RASO: an Ontology of Requirements-based Adaptive Systems

Cássio Capucho Peçanha, Bruno Borlini Duarte, Ricardo de Almeida Falbo, Vítor E. Silva Souza

Ontology & Conceptual Modeling Research Group (NEMO)
Department of Computer Science
Federal University of Espírito Santo (UFES) — Vitória, ES, Brazil

cassiocpecanha@gmail.com, bruno.b.duarte@ufes.br
{falbo,vitorsouza}@inf.ufes.br

Abstract. *There is growing interest in software that can adapt their behavior to deal with deviations between their outcome and their requirements at runtime. A systematic mapping of the literature on self-adaptation approaches based on requirements models revealed over 200 papers on this subject. However, there is still a lack of a formal and explicit representation of the concepts in this domain, which can lead to problems in communication, learning, problem-solving and interoperability. To make a clear and precise description of this domain, this paper proposes RASO: the Requirements-based Adaptive Systems Ontology. RASO was built using a well-established Ontology Engineering method, is grounded on a foundational ontology and reuses concepts from other software-related ontologies. The ontology was evaluated by mapping constructs from the most referenced approaches from the literature to its concepts, thus creating a path for interoperability among them.*

1. Introduction

For the past decades, there is growing interest in self-adaptation — systems that are able to modify their behavior/structure in response to the environment, itself and its goals — as a way to manage the ever increasing complexity and the many other challenges involved in the development and management of current software systems [de Lemos et al., 2013]. Some academic efforts in this direction (e.g., [Baresi et al., 2010; Whittle et al., 2010; Dalpiaz et al., 2012; Souza et al., 2011]) focus on Requirements Engineering (RE) for Adaptive Systems, attempting to address what adaptations are possible and how they can be realized [Cheng et al., 2009a]. A systematic mapping of the literature on requirements-based self-adaptation approaches revealed over 200 publications on this subject.

Each of these proposals may use different kinds of models and terms to represent what are the system requirements and prescribe how it should self-adapt in given situations. As a result, the vocabulary used by these methods may be very similar, but the semantics of the entities present in their models are not always the same, thus resulting in problems such as concept overloading: the same name being used to identify things that are ontologically different, i.e., that have different natures and identities in the real world. Other problems, such as construct excess, construct redundancy and incompleteness [Guizzardi et al., 2008] may lead to difficulties in communication, learning, problem-solving and interoperability, especially among requirements engineers and other actors involved in the development of adaptive systems.

An ontology as an artifact, i.e., a "formal specification of a shared conceptualization" [Borst, 1997], can help mitigate these problems and has been attracting interest in the RE community [Dermeval et al., 2016]. In particular, a *domain reference ontology* is a conceptual model developed with the goal of making a clear and precise description of domain entities, representing the consensus within a community. It is a solution-independent specification and, thus, does not maximize computational properties at the cost of truthfulness to the domain [Guizzardi, 2007].

In a previous work, Duarte et al. [2018] proposed a domain reference ontology on the use of *Requirements at Runtime* (RRT), a subject that is at the core of RE for Adaptive Systems [Cheng et al., 2009a]. Hence, we set out to extend this ontology towards the domain of requirements-based adaptive systems, with the purpose of establishing a clear and precise description of this domain.

In this paper, we present the Requirements-based Adaptive Systems Ontology (RASO). RASO was built using a well-established Ontology Engineering method, is grounded on a foundational ontology and reuses concepts from other software-related ontologies. RASO focuses on what an Adaptive System is, its distinguished components and their relationship with requirements of different nature. To establish consensus about the domain, we performed a systematic mapping of the literature on requirements-based approaches for the development of adaptive systems and used selected publications as sources for capturing the ontology's concepts. Finally, the ontology was evaluated by verifying if it satisfied its requirements, expressed as competency questions, and by validating whether its concepts are able to represent real-world entities extracted from well-known approaches in this domain.

This paper is an extended version of [Peçanha et al., 2018] and is structured as follows: Section 2 reports on the systematic mapping of the literature and briefly summarizes our domain of discourse; Section 3 describes the ontological foundations (method and reused ontologies) of RASO; Section 4 presents RASO itself (its requirements and conceptual model); Section 5 explains how RASO was evaluated; Section 6 compares our work to similar proposals in the field; and, finally, Section 7 concludes this paper.

2. Requirements-based Approaches for Adaptive Systems

With the expansion of information, software systems have to deal with issues such as integration of new technologies and continuous evolution, motivating the need for new approaches that support change. This calls for systems that are more flexible, versatile, resilient, recoverable, and self-optimizing, systems which adapt to changing operational contexts, environments or shortcomings in their own operation. Thus, self-adaptation has become an important research topic [Cheng et al., 2009a; de Lemos et al., 2013].

The element that makes self-adaptability possible is usually software. Since adaptation is associated with a specific context, it applies in several application areas and technologies, such as: user interfaces, embedded systems, mobile and autonomous robots and multi-agent systems. However, adequate adaptation is still a challenge and little endeavor has been made to create appropriate software engineering approaches to provide self-adaptation functionality [Cheng et al., 2009a].

When developing self-adaptive systems, engineers should take into account the domain logic and the adaptation logic. Requirements engineering is the first stage in the

Tabela 1. Terms used in the search string.

Id	Group of terms
ρ	"requirements model" OR "requirements engineering" OR "requirements analysis" OR "requirements reasoning" OR "goal model" OR "gore" OR "goal analysis" OR "goal reasoning"
α	"adaptive system" OR "adaptive systems" OR "self-tuning" OR "runtime adaptation" OR "self-adaptive" OR "self-adaptation" OR "self-optimization" OR "self-adaptivity" OR "software adaptation" OR "self-configuration" OR "self-healing" OR "self-protection"

life-cycle of software development and, different from traditional RE, RE for adaptive systems focuses more on defining adaptation logic and addresses what changes in the environment and the system themselves to be monitored, what to adapt, when to adapt and how to adapt [Yang et al., 2014].

In order to elicit and understand the concepts involved in approaches that use requirements models for the development of adaptive systems, we conducted a systematic mapping of the literature [Kitchenham and Charters, 2007; Petersen et al., 2015] and analyzed the state-of-the-art in this particular field. The main goal was to identify the concepts used in requirements models by different approaches with the purpose of developing the Requirements-based Adaptive Systems Ontology (RASO). In this section, we briefly describe the protocol and results of this systematic mapping, but a complete description is available in `https://goo.gl/66ve9e`.

In a previous work, Duarte et al. [2018] performed a systematic mapping study with the purpose of developing the Runtime Requirements Ontology (RRO, cf. Section 3), thus with a broader scope (any use of requirements models at runtime). Such mapping classified publications by purpose of use of requirements at runtime, separating them in two major categories: *Monitor Requirements*, which propose checking if requirements defined at design time are being achieved at runtime, and *Change Requirements*, in which requirements are used not only as guidelines to monitoring but also as rules on how the system should adapt to keep satisfying its requirements.

We based our mapping protocol on these previous results by considering the publications in the *Change Requirements* category as control papers, i.e., publications that our search string should return. We also considered the same set of publication sources: ACM Digital Library, Engineering Village, IEEE Xplore, Science Direct and Scopus. Based on the requirements for RASO, research questions were defined for the systematic mapping and the search string was thus elaborated in an iterative process, by testing it in these sources and checking if all control papers returned, otherwise refining it with key terms present in the meta-data (title, abstract, keywords) of the missing control articles. The final string is the conjunction of the two groups of terms shown in Table 1, i.e., $\rho \wedge \alpha$.

