Requirements-Driven Supervision of Socio-Technical Systems

Davide Dell'Anna

Utrecht University, Department of Information and Computing Sciences,
Utrecht, The Netherlands
d.dellanna@uu.nl

Abstract. Modern software systems are characterized by ever-changing goals and requirements. Such systems operate in an environment that is dynamic, open, partly known, unpredictable. New goals arise and others are dropped, due to changes in stakeholders' needs and priorities, government regulations, technology. Despite this dynamism, systems should meet their goals and comply with the evolving requirements. While several self-adaptation mechanisms have been proposed in the literature, they cannot be fully applied for socio-technical systems that involve autonomous (thus, non-controllable) components. This project aims at designing and developing a runtime requirements supervision framework that monitors the execution of socio-technical systems, evaluates their behavior against the overall goals and intervenes by deciding how to *revise* requirements when adaptation is not possible.

Keywords: requirements evolution, self-adaptation, autonomous software, socio-technical systems

1 Introduction

Traditional software supports the conduction of business processes within stable operational environments, where the behavior of the system is mostly predictable and changes are local. Modern software systems, instead, are being built to operate complex socio-technical systems (STSs) in increasingly dynamic settings [22]. STSs are an emerging paradigm of systems where many different, and possibly autonomous, components (both social ones, like people and organizations, and technical ones, like software) can interact, cohexist and change independently and unpredictably. For example, software for self-driving cars shall work under changing traffic and weather conditions, in unknown roads, with changing traffic regulations, and with new vehicle types. At the same time it shall constantly deal with passengers, pedestrians, other drivers, bicycles, etc.

Moreover, software requirements themselves are in constant motion [13, 27]: new functional requirements arise while others are dropped, the desired quality requirements vary, and the relative priority of the requirements evolves.

Despite this dynamism, software is expected to perform optimally and comply with the evolving requirements, or at least minimize the deviations. In highly dynamic environments, existing offline and runtime verification approaches [6,20] cannot be applied to ensure the fulfillment of the continuously evolving requirements of STSs. For runtime verification techniques, halting the system in case of non-compliance with the requirements is not an option [5, 19].

In many cases, furthermore, it is infeasible for a system designer to anticipate all the possible states that the STS and its operating environment can reach during execution [24], and to define adequate requirements for each of them. A static requirements model may often result at runtime inadequate to guarantee the overall system goals in various contexts [3, 18]. Runtime revision (incl. approximation) of requirements is therefore one of the key factors to build a versatile system capable of ensuring the stakeholders goals. Temporarily relaxing a strict requirement in previously unpredicted operating conditions may guarantee the stability of the system without the need of an adaptation of its components. Likewise, learning under which conditions requirements are more useful and supporting their autonomous evolution may increase the knowledge of the requirements engineers and improve the quality of their future work.

In this PhD project we propose a runtime requirements supervision framework that continuously monitors the execution of STSs, evaluates their behavior against the overall goals, and, when necessary, intervenes by deciding which requirements can be ignored, weakened or strengthened.

The paper is organized as follows. Section 2 describes the research problem. Section 3 presents the proposed solution, together with an illustrative example. Section 4 gives an overview of related work. Section 5 discusses the applied research methods. Section 6 concludes the paper with a progress report.

2 Problem

Consider the requirements problem $K, S \vdash R$ (the specification S, given some assumptions K about the environment, satisfies the requirements R) formulated by Zave *et al.* [26]. The unpredictable and dynamic nature of STSs makes the design-time domain knowledge incomplete. To overcome this limitation, the Self-Adaptive Systems research field proposed, in the past years, solutions in terms of automated adaptation of S in response to changes in K [12]. The proposed solutions assume that it is possible at runtime to revise S into an S' such that $K', S' \vdash R$ (where K' is the new domain knowledge acquired after deployment). Sometimes, however, such type of adaptation is not possible or desirable. For instance, when humans participate in the system, it is not always possible to directly control their behavior.

Problem statement. How to guarantee the achievement of goals of stakeholders of a STS, when the design-time domain knowledge is incomplete, and no new specification can be determined at runtime to satisfy the requirements?

As a solution to this problem, in this project we propose an automated runtime requirements revision method to perform adaptation at requirements level.

The main research question that is addressed is the following:

MRQ. *How to design and develop a runtime requirements revision method for STSs operating in highly dynamic and weakly controllable environments?*

This question includes several sub-questions:

SRQ 1. What is an expressive, tractable and non-rigid language for specifying functional and quality requirements for software systems?

SRQ 2. What are efficient runtime monitoring mechanisms for checking compliance with the requirements represented according to **SRQ 1** ?

