January 2016

Towards an Ontology of Requirements at Runtime

Bruno Borlini DUARTE [a], Vítor E. Silva SOUZA [a,1] André Luiz de Castro LEAL [b]
Ricardo de Almeida FALBO [a] Giancarlo GUIZZARDI [a] and
Renata S. S. GUIZZARDI [a]
[a] *Ontology & Conceptual Modeling Research Group (Nemo) – Department of Informatics – Federal University of Espírito Santo (Ufes)*
[b] *Department of Mathematics – Federal Rural University of Rio de Janeiro (UFRRJ)*

Abstract. The use of *Requirements at Runtime* (RRT) is an emerging research area. Many methodologies and frameworks that make use of requirements models during the execution of software can be found in the literature, but very few of them use ontologies to ground the models that are used at runtime. In this paper, we introduce the Runtime Requirements Ontology (RRO), a domain ontology that intends to represent the nature and context of RRT. Following a well-known Ontology Engineering method, we evaluate RRO using verification and validation techniques.

Keywords. Requirements, Runtime, Ontology, UFO, RRO

1. Introduction

In the last years, we have witnessed a growing interest in software systems that can monitor their environment and, if necessary, change their requirements in order to continue to fulfill their purpose [1,2]. This particular kind of software usually consists of a base system responsible for the main functionality, along with a component that monitors the base system, analyzes the data and then reacts properly to make sure that the system continues to execute its required functions.

There are many works in the literature that propose different solutions to this issue, such as adaptive or autonomic systems (e.g., [3,4,5]). We are especially interested in those that deal with this monitoring–adaptation loop using requirements models at runtime. In this context, proposals use different kinds of models and terms to represent what are the system requirements, specify what is to be monitored and prescribe how to adapt. As result, the vocabulary used by these methodologies is very similar, but the semantics of the entities present in the models used by different proposals are not always the same, thus resulting in a domain with overloaded concepts. This problem has motivated us to propose a domain reference ontology [6] on the use of *Requirements at Runtime* (RRT).

The objective and contribution of this work is to provide, through the proposed ontology, a formal representation of the use of RRT, giving a precise description of all do-

[1] Corresponding Author: Ufes - Departamento de Informática (CT7), Av. Fernando Ferrari, 514, Goiabieras, Vitória, ES, Brazil; E-mail: vitor.souza@ufes.br.

main entities and establishing a common vocabulary to be used by sofware engineers and stakeholders for better communication and enhanced problem-solving within the RRT domain. The ontology also defines the distinctions underlying the nature of requirements during the execution of a software system.

The Runtime Requirements Ontology (RRO) was developed following the process defined by the SABiO method [6] and using UFO [7] as foundation ontology. To extract consensual information on RRT concepts, we performed an extensive Systematic Mapping [8,9] of the literature and discussed the results with a group of specialists. Finally, RRO was evaluated by verification and validation as proposed in SABiO.

The rest of the paper is structured as follows. Section 2 summarizes the general concepts of the RRT domain. Section 3 describes the methodology used to build RRO. Section 4 introduces the ontological foundations for the proposed ontology. Section 5 presents RRO, the main contribution of this paper. Section 6 evaluates the ontology. Section 7 discusses related work. Finally, Section 8 concludes the paper.

2. Requirements at Runtime

Requirement Engineering (RE) is the field of Software Engineering (SE) concerned with understanding, modeling, analyzing, negotiating, documenting, validating and managing requirements for software-based systems [10]. In the RE literature, there is a strong overloading of the term *requirement* and, hence, different texts refer to requirements in different possible senses including as an artifact, as an intention, as a desire, as a property of a system, etc.

The seminal work by Zave & Jackson [11] defines the term *requirement* as a desired property of an environment, which comprehends the software system and its surroundings, that intends to express the desires and intentions of the stakeholders concerning a given software development project. A requirement, however, can be seen as an intention or a desire when acting as a high-level requirement, or as an artifact, when documented as part of a specification. In turn, such documentation can have different levels of abstraction, for instance, depending on whether it is part of an early requirements analysis or a machine-readable artifact (file) that allows reasoning over requirements during the execution of the software. Features such as the latter have motivated research (e.g., [12]) on the topic of *Requirements at Runtime* (RRT).

For instance, Feather et al. [13] monitor violation of requirements using Linear Temporal Logic expressions that are observed by a monitoring component at runtime in order to try and reconcile system requirements and behavior. Requirements are documented using a Goal-Oriented RE (GORE) approach for design-time purposes and as the aforementioned logical expressions for runtime purposes. The *Requirements Reflection* approach [14] proposes not only that requirements be reified as runtime entities but that they keep traceability to the architectural models and all the way back to high-level goals.

Souza et al. [15] define a *requirement at runtime* as the classes or scripts that represent the requirements being instantiated during the execution of a program, i.e., a compiled code artifact loaded in memory to represent the fact that someone (or something) is using the system in order to satisfy a requirement. Qureshi & Perini [16] have a similar vision, as they understand that a *requirement at runtime* is expressed by the users requests, which are captured by the software user interface and also monitored in order to check if it is in an agreement with the original specifications.

More recently, Dalpiaz et al. [1] propose, in the context of GORE, the distinction between a *Design-time Goal Model*—used to design a system—and a *Runtime Goal Model*—used to analyze a system's runtime behavior with respect to its requirements.