Once we had a satisfactory search string, we proceeded with the systematic mapping, i.e., collected search results, merged duplicate entries and applied inclusion/exclusion criteria. Figure 1 provides an overview of this process: a total of 784 entries returned from the different publication sources, resulting in 385 articles after duplicates (same paper returning in different sources) were eliminated. We then excluded

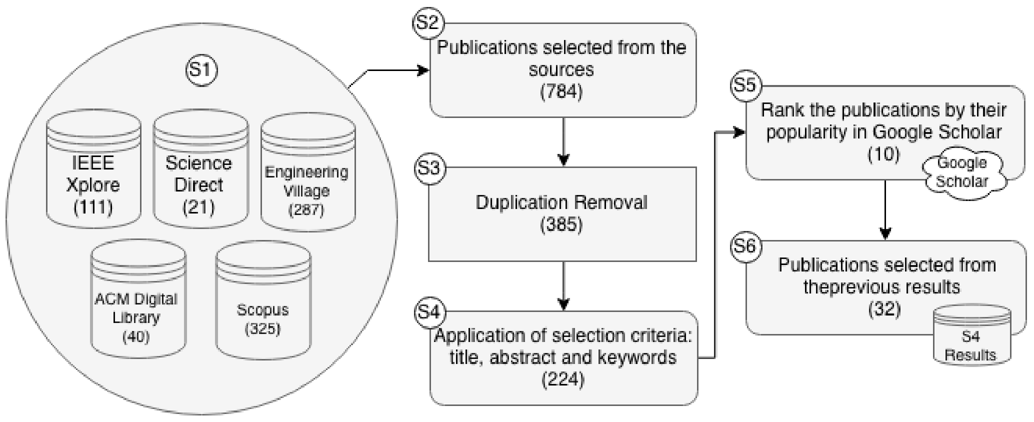
S1
IEEE Xplore (111)
Science Direct (21)
Engineering Village (287)
ACM Digital Library (40)
Scopus (325)
S2
Publications selected from the sources (784)
S3
Duplication Removal (385)
S4
Application of selection criteria: title, abstract and keywords (224)
S5
Rank the publications by their popularity in Google Scholar (10)
Google Scholar
S6
Publications selected from theprevious results (32)
S4 Results

Figura 1. Overview of the systematic mapping protocol for RASO.

161 publications that either (1) did not provide an abstract; (2) provided only an extended abstract; (3) was not written in English; (4) was an older copy or version of another selected publication; (5) was a secondary study, a tertiary study, a summary, a tutorial or an editorial; or (6) whose complete version (PDF file) was unavailable. Finally, we considered as inclusion criterion if the paper presented any kind of requirements model for adaptive systems, which excluded other 57 papers. Selection criteria were applied in two phases — first considering only meta-data (title, abstract and keywords), then reading the full paper — resulting in 224 selected publications.

Due to the large number of selected publications, we proposed a new filter in order to select a smaller set publications which would be studied as part of requirements elicitation and ontology capture for RASO. Given that an ontology should provide a conceptualization that is shared by a certain community, we ranked the publications by their popularity (citations), as follows: (1) the total number of citations was extracted from Google Scholar by searching for the publication title; (2) the age of each publication was calculated by subtracting the publication year from the current year; (3) the number of citations was divided by the age, normalizing to citations per year; (4) the publications were ranked from most cited to least cited and the top 10 most cited papers were selected.

These papers represented 8 different requirements-based approaches for the development of adaptive systems, namely: Adaptive STS, FLAGS, LoREM, Necesity, QoS-Aware Middleware, RELAX, Tropos4AS and Zanshin. Finally, we searched the complete list of (224) publications returned in the systematic mapping for other papers addressing one of these selected approaches, in order to provide a more complete content for their comprehension. The search was performed using the name of the authors of the top 10 articles and returned 22 new publications to be analyzed (8 for Zanshin, 8 for RELAX, 3 for Tropos4AS and 3 for FLAGS). The final set of 32 publications was used not only for building RASO (cf. Section 4), but also for validating it (cf. Section 5). In what follows, we briefly summarize each of the approaches.

2.1. Adaptive STS

Dalpiaz et al. [2012] propose an architecture that, based on requirements models, adds self-reconfiguring capabilities to a system using a monitor-diagnose-reconcile-compensate (MDRC) loop. A monitor component collects, filters and normalizes events/logs from the system, which serve as input to the diagnose component, responsible for identifying failures and discovering their root causes. Finally, the reconfigurator component selects, plans and deploys compensation actions in response to failures.

Adaptive-STS targets socio-technical systems (STSs), which consist of an interplay of humans, organizations, and technical systems, with an operational environment that can change unexpectedly. The approach extends Tropos [Bresciani et al., 2004] and uses goal models to represent requirements, then propose different algorithms for system reconfiguration at runtime. For instance, one such algorithm finds all valid variants to satisfy a goal and compares them based on their cost (to compensate tasks that failed or the ones that already started and will be canceled) and benefit (e.g., contribution to softgoals).

2.2. FLAGS

Based on KAOS [van Lamsweerde et al., 1991], the FLAGS approach [Baresi et al., 2010] proposes crisp (Boolean) goals (specified in Linear Temporal Logic) whose satisfaction can be easily evaluated, and fuzzy goals that are specified using fuzzy constraints, mostly associated with non-functional requirements. The key difference between crisp and fuzzy goals is that the former are firm requirements, while the latter are more flexible.

To provide semantics for fuzzy goals, FLAGS includes fuzzy relational and temporal operators. Whenever a fuzzy membership function is introduced in FLAGS, its shape must be defined by considering the preferences of stakeholders. This specifies exactly what values are considered to be "around" the desired value. Additionally, in FLAGS, adaptive goals define countermeasures to be executed when goals are not satisfied, using Event-Condition-Action rules. The approach allows for the definition of adaptive goals which execute a set of adaptation actions that can change the system's goal model in different ways — add/remove/modify goals or agents, relax a goal, etc. — and in different levels — in transient or permanent ways.

2.3. LoREM

Goldsby et al. [2008] propose the LoREM approach, which defines a systematic processes for performing (Goal-Oriented) Requirements Engineering for adaptive systems. Its name comes from the work of Berry et al. [2005], who defined four *Levels of RE for Modeling* adaptive systems — Level 1: system developers identify the goals of the system and the steady-state systems that are suitable for the domains that satisfy the goals; Level 2: adaptation scenario developers creates the set of adaptation scenarios, which represent the run-time transitions between source and target systems, including the requirements for monitoring, decision-making and adaptation; Level 3: concerned with identifying the adaptation infrastructure necessary to support the previously identified scenarios; Level 4: research done to improve the methods and techniques used in the other levels.

LoREM models requirements for Dynamically Adaptive Systems (DASs) using i* [Yu et al., 2011] goal models, working on the aforementioned four levels, to represent stakeholder objectives, non-adaptive and adaptive system behavior and adaptation mechanism needs for a DAS. The approach offers an application-driven process that assumes a mature set of adaptation mechanisms that enable a DAS to dynamically adapt.

2.4. The Necesity approach

Necesity [Botia et al., 2012] is an Ambient Assisted Living system that has been designed to monitor elders which live alone and want to keep living independently. A system capable of detecting falls or similar problems by means of sensor networks with distributed data in oder to acquire data regarding elderly home activity. Context-aware techniques process the sensor data, providing pictures of the context of the elder. One of the main contributions is related with the capability of the system to adapt its behavior to that of the monitored elder based on a set of decision rules. In Necesity, the adaptability takes into account the behavioral patterns of the monitored, using data analysis techniques to allow the system to adjust its performance.

2.5. QoS-aware Middleware

QoS-aware middleware [Nahrstedt et al., 2001] is a solution to provide quality of service (QoS) support, which is required by a new generation of QoS-sensitive applications. It presents four key aspects of a QoS-aware middleware system: QoS specification (responsible for description of application behavior and QoS parameters), QoS translation and compilation (responsible for translating specified application behavior into candidate application configurations for different resource conditions), QoS setup (responsible for appropriately selecting and instantiating a particular configuration) and QoS adaptation (responsible for adapting to runtime resource fluctuations).