SRQ 3. What are adequate runtime intervention mechanisms to revise the requirements of a STS based on learning from execution data?

SRQ 4. How to evaluate the effectiveness of the proposal on existing systems?

3 Proposed Solution

Fig. 1 sketches the main components of the runtime supervision framework that we propose. At design time **Stakeholders**, together with **Requirements Engineers**, create a **Requirements model**, based on the **Goals** and the values of the stakeholders. The **Target system** is built according to the requirements model and it is **instrumented** in order to monitor the satisfaction of the elicited requirements. At runtime the **Monitoring** component collects information about three main elements: (i) the satisfaction or violation of the instrumented requirements, (ii) the operating contexts in which they are evaluated (e.g. the hour of the day, etc.), (iii) the satisfaction of the overall system's goals.

Notice that we make a key distinction between the requirements and the goals of the system. Requirements closely reflect and guide the way the target system is built and they can be directly evaluated by monitoring the execution of the system. Goals instead represent the rationale behind the development of a software system, they are not always computable (e.g., user satisfaction) and their evaluation may be obtained from the stakeholders in a discontinuous way.

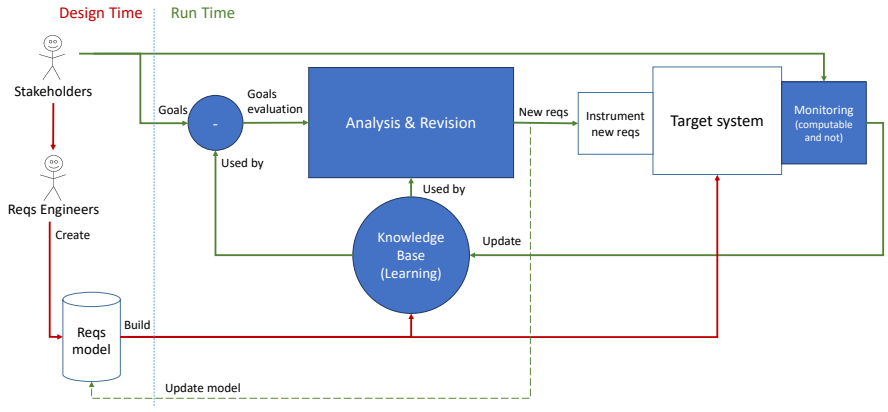

Fig. 1. The proposed requirements supervision framework.

In our framework, data collected by the monitoring component is stored into a **Knowledge Base** and used to **Learn** the correlation between the elements (i), (ii) and (iii) above described. The **Analysis & Revision** component makes use of the information learnt in order to decide if and how to revise the currently active requirements. A revision of the requirements is triggered when the currently active requirements are not able to guarantee the achievement of the goals of the system. Revision is automatically performed, in order to re-align the requirements with the system's objectives.

The novel elements of our framework are the following:

- Accurate requirement models can be obtained only at runtime and through learning. Requirements keep evolving and the relationships between them change, thereby making learning approaches a necessity.
- Requirements revision, including requirement approximation, instead of system adaptation. While early studies on requirements relaxation exist [1, 24], no concrete algorithms exist that support it.

3.1 Illustration: Narrowing Road

Consider a narrowing road with cars coming from two directions: north and south. The **Goal** of the municipality (one of the **Stakeholders**) is "*at any time, there should be less than n cars in queue in either direction*". The requirement elicitation phase produces a **Requirements model** defining different possible requirements to be satisfied. Under the assumption that most of the traffic comes from north, a requirement "*when two cars are at the opposite ends of the road, the car from north shall move first*" is selected. The target system is built and instrumented with sensors (e.g., smart cameras) and actuators (e.g., traffic lights).

At runtime, the data produced by the **Monitoring** component is stored into the **Knowledge Base**, where the relationship between the satisfaction of the requirement and the achievement of the overall goal of the municipality is **learned** under different conditions (traffic intensity, day/night, etc.).

Suppose that one month after deployment, due to a change in the road regulations of the city, an unexpected (at design time) traffic load coming from south is registered between 5 and 6 p.m.. The **Analysis & Revision** component shows that in such a time interval, the requirement is typically satisfied. However it also points out that whenever it is satisfied, the overall goal is hardly achieved. A revision of the requirement is triggered and the existing requirement is refined in two sub-requirements: an alternative requirement "*cars from south shall move first*" is selected for the critical time interval, while the initial requirement is left active for the rest of the day. The system is then adapted and instrumented to monitor the new requirements and the control loop starts over.