These and other works illustrate the diversity of concepts, models and features that have been proposed in the field of RRT, motivating us to develop a reference ontology about this domain. The purpose is to fix an interpretation for requirements and then to characterize how it relates to other intimately connected elements in the RRT domain.

3. Methodology

To build the Runtime Requirements Ontology (RRO), we used SABiO, a Systematic Approach for Building Ontologies [6]. 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) test. These phases are supported by well-known activities in the Requirements Engineering lifecycle, such as knowledge acquisition, reuse, documentation and evaluation. SABiO aims at developing both reference ontologies (phases 1 and 2) and operational ontologies (phases 3, 4 and 5). In this work, we are interested in building a domain reference ontology (phases 1 and 2 only).

The purpose of RRO is to improve human knowledge and problem-solving in the Requirements at Runtime (RRT) domain. As non-functional requirements for RRO, we defined that it would: be grounded on a well-know foundational ontology (**NFR1**); and be based on consensual work from the literature (**NFR2**).

Competency Questions (CQs) were elicited as functional requirements for RRO. CQs are questions that the ontology should answer [17] and they help to determine the scope of the ontology [6]. For RRO, CQs were identified and refined using a bottom-up strategy, starting with simpler questions and proceeded to find more complex ones. The result is listed below:

- **CQ1:** What is a running program?
- **CQ2:** Where does a running program execute?
- **CQ3:** What can be observed by a running program execution?
- **CQ4:** What is the relation between a running program and its requirements?
- **CQ5:** What is a requirement at runtime?
- **CQ6:** How are requirements used at runtime?

In order to identify proposals that use RRT, provide a primary input for knowledge acquisition and satisfy NFR2, we performed a systematic mapping of the literature. Kitchenham & Charters [9] define a Systematic Mapping as an extensive study on a specific topic that intends to identify evidence available on this theme.

Following the methodology presented in [9], we applied the steps illustrated in Figure 1. First, we defined a *search string* that intended to cover all the relevant aspects of our research. Then, we applied it in several search engines, in an attempt to find most of the literature about the research domain. 1581 studies returned (a). The search string was validated by checking if the *control articles* (6) that were chosen beforehand were retrieved from the databases (b).

Next, papers returned from all search engines were combined and duplicate entries were removed. As result from this step, 912 studies remained (c). Then, we applied two

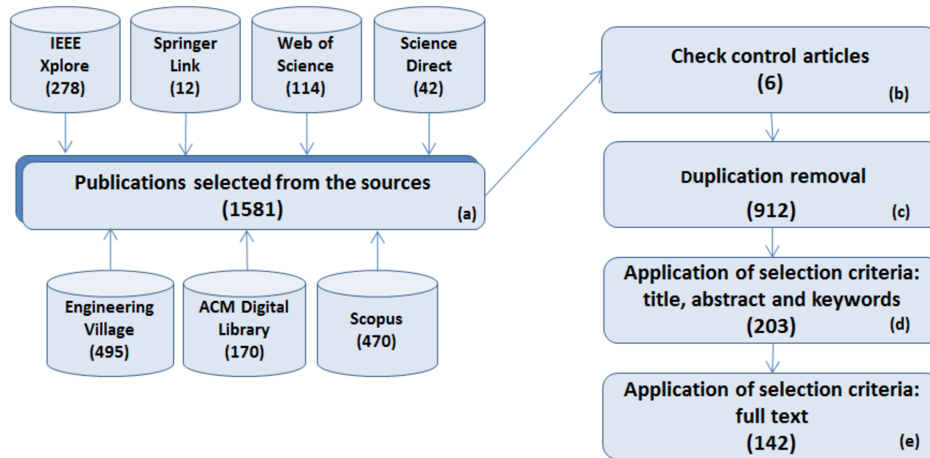

Figure 1. Steps of the Systematic Mapping process on RRT.

filters, considering inclusion and exclusion criteria established at the beginning of the process. In this first filter, only the abstracts were read to evaluate if a paper should be selected or excluded from the mapping (d). Then, the selected publications (203) were once again analyzed against the selection criteria, but now considering the full text of the publication (e). To help reduce bias, publications were analyzed in the second filter by different specialist from the first. As result, 142 publications were found to satisfy the selection criteria, which accounts for a 91,02% reduction from the starting 1581 results.

During the mapping, we classified publications by purpose of use of RRT, separating them in two major categories: *Monitor Requirements* and *Change Requirements*. The former covers publications that propose checking if requirements defined at design time are being achieved at runtime, whereas the latter covers papers in which requirements are used not only as guidelines to monitoring but also as rules on how the system should adapt in order to keep satisfying its requirements. The results of the mapping are being compiled into a paper that will soon be submitted for publication. Although briefly summarized here, the focus of this paper is the ontology, not the mapping.

During the entire process, we conducted weekly meetings in which the primary aspects and the scope of the ontology were defined. These meetings were attended by several specialists in the areas of Requirements Engineering and Ontology Engineering (expanding the initial group of four engineers), who played different roles [6] depending on the context: in requirement elicitation meetings they acted as *Domain Experts* on RRT, whereas in verification meetings, in which the many versions of the ontology were analyzed and polished, they also played the role of *Ontology Engineers*.

Given the requirements (CQs) for RRO and the results of the Systematic Mapping, we started illustrating the categories and relations of RRO. This construction process was highly iterative, using the aforementioned weekly meetings with specialists to refine the models. Results are presented later, in Section 5. Next, we present the ontological foundations that satisfy RRO's NFR1.