The operation of QoS-aware middleware systems is defined as: first, generating appropriate QoS specifications; second, translating and compiling multiple application configurations for the same application to be run in heterogeneous environments; then followed by a selection of an appropriate configuration and discovering the participating application components; and, finally, adapting QoS at multiple levels.

2.6. RELAX

The RELAX language [Sawyer et al., 2010; Whittle et al., 2010] aims at capturing uncertainty declaratively with modal, temporal, ordinal operators and uncertainty factors provided by the language, capturing uncertainty in the way requirements can be met, mainly due to environmental factors. Unlike goal-oriented approaches, it assumes that structured natural language requirements specifications, containing the SHALL statements that specify what the system must do, are available before their conversion to RELAX specifications. The modal operators available, SHALL and MAY. . .OR, specify, respectively, that requirements must hold or that there exist requirements alternatives (variability).

In RELAX, points of flexibility/uncertainty are specified declaratively, thus allowing designs based on rules, planning, etc. as well as to support unanticipated adaptations. Some requirements are deemed invariant — they need to be satisfied no matter what. Other requirements are made more flexible in order to maintain their satisfaction by using “AS POSSIBLE”-type RELAX operators (e.g., “AS EARLY AS POSSIBLE”, “AS CLOSE AS POSSIBLE”, etc.). Because of these, RELAX needs a logic with built-in uncertainty to capture its semantics. The authors chose Fuzzy Branching Temporal Logic for this purpose. It is based on the idea of fuzzy sets, which allows gradual membership functions. Temporal operators such as EVENTUALLY and UNTIL allow for temporal component in requirements specifications in RELAX.

Cheng et al. [2009b] integrate LoREM (Section 2.3) and RELAX, adding to the mix an approach to systematically explore the uncertainty form the environment to which the adaptive system will be deployed using threat modeling in KAOS. When a goal threat is identified, there are three possible mitigation strategies that can be applied: (a) add subgoals to handle the condition of the threat; (b) use RELAX to add flexibility to the goal definition; (c) create new high-level goals that capture the objective of correcting the failure. The last strategy works like a feedback loop that adapts the system whenever the goal fails at runtime.

2.7. Tropos4AS

In Tropos4AS (Tropos for Adaptive Systems), Morandini et al. [2017] propose extensions to the architectural design phase of Tropos to model adaptive systems based on the Belief-Desire-Intention (BDI) model as a reference architecture [Rao and Georgeff, 1995]. The approach introduces new goal types — namely, maintain-goals, achieve-goals and perform-goals — and a new *inhibit* relation between goals that specifies that a goal (the inhibitor) has to be stopped in order for another goal (the inhibited) to be achieved/maintained.

Tropos4AS also allows designers to model non-intentional elements using UML[1] class diagrams, specifying resources that belong to an agent and the ones that belong to the environment. The approach also allows for the modeling of undesirable (faulty) states, which are known to be possible at runtime and should trigger system adaptation. At the operational level, goal models are mapped to the Jadex[2] platform for run-time implementation.

2.8. Zanshin

Zanshin [Souza et al., 2011; Souza, 2012] is an RE-based framework created for the development of adaptive systems. Its main idea is to make elements of the feedback loop responsible to provide adaptivity important entities in the requirements models. To do that, Zanshin represent system requirements using the concepts of the Core Ontology for Requirements Engineering (CORE) [Jureta et al., 2008, 2009], but with an i*-like syntax.

Its models are augmented with new elements, called Awareness Requirements (AwReqs) — requirements that refer to the state of other requirements of a software system at runtime, i.e., responsible for representing the parts of the system that the stakeholders want it to be able to adapt — and Evolution Requirements (EvoReqs) — requirements that describe how other requirements are supposed to adapt/evolve in response to an AwReq failure, i.e., they act directly over system requirements through a set of adaptation strategies. When combined with classic GORE constructs (goals, softgoals, tasks, AND-OR refinements, and so on), these new elements are able to represent monitoring and adaptation strategies during the execution of a software system.

Figure 2 represents the requirements model of a Meeting Scheduler system created as a proof-of-concept for the Zanshin framework and used in Section 4 to illustrate the concepts of RASO. AwReqs appear in the Meeting Scheduler requirements model as small bold circles with arrows, pointing out to the elements of the system that need to

[1]The Unified Modeling Language, `http://www.uml.org/`.

[2]A BDI Agent System, `http://jadex-agents.informatik.uni-hamburg.de/`.

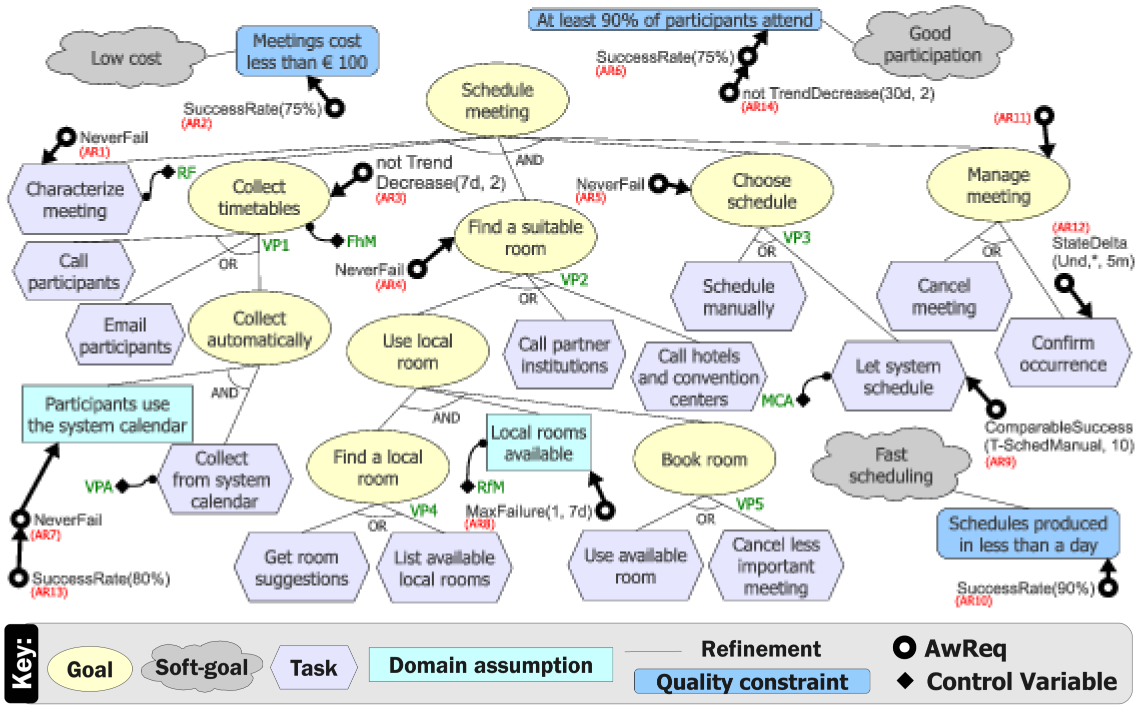

Figura 2. Requirements model of a Meeting Scheduler system, created with the Zanshin approach [Souza, 2012]

have their states monitored. EvoReqs are not graphically represented in the requirements model, however, they are implemented as a code artifact that will be executed by Zanshin when an AwReq fails. For instance, if, for some reason, a meeting cannot be properly scheduled in the system, AR1 (*NeverFail*, at the top-left corner of the model) will trigger the EvoReq *Retry Characterize Meeting*, that will make the system perform a rollback, wait for 5 seconds, and then try to schedule the meeting again.

3. Ontological Foundations

The Requirements-based Adaptive Systems Ontology (RASO) was built using SABiO: a Systematic Approach for Building Ontologies [Falbo, 2014], a well-established Ontology Engineering method. We chose SABiO because it is focused on the development of domain ontologies and has been successfully used for developing several ontologies in the Software Engineering domain (e.g., [Bringuente et al., 2011; Ruy et al., 2016]).

