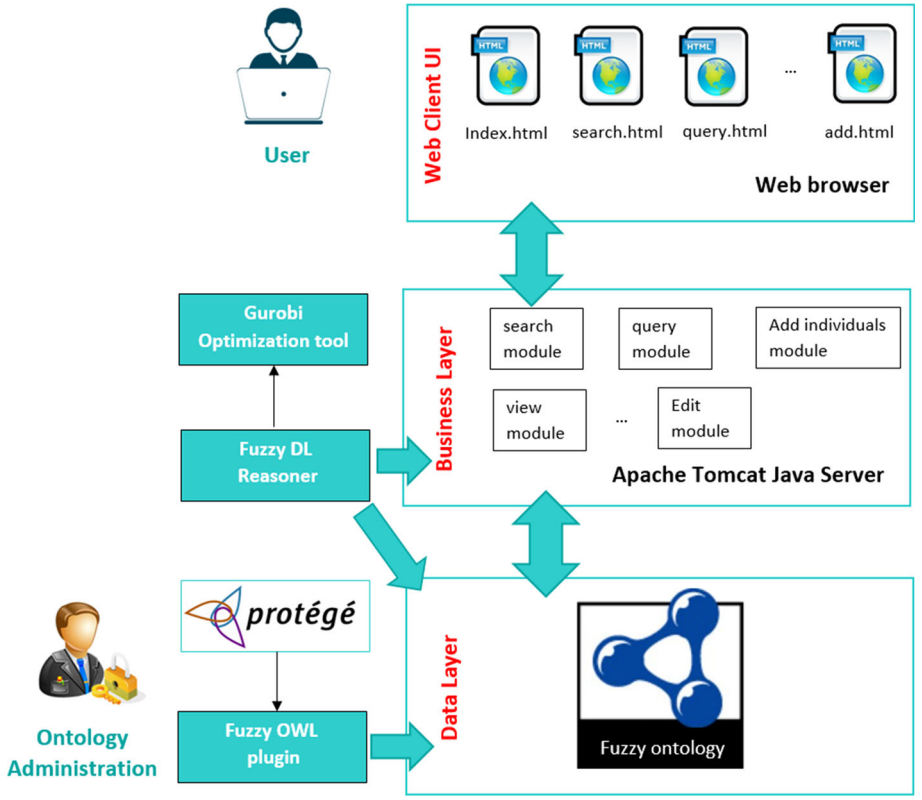

Fig. 3 Model of our tool, and linked components

The tool was developed using Eclipse as a web development platform integrating Apache Tomcat [5] as application server. The business layer is consisting of a modules that implements the principal functionalities of the tool. First of all enable *(i)* search of patterns, NFRs and pattern families query the ontology showing their *annotations*; and *(ii)* solve *Covering Answer Set* query.

Ontology Administration provides for three main functionalities: the insertion of a new pattern or a new NFR, the insertion of a fuzzy DL statements (relations intercurring between NFRs) and the editing of ontology individuals (pattern or NFR). Before executing any change on the ontology, i.e., any of the administrator functionalities, a consistency check is performed.

The linked components are the following:

- **Fuzzy DL Reasoner** is a java-based reasoner allowing to work with vague information, previously described;
- **Gurobi** is a library for mathematical programming. It is focused on linear programming solver (LP solver), quadratic programming solver (QP solver), quadratically constrained programming solver (QCP solver), mixed-integer linear programming solver (MILP solver), mixed-integer quadratic programming solver (MIQP solver), and mixed-integer

[5] https://tomcat.apache.org/.

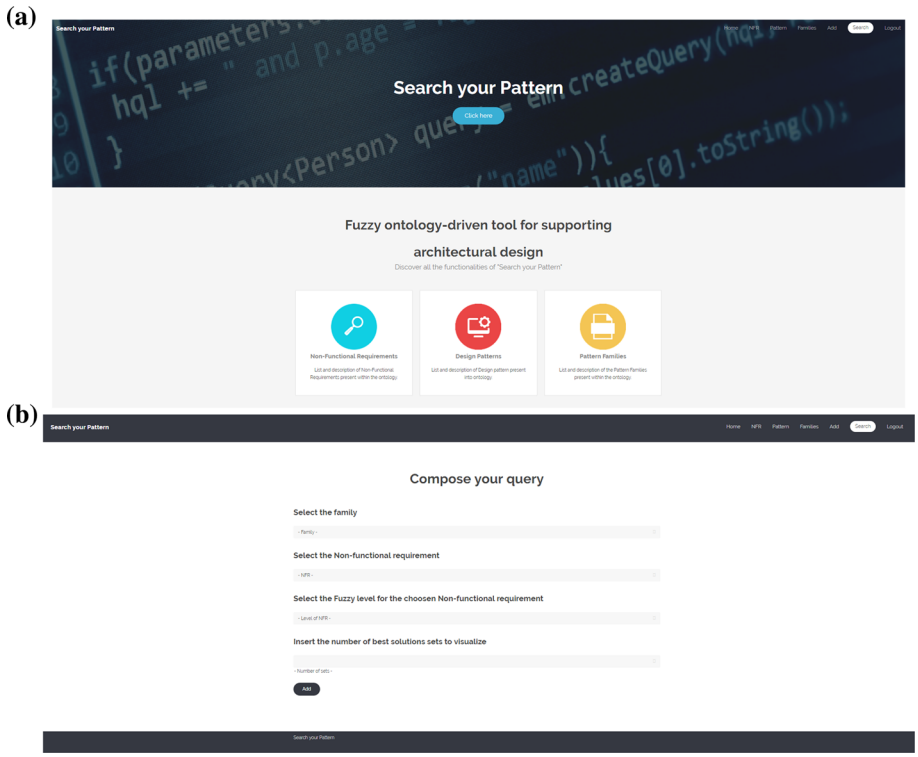

Fig. 4 **a** Home page of the tool Search your Pattern. **b** Screenshot of tool query composition

quadratically constrained programming solver (MIQCP solver) and it is used within the fuzzy DL reasoner for MILP calculations;[6]

