Requirements Models for Design- and Runtime

A Position Paper

Alexander Borgida
Rutgers University
United States
borgida@cs.rutgers.edu

Fabiano Dalpiaz
University of Toronto
Canada
dalpiaz@cs.toronto.edu

Jennifer Horkoff, John Mylopoulos
University of Trento
Italy
{horkoff, jm}@disi.unitn.it

***Abstract*—In this position paper we review the history of requirements models and conclude that a goal-oriented perspective offers a suitable abstraction for requirements analysis. We stake positions on the nature of modelling languages in general, and requirements modelling languages in particular. We then sketch some of the desirable features (...“requirements”) of design-time and runtime requirements models and draw conclusions about their similarities and differences.**

I. On Models and Modelling Languages

Abstraction is a key skill for any kind of engineering, software engineering [1] included. Modelling is the act or process of abstracting for a purpose, and models are its result. In Software Engineering (SE), we are primarily interested in analytical (class-level), rather than analogical (instance-level) models [2]. SE models describe a software system, its operational environment and requirements, its design, implementation, runtime performance, history and evolution.

Models are descriptions of a subject domain and hence need to be expressed in some form, via a *modelling language*. The following are some criteria that we and others have used to judge modelling languages: *notation* (visual vs. textual), *syntax*, *semantics*, *expressiveness*, *ontological commitments*. All of these could be the subject of in-depth discussions. In this paper, we mainly focus on the latter two, since the former have been thoroughly addressed in papers by others, including Jackson [2], and Harel & Rumpe [3].

Ontological commitments: All modelling languages are founded on a set of primitive concepts (a.k.a. ontology). Generic modelling languages, such as First Order Logic (FOL) come with domain-independent primitives (e.g., Predicate, Function) and leave it to the modeller to decide what specific concepts are appropriate for her domain. The ontological neutrality of FOL and its limits for representing a domain are discussed, e.g., in [4]. Other modelling languages offer primitives that are more domain-specific. The Entity-Relationship modelling language (ER), for instance, was intended to describe the contents of a database in a natural and direct way (rather than via database jargon): “...The entity-relationship model adopts the ... view that the real world consists of entities and relationships ...” [5]. As a consequence, its primitives include Entity Set, Relationship Set, Attribute, and uniqueness constraints. The choice proved highly successful!

In order to discuss expressive power, we need to make brief reference to model syntax and semantics.

Syntax: A modelling language has an associated set of allowable/well-formed descriptions. For ER, these are diagrams showing, for example, two entity sets A and B linked to a relationship set R, with attributes a, b and c for A, R, and B respectively (imagine!). The model depicted by this imaginary diagram could have also been presented in linear form with an expression as shown in Expression (1).

$$Ent(\mathsf{A}) \wedge Ent(\mathsf{B}) \wedge Rel(\mathsf{R}) \wedge Link(\mathsf{R},\mathsf{A}) \wedge Link(\mathsf{R},\mathsf{B}) \wedge Attr(\mathsf{A},\mathsf{a}) \wedge Attr(\mathsf{R},\mathsf{b}) \wedge Attr(\mathsf{B},\mathsf{c}) \quad (1)$$

In Extended ER, entity sets can be declared to be IsA related, as in IsA(Dog,Canine).

Semantics: Although Harel & Rumpe provide a much more thorough treatment of this in the context of SE, we mean here the assignment of a “denotation”/“meaning” to every syntactic expression. This is usually done formally, by (i) mapping expressions into mathematical structures[1]; or (ii) by translation to some other language that already has a formal semantics (e.g., FOL). The second approach can be applied to EER via Expression (1)[2]. The advantage of a semantics is that it justifies a “deducible from” relationship $\vdash$. For example, the FOL translation of EER would have the following schema representing in part “inheritance”.

$$IsA(\mathcal{A},\mathcal{B}), IsA(\mathcal{B},\mathcal{C}) \vdash IsA(\mathcal{A},\mathcal{C}) \quad (2)$$

Expressiveness of a modelling language refers to the details/distinctions it can capture. For example, one could define a more expressive version of ER by allowing disjunctions in expressions such as (1). Alternatively, instead of using predicates, we could restrict ourselves to propositional logic, meaning that only predicates with zero arguments are allowed.

So, for modelling purposes the *ontology* (chosen primitives), *expressiveness*, and *formal semantics* constitute key properties for a modelling language, indeed ones that make or break its usefulness for a particular task.