SABiO's development process is composed of five phases — (1) purpose identification and requirements elicitation; (2) ontology capture and formalization; (3) design; (4) implementation; and (5) testing — accompanied by widely accepted support processes, such as knowledge acquisition, reuse, documentation and evaluation. The first two phases of SABiO's development process aim to produce a reference domain ontology. Since we were interested in developing a reference ontology, we performed the first two phases of the method.

At the first phase, requirements for the ontology are elicited in the form of Competency Questions (CQs), which are questions that the ontology should be able to answer [Grüninger and Fox, 1995]. In the following phase, relevant concepts and relations for the domain should be identified and a conceptual model should be built. Requirements elicitation and ontology capture for RASO are presented in Section 4.

SABiO explicitly recognizes the importance of using foundational ontologies in the ontology development process to improve the ontology quality, representativeness and formality. Thus, we grounded RASO in the Unified Foundational Ontology (UFO) [Guizzardi et al., 2008], a well-established foundational ontology. UFO offers a comprehensive set of categories to tackle the specificities of the targeted domain and, as SABiO, it has been successfully employed in the development of several ontologies in the Software Engineering domain (e.g., [Bringuente et al., 2011; Ruy et al., 2016]), including the ones that were reused/extended in this work.

SABiO also suggests that existing ontologies on related domains should be reused. The Software Engineering Ontology Network (SEON) [Ruy et al., 2016] includes ontologies that are related to the target domain of this work. Thus, these ontologies were reused by RASO, facilitating its integration into SEON. The set of reused ontologies include: the Software Ontology (SwO), the Reference Software Requirements Ontology (RSRO) and the Runtime Requirements Ontology (RRO) [Duarte et al., 2018], all of which are connected to SEON's core ontology: the Software Process Ontology (SPO) [Bringuente et al., 2011]. We decided to reuse concepts from these ontologies as their domains are intrinsically related to the domain of requirements-based adaptive system. Moreover, they are all grounded in UFO, guaranteeing compatibility at the foundational level. Among these ontologies, RRO has a particular prominent role, given that it describes the use of requirements artifacts at runtime, which is key to adaptive systems. Given how close RASO is of RRO, we can say that the former is an extension of the latter.

Figure 3 shows the SEON view presenting the concepts reused by RASO, using a UML class diagram primarily for visualization (for an in-depth discussion and a more formal characterization, refer to [Duarte et al., 2018]). As shown, these concepts derive from the notion of Artifact from SPO, i.e., an Object (in the sense of UFO) intentionally made to serve a given purpose in the context of a software product or organization. We are particularly interested in three types of Artifacts: Software Items — pieces of software produced during the software process —, Documents — any written or pictorial information related to the software development, usually presented in a predefined format — and Information Items — any relevant information produced during the software process for human use.

As defined in SwO [Duarte et al., 2018], a Software System is a Software Item that *intends to implement* a System Specification, which is a Document containing a set of requirements for a system, defining its desired functions and features in an abstract way, without constraining its behavior. Software Systems are *constituted of* Programs. A Program is a Software Item which aims at producing results through execution on a computer, as prescribed by the Program Specification, a Document that describes the structure and functions of a Program. Finally, a Program is *constituted of* Code, a Software Item representing a set of computer instructions and data definitions expressed in a programming language. A Program is *constituted by* Code, but it is not identical to Code. Code can be changed (e.g., to fix a bug) without altering the identity of its Program, which is anchored to the program's essential property: its intended Program Specification, which the Code *implements*.

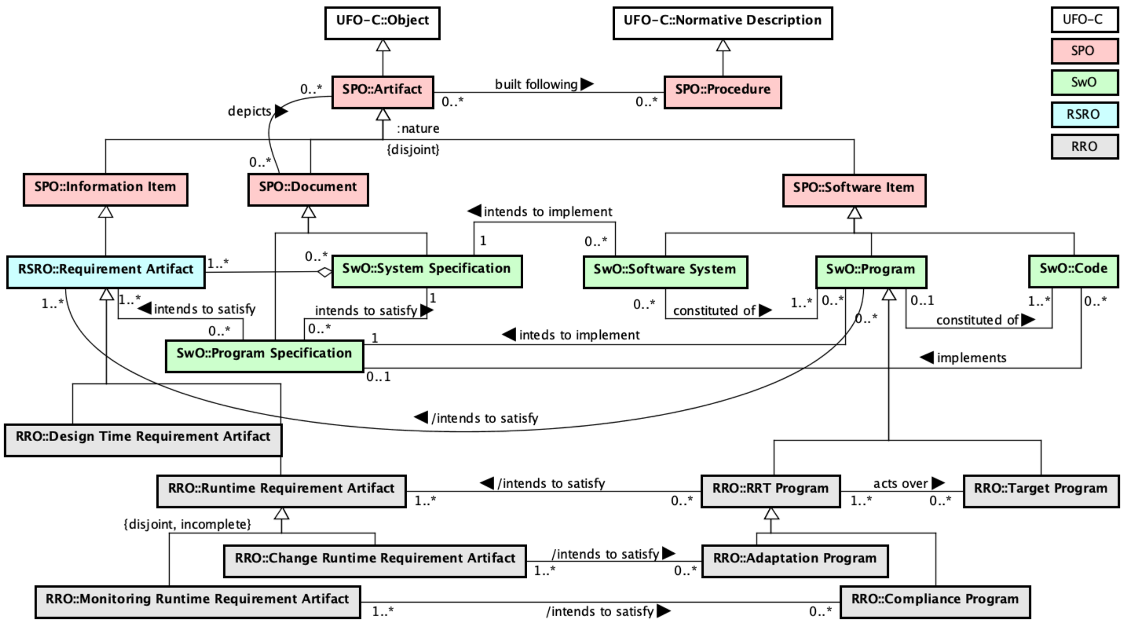

Figura 3. SEON's view showing concepts and relations reused by RASO.

Given our focus on requirements-based adaptive systems approaches, we reuse from RSRO the concept of Requirement Artifact, an Information Item that describes a stakeholder's requirement (concepts also present in RSRO, but out of scope here), most likely as the result of some requirements documentation activity in the Requirements Engineering process. In the context of adaptive systems, we are particularly interested in a special type of Requirement Artifact specified in RRO, namely, a Runtime Requirement Artifact (RRA). Unlike Design Time Requirement Artifacts, RRAs are manipulated by running Programs, i.e., at runtime. Such Programs are said RRT (Requirements at Runtime) Programs, which *intend to satisfy* one or more RRAs, *acting over* other running Programs, referred to as Target Programs. RRO specifies two particular types of use of requirements at runtime: Compliance Programs *intending to satisfy* Monitoring RRAs and Adaptation Programs *intending to satisfy* Change RRAs.

4. Requirements-based Adaptive Systems Ontology

In this section, we present the Requirements-based Adaptive Systems Ontology (RASO), a domain ontology about requirements-based approaches for adaptive systems. Following the first steps of SABiO, we defined the purpose of RASO: to be a formal and explicit representation of the concepts of adaptive systems based on requirements models. Moreover, we identified the following intended uses for RASO:

1. To serve as conceptual basis to solve interoperability problems among different requirements-based approaches for the development of adaptive systems, thus serving as a well-founded vocabulary to improve understanding and knowledge sharing in the domain of requirements for adaptive systems, being especially helpful for requirements engineers working in this field;
2. To serve as a supporting tool for the creation or reengineering of requirements-based approaches for the development of adaptive systems;

3. To serve as conceptual model for the creation and/or integration of tools supporting the intended uses described in (1) and (2) above.