– **Protégé** is an open-source ontology editor that supports the Web Ontology Language (OWL); [7]
– **FuzzyOWL plugin** is a plug-in for Protege 4.1 that allows users to edit, save fuzzy OWL 2 ontologies, and submit queries to the underlying inference engine FuzzyDL.

Screenshots in Fig. 4 show the home page of our tool. Figure 4b shows the form for selecting the main elements needed to solve the task that will be translated in the Covering Answer Set query formalism as described in Sect. 4.2.

The core component of our tool is the fuzzy ontology. We constructed a fuzzy OWL 2 ontology [9] according to the framework presented so far.[8] Table 3 summarizes the metrics associated with our fuzzy ontology, whose consistency has been checked and validated with the *fuzzyDL* Reasoner.

Figure 5a shows a screenshot of the *Protégé* editor GUI illustrating the main classes of our ontology and part of the individuals in the ABox. The fuzzy OWL 2 plug-in for Protégé made aleviated the modeling of the fuzzy sets as well as of data properties assertions as shown

[6] The library is available at www.gurobi.com.

[7] It is available for free download at http://protege.stanford.edu/download/download.html.

[8] The fuzzy ontology is available online at http://sisinflab.poliba.it/semanticweb/ontologies/architecturalpatterns/.

Table 3 Ontology metrics

Metrics	
Axiom	1672
Logical axiom count	975
Class count	3
Object property count	2
Data property count	26
Individual count	174
Class axioms	
SubClassOf axioms count	31
EquivalentClasses axioms count	23
GCI count	54
Object property axioms	
ObjectPropertyDomain axioms count	2
ObjectPropertyRange axioms count	2
Data property axioms	
FunctionalDataProperty axioms count	26
DataPropertyRange axioms count	26
Individual axioms	
ClassAssertion axioms count	174
ObjectPropertyAssertion axioms count	399
DataPropertyAssertion axioms count	289

in Fig. 5a, b. Axioms as those in Eqs. (5)–(10) and Eqs. (18)–(22) can be modeled in the *General class axioms* tab. Figure 6 reports their equivalent form with the Manchester syntax of OWL 2.[9] During the modeling, we adopted the Linked Data principles[10] and tried to reuse URIs already available in the web. For instance, we associated the URI http://dbpedia.org/resource/Non-functional_requirement to the class `NonFunctionalRequirement`. The same principle has guided the selection of the URIs for design patterns and NFRs. We referred to DBpedia[11] as it is a "de facto" standard in the representation of entities in the web.[12]

7 Discussion

In order to evaluate the degree of usefulness for the tool, we designed a controlled experiment. We sketched the Cloud-Social-Adaptable System example presented in Sect. 5 and we then proposed it to six different teams of students. The teams were assembled so that each one would be composed by three second year graduate students. All students were trained during the MSc course on software design, architectural pattern, NFRs modeling, architectural design.

[9] http://www.w3.org/TR/owl2-manchester-syntax/.

[10] http://www.w3.org/DesignIssues/LinkedData.html.

[11] http://dbpedia.org.

[12] See, e.g. the diagram available at http://lod-cloud.net.