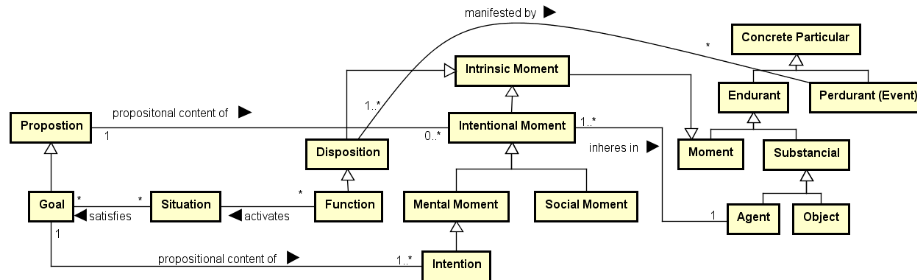

Figure 2. Fragment of UFO showing Goals, Agents and Intentions.

4. Ontological Foundations

For building the Runtime Requirements Ontology (RRO), we reused Wang et al.'s Ontology of Software Artifacts [18] and Guizzardi et al.'s interpretation of Non-Functional Requirements [19]. Moreover, we grounded RRO in the Unified Foundational Ontology (UFO) [7,20].

We choose UFO because it has been constructed with the primary goal of developing foundations for conceptual modeling. Consequently, UFO addresses many essential aspects for conceptual modeling, which have not received a sufficiently detailed attention in other foundational ontologies [7]. Examples are the notions of material relations and relational properties. For instance, this issue did not receive up to now a treatment in DOLCE [21], which focuses solely on intrinsic properties (qualities). Moreover, UFO has been sucessfully employed in a number of semantic analyses, such as the one conducted here (see detailed discussion in [22]).

In Figure 2 we present only a fragment of UFO containing the categories that are germane for the purposes of this article. Moreover, we illustrate these categories and some contextually relevant relations with UML (Unified Modeling Language) diagrams. These diagrams express typed relations (represented by lines with a reading direction pointed by ►) connecting categories (represented as rectangles), cardinality constraints for these relations, subsumption constraints (represented by open-headed arrows connecting a subcategory to its subsuming super-category), as well as disjointness constraints relating sub-categories with the same super-category, meaning that these sub-categories do not have common instances. Of course, these diagrams are used here primarily for visualization. The reader interested in an in-depth discussion and formal characterization of UFO is referred to [23,7,20,24].

Endurants and Perdurants are Concrete Individuals, entities that exist in reality and possess an identity that is unique. Endurants are entities that do not have temporal parts, but persist in time while keeping their identity (e.g., a person). Perdurants, also called Events, are composed by temporal parts (e.g., a trip) [7].

Substantials are existentially independent Endurants. They can be agentive (Agent), i.e., bear intentional properties (states) such as beliefs, desires, intentions; or non-agentive (Object). Agents can bear special kinds of moments called Intentional Moments. An Intention is a specific type of Intentional Moment that refers to a desired state of affairs for which an Agent commits to pursuing (e.g., the intention of a student

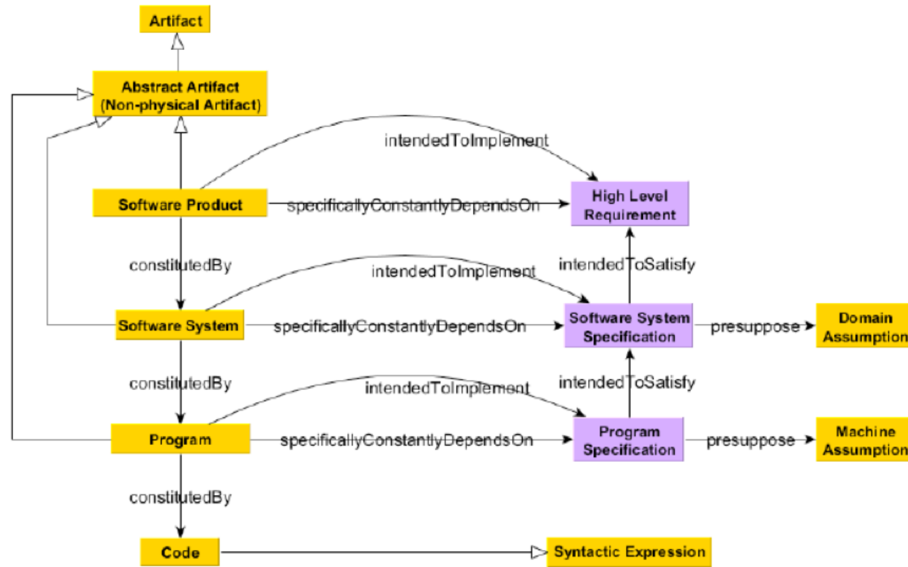

Figure 3. Fragment of the Ontology of Software Artifacts [18].

to take an exam). A Goal is a Proposition (the propositional content of an Intention) that can be satisfied by a Situation, i.e., a portion of the reality that can be comprehended as a whole, iff the Situation is the truthmaker of the Proposition expressed by the Goal [20]. Functions and Dispositions are Intrinsic Moments, i.e existentially dependent entities that have potential to be realizable through the ocurrence of an Event, this occurence brings about a Situation.

