Run-Time Model Evaluation for Requirements Model-Driven Self-Adaptation

Kristopher Welsh
School of Computing, University of Kent, UK
k.welsh@kent.ac.uk

Nelly Bencomo
INRIA Paris – Rocquencourt, France
nelly@acm.org

***Abstract*—A self-adaptive system adjusts its configuration to tolerate changes in its operating environment. To date, requirements modeling methodologies for self-adaptive systems have necessitated analysis of all potential system configurations, and the circumstances under which each is to be adopted. We argue that, by explicitly capturing and modelling uncertainty in the operating environment, and by verifying and analysing this model at runtime, it is possible for a system to adapt to tolerate some conditions that were not fully considered at design time. We showcase in this paper our tools and research results.**

Keywords-self-adaptation; self-adaptive; model-driven; run-time requirements

I. Extended Abstract

We have previously demonstrated our approach to modeling uncertainty in a self-adaptive system's operating environment [1]. We achieve this by making statements about system components, the operating environment, or the suitability of a particular component to particular operating conditions. These statements are modelled as *claims*, which were first introduced in the NFR framework [2]. Claims are written as simple statements of fact, and can be included in an i* [3] Strategic Rationale model, to record the rationale behind the decision to favour one goal satisfaction strategy over another. Claims can be supported by other claims, to form a hierarchy joined with simple AND, OR, MAKE and BREAK contributions. We have also previously demonstrated a means of deriving system configurations, specified in adaptation *policies*, from the models directly [4].

Building on this previous work, we are now able to demonstrate a means for a system to evaluate these models at run time, inferring the configuration specification by model analysis, bypassing the need for an adaptation policy completely. Furthermore, we demonstrate that a suitably-equipped self-adaptive system may transform the run-time requirements model in response to changes in the operating environment to yield updated adaptive behaviour. We use *claim monitoring* to drive this model transformation, with suitable monitors verifying or invalidating those claims for which monitoring is feasible.

Although all claims in the model are assumed true by default, the degree of confidence held by the analyst in a claim may vary. Some claims are axiomatic, some represent well-researched and well-reasoned arguments behind a decision, and some are little more than conjecture deemed necessary to allow a specification to be reached in the face of a uncertain and/or volatile environment. At design time, it may prove valuable to classify claims by confidence, and to carry out validation (such as research, testing, or stochastic modelling) to improve confidence in the less certain claims. At runtime, a claim in which a low degree of confidence is held is a prime candidate for monitoring. Claim monitoring is akin to requirements monitoring [5] [6], where the degree to which system requirements are achieved is observed and recorded by the system. Monitoring a claim, however, involves using monitoring data to prove or disprove the claim in question. Like much requirements monitoring literature, we do not specify the form of a claim monitor, merely its interface with our tools, which are event-based.

A claim deemed, through analysis of monitoring data to be inaccurate, is said to be falsified. In essence, claim monitoring provides a means of run-time verification of a claim, and our model transformation and evaluation tool provides a means for a self-adaptive system to take action in the event of a claim's negative verification.

Our approach in creating a run-time requirements model-driven self-adaptive system involves the system loading its own i* Strategic Rationale model, complete with a claim hierarchy, and monitoring the claims within. Synchronisation between the in-memory model and the claim hierarchy is maintained by our tool using events. Claims deemed to be monitorable are associated with a specific event to be fired by a monitor if its data indicates the claim no longer holds. When this event is fired, the claim is falsified on the model, the model is transformed to reflect this fact, and the other claims in the model supported by the now falsified claim are checked to see if they are still sufficiently supported by valid claims, or their own monitoring data. If they are no longer sufficiently supported, they too are falsified by propagation. Finally, the goal model must be evaluated to see if the currently preferred goal satisfaction strategies (i.e. the current configuration) are still justified. If a goal selection strategy is justified only by a falsified claim, another goal satisfaction strategy may be chosen, and the system adapts to a configuration in which the replacement goal satisfaction strategy is utilised. Fig. 1 shows an overview of the approach used at design-time and runtime.

Let's study an example. Fig. 2 depicts a fragment of an i* goal model for a video streaming system. The system uses self-adaptation to vary the bitrate (and thus the video quality) of the streamed video. The adaptation in a system this simple could be achieved parametrically or compositionally, or the behaviour could even be hard-coded. However, our interest lies in modeling the character of, and in controlling the system's self-adaptation rather than in the implementation details of the method by which self-adaptation is effected.

In Fig. 2, The "Encode h264 video" goal can be achieved by either "Encode at 1200kbps" or "Encode at 1800kbps" tasks. Naturally, encoding video at a higher bitrate yields better video quality at the expense of more bandwidth, each modelled as a softgoal. In Fig. 2, the contributions between the tasks and the softgoals are deadlocked, with no clearly preferable goal satisfaction strategy.