Once its purpose and intended uses were defined, requirements for RASO were elicited and documented in the form of Competency Questions (CQs) through a highly iterative process of requirements elicitation/documentation and ontology capture/formalization. The CQs for RASO are:

- **CQ1:** What is an adaptive system?
- **CQ2:** What are the components of an adaptive system?
- **CQ3:** How are adaptations performed in an adaptive system?
- **CQ4:** How are adaptive systems specified?
- **CQ5:** What requirements does the adaptive system handle at runtime?

Figure 4 shows the conceptual model of RASO, relating concepts of its domain to the ones reused from the Software Engineering Ontology Network (SEON, cf. Section 3). Next, we show how RASO provides a clear and precise description of this domain.

An Adaptive System is a special kind of Software System, *constituted of* one or more Adaptive Feedback Loop Control Programs (*feedback loop* or *controller*) and *base programs* (often referred to in the literature as *target systems* [Souza, 2012]), represented in RASO as Adaptable Target Program. Once the *base program* is integrated into a *feedback loop* (in other words, it is made adaptive using any given approach for the development of adaptive systems [Souza and Mylopoulos, 2011]), it becomes an Adaptive System. Hence, the programs that need to perform some adaptation will be the Adaptable Target Programs. Considering the Zanshin approach (cf. Section 2.8), for example, this integration consists of implementing a callback API specified by Zanshin.

As Programs, both Adaptable Target Program and Adaptive Feedback Loop Control Program *intend to implement* a Program Specification in order to *satisfy* Requirement Artifacts. The former implements what we informally call the *main functions* of the system, e.g., schedule meetings, whereas the latter implements *adaptation functions*, i.e., monitor the main functions and adapt them if needed, forming a *feedback loop* (e.g., the MAPE loop [Kephart and Chess, 2003]). For instance, if a self-adaptive Meeting Scheduler (adaptive system composed of base program and controller program) detects a low participation rate in meetings, it could send e-mails to invited participants to properly collect their timetables.

Software Systems *intend to implement* a System Specification [Duarte et al., 2018]. As such, an Adaptive System *intends to implement* an Adaptive System Specification. In particular, besides specifying the main functions of the system, an Adaptive System Specification *is composed of* one or more Runtime Requirement Artifacts (RRAs) regarding the system's self-adaptation functions. This particular kind of specification is *built following* an Adaptive System Design Framework, which SPO [Bringuente et al., 2011] considers a Procedure. For instance, in [Souza, 2012], the author presents an Adaptive System Specification for a Meeting Scheduler Adaptive System, based on the Zanshin Adaptive System Design Framework, containing *Awareness Requirements* (Monitoring RRAs and Analyze RRAs) to monitor some of the Meeting Scheduler's functions (e.g., low participation in scheduled meetings) and *Evolution Requirements* (Plan RRAs and Change RRAs) to adapt these functions if necessary (e.g., start collecting timetables properly via e-mail).

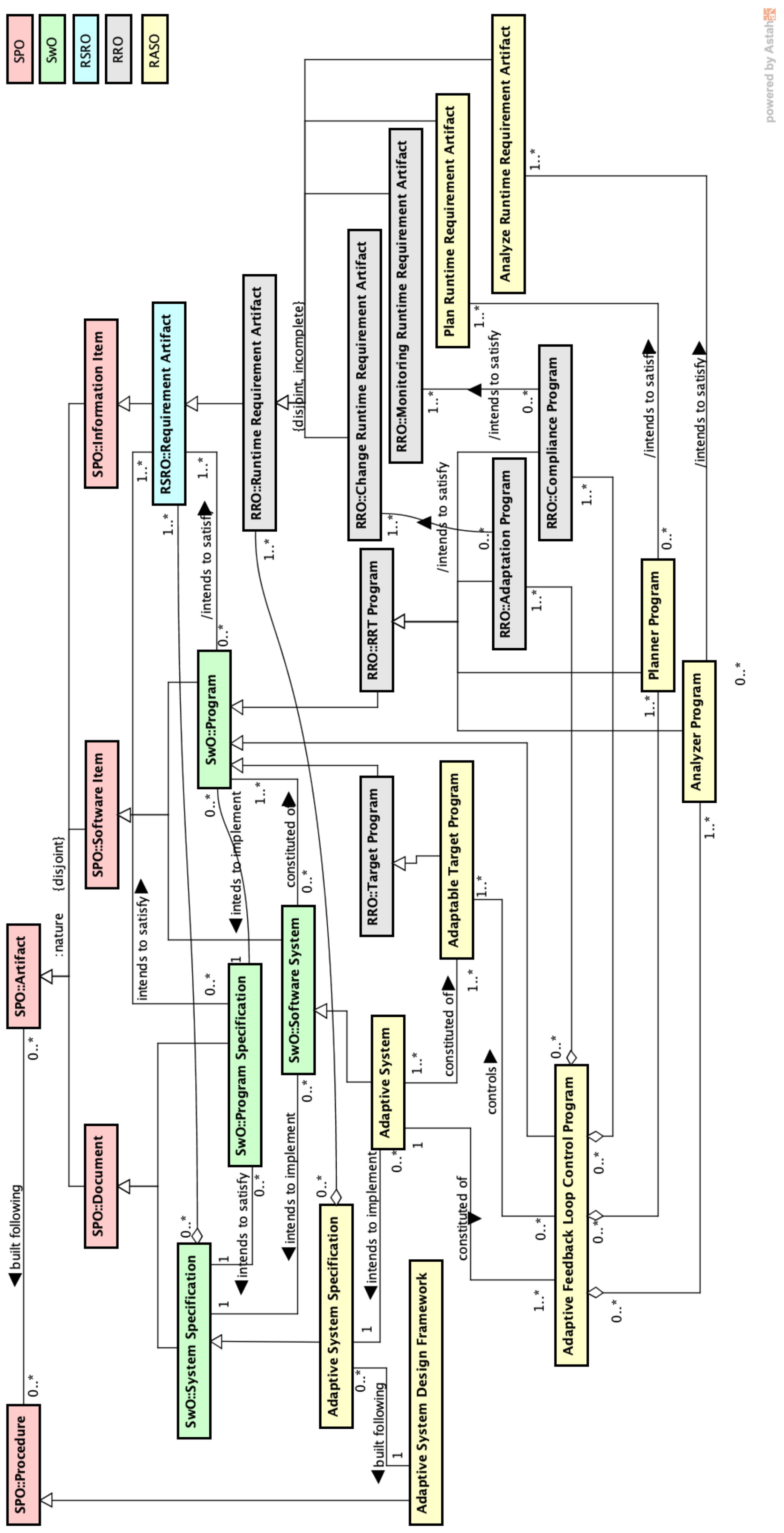

Figura 4. The Requirements-based Adaptive Systems Ontology (RASO).

Finally, the self-adaptation functions in the Adaptive System Specification are implemented by the Adaptive System by means of its Adaptive Feedback Loop Control Programs, which are *constituted by* four particular kinds of RRT (Requirements at Runtime) Programs: Compliance Programs *intend to satisfy* the Monitoring RRAs in the specification, Analyzer Programs *intend to satisfy* the Analyze RRAs in the specification, Planner Programs *intend to satisfy* the Plan RRAs in the specification and Adaptation Programs *intend to satisfy* the Change RRAs in the specification. Again, using Souza's work as example [Souza, 2012], the Zanshin approach provides an implementation of an Adaptive Feedback Loop Control Program based on the Eclipse[3] platform and constituted by OSGi[4] bundles, among which the *Monitoring Bundle* (Compliance Program and Analyzer Program) monitors the satisfaction of *Awareness Requirements* and reports to the *Adaptation Bundle* (Planner Program and Adaptation Program), which analyze *Evolution Requirements* in order to adapt.