We focus next on *requirements* models, their nature and their uses. Given that they represent stakeholder

[1] e.g., Denotational or Tarski-style semantics

[2] For those recognizing the higher-order syntax, we make the standard reference to Henkin-style semantics

wishes/needs/wants, requirements constitute the least tangible artifact SE has to cope with, but also the most critical in determining project success/failure. Not surprisingly, models and modelling languages have been used in Requirements Engineering (RE) since the early days of SE, e.g., SADT [6], and have evolved over the years with respect to the primitives they are based on, their degree of formality, their expressiveness, and the kinds of analysis they support.

The main objective of this position paper is to examine the role and nature of requirements models at design and runtime to make a case for the following claims:

- As in other areas of SE, the ontology on which a Requirements Modelling Language (RML) is founded on is critical. Unlike other areas of SE, expressiveness for RMLs is better to be restrained . . . (Expressively) small is beautiful in RE.
- There are fundamental differences between design- and runtime requirements models, in what they capture, the kinds of reasoning they support, and the uses they see.

All claims made herein come with caveats. With respect to its review of the literature, conclusions drawn and positions staked, this paper presents the work and views of its authors, rather than the RE community at-large. Moreover, our views have been inspired by knowledge representation research in Artificial Intelligence (AI). Finally, given the name and focus of the venue, we limit our discussion to models and modelling languages rather than methods. But, of course, we subscribe wholeheartedly to methods that use our models to engage stakeholders throughout the RE process.

The rest of this paper is structured as follows. In Section II we present some historical background and draw early conclusions on the nature of RMLs. Sections III and IV study respectively design- and runtime models, while Section V summarizes the positions staked in this work.

II. History, and Lessons Learnt

The Structured Analysis and Design Technique (SADT) was used in practice to define requirements for US government contracts in the mid-70s. According to its author, Douglas Ross, SADT was meant to be "a language for communicating ideas", not just model software. Indeed, Ross had noted that when one talks about requirements for a system-to-be, one is talking about aspects of the real world: organizational, physical and/or technical. Hence the modelling language to be used had to be suitable for communicating any idea, rather than merely software-specific ones.

The primitives supported by SADT include 'activities' and 'data', along with associated links, such as 'inputs', 'outputs' and 'controls'. Both activities and data can be decomposed to sub-elements of the same kind thereby conforming to the structured methodology dictum that complexity can be harnessed through decomposition of complex concepts into simpler ones, leading to hierarchical structures. Activities/data near the top of such hierarchies represent coarse grain activities/objects, such as Running-a-Farm or Farm-House. Through decomposition these are refined to finer-grain elements such as Pay-Bills or Barn, and some of these finer grain elements define the functional requirements for the system-to-be, while others define the kinds of data the system needs to process. Despite its ambitious goal, SADT was not terribly expressive, missing things such as disjunction, negation, etc. Nor was it in any way formal, beyond supporting the syntactic rules of SADT diagrams. Nevertheless, it had tremendous impact on RE research and practice. In practice, it inspired box-and-arrow notations that dominated for almost two decades. For research, it was perhaps the most influential work in launching the area we now call RE. From a modelling perspective, SADT advocated that requirements elicitation involves modelling the domain as it is supposed to be when the new system is operational (i.e., how will the farm be running when the system is operational?), and then deriving through a decomposition process the functional and data-handling requirements of the system-to-be.

The task-at-hand for RE researchers in the 80s was to formalize SADT-like languages. Two of the authors of this paper were among those trying. Our efforts were successful [7], except for a small detail: because we were using techniques inspired by semantic networks (inheritance), as well as FOL, to formalize SADT, the result ended up being an object-oriented modelling language, rather than a structured one. Predictably (and understandably!) Douglas Ross was not happy when he saw the result . . .

The 90s saw at least two notable advances in RE research. Firstly, Jackson & Zave changed the RE landscape by defining the requirements problem [8]. In a nutshell, the requirements problem takes as given a set of requirements R and a set of environment properties E, and asks for a specification (solution) S of a system-to-be such that

$$\mathsf{E}, \mathsf{S} \vdash \mathsf{R} \tag{3}$$

In other words, "... satisfaction of the requirements can be deduced from the satisfaction of the specification, together with the environment properties..." [8]. The specification here includes the functions the system-to-be ought to support, while environment properties include properties of the domain (such as laws of gravity for a spacecraft design), as well as assumptions made to support the iterative process through which a specification is derived from R and E. The contribution here lies in the distinction introduced between requirements and specification, as well as the formal condition under which a given specification indeed fulfills requirements. Also, in giving a crisp definition of what RE is all about and what problem(s) it is trying to solve.