Based on UFO, Guizzardi et al. [19] present an interpretation of the difference between Functional and Non-Functional Requirements (FRs/NFRs), a frequently used categorization scheme in Requirements Engineering. According to the authors, requirements are Goals (as in UFO) and can be *functional* and/or *non-functional requirements*. Functional Requirements are those which refer to Functions, whereas NFRs refer to Qualities taking Quality Values in particular Quality Regions.

Finally, we also reused the conceptualization established in Wang et al.'s Ontology of Software Artifacts [18]. According to them, software is a special kind of entity that is capable of changing while maintaining its numerical identity. These changes (i.e., a simple bug fixing or an entirely new release) are necessary for the natural evolution of software and are also one of the engines that move the software industry. When we start to think in specific software such as, e.g., Microsoft Excel, this fact becomes clearer: Excel has many releases and even more versions throughout its 30 years of existence, but has always maintained its identity as Microsoft's spreadsheet software.

To try to answer questions like "what does it mean for software to change?", "What is the difference between a new release and a new version?" and to address the ontological reasons for software being able to change while maintaining its nature, Wang et al. [18] propose an ontology of software inspired by the Requirements Engineering literature, depicted in Figure 3. This ontology makes the distinction between three artifacts:

Software Product, Software System and Program. It also deals with the differences between a Program seen as a piece of Code and as a process, running in a medium.

A Program is a special type of Abstract Artifact, i.e., a non-physical entity created by humans [25], with temporal properties and constituted by Code (a sequence of machine instructions). A Program is created with the purpose of performing a function of a given type, which is specified in a Program Specification. The Program is considered an Abstract Artifact because of its complex nature: it behaves as a universal since it has characteristics that are repeatable in its copies, but it also does not exists outside space and time, unlike a universal.

A Software System is constituted by a set of Programs. A Software System intends to determine the behavior of the machine towards the external environment. This behavior is specified by the Software System Specification. The Software Product is considered an Abstract Artifact that intends to implement the High Level Requirements, which represent the stakeholders intentions and goals.

5. The Runtime Requirements Ontology (RRO)

In this section, we present RRO. To try and manage the complexity of the model, it is divided in figures 4 and 5. Concepts from the ontologies described in Section 4 were imported and are prefixed by the acronym of their original ontology, using NFR for Guizzardi et al.'s ontological interpretation of Non-Functional Requirements [19] and OSA for Wang. et al.'s Ontology of Software Artifacts [18].

Figure 4 explains the nature of requirements at runtime. As in [19], a Requirement is a Goal in the sense of UFO, i.e., the propositional content of an Intention. Once it is documented in some kind of requirements specification (e.g., as result of a requirements phase of a software process), there is a Requirement Artifact describing the Requirement. A Requirement Artifact is an Artifact in the sense of OSA, which we also infer to be an Object in the sense of UFO.

The description of requirements in artifacts can occur in several ways, like a text in natural language that is written in the software requirements specification or in a computational file that could be processed by a program. The latter, if meant to be used at runtime, constitutes a Runtime Requirement Artifact. Thus, a Runtime Requirement Artifact is responsible for describing at runtime, in some way, a Requirement that represents a stakeholder goal.

On the other side of the figure, a Functional Requirement refers to a Software Function Universal (i.e., it describes characteristics that are common to all function individuals inhering in specific machines). A Program intends to fulfill a set of Software Function Universals, which, collectively, can be considered as an abstract requirements specification for this program (one can make it concrete by describing each Requirement using a Requirement Artifact).

Wang et al.'s [18] Program is an Abstract Artifact constituted by code written for a specific machine environment (e.g., Microsoft Excel for MacOS). To get to runtime, one must own a *copy* of the program and *execute it*. Irmak [25] defines such *copy* as physical dispositions of particular computer components (e.g., the hard drive) to do certain things. He then describes the *execution* of the program as a kind of *event*, the physical manifestation of the aforementioned dispositions.