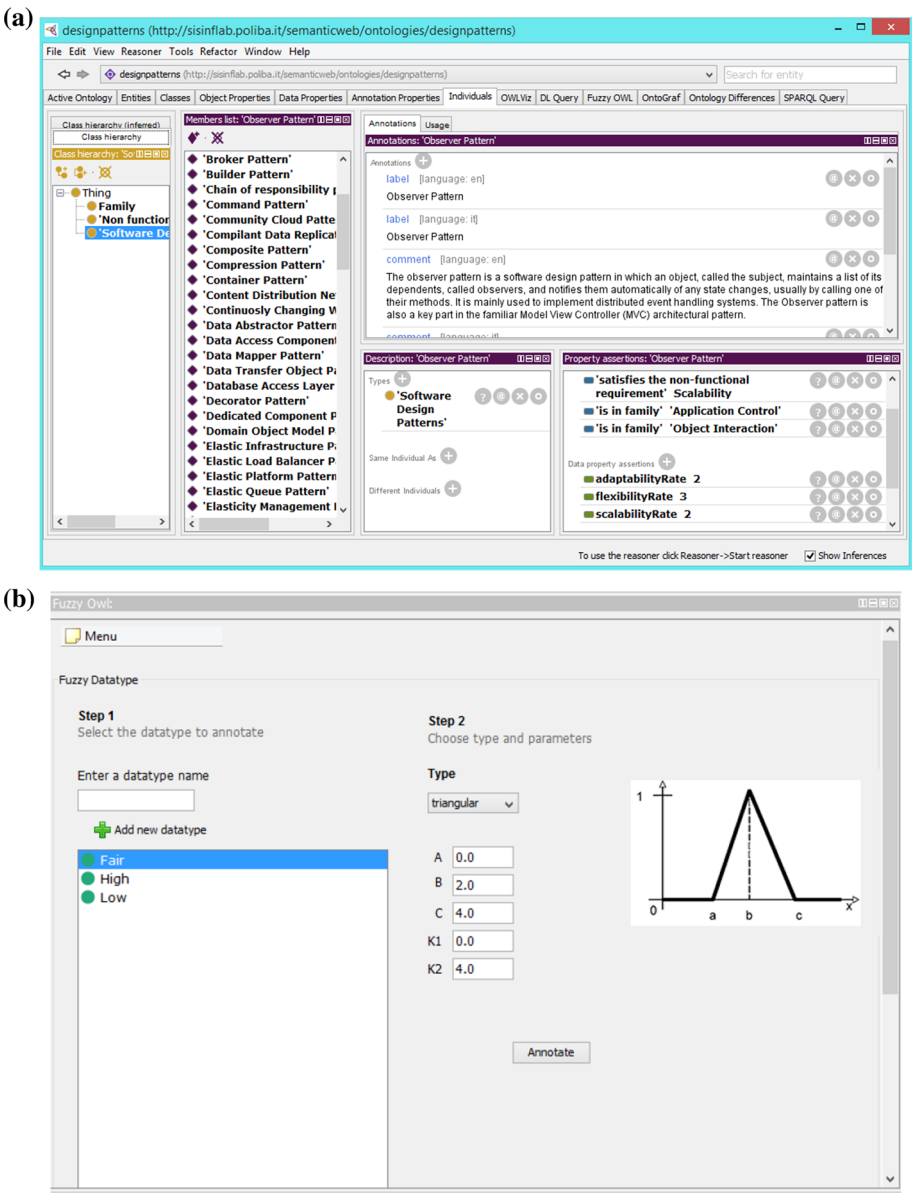

Fig. 5 **a** Individuals tab, **b** creation of a fuzzy datatype *Fair* with the fuzzy OWL plug-in

Three teams (T_1, T_2, T_3) solved the problem supported by the tool while the other three teams (T_4, T_5, T_6) solved the problem using only their own experience. The teams had no idea that other teams were working on the same use case and they had been instructed not to comment the experiment with anyone else. The solution to each design problem was provided as the design of an architecture considering quality aspects and choosing the patterns that best model NFRs suitable for the given context, domain and goals.

The teams T_1, T_2, T_3 were provided with an abstract description in natural language of the use-case scenario and each team formulated the query to be submitted to the tool. As shown

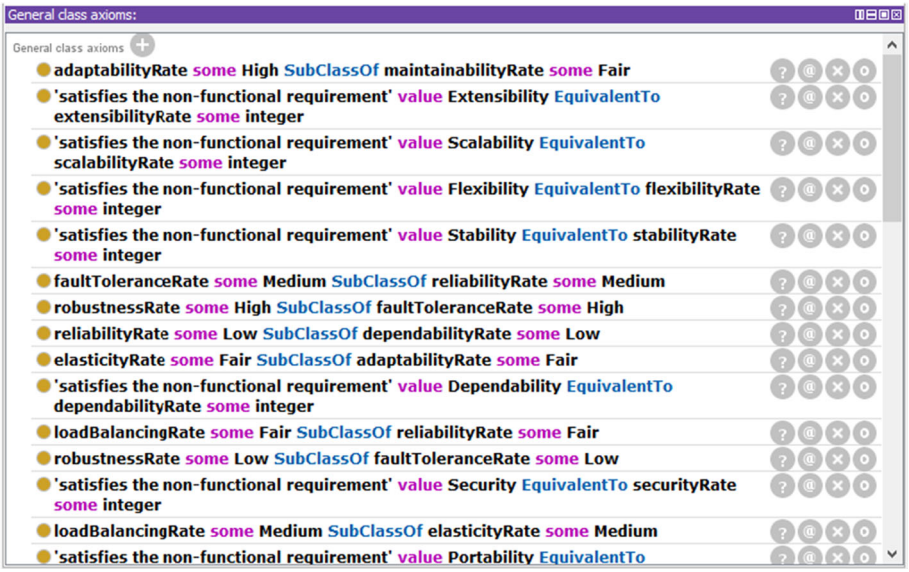

Fig. 6 General class axioms tab

in Table 4, all the three teams composed a query that slightly differs from one another only for the NFR degree (*high*, *medium*, *low*).

The other three teams—T_4, T_5, T_6, being provided with the entire query, solved the problems using only their own experience.

The three solutions provided by the last teams are not expected to be the same, since the answer derives from the experience, from their reasoning, and from creativity. Measuring the goodness of results is not immediate. We compared the solutions provided by the three teams which were supported by the tool with those provided by the teams not supported by the tool and with the human expert-provided solution. Based on the previously described analysis in Sect. 5, the best formulated query is composed as follows:

- Families: *Distribution Infrastructure*, *Application Control*, *Adaptation and Extension*, *Cloud Patterns*;
- NFRs: a low degree of *coupling*, a high level of *adaptability*, a high *fault tolerance*, a high level of *elasticity*.

The set of top-3 ranked answers Q for the provided query is the same as the one reported at the end of Sect. 5.