The other, related, advance was to challenge the orthodoxy of SADT et al. by noting that requirements are not functions but rather what the stakeholders want. Accordingly, requirements ought to be modelled as goals, "desired states of affairs" [9]. Here, we have a shift in the ontology that RMLs ought to be founded on. Such a shift means that in eliciting requirements from stakeholders, we do not ask them what functions/activities the system-to-be ought to perform (... which they probably will not know anyway). Rather, we

ask them what kind of problem they want the system to solve. Then through refinement we flesh out that problem into more concrete ones until we have problems that an external actor can solve by carrying out an action she is capable of, or the system-to-be can solve by executing one of its required functions. This was the philosophy of KAOS [10].

Note that here requirements are desired goals (= problems) and analysis through refinement is basically defining a problem space since there are many different ways to refine a problem. Also note that refinement is more general than decomposition. For example, you can refine a problem into a single sub-problem, Find-a-Book $\rightarrow$ Find-a-Book-in-a-Library-Catalogue. However, you can not decompose a thing into a single part (principle of weak supplementation [11]). The result of goal analysis is a hierarchical structure with stakeholder requirements as roots, actor and system actions as leafs, and conflict links relating goals that can not co-exist in a proposed solution. Solutions, by the way, consist of collections of actions that together fulfill root-level goals. Each such solution constitutes a specification in Jackson & Zave terminology.

In essence, KAOS offered a concrete methodology for solving the requirements problem, one that included new concepts (goal, agent, ...) and a process for eliciting goals, refining them, etc. until a full goal model is built and a specification can be selected. KAOS also came with a rich ontology beyond the primitives discussed here and a rich logical sublanguage including Linear Temporal Logic (LTL) for describing formally the elements of a KAOS model.

The shift from activities and data to goals and actors has had a major impact on RE. It changed the way we view requirements, the types of questions we ask to elicit them, the kinds of models we build, and the kinds of analysis we perform. This leads to a first conclusion, to be drawn from this brief review of RE history. The ontology offered by an RML is all-important.

Position 1. *The ontology offered by a modelling language matters, for it guides the modeller on how she conceptualizes her domain, what questions she asks, what kinds of models she builds and what kinds of analysis she does. "Vanilla" languages such as FOL do not cut it for RE.*

We suspect that generic modelling languages will not cut it for other SE purposes either.

In parallel to the seminal work of van Lamsweerde et al., the "Toronto group" (Lawrence Chung, Brian Nixon, Eric Yu, John Mylopoulos) proposed ways for modelling non-functional requirements, as well as stakeholders, along with their requirements. Non-functional requirements are to be represented as softgoals—goals that do not have a clear-cut criterion for fulfillment [12]. For example "The system should be secure" is a plausible wish coming from a stakeholder, but it is clearly vague. Nevertheless, it makes sense to talk about modelling such a goal, refining it, relating it to other goals (e.g., this security softgoal impacts on user-friendliness, another softgoal). Softgoal models share many features with their (hard)goal cousins, but add qualitative contribution relationships among goals, as in "softgoal SG_1 helps/hurts softgoal SG_2". Such models support a weak form of analysis by adopting qualitative reasoning techniques.

Eric Yu's thesis [13], completed in the mid 90s, builds on this idea but adds that stakeholders need to be modelled too, as actors, along with their goals and softgoals. Moreover, each actor has her own view of the goals she wants to fulfill, how they can be refined and how they can be fulfilled through delegations to other actors. Delegations introduce a social dimension to the requirements problem. We are no longer concerned with going from R and E to S (as in Jackson & Zave). Rather, we are looking for ways of having a collection of (cooperative) actors fulfill their respective goals through delegations to each other and to new actors, as well as delegations to the system-to-be. Volha Bryl's PhD thesis studies this social (i.e., multi-actor) version of the requirements problem [14].

An important theme of this work has been the realization that stakeholder ("early") requirements are necessarily vague, informal, self-contradictory, etc. (...in short "scruffy"), but they are requirements none-the-less. They can not be thrown away, nor can they be instantly cleansed into something more "respectable". Rather, they need to be modelled, analyzed and refined into a specification, just like Jackson & Zave advocated. Therefore calls for only tolerating non-functional requirements that have been metricized, or only requirements that are observably satisfiable/falsifiable are misguided. Requirements are generally born scruffy. It is the task of RE to bring them into a respectable state, through modelling, analysis and refinement. Unlike much of the rest of SE and Computer Science, RE is primarily a *problem definition* activity, rather than a *problem-solving* one (...though we do that too!).