4 Related Work

In order to make possible requirements evolution at runtime, monitoring requirements satisfaction is essential. The notion of requirements at runtime emerged

in the past years in requirements engineering literature: specification of software has been extended with annotation for monitoring (e.g. Tropos [9], goal models [10,15], etc.). Adopting these techniques, requirements can be kept alive after deployment and can be integrated in the software to be monitored and analyzed [21]. Some authors [14, 23] propose frameworks for run-time monitoring and diagnosis of non-functional requirements and to detect changes that require adaptation. We use runtime monitoring to collect data about the behavior of a STS and about requirements satisfaction in different operating contexts.

Early studies on relaxation of requirements of a software system are presented by Whittle *et al.* [24]. The authors present a requirements language for self-adaptive systems (RELAX) that allows to specify relaxed versions of a requirement during the requirement elicitation phase.

Ali *et al.* [3] show that causes for requirements evolution include design time assumptions invalidated at runtime. They also discuss the importance of keeping track of the relationship between context and requirements at runtime [2].

The majority of the existing approaches to self-adaptation at requirements level mainly focus on non-functional requirements [4, 7] and on techniques to guarantee compliance with requirements by adapting the system.

Bencomo *et al.* [8] employ Dynamic Decision Networks to suggest a revision of the priorities associated to non-functional requirements based on a degree of uncertainty of events in the environment.

Dalpiaz *et al.* [11] introduce an architecture for adaptive STSs, able to switch between different requirements configuration when needed.

Knauss *et al.* [16] discuss the mining of optimal contexts for contextual requirements, and propose a revision of their contextual condition of applicability. In this project we propose revision concerning all attributes that characterize requirements, including the way they are refined into sub-requirements.

5 Research Methods

The research methods we follow are based on the design science research methodology [25]. The first step is the *investigation* of the most common and critic scenarios, in order to identify the main *stakeholders* and *goals*. We make use of two major case studies of STSs. The former concerns a workflow analyzing immigration applications, where laws and regulations keep changing and relaxations of requirements are necessary when the number of pending applications exceeds the processing capacity. The latter concerns traffic regulation for smart cities, where different autonomous vehicles coexist and interact to achieve their own tasks in a shared environment, and revision of the requirements is necessary to achieve goals of the overall city, such as avoiding road congestions or ensuring safety of pedestrians. After surverying the existing applicable state-of-art solutions in literaure, major existing problems or limitations is identified for the *problem context*. An *artifact*, the runtime supervision framework, is therefore designed in order to overcome the existing gap, by studying the relationship between the proposed solution and the context of the problem. Both theoretical analysis and

the described case studies are used to validate the designed artifact and to trigger changes in areas that require improvements, guided by the research questions. The framework is *evaluated* with the help of real-scale case studies.

6 Progress

Two main application scenarios (immigration applications and traffic regulation) of interest for both research and industry have been identified and used to determine the main research problems to face. The proposed framework is now being designed. We are currently focusing on the two main contributions of the project: (i) learning a requirements model whose underlying assumptions are validated by data and where the requirements satisfaction is coherent with the achievement of the goals, and (ii) requirements revision.

We are using different simulated scenarios for smart traffic regulation based on the SUMO simulator [17] to generate data to analyze. Bayesian Networks are currently employed to store and learn information about satisfaction of requirements and their relationship with the stakeholders' goals in different operating context. We formalized an initial proposal of procedure for the diagnosis and the usage of the acquired knowledge for the suggestion of a revision of requirements. The defined suggestion mechanism partly answers **SRQ 3**. We are working on the definition of an opportune language to express requirements for STSs (**SRQ 1**), in order to answer **SRQ 2** and to complete the **SRQ 3** with the automatic generation of new requirements based on the suggested revision.

Acknowledgments. I would like to thank my supervisors Dr. F. Dalpiaz and Dr. M.M. Dastani for their support and advice.

References

1. Alechina, N., Dastani, M., Logan, B.: Norm approximation for imperfect monitors. In: Proc. of AAMAS, 2014. pp. 117–124 (2014)
2. Ali, R., Dalpiaz, F., Giorgini, P.: Reasoning with contextual requirements: Detecting inconsistency and conflicts. Information & Software Technology 55(1), 35–57 (2013)
3. Ali, R., Dalpiaz, F., Giorgini, P., Souza, V.E.S.: Requirements evolution: From assumptions to reality. In: Proc. of EMMSAD, 2011. pp. 372–382 (2011)
4. Almeida, A., Bencomo, N., Batista, T.V., Cavalcante, E., Dantas, F.: Dynamic decision-making based on NFR for managing software variability and configuration selection. In: Proc. of SAC, 2015. pp. 1376–1382 (2015)
5. Basin, D.A., Jugé, V., Klaedtke, F., Zalinescu, E.: Enforceable security policies revisited. ACM Trans. Inf. Syst. Secur. 16(1), 3:1–3:26 (2013)
6. Bauer, A., Leucker, M., Schallhart, C.: Runtime verification for LTL and TLTL. ACM Trans. Softw. Eng. Methodol. 20(4), 14:1–14:64 (2011)
7. Bencomo, N.: Quantun: Quantification of uncertainty for the reassessment of requirements. In: Proc. of RE, 2015. pp. 236–240 (2015)