At the end of the modeling process, each team was supplied with the solution proposed by the others. A qualitative evaluation of the approaches was asked to the teams. Table 5 summarizes the results for this qualitative evaluation: the solutions provided with the support of the tool where judged fully correct and complete by the others teams; on the other hand, among the solutions provided by the teams not using the tool, one was considered fully acceptable by all of the other teams while the other two needed, respectively, minor/ major revisions.

Note that the initial queries for the first three teams were slightly different since each team faced the problem starting from a different point of view. Nevertheless, the answers for these queries was the same, corresponding to the best solution.

Table 4 Queries formulated by each team and corresponding answers (best solution)

Team	Query		Best solution
	Families	NFRs	
T_1	*Distribution Infrastructure*, *Application Control*, *Adaptation and Extension*, *Cloud Patterns*	low *coupling*, high *adaptability*, high *fault tolerance*, high *elasticity*	Hypervisor, Broker
T_2	*Distribution Infrastructure*, *Application Control*, *Adaptation and Extension*, *Cloud Patterns*	medium *coupling*, high *adaptability*, high *fault tolerance*, high *elasticity*	Hypervisor, Broker
T_3	*Distribution Infrastructure*, *Application Control*, *Adaptation and Extension*, *Cloud Patterns*	low *coupling*, medium *adaptability*, medium *fault tolerance*, high *elasticity*	Hypervisor, Broker
T_4	*Distribution Infrastructure*, *Application Control*, *Adaptation and Extension*, *Cloud Patterns*	low *coupling*, high *adaptability*, high *fault tolerance*, high *elasticity*	Hypervisor, Broker
T_5	*Distribution Infrastructure*, *Application Control*, *Adaptation and Extension*, *Cloud Patterns*	low *coupling*, high *adaptability*, high *fault tolerance*, high *elasticity*	Reflection, Broker
T_6	*Distribution Infrastructure*, *Application Control*, *Adaptation and Extension*, *Cloud Patterns*	low *coupling*, high *adaptability*, high *fault tolerance*, high *elasticity*	Integration Provider Pattern, Observer

Table 5 Cross evaluation among teams

	Full acceptance	Minor revision	Major revision
T_1, T_2, T_3 Solutions	3	0	0
T_4, T_5, T_6 Solutions	1	1	1

Table 6 Elapsed time to solve the problem

Team	Elapsed time
T_1	2 days
T_2	1 day and a half
T_3	2 days
T_4	4 days
T_5	5 days
T_6	4 days and a half

Table 6 shows the time used by the teams to solve the problem. We measured the efficiency of the process, in terms of time elapsed between the starting point of the work and the final solution provided by each team.

Subsequently, in order to measure the matching degree of retrieved patterns with the three ones proposed by the human expert, we measured the similarity between the proposed solutions by the six teams with respect to these three. To this end we used a combination of Jaccard coefficients [49]. We recap that the Jaccard similarity between two sets A and B is defined as:

$$J(A, B) = \frac{| A \cap B |}{| A \cup B |}$$

In detail, we compare the similarity of each solution with the ranking provided by the covering answer set algorithm; w.r.t. the standard Jaccard similarity coefficient we introduce an exponential decay for each retrieved covering answer set. Formally, being:

- k, the number of covering answer set to retrieve;
- ans^{Tj} the solution set provided by the j-th Team (T_j);
- Q_i the i-th answer set composing the *top-k covering answer set* Q;

the similarity between the solutions set provided by each Team ans^{Tj} and the top-k ranked answers Q is defined as in the following:

$$sim(ans^{Tj}, Q) = \frac{\sum_{i=1}^{k} J(ans^{Tj}, Q_i) \cdot e^{-(i-1)}}{k}, \text{ with } i = 1 \ldots k \quad (1)$$

Table 7 illustrates the similarity values obtained by using Eq. (1).

In particular, the first column illustrates the similarity value with respect to the best solution Q_1. Note that the similarity analysis is in line with the qualitative analysis carried out in the first step and that the teams using the tool performed better than those without it.

8 Related work

Existing approaches for knowledge representation in software engineering are listed in this section. An emphasis is posed on modeling of NFRs, modeling of design pattern, relationships

Table 7 Similarity values with respect to Q_1 (best solution) and Q (the top-3 ranked answers)

Team	$J(ans^{T_j}, Q_1)$	$J(ans^{T_j}, Q)$
T_4	1.00	0.389
T_5	0.333	0.249
T_6	0.00	0.056

among NFRs and design patterns, approaches for supporting decision making in software architecture design and existing tools.

A systematic mapping study about application of method based on knowledge representation to solve typical problems of software architecture is proposed in [50]. The authors outline the most relevant research directions and the weaknesses in the two domains and in their combination. The study witnesses an increased use of knowledge representation based methods to modeling software architectures, especially concerning the model of architectural essentials and relationships among them. In particular, (1) the study calls for more deep investigation on use of knowledge representation to study impact analysis of software architectures. (2) also approaches of knowledge recovery needs to be further explored; (3) the architectural implementation can benefit from knowledge sharing; (4) automatic and semi-automatic reasoning should be improved; (5) the study shows the need of deep studies about benefit of knowledge approach to software architecture; (6) knowledge-based approaches could be implied in several new domains.