For this purpose, the requirements engineer needs to communicate with stakeholders, and this is best done in simple phrases (see Fig. 1), rather than complex formal expressions. Moreover, goal refinements can be captured with formulas of the form $\alpha_1 \wedge \ldots \wedge \alpha_n \rightarrow \beta$. Interestingly, so can inconsistency (hat-is-black $\wedge$ hat-is-white $\rightarrow \bot$). Propositional Horn Logic is the least "complex" form of logic that accommodates these needs and ventures beyond conjunctions.

Position 2. *The requirements elicited from stakeholders are generally vague, informal and often contradictory. Accordingly, the RMLs used to model and analyze such requirements need to accommodate such traits by limiting expressiveness while tolerating inconsistency.*

For us, limited expressiveness for early requirements and goal models has meant starting from Propositional Horn Logic, though we have found it useful to embellish the actions/tasks with formal annotations [15], as well as adding paraconsistency—i.e., tolerance to inconsistency—and priorities. In the next section, we illustrate our first two positions by presenting the family of RMLs that we and our colleagues have devised in previous work.

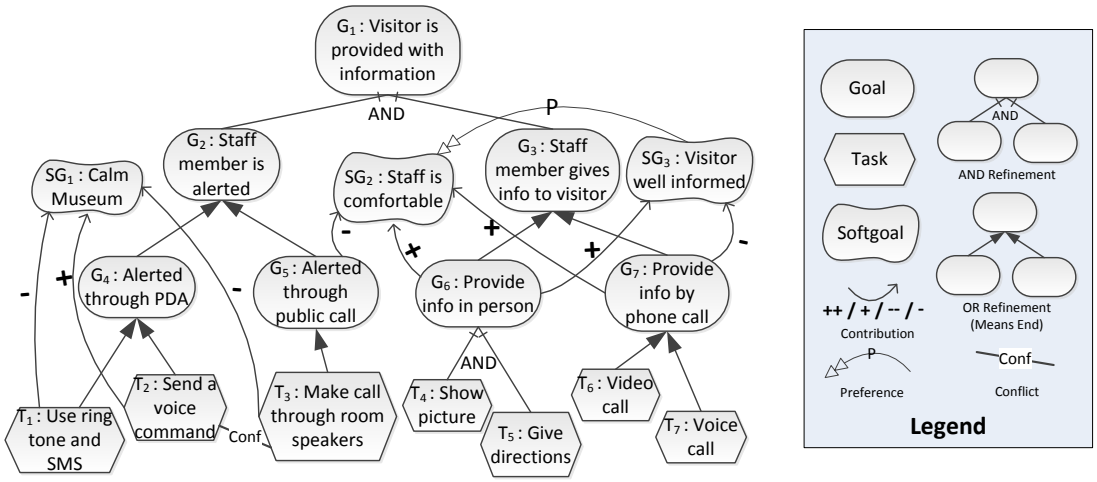

Fig. 1. Design-Time: goal model for a mobile museum guide

III. Design-Time Requirements Models

At design-time, requirements models circumscribe a problem space. Minimally, this includes requirements (functional, non-functional) and their relationships, including refinements, contributions, and conflicts. Among these, refinement is particularly important in that it reduces high-level needs (R) to actionable items (S).

We introduce a simple requirements model to illustrate our goal-oriented perspective on RE. Fig. 1 shows a goal model capturing a subset of the requirements for a museum guide mobile information system, adapted from [16]. Here, the primary requirement is to provide the visitor with information (G_1), for example, about a particular item in the museum. We can refine this *goal* (oval shape) into more detailed requirements, in this case: staff member is alerted (G_2) and staff member gives info to visitor (G_3). We denote this *refinement* with AND, meaning that both subgoals must be achieved to satisfy the primary goal. Further refinements down the tree of goals produce actionable items, or *tasks* (hexagonal shape).

There are usually several ways to solve a problem, and this is captured in goal models through multiple AND/OR refinements. For example, to give information to a visitor (G_3), the staff member can provide information in person (G_6) or through a phone call (G_7). Given the model in Fig. 1, and the requirements R = $\{G_1\}$, a possible solution to the requirements problem includes T_4: 'Show picture', and T_5: 'Give directions'. Another solution includes making a video call (T_6), and yet another one making a voice call (T_7).

