RE at runtime

the challenge of change

Jeff Kramer
Department of Computing
Imperial College London
UK
j.kramer@imperial.ac.uk

Abstract—**Providing rigorous techniques and tools to support RE so that it could potentially be performed online, at runtime, is certainly challenging. However the rewards could be great. Performing RE at runtime has the potential not only to enable software self-adaptation, but also to provide support for change in general by suggesting possible remedies to engineers. This talk will present our motivation and vision, and suggest possible approaches using techniques such as model checking, learning and synthesis to try to support RE change and adaptation.**

***Index Terms*—requirements, runtime, models, software architecture, synthesis, change, learning.**

I. Introduction

Requirements engineering has perhaps fallen from grace over recent years. With the rise of the agile code and test methodologies and rapid application development technologies, many believe that its role has become less important: just a small part of the initial setting of overall project requirements. However, I believe that this is a major misjudgment and that a systematic, rigorous and formal approach to requirements engineering is even more important. In our digital world, rigorous RE holds the key to sound software development for complex applications such as cyber-physical systems, especially in the face of a changing world!

II. A vision for software development

In their seminal work on requirements, Zave and Jackson [1] clearly set out the need to consider the environment, providing a precise characterization in the formula

$$\mathbf{S_i}, \mathbf{E} \vdash \mathbf{R}. \quad (1)$$

Here E represents the domain knowledge in the form of properties and assumptions about the environment, distinguishing between those actions that can and those that cannot be controlled in the governed world ie. software vs environment controlled respectively. S_i represents the set of specifications for the software-to-be with environment interface i of actuators and sensors respectively. Finally, R represents the set of requirements specifications to be satisfied. Thus, S_i and E must be sufficient to guarantee satisfaction of R, and the challenge is to develop software S_i.

One attractive development approach is the use of software architectures to configure software components according to need. For instance, the Darwin [2] architectural approach was successfully employed by Philips for consumer products in the form of the Koala platform [3]. Furthermore, if formal behavior is associated with the components, then composition can be used to model the composite behavior [4]. More recently, rapid application development (RAD) and No-Code development platforms are available which aim to provide drag-and-drop tools to enable users in a constrained domain – such as business processes – to develop applications by assembling pre-coded components. This prototyping approach is very appealing for rapid development in a bottom-up manner by both expert and non-expert developers.

These compositional, assembly approaches offer early operational feedback on the adequacy of the software with respect to the requirements. Can they be used in complex domains when more stringent requirements must be achieved, such as embedded cyber-physical systems where systematic interaction is crucial to properly govern its environment? Is there a way of combining the use of rigorous (top-downish) requirements specification and analysis with this attractive (bottom-upish) approach for software component assembly so that it can accomplish the particular requirements [5]?

We suggest that software synthesis, and controller (or plan) synthesis in particular, could hold the key. Synthesis of controllers can be used for a large class of systems to coordinate actions and monitor the environment [6]. The formula is familiar:

$$\mathbf{E}, \mathbf{X_i} \vdash \mathbf{R}. \quad (2)$$

Again, E represents the governed world or environment and R the requirements. The synthesized controller X_i interacts with E through interface i in order to satisfy R. X_i accomplishes this by controlling and coordinating an assembly of components, selected specifically to perform the tasks required of the controller.

The need for rigor and formality is crucial. Formal specifications and models of the environment, the requirements and the capabilities of the software X_i available through interface i, are necessary to support synthesis. Although the real world is inherently informal and any formal

specification or model of the governed part of the environment will necessarily be an abstraction and incomplete [1], nevertheless it can still provide the basis for reasonable synthesis of the controller software X_i.

Further, the effort in developing formal specifications and models is worthwhile as it also enables reasoning and the potential use of automated software tools to support requirements elaboration and revision. For instance, the generation of obstacle conditions for requirements completeness can be supported using a combination of model checking and symbolic machine learning [7]. Model checking can be used for diagnosis to provide both anomalous and witness behavior traces to some goal; while learning can help to suggest obstacles that declaratively specify the behavior that should be avoided by the software-to-be.

III. What about Change? Uncertainties? Unknowns?

As mentioned, it is certainly a challenge to develop adequate formal models of the governed world E; this is especially the case as it is a changing world that includes uncertainty and unknowns. For instance, while the system is running, assumptions may be violated and/or the goals of the system may change or become unachievable, uncertainties may be resolved and unanticipated events may occur. This requires some form of revision and update. The process of requirements revision and software redevelopment is most often performed offline [8], with redeployment performed by stopping the old system and replacing it with the new. However, this is not always desirable or possible.