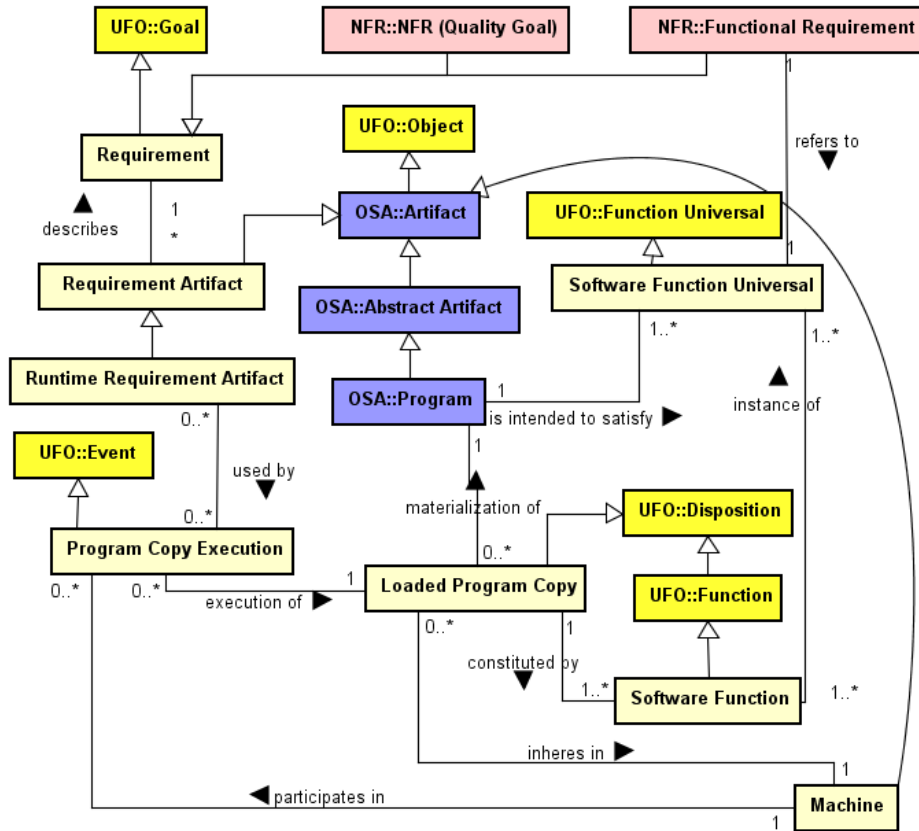

Figure 4. Fragment of RRO describing the nature of Requirements at Runtime.

In the RRT domain (cf. Section 2), requirements are used not only to monitor (which can be done by observing the *events* referred to by Irmak) but also to adapt (i.e., change) the program. In that case, neither *events* (which are immutable) nor the *copy* at the hard drive (which is not running) can help. We are, thus, interested in the Loaded Program Copy as the materialization of a Program, inhering in a Machine, e.g., a copy of Microsoft Excel loaded in primary memory by MacOS on my MacBook.

The Loaded Program Copy is a complex Disposition, constituted by one or more Software Functions which are, in turn, instances of Software Function Universals. When the software development process is done correctly, the functions that constitute the (loaded) program copy are instances of the exact universals the program is intended to fulfill. Moreover, the Loaded Program Copy, being a *Disposition*, is a kind of endurant and, thus, can change qualitatively while maintaining its identity.

Finally, as in [25], the Program Copy Execution is an Event in which the Machine participates. Here, RRO defines a characteristic that distinguishes Runtime Requirement Artifacts from their non-runtime counterparts: they can be *used by* Program Copy Executions at runtime.

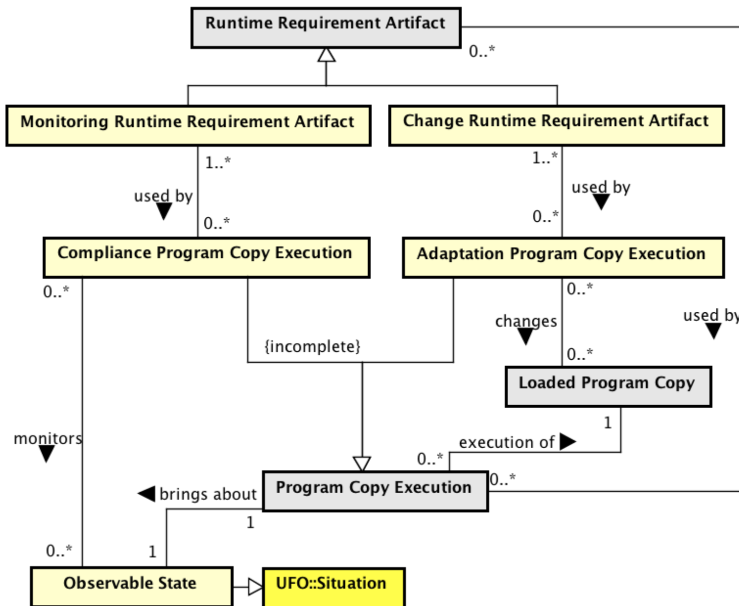

Figure 5. Fragment of RRO showing the relation between requirements at runtime and running programs.

The second fragment of RRO, shown in Figure 5, describes the relation between requirements and programs at runtime. Runtime Requirement Artifact is further specialized into two specific subtypes: a Monitoring Runtime Requirement Artifact defines the criteria for verifying if a requirement is being satisfied or not at runtime; on the other hand, a Change Runtime Requirement Artifact specifies adaptations on the system's behavior, in order for the software system to keep fulfilling its mandate. RRT proposals from the Systematic Mapping categorized as *Monitoring Requirements* present only Monitoring Runtime Requirement Artifacts in their models, whereas *Change Requirements* proposals present both types of runtime requirements artifacts.

The aforementioned artifacts are used by two important events, i.e. the Compliance Program Copy Execution and the Adaptation Program Copy Execution. The Program Copy Execution brings about a particular type of situation (in the sense of UFO-B discussed in [24]). We term this situation here an Observable State. As discussed in [24], a situation is a particular configuration of a part of reality that can be understood as a whole, akin to notion of state of affairs in the philosophical literature. Situations can be characterized by the presence of objects, their intrinsic and relational moments, by the values that the qualities of these objects assume in certain quality regions, etc. (e.g., the situation in which "John has 38°C of fever", or that "a particular network connection is inactive" or the situation in which "the free space on the hard disk is null").

We assume here that an Observable State is a situation involving qualities and quality values (qualia) of the Machine in which the Loaded Program Copy inheres as well as of entities residing in this Machine (including the Loaded Program Copy itself). The Compliance Program Copy Execution monitors the Observable State to verify if the runtime requirements comply with the criteria specified in the Monitoring Runtime Requirements Artifact. If one or more requirements are not being fulfilled accordingly, the

Adaptation Program Copy Execution changes the Loaded Program Copy, following the Change Runtime Requirement Artifact specification.

