Experimentation with Self-configuring Systems

Erik Kamsties[1] and Fabian Kneer[1]

[1]University of Applied Science and Arts Dortmund,
Emil-Figge-Strasse 42, 44227 Dortmund, Germany
erik.kamsties@fh-dortmund.de, fabian.kneer@fh-dortmund.de

1 Introduction

The Internet of Things (IoT) has received increased interest over the last years. Objects of the real world (e.g., cars, traffic lights, buildings, mobile phones, etc.) become connected to the Internet. It is expected that self-configuration becomes an increasingly important subject with the IoT.

Self-configuring systems (SCS) adapt their behavior to changes in the environment by dynamically change properties, deploying components, or removing them for instance. For this purpose, SCS use policies provided by a requirements engineer.

SCS can be found in various application areas. Often, they are used in situations, where maintenance activities cannot be carried out by a human. One reason is lack of time, e.g. the user expects instantaneous adaption to his needs, which is the case with service-oriented systems or some kinds of embedded systems.

Runtime representations of requirements (e.g. feature or goal models) are used as policies for SCS. They describe runtime variation points and indicators that can lead to a new configuration.

This poster presents a proposal for experimentation with self-configuring systems targeting at both researchers and practitioners in the field. The contribution from th RE perspective is a sandbox to study requirements at runtime.

2 Poster

Our proposal for experimentation consists of two parts, (1) a framework for prototyping a self-configuring system and (2) a case study for application.

Framework for Prototyping. The framework for prototyping is based on a feedback loop and an analysis of commonalities and differences of the approaches suggested for self-configuring systems, e.g., [1]. It is implemented in the Python programming language as it allows for a wide range of deployments and is frequently used for prototyping (yet not limited to that purpose).

The framework covers two stages (see Fig. 1). First a *development time* stage for *describing* self-configuration requirements and other relevant entities. Second, a *runtime stage* that contains artifacts *executed* at runtime. A *generator* is

available to derive runtime artifacts such as monitoring rules from development time requirements.

A few approaches to self-configuration, e.g., based on feature models, are already implemented and thus can be used out of the box. Further can be added easily using Python.

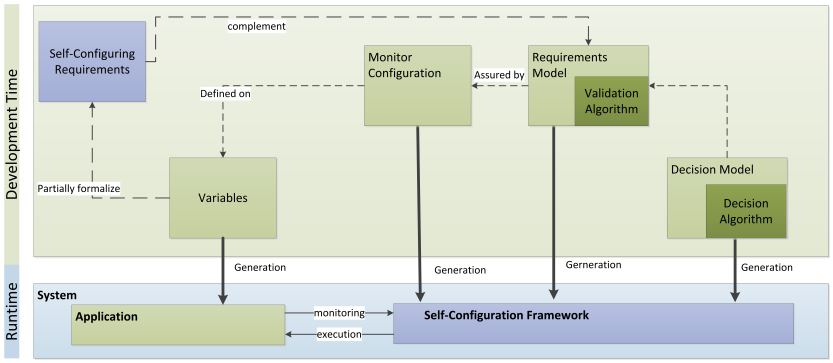

Fig. 1. Framework for Prototyping Self-configuring Systems

Case Study. The aim of the case study is to provide a certain degree of complexity and to be easy to understand by users. We selected the *Smart City* domain as an instantiation of the Internet of Things and *Smart Street Lighting* as a subdomain to start with.

We developed a demonstrator to benchmark algorithms for self-configuration. This demonstrator is a combination of real physical hardware (1:10 model of a street light) and a software simulation to simulate additional street lights. The street light is capable to spot moving vehicles to adapt the lighting and it detects parked vehicles to allow for management of parking spaces.

The street light employs a Raspberry Pi and has LEDs and a couple of sensors. Several street lights can be connected over wireless.

References

[1] N. Bencomo and A. Belaggoun. "Supporting Decision-Making for Self-Adaptive Systems: From Goal Models to Dynamic Decision Networks". In: *Requirements Engineering: Foundation for Software Quality*. Ed. by J. Doerr and A. Opdahl. Vol. 7830. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2013, pp. 221–236.