Ontological approaches Ontologies are frequently used in several contexts of software engineering. [59] provides a complete analysis for using ontologies in software engineering, especially in the development process. In [32] the use of ontologies in software engineering is surveyed, by considering all the phases of software development Kruchten [47] models architectural design decision in software-intensive systems using an ontology in which each architectural design decision belongs to one of the following categories: existence decisions, behavior decisions, property decisions. The ontology is supported by a tool able to list decision and relations, visualizing design structure and temporal view of design decisions. The works [41,42] propose the modeling of patterns by means of ontologies, but only for a set of patterns, i.e., the usability design patterns. A metamodel for design pattern language in proposed in [42]. In [46] a *Design Pattern Intent Ontology* (DPIO) to formalize relationships among *Gang of Four's* (GoF) patterns is proposed, the ontology is used to suggest a pattern to solve a given design problem. An extended version of DPIO ontology to suggest patterns to solve integration problems is in [36]. Formalization of web design pattern is modeled in [55] using ontologies. A pattern scanner for the Java language based on a OWL design pattern ontology is proposed in [25] for recognizing patterns in source code. In [4] an ontology for modeling the enterprise architecture domain is proposed, while an ontology to model architectural design decisions is described in [48]. More recently, in [35] NFRs are defined from an ontological point of view and a language is proposed to model them. Differently from our approach in [35] there is no relation with patterns and their families. The authors identify the fuzziness of NFR specifications but their approach is to a more conceptual level. Analogously, the authors of [60] define an annotated ontological vocabulary of NFRs with the main aim of classifying natural language sentences with reference to software specifications. Chi-Lun Liu proposes in [51] a modeling of NFRs that mixes ontologies with rules with the goal of catching inconsistencies in information systems specifications. Unfortunately, neither the overall modeling does not take into account the fuzzy nature of NFRs nor considers relations with architectural patterns. The same issues can be found in [26] where an ontology to model NFRs is proposed.

Design patterns and NFRs modeling In [54] features of design pattern are modeled using formal methods that capture temporal properties. The authors in [66] propose a language named *Balanced Pattern Specification Language* (BPSL) for specification of behavioral features and structural aspects of patterns [68]. The language derives from Temporal Logics of Action and First Order Logic. In [38] the authors study relationships between design pattern: they consider application of pattern and tactics by using an entity diagram for annotating information about used tactics.

Use of NFRs is studied by Chung et al. [18] and Botella et al. [11].

While [29,33] survey several definitions of NFRs. Mylopoulos et al. [56] describe the use of NFRs with process or product oriented approaches. Several automated tools have been proposed for supporting the enterprise architect in architectural modeling, such as [16,24,45,70].

Eventually, relationship between patterns and NFRs has been addressed in [61] that defines the policy to specify how an attribute affects the quality of non-functional properties. [34] studies the relationships between NFRs and design patterns. A framework to formalize relations between patterns and tactics is proposed by Harrison et al. [40] to study the impact at implementation level. The impact of a tactic on quality attributes is described in [40].

9 Conclusion

In this work was proposed as main goal the development of a decision support system for supporting designers and software architect—having greater or lesser experience—in the process of modeling a software system's architecture. Achievement of the proposed goal was supported by listing some useful research questions. We describe hereby how the five research questions have been addressed throughout the work. For the first two questions, Q1, and Q2 we studied state of the art concerning main categories of requirements—functional and non-functional—by posing great interest in non-functional requirements. Answer to question Q1 can be found in the Sect. 3 and in a more extended study in the related work in Sect. 8, where the use of catalogues of NFRs was found as the more relevant approach. To answer question Q2, we reviewed some state-of-the-art approaches for pattern categorization or classification. The results of this study are summarized in Sect. 3, where pattern, pattern languages, problem area and frameworks available are cited; a more extended study is reported in the related work in Sect. 8. The answer to question Q3 was derived from studying all the approaches to relate NFRs and pattern described in Sect. 3: relationship between NFRs and reusable schema is introduced as the starting point of the work we developed. This study allowed us to build the theoretical framework of our approach that is the answer to question Q4 by defining the fuzzy ontology in which we catalog the NFRs, the families and the patterns according to catalogues and categorizations found in the state-of-the-art descriptions. By defining the *Covering Answer Set* algorithm for retrieval of pattern from the fuzzy ontology, we solved the problem of facilitating the decision-making problem in architectural design.

Question Q5 led us to implement and validate the decision support tool described in the paper through the use-case scenario and preliminary performed experiment on the method.

Main strengths of our approach is the use of *fuzzy ontologies* for modeling knowledge that relates NFRs with patterns and with the families they belong to. The fuzzy nature of the adopted language makes it possible to express and reason with concepts like "the Proxy pattern has a good load balancing level" or that "the load balancing level is directly proportional to the level of reliability". The proposed formalism and the reasoning service was implemented in

a tool for supporting the human expert, the architect or the designer to select a set of patterns that matched the desired requirements. To this end, we have also defined a novel reasoning task able to retrieve a ranked set of patterns that match the user requirements expressed in terms of NFRs. The use of a formal approach inherently guarantees the correctness and consistency of the data entered and the trust of the approach. The reasoner used to build the ontology is able to detect and prevent inconsistent data and incongruences. At last, we have presented a use case and have developed a fuzzy OWL 2 knowledge base describing 28 pattern families, 37 non-functional requirements, 109 design patterns and their relations. The use case was experienced using the tool by an expert whose solution is provided in the work and by a number of teams whose solution were compared with those provided by the expert supported by the tool. Results proved that solutions proposed by the tool-supported teams were more efficient in terms of elapsed time and similarity.