Recent research on adaptive self-managing systems aims to handle unexpected and unplanned changes that occur at run-time, and to support online revision at runtime [9,10]. These unexpected changes may be in the environment in which the system operates, or in the requirements and goals that the system should achieve, or in the capabilities of the system itself. What we need are rigorous, comprehensive, and pragmatic approaches to deal with the challenges that run-time change presents. The foundation necessary to support such approaches is a sound *runtime* architecture [11,12] that can support runtime change in aspects such as requirements goals and goal revision [13], environment domain modeling and model revision, planning and plan revision, and software configuration and reconfiguration [14]. Furthermore, the specifications and models that were used during software development must be available at runtime. Known as models@runtime, it is essential that they are available and updateable online, as the system executes, to enable and support reasoning@runtime.

Can this really be achieved? RE at runtime is an interesting but difficult challenge, even with human support and intervention. However, perhaps the running system can help in this regard by providing observations from previous responses of the governed environment. Such observations can be used to perform probabilistic rule learning to hypothesize general rules which explain the behavior and so update and correct the domain models [15]. Further, rather than use model checking to uncover anomalous behavior, the violation of an assumption could perhaps be detected and, with a combination of learning and synthesis, used to revise both assumptions and goals to make them achievable [16].

IV. CONCLUSION

RE@runtime opens up a rich and exciting research topic, which seems to have the potential to radically challenge the way in which we perform requirements engineering in our changing digital world. We will need to draw on multiple disciplines in software engineering, AI, control and elsewhere if we are to provide rigorous techniques and tools.

References

[1] P. Zave and M. Jackson, “Four dark corners of requirements engineering,” ACM TOSEM, vol. 6, pp. 1-30, January 1997.

[2] J. Magee, N. Dulay, S. Eisenbach, J. Kramer, “Specifying distributed software architectures”, 5th ACM ESEC, LNCS 989, Springer-Verlag, pp. 137-153, September 1995.

[3] R. Van Ommering, F. van der Linden, J. Kramer, and J. Magee, “The Koala component model for consumer electronics software”, IEEE Computer vol. 33, no.3, pp. 78-85, March 2000.

[4] J. Magee, and J. Kramer, Concurrency: state models & Java programs, 2nd Edition, John Wiley & Sons (Worldwide Series in Computer Science), April 2006.

[5] B.Nuseibeh, "Weaving together requirements and architectures," in IEEE *Computer*, vol. 34, no. 3, pp. 115-119, March 2001.

[6] N. D'Ippolito, V. Braberman, N. Piterman, and S. Uchitel, “Synthesis of live behaviour models”, 18th ACM SIGSOFT FSE, pp 77-86, November 2010.

[7] D. Alrajeh, J. Kramer, A. van Lamsweerde, A. Russo and S. Uchitel, “Generating obstacle conditions for requirements completeness”, 34th IEEE/ACM ICSE, pp. 705-715, May 2012.

[8] D. Alrajeh, A. Cailliau, and A. van Lamsweerde, “Adapting requirements models to varying environments”, 42nd IEEE/ACM ICSE, May 2020.

[9] D. Sykes, W. Heaven, J. Magee, and J. Kramer, “From goals to components: a combined approach to self-management”, IEEE/ACM SEAMS, pp. 1-8, May 2008.

[10] B. Cheng, et al, “Software Engineering for Self-Adaptive Systems: a research roadmap”, In SEfSAS, Springer, LNCS 5525, pp. 1-26, 2009.

[11] J. Kramer and J. Magee, "Self-managed systems: an architectural challenge," Future of Software Engineering (FOSE '07), Minneapolis, pp. 259-268, May 2007.

[12] N. D’Ippolito, J. Kramer, V. Braberman, D. Sykes D, and S. Uchitel, “MORPH: a reference architecture for configuration and behaviour self-adaptation”, CTSE Workshop, ESEC/FSE, pp. 9-16, September 2015.

[13] D. Alrajeh, J. Kramer, A. van Lamsweerde, A. Russo and S. Uchitel, “Risk-driven revision of requirements models”, 38th IEEE/ACM ICSE, San Francisco, pp. 855-865, May 2016.

[14] L. Nahabedian, et al, “Assured and Correct Dynamic Update of Controllers”, 11th SEAMS, Austin, pp. 96-107, May 2016.

[15] D. Sykes, et al, “Learning revised models for planning in self-adaptive systems”, 35th IEEE/ACM ICSE, pp. 63-71, May 2013.

[16] D. Alrajeh, J. Kramer, and S. Uchitel, ,”Enabling software self-adaptation: learning and synthesis for runtime system goal evolution”, in preparation, unpublished.