Selection between alternative solutions can use non-functional requirements and/or preferences. In our example, we add several non-functional requirements or softgoals (cloud shapes), for example, 'Visitor well informed' (SG_3). We show the effects of refinements on these softgoals using contribution links (+ for positive effect, − for negative). For example, G_6 has a positive effect on 'Staff is comfortable' (SG_2). Of course, a system will have many, competing non-functional requirements, increasing the complexity of the selection process.

Several existing techniques make use of refinements, softgoals and contributions in order to help requirements engineers make selections over alternative requirements [12], [17], [18], [19]. Such techniques support exploratory, "what if?" analysis, using the effects of alternative refinements on non-functional requirements to help accept or reject possible sets of solutions.

For example, if we were to explore the effects of selecting a solution that includes T_4 and T_5, we would satisfy G_6, which would satisfy G_3. This option would contribute positively towards both SG_2 and SG_3. However, in order to satisfy our top goal, we must also pick solutions for the left hand side of the model. In this case, if we were to select T_1, we would satisfy our top goal, but provide a negative effect to softgoal SG_1. The analysis process can continue until an alternative is found that makes effective tradeoffs between competing softgoals.

Other approaches refine softgoals into measurable quality constraints, in order to allow modellers to find a solution (selection of task and quality constraints) which satisfies all mandatory goals [20], [21]. In this way, scruffy goal model concepts are moved from "early" to later RE. For example, SG_1 may be refined into a measurable requirement such as "noise level always below 20 decibels". In this type of analysis, instead of exploring the effects of a particular solution, the structure of the model is used to find solutions which are guaranteed to satisfy mandatory requirements. Such techniques can select an optimal solution by further considering the presence of optional goals, conflicts, and preferences between goals. For example, our model includes a *conflict* link between T_3 and T_2, as doing both would confuse the user. As such, a solution may not select both tasks. Similarly, an *optional* requirement does not have to be included in a solution, although such requirements are nice to have. Our model also contains a *preference*, SG_3 preferred over SG_2 (double arrow). An optimal solution would satisfy both requirements, but if this is not possible, we prefer those that satisfy SG_3.

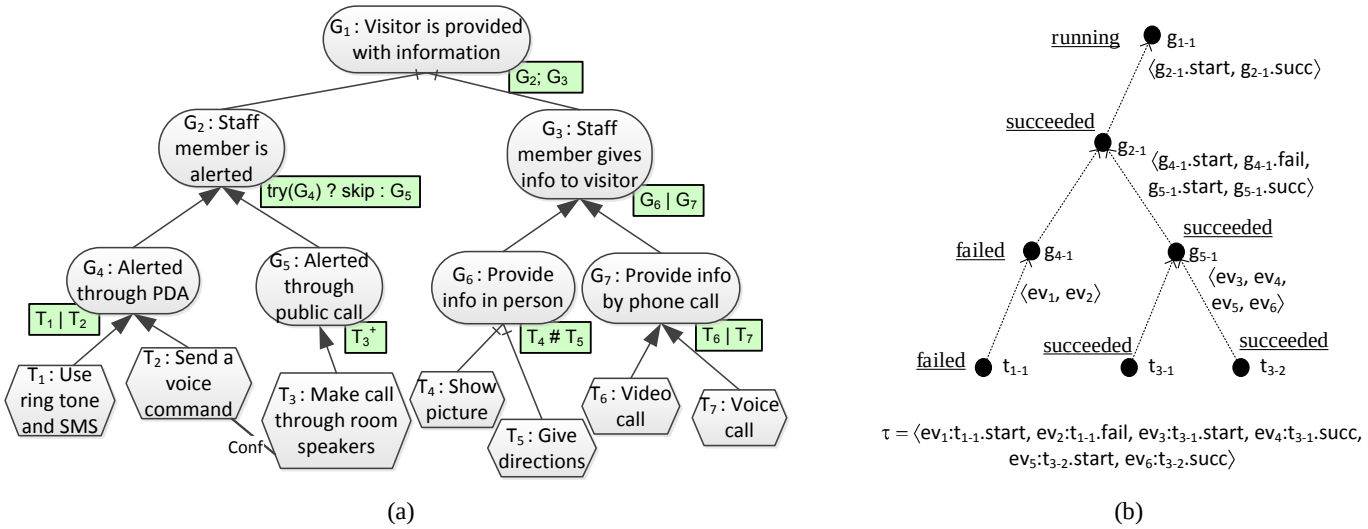

Fig. 2. (a) Runtime goal model and (b) runtime goal instance for the design-time goal model of Fig. 1

Position 3. *Design-time requirement models define a problem space from which a specification is chosen. Concepts such as goals, softgoals, preferences and priorities define what constitutes a solution.*

Further approaches (see [22] for an overview) have extended the ontology and expressiveness of goal modelling with dependencies, probabilities, risks, costs, and obstacles.