Acknowledgements Francesco Nocera is supported by Exprivia S.p.A Ph.D Grant 2018.

References

1. Avgeriou P, Grundy J, Hall JG, Lago P, Mistrík I (2011) Relating software requirements and architectures. Springer, Berlin
2. Avgeriou P, Zdun U (2005) Architectural patterns revisited—a pattern language. In: Proceedings of the 10th European conference on pattern languages of programs (EuroPLoP), pp 431–470
3. Baader F, Calvanese D, McGuinness D, Nardi D, Patel-Schneider PF (eds) (2003) The description logic handbook: theory, implementation, and applications. Cambridge University Press
4. Bakhshandeh M, Antunes G, Mayer R, Borbinha J, Caetano A (2013) A modular ontology for the enterprise architecture domain. In: 2013 17th IEEE international enterprise distributed object computing conference workshops (EDOCW). IEEE, pp 5–12
5. Bass L, Clements P, Kazman R (2005) Software architecture in practice. Addison-Wesley, Boston (Munich [u.a.])
6. Bass L, Klein M, Bachmann F (2002) Quality attribute design primitives and the attribute driven design method. In: Revised papers from 4th International workshop on software product-family engineering, vol 2290. Springer, pp 169–186
7. Berners-Lee T, Hendler J, Lassila O (2001) The semantic web. Sci Am 284(5):34–43
8. Bobillo F, Cerami M, Esteva F, García-Cerdaña À, Peñaloza R, Straccia U (2015) Fuzzy description logics in the framework of mathematical fuzzy logic. In: Petr Cintula Christian Fermüller CN (ed) Handbook of mathematical fuzzy logic, Volume 3, Studies in logic, mathematical logic and foundations, vol 58, chap. 16. College Publications, pp 1105–1181
9. Bobillo F, Straccia U (2011) Fuzzy ontology representation using OWL 2. Int J Approx Reason 52:1073–1094
10. Bobillo F, Straccia U (2016) The fuzzy ontology reasoner fuzzyDL. Knowl Based Syst 95:12–34. https://doi.org/10.1016/j.knosys.2015.11.017
11. Botella P, Burgues X, Franch X, Huerta M, Salazar G (2001) Modeling non-functional requirements. In: Proceedings of Jornadas de Ingenieria de Requisitos Aplicada JIRA **2001**
12. Brooks FP Jr (1987) No silver bullet essence and accidents of software engineering. Computer 20(4):10–19. https://doi.org/10.1109/MC.1987.1663532
13. Buschmann F, Henney K, Schmidt DC (2007) Pattern-oriented software architecture, volume 4, a pattern language for distributed computing. Wiley, New York
14. Buschmann F, Meunier R, Rohnert H, Sommerlad P, Stal M (1996) Pattern-oriented software architecture: a system of patterns. Wiley, New York
15. Calegari S, Sanchez E (2007) A fuzzy ontology-approach to improve semantic information retrieval. In: Proceedings of the third international conference on uncertainty reasoning for the semantic web-volume 327. pp 117–122. CEUR-WS. org
16. Capilla R, Nava F, Pérez S, Dueñas JC (2006) A web-based tool for managing architectural design decisions. ACM SIGSOFT Softw Eng Notes 31(5):4
17. Chung L, Nixon BA, Yu E, Mylopoulos J (2012) Non-functional requirements in software engineering, vol 5. Springer, Berlin