8. Bencomo, N., Belaggoun, A., Issarny, V.: Dynamic decision networks for decision-making in self-adaptive systems: a case study. In: Proc. of SEAMS, 2013. pp. 113–122 (2013)
9. Bresciani, P., Perini, A., Giorgini, P., Giunchiglia, F., Mylopoulos, J.: Tropos: An agent-oriented software development methodology. Autonomous Agents and Multi-Agent Systems 8(3), 203–236 (2004)
10. Dalpiaz, F., Borgida, A., Horkoff, J., Mylopoulos, J.: Runtime goal models: Keynote. In: Proc. of RCIS, 2013. pp. 1–11 (2013)
11. Dalpiaz, F., Giorgini, P., Mylopoulos, J.: Adaptive socio-technical systems: a requirements-based approach. Requir. Eng. 18(1), 1–24 (2013)
12. De Lemos, R., Giese, H., Müller, H.A., Shaw, M., Andersson, J., Litoiu, M., Schmerl, B., Tamura, G., Villegas, N.M., Vogel, T., et al.: Software engineering for self-adaptive systems: A second research roadmap. In: Software Engineering for Self-Adaptive Systems II, pp. 1–32. Springer (2013)
13. Ernst, N.A., Borgida, A., Jureta, I., Mylopoulos, J.: An overview of requirements evolution. In: Evolving Software Systems, pp. 3–32 (2014)
14. Filieri, A., Tamburrelli, G., Ghezzi, C.: Supporting self-adaptation via quantitative verification and sensitivity analysis at run time. IEEE Trans. Software Eng. 42(1), 75–99 (2016)
15. Giorgini, P., Mylopoulos, J., Nicchiarelli, E., Sebastiani, R.: Reasoning with goal models. ER 2, 167–181 (2002)
16. Knauss, A., Damian, D., Franch, X., Rook, A., Müller, H.A., Thomo, A.: Acon: A learning-based approach to deal with uncertainty in contextual requirements at runtime. Information & Software Technology 70, 85–99 (2016)
17. Krajzewicz, D., Erdmann, J., Behrisch, M., Bieker, L.: Recent development and applications of sumo-simulation of urban mobility. International Journal On Advances in Systems and Measurements 5(3&4), 128–138 (2012)
18. Letier, E., van Lamsweerde, A.: Reasoning about partial goal satisfaction for requirements and design engineering. In: Proc. of SIGSOFT, 2004. pp. 53–62 (2004)
19. Ligatti, J., Bauer, L., Walker, D.: Edit automata: enforcement mechanisms for run-time security policies. Int. J. Inf. Sec. 4(1-2), 2–16 (2005)
20. Pnueli, A., Zaks, A.: PSL model checking and run-time verification via testers. In: Proc. of FM, 2006. pp. 573–586 (2006)
21. Sawyer, P., Bencomo, N., Whittle, J., Letier, E., Finkelstein, A.: Requirements-aware systems: A research agenda for RE for self-adaptive systems. In: Proc. of RE. pp. 95–103 (2010)
22. Sommerville, I., Cliff, D., Calinescu, R., Keen, J., Kelly, T., Kwiatkowska, M.Z., McDermid, J.A., Paige, R.F.: Large-scale complex IT systems. Commun. ACM 55(7), 71–77 (2012)
23. Wang, Y., McIlraith, S.A., Yu, Y., Mylopoulos, J.: Monitoring and diagnosing software requirements. Autom. Softw. Eng. 16(1), 3–35 (2009)
24. Whittle, J., Sawyer, P., Bencomo, N., Cheng, B.H.C., Bruel, J.: RELAX: a language to address uncertainty in self-adaptive systems requirement. Requir. Eng. 15(2), 177–196 (2010)
25. Wieringa, R.: Design Science Methodology for Information Systems and Software Engineering. Springer (2014)
26. Zave, P., Jackson, M.: Four dark corners of requirements engineering. ACM Trans. Softw. Eng. Methodol. 6(1), 1–30 (1997)
27. Zowghi, D., Gervasi, V.: On the interplay between consistency, completeness, and correctness in requirements evolution. Information & Software Technology 45(14), 993–1009 (2003)