The limited expressiveness and complex social nature of early requirements models means that such models may not be complete and that analysis could be supplemented with user input. As such, several "early" goal model analysis procedures deliberately call for stakeholder judgment as part of analysis (e.g., [12], [18]).

Once a solution is found, its elements can be further elaborated into a traditional Software Requirements Specification ("The system shall…"), using some combination of manual and automatic generation techniques (e.g., [23], [24]).

IV. Runtime Requirements Models

The role of requirements models is not limited to the design of a software system; requirements models have a key role at runtime to support monitoring and adaptation of a running system. We use the term *runtime requirements models* to denote requirements models that are used at runtime. They can be used either online by the system itself—e.g., to detect when and how to adapt to a different configuration (e.g., [25], [26], [27], [28])—, or offline by analysts to determine how well the system is doing with respect to requirements.

Design-time requirements models are not sufficiently detailed for this task. Such models, for example, define what tasks/functions the system shall implement, but do not capture information on the status of requirements as the system is executing, nor on the history of an execution.

The root cause for such shortcomings is that whereas design-time models describe a system in terms of *classes* of tasks, a system operates by temporally interleaving *individual instances* of tasks. Although design-time models may consider instances via universal/existential quantification, they are not expressive enough to consider the status of individual instances. Moreover, since tasks are meant to fulfill goals, the class/instance distinction applies to goals as well. Accordingly, runtime requirements models (here, Runtime Goal Models or RGMs) extend [29] design-time requirements models (here, Design-time Goal Models or DGMs) with a behavioral specification of how task and goal instances—of classes in a DGM—can be intertwined. This includes expressing how instances are temporally interleaved (e.g., sequential, concurrent), how many instances of a given class have been created, and whether and when failures of task and goal instances are admitted.

Fig. 2(a) shows an RGM for the DGM in Fig. 1. The RGM shows only functional requirements (hard-goals and tasks), and imposes constraints on how instances of those goal and task classes can be interleaved. For example, an instance of G_1 is successful if a successful instance of G_2 precedes a successful instance of G_3. In other words, in order to provide a visitor with information, a staff member shall be alerted, and then the staff member has to inform the visitor. In turn, an instance of G_2 is successful if an instance of G_4 succeeds (the staff member is alerted via its personal digital assistant), or if it fails but later an instance of G_5 succeeds (a public call is made). The annotation for G_5 expresses that multiple (but at least one) public calls can be made through speakers (these calls are instances of T_3). Details about the annotations can be found in [29].

RGMs are still design-time artifacts. They act as a schema for determining which system executions (traces of events about tasks) comply with requirements. At runtime, the system trace can be interpreted with respect to an RGM to define a Runtime Goal Instance (RGI) [29], i.e., an abstraction of the execution of the system relative to its requirements. Importantly, RGIs are defined in terms of individual instances

of the goal/task classes in an RGM. Unlike DGMs and RGMs, RGIs are runtime artifacts that include information about:

- *State:* task and goal instances are stateful entities, for they are created, pursued, possibly suspended and resumed, and eventually terminate (successfully or not). RGIs store information about the state of instances.
- *History:* a goal or task instance may end up being in a certain state in different ways. To provide accurate diagnostic information, RGIs also keep track of historical information about change of state of tasks and instances. This way, one can reconstruct how a given goal ended up in a certain state based on the history of events related to its sub-goals and tasks.

Fig. 2(b) shows an RGI that has been reconstructed from the trace of task events τ, and based on the RGM in Fig. 2(a). An RGI is constructed from a trace in a bottom-up fashion by analyzing events in the trace, and captures the state (failed, succeeded, running) of each goal instance (node in the tree), and the history of events (between $\langle\ \rangle$) that led a goal instance to its current state. In Fig. 2(b), for instance, task $t_{1\text{-}1}$[3] is in state failed. In turn, this has lead to the failure of goal $g_{4\text{-}1}$. However, goal instance $g_{2\text{-}1}$—the parent of $g_{4\text{-}1}$—has succeeded, due to the success of $g_{5\text{-}1}$. Indeed, the annotation for G_2 in Fig. 2(a) indicates that the system should try to achieve an instance of G_4 and, if this option fails, an instance of G_5 shall be fulfilled.