5. Evaluation

In order to evaluate the Requirements for Adaptive Systems Ontology (RASO), we applied verification and validation techniques, as prescribed by SABiO. For verification, SABiO suggests a table that shows the ontology elements that are required to answer the competency questions (CQs) that were raised, thus demonstrating that the ontology satisfies the requirements as documented. For validation, the ontology should be instantiated using real-world entities, demonstrating that it is able to represent real-world situations.

Table 2 shows how each of the predefined CQs are answered by concepts and relations of RASO. This table can also be used as a traceability tool, supporting ontology change management. It is worthwhile to pointy out that all concepts of RASO are mentioned in Table 2. This shows they are both necessary and sufficient to fulfill the ontology's requirements.

For validation, we studied the 32 publications about the 8 different approaches selected in our systematic mapping of the literature (cf. Section 2). Then, we identified instances of concepts from RASO (and a few key concepts of RRO as well) in each of the approaches, producing the instantiation table divided in tables 3 and 4. Note that an m-dash (—) in a cell does not mean that the concept is not present at all in the respective approach, but instead that it is not explicitly named.

The successful instantiation of RASO with entities from these approaches is an indication of the appropriateness of the proposed ontology as a reference model of this domain. Tables 3 and 4 also show that RASO, as a conceptual model, does not present the problems discussed by Guizzardi [Guizzardi et al., 2008]:

- *Construct Overload*: the concepts of RASO represent a single entity in the approaches, i.e., they are not overloaded with meaning;
- *Construct Excess*: all concepts of RASO have instances in most of the approaches, i.e., there is no unnecessary concepts in the ontology with respect to the domain;
- *Construct Redundancy*: no two concepts of RASO point to the same entity in any of the approaches, i.e., the concepts of the ontology are not redundant;

[3]`http://www.eclipse.org`
[4]`http://www.osgi.org`

Tabela 2. Verification of RASO's Competency Questions.

CQ	Concepts and *Relations*
CQ1 - What is an adaptive system?	Adaptive System *subtype of* Software System;
	Adaptive System *intends to implement* Adaptive System Specification.
CQ2 - What are the components of an adaptive system?	Adaptive System *subtype of* Software System;
	Software System *constituted of* Program;
	Adaptive System *constituted of* Adaptable Target Program and Adaptive Feedback Loop Control Program;
	Adaptable Target Program *subtype of* Target Program *subtype of* Program;
	Adaptive Feedback Loop Control Program *subtype of* Program.
CQ3 - How are adaptations performed in an adaptive system?	Adaptive Feedback Loop Control Program *controls* Adaptable Target Program.
CQ4 - How are adaptive systems specified?	Adaptive System Specification *subtype of* System Specification;
	Runtime Requirement Artifact *part of* Adaptive System Specification;
	Adaptive System Specification *built following* Adaptive System Design Framework.
CQ5 - What requirements does the adaptive system handle at runtime?	Adaptive System *constituted of* Adaptable Target Program and Adaptive Feedback Loop Control Program;
	{Compliance Program, Analyzer Program, Planner Program, Adaptation Program} *part of* Adaptive Feedback Loop Control Program;
	Compliance Program *intends to satisfy* Monitoring RRA;
	Analyzer Program *intends to satisfy* Analyze RRA;
	Planner Program *intends to satisfy* Plan RRA;
	Adaptation Program *intends to satisfy* Change RRA;
	{Monitoring RRA, Analyze RRA, Plan RRA, Change RRA} *subtype of* Runtime Requirement Artifact.

- *Incompleteness*: the entities related to the domain of adaptive systems in the different approaches were all mapped to concepts of RASO, i.e., the ontology is complete with respect to its given scope.

As such, it can serve as conceptual basis to solve interoperability problems between different approaches for the development of adaptive systems — tables 3 and 4 even serve as starting point for mapping concepts among the 8 selected approaches. RASO can also be used to develop or integrate existing software tools in this domain, as done, e.g., in the domain of software measurement [Fonseca et al., 2017].

Another interesting use of the ontology is to perform ontological analysis of the modeling languages used in these approaches, offering recommendations to clarify their semantics and ensure expressiveness, as done, for instance, in the domain of service-oriented enterprise architectures [Nardi et al., 2014]. Tables 3 and 4 even provide us with hints regarding the 8 selected approaches, as they could also be evaluated with respect to lucidity, soundness, laconicity and completeness [Guizzardi et al., 2008], as we did with RASO above. For instance, we were not able to instantiate any of the Program concepts for the RELAX approach, as the papers define the specification language, but do not provide an implementation. As another example, the QoS-aware Middleware approach does not make explicit their Monitoring Runtime Requirement Artifact. Such feedback could help improve these approaches.

In general, the evaluation of RASO shows that it can act as a well-founded vocabulary of the domain of requirements-based adaptive systems design, improving communication, learning and problem-solving in this domain.

Tabela 3. Validation of RASO by instantiating its concepts according to popular approaches for the development of adaptive systems (part 1).

Concept	**Approach for the Development of Adaptive Systems**			
Adaptive System Design Framework	**Adaptive STS**	**FLAGS**	**LoREM**	**Unnamed approach applied to Necesity**
Adaptive System	Combination of the Smart Home example with the adaptation controller	Combination of the Laundry System example with the adaptation controller	The GridStix system	The Necesity system and the Context Management Middleware
Adaptive System Specification	Plan specifications for the Smart Home example	Goal model specification for the Laundry System example	Goal model specification for GridStix	Adaptive timers algorithm
Adaptable Target Program	Smart Home example implementation	Laundry System example implementation	GridStix's steady-state system implementation	The Necesity system implementation
Adaptive Feedback Loop Control Program	Self-Reconfiguration component (Monitor-Diagnose-Reconcile-Compensate cycle)	FLAGS framework implementation	Adaptation infrastructure	Context Management Middleware
Analyze Runtime Requirement Artifact	—	Condition (Adaptive Goal)	—	—
Analyzer Program	—	Data Collector	—	—
Plan Runtime Requirement Artifact	—	Objective (Adaptive Goal)	—	—
Planner Program	—	Monitor	—	—
Compliance Program	Monitor component	Monitor	Monitoring mechanism	Monitoring software
Monitoring Runtime Requirement Artifact	Fulfillment conditions	Triggers (Adaptive Goal)	Decision-making mechanism	Set of decision rules
Adaptation Program	Reconfiguration component	Monitor	Adaptation mechanism	Adapter component
Change Runtime Requirement Artifact	Activation rules	Actions (Adaptive Goal)	Adaptive step	Parameters which are set based on the history of use of the system

6. Related Work

Qureshi et al. [2011] proposed a new version of the Core Ontology for Requirement Engineering (CORE) [Jureta et al., 2008, 2009], introducing two new concepts (*context* and *resource*) and relations (*relegation* and *influence*) on top of CORE (which concerns Requirements Engineering in general) in order to properly represent possible changes that might occur in requirements, at runtime. The authors claim that combining the new elements with the ones that are originally used by the goal modeling language Techne [Jureta et al., 2010], they are able to support the definition of the runtime requirements adaptation problem. In comparison with RASO, the ontology of Qureshi et al. [2011] does not represent concepts that were found in the approaches selected in our systematic mapping of the literature, such as Monitoring RRA and Change RRA, for example. Instead, it includes concepts and relations that are not strongly related to this domain (i.e., were not commonly found in the selected approaches), such as *resource* and *relegation*.

Soares et al. [2016] propose a core ontology to assist requirements elicitation and specification for adaptive systems, based on the ontology of Qureshi et al. [2011] — which they deemed incomplete —, adding concepts extracted from the modeling dimensions for

Tabela 4. Validation of RASO by instantiating its concepts according to popular approaches for the development of adaptive systems (part 2).