Considering the models of RRO presented in figures 4 and 5, it is important to observe that:

- The RRT literature uses terms such as *base* or *target system* to refer to the program whose execution's observable state is monitored or whose loaded copy is changed. We do not include a specific concept to represent a *target system* in RRO. One can easily identify which instances of Program (or Loaded Program Copy) are *target systems* from the above description;
- Specializations of Program Copy Execution (e.g., Compliance Program Copy Execution) are, naturally, *executions of* some sort of Loaded Program Copy (e.g., a Compliance Loaded Program Copy) which, in turn, are materializations of some sort of Program (e.g., a Compliance Program). Specializations of Loaded Program Copy and Program, however, are not included in the models for simplicity reasons (one can derive them in a straightforward manner);
- Compliance and adaptation programs can be internal components of the *target application* or an external application that communicates with the *target application* through specific channels. We have considered this out of the scope of the ontology, therefore RRO does not make this particular distinction;
- Compliance and adaptation program copies, when executing, are also subject to being monitored/changed by other executions of compliance/adaptation programs, forming hierarchies of requirements monitoring/adaptation frameworks. RRO considers this situation.
- Only categories and relations are shown. We intend to provide a formal characterization for the domain specific categories and eventual formal constraints involving them in an extension of this paper. However, by reusing the foundational categories present in UFO, the semantics of the domain-related terms are already constrained by the inherited semantics of the corresponding terms in UFO.

Lastly, for a better understanding of the ontology, its categories were instantiated using the Meeting Scheduler example used by the *Zanshin* framework [26]. Table 1 illustrates the results of this instantiation.

6. Evaluation

For ontology verification, SABiO suggests a table that shows the ontology elements that are able to answer the competency questions (CQs) that were raised. For validation, the reference ontology should be instantiated to check if it is able to represent real-world situations.

Table 2 illustrates the results of verification. Moreover, it can also be used as a traceability tool, supporting ontology change management. The table shows that RRO answers all of its CQs.

Regarding validation, RRO was instantiated with individuals extracted from the Meeting Scheduler, mentioned earlier in Section 5. This evaluation intended to check if the ontology was able to represent real world situations.

Table 1. Results of RRO instantiation using the Meeting Scheduler example presented in [26].

Concept	Instance
Requirement	A stakeholder wants a software system that allows users to (among other things) characterize meetings before scheduling them.
Software Function Universal	The function of producing meeting characterizations from appropriate user input as required by the stakeholders.
Program	An implementation of the Meeting Scheduler system for a specific machine environment (e.g., a specific operating system).
Software Function	The disposition of the specific Meeting Scheduler implementation to produce characterizations when given proper inputs.
Loaded Program Copy	Materialization of a Meeting Scheduler implementaton loaded in a machine's main memory.
Program Copy Execution	The event of the loaded copy of the Meeting Scheduler executing in the machine.
Observable State	When given the characteristics of a meeting, the Meeting Scheduler produces the record of a new meeting (e.g., in a database).
Requirement Artifact	*Characterize Meeting* task, from the requirements (goal) model.
Runtime Requirement Artifact	*Characterize Meeting* task, represented in XML to be consumed by *Zanshin* components at runtime.
Compliance Program Copy Execution	The Monitor component of *Zanshin* running in some machine.
Monitoring Runtime Requirement Artifact	*Characterize meeting should never fail* Awareness Requirement [2], represented in XML.
Adaptation Program Copy Execution	The Adapt component of *Zanshin* running in some machine.
Change Runtime Requirement Artifact	*Retry Characterize Meeting after 5 seconds* Evolution Requirement [15], represented in XML.

Table 2. Results of RRO verification.

CQ	Concepts and *Relations*
CQ1	Loaded Program Copy is a *subtype of* Disposition and a *materialization of* Program. Program Copy Execution is a *subtype of* Event and an *execution of* Loaded Program Copy.
CQ2	Loaded Program Copy *inheres in* Machine, which is a *subtype of* Object. Machine *participates in* Program Copy Execution.
CQ3	Observable State is a *subtype of* Situation. Program Copy Execution *brings about* Observable State. Compliance Program Copy Execution *monitors* Observable State.
CQ4	Loaded Program Copy is *constituted by* Software Functions, which are *instances of* Software Function Universals. Loaded Program Copy is a *materialization of* Program, which *intends to fulfill* Software Function Universals. Functional Requirement, a *subtype of* Requirement, *refers to* Software Function Universals.
CQ5	Runtime Requirement Artifact is a *subtype of* Requirement Artifact, which is a *subtype of* Artifact, which is a *subtype of* Object. Runtime Requirement Artifact is *used by* a Program Copy Execution.
CQ6	Monitoring Runtime Requirement Artifact is a *subtype of* Runtime Requirement Artifact and is *used by* a Compliance Program Copy Execution to *monitor* Observable States. Change Runtime Requirement Artifact is a *subtype of* Runtime Requirement Artifact and is *used by* an Adaptation Program Copy Execution to *change* the Loaded Program Copy.

7. Related Work

During the systematic mapping of the literature and the early stages of RRO's development process we have taken in consideration ontologies and ontological analysis of requirements. In this section we will present some of these ontologies that were relevant to our research.