Reasoning with RGMs and RGIs is significantly different from reasoning with DGMs. A basic form of reasoning consists of reconstructing an RGI from an RGM given a trace. The output of this analysis is an RGI structure, such as the one in Fig. 2(b). As long as the top-level goal instance (the root of the RGI tree) is not in state failed, the system is acting in compliance with its specification in the RGM. One can then build on this basic information and answer more sophisticated diagnostic questions such as: "Are there intermediate (non-leaf, non-root) goals that have failed?", "What are possible next events?", "What is the success/failure rate for a goal G over period T?", and "What is the trend of success/failure for a goal G over period T?"

These questions can be asked by an analyst which wants to assess compliance of a system with its requirements, which functionalities that are rarely used, which functionalities need to be improved (as they lead to requirements failure). Moreover, adaptive systems can use RGIs as a live representation of their requirements, and reflect upon them for determining when an adaptation is needed and which tasks are most likely to succeed in the current context [30].

We generalize our description of and observations about RGMs and RGIs and present our position on the nature and requirements of runtime requirements models.

Position 4. *Runtime requirements models are intended to support monitoring and diagnosis of the runtime system behavior. As such, they need to capture instance-level information about state, behavior, and history of requirements during system execution.*

[3]In Fig. 2(b), $g_{x\text{-}y}$ indicates an instance of goal class G_x with identifier y

V. Discussion

We have argued that for requirements modelling languages, the ontology offered is all-important, but expressiveness is not, since requirements models have to be kept simple. Simplicity is essential (Position 2) for facilitating communication with stakeholders, who are typically uncomfortable with complex modelling languages. Ontology matters (Position 1), for the modelling primitives guide the modeller on how the domain is conceptualized and analyzed. For instance, a goal model is more than a graph, for its nodes and edges carry a specific semantics (e.g., some nodes are goals—desired states of affairs—and some edges are refinement relationships that enable decomposing high-level goals into subgoals and tasks/functions).

We have also argued that runtime requirements models are substantially different from their design-time cousins as they need to capture state, behavior and history information for system executions. This distinction exists due to the different purpose of these two artifacts. While design-time requirements models are meant for representing and analyzing requirements as well as exploring design alternatives (Position 3), runtime requirements models are used to monitor a running system and identify deviations between actual and desired system executions (Position 4).

Acknowledgement

We are grateful to our colleagues Sol Greenspan, Eric Yu, Paolo Giorgini, Nicola Guarino, Ivan Jureta, Neil Ernst, Sotiris Liaskos and many others for helping us shape and refine our scruffy ideas towards respectability. Thanks to the anonymous reviewers for their valuable comments. This work was partially funded by ERC advanced grant 267856, titled "Lucretius: Foundations for Software Evolution", and by the Natural Sciences and Engineering Research Council (NSERC) of Canada through the Business Intelligence Network.

References

[1] J. Kramer, "Is abstraction the key to computing?" *Communications of the ACM*, vol. 50, no. 4, pp. 36–42, 2007.

[2] M. Jackson, "Some notes on models and modelling," *Conceptual Modeling: Foundations and Applications*, vol. 5600, pp. 68–81, 2009.

[3] D. Harel and B. Rumpe, "Meaningful modeling: what's the semantics of," *Computer*, vol. 37, no. 10, pp. 64–72, 2004.

[4] N. Guarino, "Formal ontology, conceptual analysis and knowledge representation," *International Journal of Human Computer Studies*, vol. 43, no. 5–6, pp. 625–640, 1995.

[5] P. P.-S. Chen, "The entity-relationship model–toward a unified view of data," *ACM Transactions on Database Systems*, vol. 1, no. 1, pp. 9–36, 1976.

[6] D. T. Ross and K. E. Schoman Jr, "Structured Analysis for Requirements Definition," *Transactions on Software Engineering*, no. 1, pp. 6–15, 1977.

[7] S. J. Greenspan, J. Mylopoulos, and A. Borgida, "Capturing more world knowledge in the requirements specification," in *Proceedings of the 6th International Conference on Software Engineering (ICSE)*, 1982, pp. 225–234.

[8] M. Jackson and P. Zave, "Deriving specifications from requirements: an example," in *Proceedings of the 17th International Conference on Software Engineering (ICSE)*, 1995, pp. 15–24.

[9] A. van Lamsweerde, "Goal-oriented requirements engineering: a guided tour," in *Proceedings of the 5th IEEE International Symposium on Requirements Engineering (RE)*, 2001, pp. 249–263.