18. Chung L, do Prado Leite JCS (2009) On non-functional requirements in software engineering. In: Borgida AT, Chaudhri VK, Giorgini P, Yu ES (eds) Conceptual modeling: foundations and applications. Lecture notes in computer science, vol 5600. Springer, Berlin, Heidelberg, pp 363–379
19. Cuenca-Grau B, Horrocks I, Motik B, Parsia B, Patel-Schneider P, Sattler U (2008) OWL 2: the next step for OWL. J Web Semant 6(4):309–322
20. Cysneiros LM (2007) Evaluating the effectiveness of using catalogues to elicit non-functional requirements. In: Proc. of 10th Workshop in requirements engineering. pp 107–115
21. Davis AM (1993) Software requirements: objects, functions, and states. Prentice-Hall Inc., Englewood Cliffs
22. Di Noia T, Mongiello M, Di Sciascio E (2014) Ontology-driven pattern selection and matching in software design. In: European conference on software architecture. Springer, Berlin, pp. 82–89. https://doi.org/10.1007/978-3-319-09970-5_8
23. Di Noia T, Mongiello M, Straccia U (2015) Fuzzy description logics for component selection in software design. In: Software engineering and formal methods: SEFM 2015 collocated workshops, revised selected papers (Lecture notes in computer science), vol 9509, pp 228–239. Springer, Berlin. https://doi.org/10.1007/978-3-662-49224-619
24. Diaz-Pace A, Kim H, Bass L, Bianco P, Bachmann F (2008) Integrating quality-attribute reasoning frameworks in the ArchE design assistant. In: Becker S, Plasil F, Reussner R (eds) Quality of software architectures. Models and architectures. QoSA 2008. Lecture notes in computer science, vol 5281. Springer, Berlin, Heidelberg, pp 171–188
25. Dietrich J, Elgar C (2005) A formal description of design patterns using owl. In: Proceedings software engineering conference, 2005 Australian. IEEE, pp. 243–250
26. Dobson G, Hall S, Kotonya G (2007) A domain-independent ontology for non-functional requirements. In: Proceedings of ICEBE 2007, IEEE international conference on e-business engineering and the workshops SOAIC 2007, SOSE 2007, SOKM 2007, 24–26 October, 2007, Hong Kong, China, pp. 563–566
27. Egyed A, Grunbacher P (2004) Identifying requirements conflicts and cooperation: how quality attributes and automated traceability can help. Softw IEEE 21(6):50–58
28. Fehling C, Leymann F, Retter R, Schupeck W, Arbitter P (2014) Cloud computing patterns: fundamentals to design, build, and manage cloud applications. Springer, Vienna
29. Franch X (1998) Systematic formulation of non-functional characteristics of software. In: Proceedings of the third international conference on requirements engineering. IEEE, pp 174–181
30. Gamma E, Helm R, Johnson R, Vlissides J (1994) Design patterns: elements of reusable object-oriented software. Pearson Education, London
31. Garlan D, Shaw M (1994) An introduction to software architecture. Technical report
32. Gašević D, Kaviani N, Milanović M (2009) Ontologies and software engineering. In: Staab S, Studer R (eds) Handbook on Ontologies. Springer, Berlin, Heidelberg, Germany, pp 593–615
33. Glinz M (2007) On non-functional requirements. In: 15th IEEE international conference on requirements engineering, 2007. RE'07. IEEE, pp 21–26
34. Gross D, Yu E (2001) From non-functional requirements to design through patterns. Requir Eng 6(1):18–36
35. Guizzardi RSS, Li F, Borgida A, Guizzardi G, Horkoff J, Mylopoulos J (2014) An ontological interpretation of non-functional requirements. In: formal ontology in information systems—proceedings of the eighth international conference, FOIS 2014, September, 22–25, 2014, Rio de Janeiro, Brazil, pp 344–357
36. Harb D, Bouhours C, Leblanc H (2009) Using an ontology to suggest software design patterns integration. In: Chaudron MRV (ed) Models in software engineering. MODELS 2008. Lecture notes in computer science, vol 5421. Springer, Berlin, Heidelberg, pp 318–331
37. Harrison N, Avgeriou P (2007) Pattern-driven architectural partitioning: balancing functional and non-functional requirements. In: Second international conference on digital telecommunications 2007. ICDT '07. IEEE, pp 21–26
38. Harrison NB, Avgeriou P (2010) How do architecture patterns and tactics interact? a model and annotation. J Syst Softw 83(10):1735–1758
39. Harrison NB, Avgeriou P (2010) Implementing reliability: the interaction of requirements, tactics and architecture patterns. In: Casimiro A, de Lemos R, Gacek C (eds) Architecting dependable systems VII. Lecture notes in computer science, vol 6420. Springer, Berlin, Heidelberg, pp 97–122
40. Harrison NB, Avgeriou P, Zdun U (2010) On the impact of fault tolerance tactics on architecture patterns. In: Proceedings of the 2nd international workshop on software engineering for resilient systems. ACM, pp 12–21
41. Henninger S, Ashokkumar P (2005) An ontology-based infrastructure for usability design patterns. In: Proceedings of the semantic web enabled software engineering (SWESE), Galway, Ireland, pp 41–55