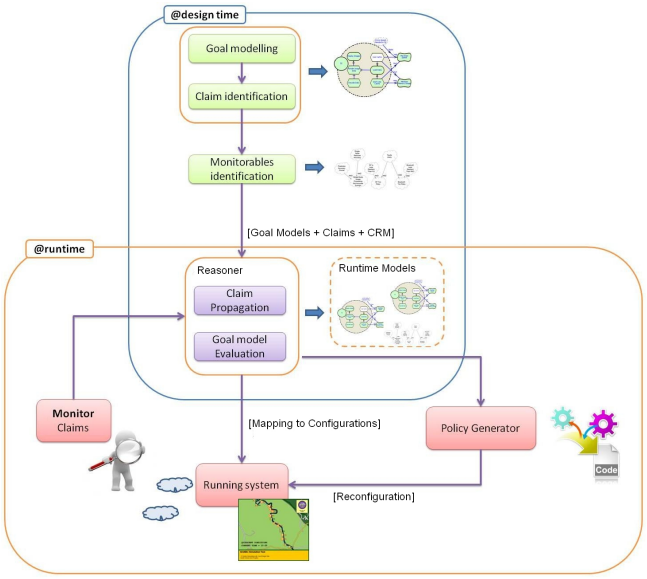

Figure 1. Overview of the Approach

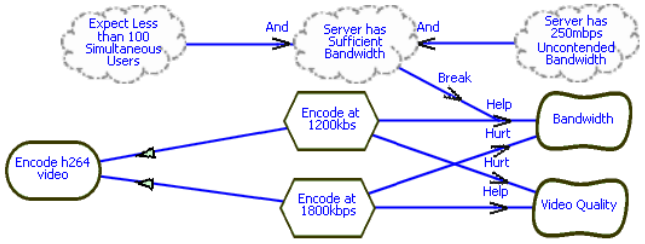

Figure 2. Fragment of video streaming system's i* goal model, with claim hierarchy

The deadlock in Fig. 2 is broken by the "Server has Sufficient Bandwidth" claim which, by virtue of it being connected to the model via its own *Break* link, means that the positive (*help)* contribution "Encode at 1200kbps" makes to "Bandwidth" is disregarded. The claim could equally be attached to the "Encode at 1800kbps" task's *hurt* contribution to the "Bandwidth" softgoal, again with a *Break* link. The "Server has Sufficient Bandwidth" claim is supported by two *AND*-ed claims: "Expect less than 100 Simultaneous Users" and "Server has 250mbps Uncontended Bandwidth". The "Expect less than 100 Simultaneous Users" claim is manifestly shaky, is based on prediction, and as such makes a good candidate for monitoring. It is trivial to imagine a monitor capable of counting the current number of simultaneous users, and should that monitor indicate that the "Expect less than 100 Simultaneous Users" claim is false, an event is fired, and the claim is falsified. In this scenario, the "Server has Sufficient Bandwidth" claim is no longer sufficiently supported, and its link to the model is reversed to become a *Make*. Evaluation of the transformed model will yield that encoding video at 1200kbps is now the preferable goal satisfaction strategy, and an event is fired instructing the system to adapt to use the lower bitrate.

By explicitly modelling known uncertainty about the system and its operating environment as monitorable claims, our approach makes it possible for a system to adapt to various combinations of circumstances (combinations of claims falsified, or otherwise), that weren't explicitly foreseen at design time. Thus, in some limited circumstances, a run-time requirements model-driven self-adaptive system is capable of tolerating uncertainty that was only partially foreseen.

Demonstration: We demonstrate the tools developed and their use for several case studies. Particularly, using our tools, we demonstrate how a self-adaptive system is capable of loading its i* Strategic Rationale model, complete with claims, created in the OME3 [8] i* modelling tool. We showcase the (runtime) model evaluation performed by the system in the event of a monitored claim being falsified, and the resultant self-adaptation performed by the system. Crucially, we show how a system reconfigures to a configuration that was not necessarily defined at design time.

ACKNOWLEDGMENT

This research is partially supported by the Marie Curie Fellowship "Requirements@run.time". We thanks Pete Sawyer for his support.

REFERENCES

[1] K. Welsh and P. Sawyer, "Understanding the scope of uncertainty in dynamically adaptive systems," in REFSQ, 2010. Proceedings of the 16th International Working Conference on Requirements Engineering: Foundation for Software Quality.

[2] L. Chung, B. A. Nixon, E. Yu, and J. Mylopoulos, "Non-Functional Requirements in Software Engineering". Springer, 1999, vol. 5.I.

[3] E. S. K. Yu, "Towards modeling and reasoning support for early-phase requirements engineering," in RE '97: Proceedings of the 3rd IEEE International Symposium on Requirements Engineering (RE'97)

[4] K. Welsh and P. Sawyer, "Deriving adaptive behaviour from i* models" Fourth International i* Workshop, Hammammet, Tunisia, 2010.

[5] W. Robinson, "A requirements monitoring framework for enterprise systems," Requirements Engineering, vol. 11, no. 1, pp. 17 – 41, 2005..

[6] S. Fickas and M. Feather, "Requirements monitoring in dynamic environments," in Second IEEE International Symposium on Requirements Engineering (RE'95), 1995.

[7] K. Welsh , P. Sawyer, N. Bencomo, "Towards Requirements Aware Systems: Run-time Resolution of Design-time Assumptions", 26th IEEE/ACM International Conference On Automated Software Engineering, USA, 2011.

[8] Available at: http://www.cs.toronto.edu/km/ome/.