[10] A. Dardenne, A. van Lamsweerde, and S. Fickas, "Goal-directed requirements acquisition," *Science of Computer Programming*, vol. 20, no. 1-2, pp. 3–50, 1993.

[11] A. C. Varzi, "Parts, wholes, and part-whole relations: the prospects of Mereotopology," *Data & Knowledge Engineering*, vol. 20, no. 3, pp. 259–286, 1996.

[12] L. Chung, B. A. Nixon, E. Yu, and J. Mylopoulos, *Non-functional Requirements in Software Engineering*. Springer, 2000.

[13] E. S.-K. Yu, "Modelling strategic relationships for process reengineering," Ph.D. dissertation, University of Toronto, 1996.

[14] V. Bryl, P. Giorgini, and J. Mylopoulos, "Designing socio-technical systems: from stakeholder goals to social networks," *Requirements Engineering*, vol. 14, no. 1, pp. 47–70, 2009.

[15] S. Liaskos, S. McIlraith, S. Sohrabi, and J. Mylopoulos, "Integrating preferences into goal models for requirements engineering," in *Proceedings of the 18th IEEE International Requirements Engineering Conference (RE)*, 2010, pp. 135–144.

[16] R. Ali, F. Dalpiaz, and P. Giorgini, "A goal-based framework for contextual requirements modeling and analysis," *Requirements Engineering*, vol. 15, no. 4, pp. 439–458, 2010.

[17] P. Giorgini, J. Mylopoulos, E. Nicchiarelli, and R. Sebastiani, "Reasoning with goal models," in *Proceedings of the 21st International Conference on Conceptual Modeling (ER)*, 2002, pp. 167–181.

[18] J. Horkoff and E. Yu, "Interactive analysis of agent-goal models in enterprise modeling," *International Journal of Information System Modeling and Design*, vol. 1, no. 4, pp. 1–23, 2010.

[19] E. Letier and A. Van Lamsweerde, "Reasoning about partial goal satisfaction for requirements and design engineering," *ACM SIGSOFT Software Engineering Notes*, vol. 29, no. 6, pp. 53–62, 2004.

[20] N. A. Ernst, J. Mylopoulos, A. Borgida, and I. J. Jureta, "Reasoning with optional and preferred requirements," *Proceedings of the 29th International Conference on Conceptual Modeling (ER)*, pp. 118–131, 2010.

[21] I. J. Jureta, A. Borgida, N. A. Ernst, and J. Mylopoulos, "Techne: towards a new generation of requirements modeling languages with goals, preferences, and inconsistency handling," in *Proceedings of the 18th IEEE International Requirements Engineering Conference (RE)*, 2010, pp. 115–124.

[22] J. Horkoff and E. Yu, "Analyzing goal models: different approaches and how to choose among them," in *Proceedings of the 2011 ACM Symposium on Applied Computing (SAC)*, 2011, pp. 675–682.

[23] E. Letier and A. van Lamsweerde, "Deriving operational software specifications from system goals," *ACM SIGSOFT Software Engineering Notes*, vol. 27, no. 6, pp. 119–128, 2002.

[24] A. van Lamsweerde, "Elaborating security requirements by construction of intentional anti-models," in *Proceedings of the 26th International Conference on Software Engineering (ICSE)*, 2004, pp. 148–157.

[25] D. Cohen, M. S. Feather, K. Narayanaswamy, and S. S. Fickas, "Automatic Monitoring of Software Requirements," in *Proc. of ICSE*, 1997, pp. 602–603.

[26] M. S. Feather, S. Fickas, A. van Lamsweerde, and C. Ponsard, "Reconciling System Requirements and Runtime Behavior," in *Proc. of IWSSD*, 1998, pp. 50–59.

[27] Y. Wang, S. McIlraith, Y. Yu, and J. Mylopoulos, "Monitoring and Diagnosing Software Requirements," *Automated Software Engineering*, vol. 16, no. 1, pp. 3–35, 2009.

[28] F. Dalpiaz, P. Giorgini, and J. Mylopoulos, "Adaptive Socio-Technical Systems: a Requirements-driven Approach," *Requirements Engineering*, vol. 18, no. 1, pp. 1–24, 2013.

[29] F. Dalpiaz, A. Borgida, J. Horkoff, and J. Mylopoulos, "Runtime Goal Models," 2013, unpublished.

[30] N. Bencomo, J. Whittle, P. Sawyer, A. Finkelstein, and E. Letier, "Requirements Reflection: Requirements as Runtime Entities," in *Proc. of ICSE*, 2010, pp. 199–202.