42. Henninger S, Ashokkumar P (2006) An ontology-based metamodel for software patterns. In: 8th international conference on software engineering and knowledge engineering (SEKE2006)
43. Henninger S, Corrêa V (2007) Software pattern communities: current practices and challenges. In: Proceedings of the 14th conference on pattern languages of programs. ACM, p 14
44. Horrocks I, Kutz O, Sattler U (2006) The even more irresistible $\mathcal{SROIQ}$. In: Proceedings of the 10th international conference on principles of knowledge representation and reasoning (KR-06). AAAI Press, pp 57–67
45. Jansen A, Van Der Ven J, Avgeriou P, Hammer DK (2007) Tool support for architectural decisions. In: The working IEEE/IFIP conference on software architecture, 2007. WICSA'07, pp 4–4
46. Kampffmeyer H, Zschaler S (2007) Finding the pattern you need: the design pattern intent ontology. In: Engels G, Opdyke B, Schmidt DC, Weil F (eds) Model driven engineering languages and systems. MODELS 2007. Lecture notes in computer science, vol 4735. Springer, Berlin, Heidelberg, pp 211–225
47. Kruchten P (2004) An ontology of architectural design decisions in software intensive systems. In: 2nd Groningen workshop on software variability. Groningen, The Netherlands, pp 54–61
48. Kruchten P (2004) An ontology of architectural design decisions in software intensive systems. In: 2nd Groningen workshop on software variability, pp 54–61
49. Levandowsky M, Winter D (1971) Distance between sets. Nature 234(5323):34–35
50. Li Z, Liang P, Avgeriou P (2013) Application of knowledge-based approaches in software architecture: a systematic mapping study. Inf Softw Technol 55:777–794
51. Liu C (2010) Ontology-based conflict analysis method in non-functional requirements. In: 9th IEEE/ACIS international conference on computer and information science, IEEE/ACIS ICIS 2010, 18–20 August 2010, Yamagata, Japan, pp 491–496
52. López C, Cysneiros LM, Astudillo H (2008) Ndr ontology: sharing and reusing NFR and design rationale knowledge. In: First international workshop on managing requirements knowledge, 2008. MARK'08. IEEE, pp 1–10
53. Mairiza D, Zowghi D, Nurmuliani N (2009) Managing conflicts among non-functional requirements. In: 12th Australian workshop on requirements engineering. University of Technology, Sydney, pp 11–19
54. Mikkonen T (1998) Formalizing design patterns. In: Proceedings of the 20th international conference on software engineering. IEEE computer society, pp 115–124
55. Montero S, Díaz P, Aedo I (2003) Formalization of web design patterns using ontologies. In: Menasalvas E, Segovia J, Szczepaniak PS (eds) Advances in web intelligence. AWIC 2003. Lecture notes in computer science (Lecture notes in artificial intelligence), vol 2663. Springer, Berlin, Heidelberg, pp 179–188
56. Mylopoulos J, Chung L, Nixon B (1992) Representing and using nonfunctional requirements: a process-oriented approach. IEEE Trans Softw Eng 18(6):483–497
57. Naragani DP, Uniyal P (2013) Comparative analysis of software quality models. Int J Comput Sci Manag Res 2(3):5634–5638
58. Nocera F (2016) Fuzzy ontology-driven web-based framework for supporting architectural design: student research abstract. In: Proceedings of the 31st annual ACM symposium on applied computing. ACM, pp. 1361–1362. https://doi.org/10.1145/2851613.2852014
59. Pan JZ, Staab S, Aßmann U, Ebert J, Zhao Y (2013) Ontology-driven software development. Springer, Berlin
60. Rashwan A, Ormandjieva O, Witte R (2013) Ontology-based classification of non-functional requirements in software specifications: a new corpus and svm-based classifier. In: 37th annual IEEE computer software and applications conference, COMPSAC 2013, Kyoto, Japan, July 22–26, 2013, pp 381–386
61. Rosa NS, Cunha PR, Justo GR (2002) Process NFL: a language for describing non-functional properties. In: Proceedings of the 35th annual Hawaii international conference on system sciences, 2002. HICSS. IEEE, pp 3676–3685
62. Sánchez D, Tettamanzi AG (2006) Fuzzy quantification in fuzzy description logics. Capturing Intell 1:135–159
63. Shaw M, Garlan D (1996) Software architecture: perspectives on an emerging discipline, vol 1. Prentice Hall, Englewood Cliffs
64. Straccia U (2005) Description logics with fuzzy concrete domains. In: Bachus F, Jaakkola T (eds) 21st conference on uncertainty in artificial intelligence (UAI-05). AUAI Press, Edinburgh, Scotland, pp 559–567
65. Straccia U (2013) Foundations of fuzzy logic and semantic web languages. CRC studies in informatics series. Chapman & Hall, Boca Raton
66. Taibi T, Ngo DCL (2003) Formal specification of design patterns—a balanced approach. J Object Technol 2(4):127–140
67. Taylor RN, Medvidovic N, Dashofy EM (2009) Software architecture: foundations, theory, and practice. Wiley, New York

68. Tichy WF (1997) A catalogue of general-purpose software design patterns. In: Proceedings of the technology of object-oriented languages and systems, 1997. TOOLS 23. IEEE, pp 330–339
69. Torra V, Narukawa Y (2007) Information fusion and aggregation operators. Cognitive technologies. Springer, Berlin
70. Tran Q, Chung L (1999) NFR-assistant: tool support for achieving quality. In: Proceedings of the 1999 IEEE symposium on application-specific systems and software engineering and technology, 1999. ASSET'99. IEEE, pp 284–289
71. Zadeh LA (1965) Fuzzy sets. Inf Control 8(3):338–353

Tommaso Di Noia is an Associate Professor of Information Technology Engineering with the Polytechnic University of Bari, Bari, Italy. He has authored several papers in international journals and prominent conferences in his research areas. His current research interests include knowledge representation and automated reasoning, semantic web technologies, mobile and ubiquitous computing, personalized information access, and linked data.

Marina Mongiello is an Assistant Professor at Politecnico di Bari, since 2002. Her recent research interests include: software engineering and software architecture; formal methods and model checking, mobile and distributed reasoning and applications, software architecture for self-adaptive and context-aware software. She is associate editor of the *IET Software Journal*, she has served in organization and in the Program Committee of several international conferences and workshops.

Francesco Nocera is a Ph.D. student with particular interests in Software Engineering and Artificial Intelligence fields. Main research topics are: Information Flow Processing (IFP) Systems, Knowledge Representation and description logics, Knowledge Representation Systems and Applications for the Semantic Web, Self-adaptive Systems and software architecture design. He holds a master's degree in Computer Engineering from Polytechnic University of Bari. He has taken part in the organization and in the Program Committee of conferences and workshops. He has served as a reviewer for different scientific journals.

Umberto Straccia is a researcher at ISTI-CNR (the Istituto di Scienza e di Tecnologie dellInformazione—ISTI, an Institute of the National Research Council of Italy—CNR). He received a Ph.D. in computer science from the University of Dortmund, Germany. His research interests include logics for knowledge representation and reasoning (description logics, logic programming, answer set programming), semantic web languages (OWL, RDFS, RuleML), fuzzy Logic, machine learning, their combination and application.