By definition, a core ontology is a mid-term ontology, that is not as specific as a domain ontology but also not so domain-independent as a foundational ontology. In [27], Jureta et al. propose a new core ontology for requirements (CORE), based on Zave & Jackson's work [11] and grounded in DOLCE [21]. The authors extend Zave & Jackson's formulation of the requirements problem, in order to "establish new criteria for determining whether RE has been successfully completed" [27]. CORE covers all types of basic concerns that stakeholders communicate to requirements engineers, thus establishing a new framework for the Requirements Engineering process. CORE was, by far, the ontology that was used the most as basis for works included in the results of the systematic mapping, including, e.g., *Zanshin* [26,2,15]. However, CORE does not cover concepts that are specific of the RRT domain.

Guizzardi et al. [19] propose an ontological interpretation of non-functional requirements (NFRs) in the light of UFO. As briefly mentioned in Section 4, NFRs and functional requirements (FRs) are seen as goals, with the major difference that the former refer to qualities and the latter to functions. In UFO, qualities and functions are both sub-categories of intrinsic moments, however, qualities are properties that are manifested whenever they exist, whereas functions are dispositional properties that are manifested only through the execution of an event. In their work, the authors advance the work of CORE and motivate the choice for adopting UFO as opposed to DOLCE in this domain. In our work, we have imported the distinction of FRs and NFRs in order to relate requirements to functions, but we have not explored the ontology of [19] extensively, which is subject to future work.

Nardi & Falbo [28] propose a new version of the Software Requirements Ontology (SRO), reengineering an existing ontology and grounding it in UFO. SRO is a domain ontology that was built for formalizing the knowledge in the RE domain and to support the integration and development os RE tools. Compared with RRO, SRO treats Requirements as Artifacts, however, in RRO, we made a distinction in which we defined Requirements as Goals and their descriptions as the Artifacts that are mentioned in SRO. Despite this difference, since both ontologies are grounded in UFO, they share similar conceptualizations and could be reused together if necessary.

In effect, RRO, SRO and the interpretation of NFRs [19] are complementary to each other: they represent different aspects of Requirements Engineering, which could be combined if necessary.

8. Conclusions

The main contribution of this paper is the definition of RRO, a domain reference ontology about the use of requirements at runtime. To build it, we followed the SABiO approach for identifying the purpose, eliciting requirements, capturing, formalizing, verifying and validating the ontology. During requirements elicitation, we performed a sys-

tematic mapping of the literature on requirements at runtime, which played a key role in our work, since it was the primary source of shared knowledge about the domain.

Given that RRO was based on an extensive mapping of the literature and evaluated positively through verification and validation techniques proposed by SABiO, we believe it fits well its purpose as a common vocabulary about the RRT domain and a precise description about its concepts, successfully representing the state-of-the-art on requirements at runtime.

As future work, we intend to: (a) extend on the validation of this ontology by using formal validation techniques (e.g., Alloy); (b) provide a full formal characterization of RRO, including eventual domain-related formal contraints; (c) use RRO in particular implementations under the domain of Adaptive Systems, which serves as an additional form of evaluation; (d) analyze the need/possibility of developing an Ontology Pattern Language (OPL) for this domain, as done by out research group in other software-related domains.

Regarding item (c) above, our intention is to create an ontology that combines concepts of RRO with concepts from Goal-Oriented Requirements Engineering (GORE), a paradigm that is very popular in Requirements Engineering research (fact that is confirmed by the systematic mapping results). At first hand, we decided to not base our ontology in GORE or any other specific RE approach, with the purpose of making it as generic as possible, not excluding any potential users. Then, we plan on deriving metamodels from this ontology in order to develop a new version of the *Zanshin* framework for adaptive systems, now properly grounded on an ontology about RRT and GORE.

9. Acknowledgments

Nemo (`http://nemo.inf.ufes.br`) is currently supported by Brazilian research agencies Fapes (# 0969/2015), CNPq (# 461777/2014-2), and by Ufes' FAP (# 6166/2015). We would like to thank Nicola Guarino, Pedro Negri and Beatriz Franco Martins for their participation during discussions about the research contained herein.

References

[1] F. Dalpiaz, A. Borgida, J. Horkoff, and J. Mylopoulos. Runtime goal models. In *Proc. of the IEEE 7th International Conference on Research Challenges in Information Science*, pages 1–11, Paris, France, may 2013. IEEE.

[2] V. E. S. Souza, A. Lapouchnian, W. N. Robinson, and J. Mylopoulos. Awareness Requirements. In R. Lemos, H. Giese, H. A. Müller, and M. Shaw, editors, *Software Engineering for Self-Adaptive Systems II*, volume 7475 of *Lecture Notes in Computer Science*, pages 133–161. Springer, 2013.

[3] M. C. Huebscher and J. A. McCann. A survey of Autonomic Computing—Degrees, Models, and Applications. *ACM Computing Surveys*, 40(3):1–28, 2008.

[4] B. H. C. Cheng, R. de Lemos, H. Giese, P. Inverardi, and J. Magee, editors. *Software Engineering for Self-Adaptive Systems*, volume 5525 of *Lecture Notes in Computer Science*. Springer, 2009.

[5] R. de Lemos, H. Giese, H. A. Müller, and M. Shaw, editors. *Software Engineering for Self-Adaptive Systems II*, volume 7475 of *Lecture Notes in Computer Science*. Springer, 2013.