Concept	Approach for the Development of Adaptive Systems			
Adaptive System Design Framework	**QoS-aware Middleware**	**RELAX**	**Tropos4AS**	**Zanshin**
Adaptive System	Combination of a video streaming application with the QoS Middleware	—	Combination of the iCleaner example with the Tropos4AS Middleware	Combination of the Meeting Scheduler example with the Eclipse/OSGi-based implementation
Adaptive System Specification	Qos Specification	Goal model for the Ambient Assisted Living example	Goal model for the iCleaner example	Goal model for the Meeting Scheduler example
Adaptable Target Program	A video streaming application	Ambient Assisted Living (ALL) example	iCleaner example implementation	Meeting Scheduler example implementation
Adaptive Feedback Loop Control Program	QoS Middleware (QoS-aware resource management)	—	Tropos4AS Middleware (MAPE Loop)	The Eclipse/OSGi-based implementation (MAPE Loop)
Analyze Runtime Requirement Artifact	QoS parameter	Specification using REL operator	Failure Conditions or Drop Conditions	—
Analyzer Program	QoS translation and compilation	—	—	—
Plan Runtime Requirement Artifact	—	—	Plan	—
Planner Program	QoS setup	—	—	—
Compliance Program	Observer component	—	Monitoring BDI agent	Monitoring OSGi bundle
Monitoring Runtime Requirement Artifact	—	Specification using MON operator	Satisfaction conditions	Awareness Requirement
Adaptation Program	QoS adaptation	—	Adaptation BDI agent	Adaptation OSGi bundle
Change Runtime Requirement Artifact	Application adaptation policies	Relaxed requirements	Recovery activities	Evolution Requirement

adaptive systems [Andersson et al., 2009]. Besides inheriting the aforementioned problems of [Qureshi et al., 2011], the ontology is not properly based on a foundational ontology, is presented only in its operational form (in OWL) and includes concepts pertaining to the area of context-aware systems (claiming they subsume adaptive systems, a claim which we find debatable). On the other hand, RASO is a reference ontology, founded on UFO and focused exclusively on the concepts of the adaptive systems domain.

Reinhartz-Berger et al. [2013] propose a conceptual model of software behavior based on the foundational ontology by Bunge [1977], which can be used to model the expected behavior of a system (its requirements) and alternative behaviors that the running program can actually perform, thus supporting self-adaptation decisions by comparing alternatives on: (1) how well they meet the requirements and (2) the effort needed to switch behaviors. Their work uses an ontological approach to model the system's behavior in order to foster self-adaptation, i.e., it is a requirements-based approach for the development of adaptive systems. Our work instead provides an ontology for the domain of requirements for adaptive-systems, i.e., it describes concepts used by different approaches (including [Reinhartz-Berger et al., 2013], which could be instantiated like the works included in tables 3 and 4).

Finally, regarding the systematic mapping study we present in Section 4, Yang et al. [2014] performed a systematic literature review to investigate what modeling methods, activities, quality attributes, application domains and research topics have been studied in the area of requirements engineering for adaptive systems and how well these studies have been conveyed. Our mapping, although more superficial than a review, bears some similarities with their work, such as the research method [Kitchenham and Charters, 2007]. We did consider the possibility of using their study as the knowledge base for building RASO instead of performing a new one. However, since this systematic mapping is already five years old, we decided that a new study was needed in order to not neglect the recent advances that were achieved in this research area. Moreover, updating their review would have taken much more effort than producing a new mapping.

Our systematic mapping study showed that a almost a fifth of the requirements-based approaches for the development of adaptive systems make use of ontologies as part of their proposed methods. Among them, CORE [Jureta et al., 2008, 2009] is the most cited work. However, as mentioned before, CORE represents the domain of Requirements Engineering in general, whereas RASO focuses on Requirements Engineering for adaptive systems (based on RSRO [Duarte et al., 2018] for the reasons discussed in Section 3). This shows both a growing interest in the use of ontologies in this domain and a gap for ontologies that are specifically tailored for adaptive systems, which RASO intends to fill.

7. Conclusions

This paper presented RASO, the Requirements for Adaptive Systems Ontology, a domain reference ontology about requirements-based development of adaptive systems. RASO was built using the SABiO ontology engineering method, reusing concepts from existing ontologies on Software Engineering and built on the foundational ontology UFO. Requirements elicitation and ontology capture were based on selected publications from a systematic mapping of the literature regarding this domain. The ontology was evaluated by verifying that its proposed conceptual model answers all its Competency Questions and by instantiating real-world entities extracted from the most cited approaches from the systematic mapping of the literature using the concepts defined by RASO.

Given how RASO was elicited and validated, we consider that it successfully represents the state-of-the-art on requirements-based approaches for the development of adaptive systems, achieving its purposes of serving as a well-founded vocabulary to improve understanding and knowledge sharing in this domain and as conceptual basis to solve interoperability problems or for the creation or reengineering of approaches for adaptive systems development.

As future work, we are considering: (a) studying the approaches returned in our systematic mapping study that did not make the top 10 citations per year cut, in order to further validate RASO or complement its conceptual model; (b) providing a more formal characterization of RASO; (c) properly integrating RASO in the Software Engineering Ontology Network mentioned in Section 3; (d) building an operational version of RASO in order to implement a prototype on interoperability among different requirements-based approaches for adaptive systems; (e) combined with the Goal-Oriented Requirements Ontology (GORO) [Negri et al., 2017], performing analysis and reengineering of the Zanshin approach, including its modeling language and Adaptive Feedback Loop Control Program implementation; and (f) building an method for evaluating adaptive systems, as well as creating metrics to qualitatively measure these systems.

Acknowledgments

NEMO (`http://nemo.inf.ufes.br`) is currently supported by Brazilian research funding agencies CNPq (process 407235/2017-5), CAPES (process 23038.028816/2016-41), and FAPES (process 69382549/2015).

Referências

Andersson, J., de Lemos, R., Malek, S., and Weyns, D. (2009). Modeling Dimensions of Self-Adaptive Software Systems. In *Software Engineering for Self-Adaptive Systems*, volume 5525, pages 27–47. Springer.

Baresi, L., Pasquale, L., and Spoletini, P. (2010). Fuzzy Goals for Requirements-driven Adaptation. In *Proc. of the 18th IEEE International Requirements Engineering Conference*, pages 125–134. IEEE.

Berry, D. M., Cheng, B. H. C., and Zhang, J. (2005). The Four Levels of Requirements Engineering for and in Dynamic Adaptive Systems. In *Proc. of the 11th International Workshop on Requirements Engineering: Foundation for Software Quality*, pages 95–100.

Borst, W. N. (1997). *Construction of engineering ontologies for knowledge sharing and reuse*. PhD thesis, Institute for Telematica and Information Technology, University of Twente, Enschede, The Netherlands.

Botia, J. A., Villa, A., and Palma, J. (2012). Ambient Assisted Living system for in-home monitoring of healthy independent elders. *Expert Systems with Applications*, 39(9):8136–8148.

Bresciani, P., Perini, A., Giorgini, P., Giunchiglia, F., and Mylopoulos, J. (2004). Tropos: An Agent-Oriented Software Development Methodology. *Autonomous Agents and Multi-Agent Systems*, 8(3):203–236.

Bringuente, A. C. O., Falbo, R. A., and Guizzardi, G. (2011). Using a Foundational Ontology for Reengineering a Software Process Ontology. *Journal of Information and Data Management*, 2(3):511–526.

Bunge, M. (1977). Treatise on basic philosophy, Vol. 3: Ontology I: The furniture of the world.

Cheng, B. H. C. et al. (2009a). Software Engineering for Self-Adaptive Systems: A Research Roadmap. In *Software Engineering for Self-Adaptive Systems*, volume 5525, pages 1–26. Springer.

Cheng, B. H. C., Sawyer, P., Bencomo, N., and Whittle, J. (2009b). A Goal-Based Modeling Approach to Develop Requirements of an Adaptive System with Environmental Uncertainty. In *Proc. of the 12th International Conference on Model Driven Engineering Languages and Systems (MODELS 09)*, pages 468–483. Springer.