[6] R. A. Falbo. SABiO: Systematic Approach for Building Ontologies. In G. Guizzardi, O. Pastor, Y. Wand, S. de Cesare, F. Gailly, M. Lycett, and C. Partridge, editors, *Proc. of the Proceedings of the 1st Joint Workshop ONTO.COM / ODISE on Ontologies in Conceptual Modeling and Information Systems Engineering*, Rio de Janeiro, RJ, Brasil, sep 2014. CEUR.

[7] G. Guizzardi. *Ontological Foundations for Structural Conceptual Models*. Phd thesis, University of Twente, The Netherlands, 2005.

[8] B. A. Kitchenham, D. Budgen, and O. P. Brereton. The value of mapping studies: A participantobserver case study. In *Proc. of the 14th International Conference on Evaluation and Assessment in Software Engineering*, EASE'10, pages 25–33, Swinton, UK, UK, 2010. British Computer Society.

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

[10] B. H. C. Cheng and J. M. Atlee. Research Directions in Requirements Engineering. In *Future of Software Engineering (FOSE '07)*, pages 285–303. IEEE, may 2007.

[11] P. Zave and M. Jackson. Four Dark Corners of Requirements Engineering. *ACM Transactions on Software Engineering and Methodology*, 6(1):1–30, jan 1997.

[12] N. Bencomo, E. Letier, A. Finkelstein, J. Whittle, and K. Welsh, editors. *Proceedings of the 2nd International Workshop on Requirements@Run.Time*. IEEE, 2011.

[13] M. Feather, S. Fickas, A. van Lamsweerde, and C. Ponsard. Reconciling system requirements and runtime behavior. In *Proc. of the 9th International Workshop on Software Specification and Design*, pages 50–59. IEEE, 1998.

[14] N. Bencomo, J. Whittle, P. Sawyer, A. Finkelstein, and E. Letier. Requirements Reflection: Requirements as Runtime Entities. In *Proc. of the 32nd ACM/IEEE International Conference on Software Engineering (ICSE '10)*, volume 2, pages 199–202, Cape Town, South Africa, may 2010. ACM.

[15] V. E. S. Souza, A. Lapouchnian, K. Angelopoulos, and J. Mylopoulos. Requirements-driven software evolution. *Computer Science - Research and Development*, 28(4):311–329, nov 2013.

[16] N. A. Qureshi and A. Perini. Requirements Engineering for Adaptive Service Based Applications. In *Proc. of the 18th IEEE International Requirements Engineering Conference*, pages 108–111, Sydney, Australia, sep 2010. IEEE.

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

[18] X. Wang, N. Guarino, G. Guizzardi, and J. Mylopoulos. Towards an Ontology of Software: a Requirements Engineering Perspective. In P. Garbacz and O. Kutz, editors, *Proc. of the 8th International Conference on Formal Ontology in Information Systems*, volume 267, pages 317–329, Rio de Janeiro, RJ, Brasil, sep 2014. IOS Press.

[19] R. S. S. Guizzardi, F.-L. Li, A. Borgida, G. Guizzardi, J. Horkoff, and J. Mylopoulos. An Ontological Interpretation of Non-Functional Requirements. In P. Garbacz and O. Kutz, editors, *Proc. of the 8th International Conference on Formal Ontology in Information Systems*, volume 267, pages 344–357, Rio de Janeiro, RJ, Brasil, sep 2014. IOS Press.

[20] G. Guizzardi, R. de Almeida Falbo, and R. S. Guizzardi. 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, 2008.

[21] C. Masolo, S. Borgo, A. Gangemi, N. Guarino, A. Oltramari, and L. Schneider. Dolce: a descriptive ontology for linguistic and cognitive engineering. *WonderWeb Project, Deliverable D17 v2*, 1, 2003.

[22] G. Guizzardi, G. Wagner, J. P. A. Almeida, and R. S. S. Guizzardi. Towards ontological foundation for conceptual modeling: The unified foundational ontology (ufo) story. *Applied Ontology*, 10(3–4):259–271, 2015.

[23] A. B. Benevides, G. Guizzardi, B. F. B. Braga, and J. P. A. Almeida. Validating Modal Aspects of OntoUML Conceptual Models Using Automatically Generated Visual World Structures. *Journal of Universal Computer Science*, 16(20):2904–2933, 2010.

[24] G. Guizzardi, G. Wagner, R. Almeida Falbo, R. S. S. Guizzardi, and J. P. A. Almeida. Towards Ontological Foundations for the Conceptual Modeling of Events. In *Proc. of the 32th International Conference on Conceptual Modeling*, pages 327–341. Springer, 2013.

[25] N. Irmak. Software is an abstract artifact. *Grazer Philosophische Studien*, 86(1):55–72, 2013.

[26] V. E. S. Souza. *Requirements-based Software System Adaptation*. Phd thesis, University of Trento, Italy, 2012.

[27] I. Jureta, J. Mylopoulos, and S. Faulkner. Revisiting the Core Ontology and Problem in Requirements Engineering. In *Proc. of the 16th IEEE International Requirements Engineering Conference*, pages 71–80. IEEE, 2008.

[28] J. C. Nardi and R. A. Falbo. Evolving a Software Requirements Ontology. In *Proc. of the 34th Conferencia Latinoamericana de Informatica (CLEI 08)*, Santa Fe, Argentina, sep 2008.