Dalpiaz, F., Giorgini, P., and Mylopoulos, J. (2012). Adaptive socio-technical systems: a requirements-based approach. *Requirements Engineering*, 18(1):1–24.

de Lemos, R. et al. (2013). Software Engineering for Self-Adaptive Systems: A Second Research Roadmap. In *Software Engineering for Self-Adaptive Systems II*, pages 1–32. Springer.

Dermeval, D., Vilela, J., Bittencourt, I. I., Castro, J., Isotani, S., Brito, P., and Silva, A. (2016). Applications of ontologies in requirements engineering: a systematic review of the literature. *Requirements Engineering*, 21(4):405–437.

Duarte, B. B., Leal, A. L. d. C., Falbo, R. d. A., Guizzardi, G., Guizzardi, R. S. S., and Souza, V. E. S. (2018). Ontological Foundations for Software Requirements with a Focus on Requirements at Runtime. *Applied Ontology*, preprint(preprint):1–33.

Falbo, R. A. (2014). SABiO: Systematic Approach for Building Ontologies. In *Proc. of the Proceedings of the 1st Joint Workshop ONTO.COM / ODISE on Ontologies in Conceptual Modeling and Information Systems Engineering*. CEUR.

Fonseca, V. S., Barcellos, M. P., and de Almeida Falbo, R. (2017). An ontology-based approach for integrating tools supporting the software measurement process. *Science of Computer Programming*, 135:20–44.

Goldsby, H. J., Sawyer, P., Bencomo, N., Cheng, B. H. C., and Hughes, D. (2008). Goal-Based Modeling of Dynamically Adaptive System Requirements. In *Proc. of the 15th Annual IEEE International Conference and Workshop on the Engineering of Computer Based Systems*, pages 36–45. IEEE.

Grüninger, M. and Fox, M. (1995). Methodology for the Design and Evaluation of Ontologies. In *IJCAI'95, Workshop on Basic Ontological Issues in Knowledge Sharing*.

Guizzardi, G. (2007). On ontology, ontologies, conceptualizations, modeling languages, and (meta) models. *Frontiers in artificial intelligence and applications*, 155:18.

Guizzardi, G., Falbo, R. d. A., and Guizzardi, R. S. S. (2008). Grounding software domain ontologies in the unified foundational ontology (ufo): The case of the ode software process ontology. In *Proc. of the 11th Iberoamerican Conference on Software Engineering (CIbSE)*, pages 127–140.

Jureta, I., Mylopoulos, J., and Faulkner, S. (2008). Revisiting the core ontology and problem in requirements engineering. In *International Requirements Engineering, 2008. RE'08. 16th IEEE*, pages 71–80. IEEE.

Jureta, I. J., Borgida, A., Ernst, N. A., and Mylopoulos, J. (2010). Techne: Towards a new generation of requirements modeling languages with goals, preferences, and inconsistency handling. In *Requirements Engineering Conference (RE), 2010 18th IEEE International*, pages 115–124. IEEE.

Jureta, I. J., Mylopoulos, J., and Faulkner, S. (2009). A core ontology for requirements. *Applied Ontology*, 4(3-4):169–244.

Kephart, J. O. and Chess, D. M. (2003). The vision of autonomic computing. *Computer*, 36(1):41–50.

Kitchenham, B. A. and Charters, S. (2007). Guidelines for performing Systematic Literature Reviews in Software Engineering. Technical report, Keele University, UK.

Morandini, M., Penserini, L., Perini, A., and Marchetto, A. (2017). Engineering requirements for adaptive systems. *Requirements Engineering*, 22(1):77–103.

Nahrstedt, K., Xu, D., Wichadakul, D., and Li, B. (2001). QoS-Aware Middleware for Ubiquitous and Heterogeneous Environments. *IEEE Communications Magazine*, 39(11):140–148.

Nardi, J. C., de Almeida Falbo, R., and Almeida, J. P. A. (2014). An Ontological Analysis of Service Modeling at ArchiMate's Business Layer. In *Proc. of the 18th International IEEE Enterprise Distributed Object Computing Conference*, pages 92–100. IEEE.

Negri, P. P., Souza, V. E. S., Leal, A. L. d. C., Falbo, R. A., and Guizzardi, G. (2017). Towards an Ontology of Goal-Oriented Requirements. In *Proc. of the 20th Ibero-American Conference on Software Engineering, Requirements Engineering track*.

Peçanha, C. C., Duarte, B. B., and Souza, V. E. S. (2018). RASO: an Ontology on Requirements for the Development of Adaptive Systems. In *Proc. of the 21st Workshop on Requirements Engineering (WER 2018)*, pages 1–14, Rio de Janeiro, RJ, Brasil. PUC-RIO.

Petersen, K., Vakkalanka, S., and Kuzniarz, L. (2015). Guidelines for conducting systematic mapping studies in software engineering: An update. *Information and Software Technology*, 64:1–18.

Qureshi, N. A., Jureta, I. J., and Perini, A. (2011). Requirements engineering for self-adaptive systems: Core ontology and problem statement. In *International Conference on Advanced Information Systems Engineering*, pages 33–47. Springer.

Rao, A. S. and Georgeff, M. P. (1995). BDI Agents: From Theory to Practice. In *Proc. of the 1st International Conference on Multiagent Systems*, San Francisco, CA, USA. AAAI.

Reinhartz-Berger, I., Sturm, A., and Wand, Y. (2013). Comparing functionality of software systems: An ontological approach. *Data & Knowledge Engineering*, 87:320–338.

Ruy, F. B., Falbo, R. d. A., Barcellos, M. P., Costa, S. D., and Guizzardi, G. (2016). SEON: A Software Engineering Ontology Network. In *Proc. of the 20th International Conference on Knowledge Engineering and Knowledge Management*, pages 527–542. Springer.

Sawyer, P., Bencomo, N., Whittle, J., Letier, E., and Finkelstein, A. (2010). Requirements-Aware Systems: A research agenda for RE for self-adaptive systems. In *Proc. of the 18th IEEE International Requirements Engineering Conference*, pages 95–103. IEEE.

Soares, M., Vilela, J., Guedes, G., Silva, C., and Castro, J. (2016). Core Ontology to Aid the Goal Oriented Specification for Self-Adaptive Systems. In *New Advances in Information Systems and Technologies*, pages 609–618. Springer.

Souza, V. E. S. (2012). *Requirements-based Software System Adaptation*. PhD thesis, University of Trento, Italy.

Souza, V. E. S., Lapouchnian, A., Robinson, W. N., and Mylopoulos, J. (2011). Awareness Requirements for Adaptive Systems. In *Proc. of the 6th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, pages 60–69. ACM.

Souza, V. E. S. and Mylopoulos, J. (2011). From Awareness Requirements to Adaptive Systems: a Control-Theoretic Approach. In *Proc. of the 2nd International Workshop on Requirements@Run.Time*, pages 9–15. IEEE.

van Lamsweerde, A., Dardenne, A., Delcourt, B., and Dubisy, F. (1991). The KAOS Project: Knowledge Acquisition in Automated Specification of Software. In *Proc. of the AAAI Spring Symposium Series, Stanford University*, pages 59–62. AAAI.

Whittle, J., Sawyer, P., Bencomo, N., Cheng, B. H. C., and Bruel, J.-M. (2010). RELAX: a language to address uncertainty in self-adaptive systems requirement. *Requirements Engineering*, 15(2):177–196.

Yang, Z., Li, Z., Jin, Z., and Chen, Y. (2014). A systematic literature review of requirements modeling and analysis for self-adaptive systems. In *International Working Conference on Requirements Engineering: Foundation for Software Quality*, pages 55–71. Springer.

Yu, E. S. K., Giorgini, P., Maiden, N., and Mylopoulos, J. (2011). *Social Modeling for Requirements Engineering*. MIT Press, 1st edition.