CADS*: Computer-Aided Development of Self-* Systems

Radu Calinescu and Marta Kwiatkowska

Computing Laboratory, University of Oxford, UK
{Radu.Calinescu, Marta.Kwiatkowska}@comlab.ox.ac.uk

Abstract. We present the prototype tool CADS* for the computer-aided development of an important class of self-* systems, namely systems whose components can be modelled as Markov chains. Given a Markov chain representation of the IT components to be included into a self-* system, CADS* automates or aids (a) the development of the artifacts necessary to build the self-* system; and (b) their integration into a fully-operational self-* solution. This is achieved through a combination of formal software development techniques including model transformation, model-driven code generation and dynamic software reconfiguration.

1 Introduction

The ever growing complexity of today's IT systems has led to unsustainable increases in their management and operation costs. Software architects and developers aim to alleviate this problem by building *self-** (or *autonomic*) *systems*, i.e., systems that self-configure, self-optimise, self-protect and self-heal based on a set of high-level, user-specified objectives [4, 8].

The architecture of a self-* system is depicted in Fig. 1. Given a set of user-specified system objectives (or *policies*), an *autonomic manager* monitors the system components through *sensors*, uses its *knowledge* (i.e., *model* of the system) to analyse their state and to plan changes in their configurable parameters, and implements these changes through *effectors*. In recent work, we introduced a general-purpose implementation of an autonomic manager as a reconfigurable *policy engine* [1], and we described how quantitative analysis methods from the area of probabilistic model checking can be used to implement user-specified policies in a self-* system [2, 3].

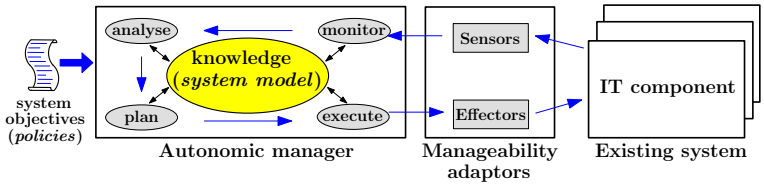

Fig. 1. High-level architecture of a self-* system

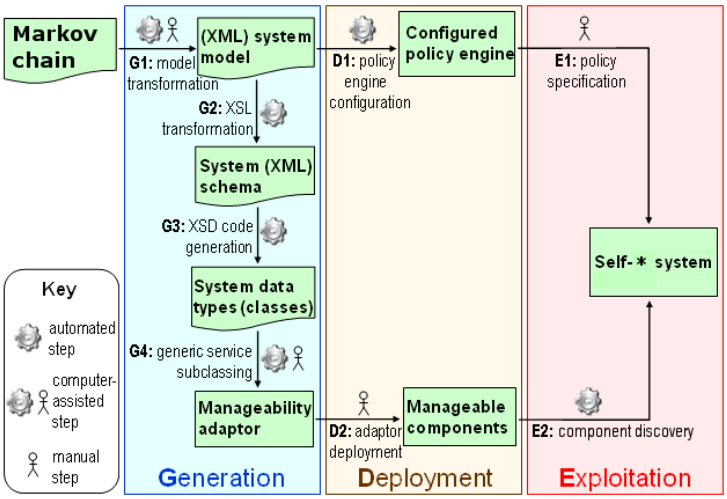

Fig. 2. The CADS* development process

In this paper, we introduce CADS*, a tool for the computer-aided development of self-* systems whose components can be modelled as Markov chains. The tool takes as input a continuous- or discrete-time Markov chain (CTMC or DTMC) describing the behaviour of the IT components to be included in the self-* system. This Markov chain is expressed in the high-level modelling language used by the probabilistic model checker PRISM [5], and may be available already from the verification of the IT components.[1] Alternatively, the Markov chain can be built as described in [6, 5].

Starting from the Markov chain mentioned above, CADS* reduces the effort and expertise required to develop self-* systems by automating or guiding: (a) the generation of the system model used to set up the reconfigurable policy engine from [1]; (b) the generation of the manageability adaptors from Fig. 1; and (c) the configuration of the policy engine for the planned self-* solution.

2 CADS* Development Process

The three-stage development process implemented by CADS* is shown in Fig. 2, and presented below.

Generation stage This stage starts with the developer uploading into CADS* a Markov chain describing the behaviour of the targeted IT components, and expressed in the PRISM modelling language [5]. In a first step (labelled **G1** in Fig. 2), CADS* employs a model transformation to derive an XML-encoded model of the system to be managed, as described by the mapping

$$modelTransformation : MarkovChain \rightarrow SystemModel, \quad (1)$$

[1] PRISM models for a wide range of system components are available from [9].

where $MarkovChain$ and $SystemModel$ represent the set of Markov chains accepted by PRISM and the set of models used to configure the policy engine from [1]. Step **G1** is computer-assisted, i.e., the developer is required to allocate the system parameters that CADS* identifies in the Markov chain to IT components, and to partition them into read-only *state parameters* and read-write *configuration parameters*.

In step **G2**, CADS* uses an XSL transformation to automatically extract an XML schema specification for the targeted IT components from the result of (1):

$$schemaGen : SystemModel \rightarrow XmlSchema. \tag{2}$$

In step **G3**, the tool runs an instance of the XML Schema Definition tool [7] to generate the set of data types associated with the XML schema:

$$dataTypeGen : XmlSchema \rightarrow 2^{DataType}. \tag{3}$$

The result is a set of .NET classes. Finally, we implemented a model-driven code generation module that CADS* uses in step **G4** to automate the generation of web service stubs for the manageability adaptors in Fig. 1:

$$adaptorGen : XmlSchema \rightarrow 2^{ManageabilityAdaptor}. \tag{4}$$

These stubs subclass a generic abstract web service *ManagedResource*<*T*> that implements the bulk of the sensor and effector functionality associated with the manageability adaptor for an IT component (or *resource*). At the end of this computer-assisted step, CADS* requires that the developer implements manually a couple of simple, component-specific methods that are declared `abstract` in *ManagedResource*<*T*>—the work involved is described in [1].

Deployment stage In step **D1**, the developer provides the URL of a running instance of the policy engine from [1], and CADS* calls the appropriate web method to supply the model from (1) to this policy engine. In step **D2**, the manageability adaptors from step **G4** are deployed manually, and configured to access the IT components to which autonomic capabilities are being added.

Exploitation stage In step **E1**, user-specified policies are forwarded by CADS* to the policy engine configured in step **D1**. These policies are specified in the policy expression language described in [2, 1], by using a combination of arithmetic, logic, relational and string operators, and optimisation functions such as Min, Max and Goal to construct well-defined policies as expressions of the system parameters identified in step **G1**. Finally, the policy engine applies these policies to the IT components exposed by the manageability adaptors it discovers automatically in step **E2**.

3 Tool Validation

In order to assess the effectiveness of CADS*, we used it to re-implement two self-* systems that we had previously developed manually in [2, 3]. The first system was a self-configuring/self-protecting system whose objective was to maintain user-specified levels of availability for a set of data-centre clusters. This

objective was achieved by automatically adapting the number of servers allocated to each cluster to changing cluster workloads, priorities and target availabilities. The second system was a self-optimising system involving the dynamic power management of a disk drive, and had as objective the optimisation of user-specified trade-offs between the performance and the power usage of a disk drive exposed to variable workloads.

In both cases, we started from existing PRISM CTMCs from [9], and we successfully devised operational prototypes of the planned self-* system in approximately a tenth of the time taken to develop an equivalent solution manually (i.e., under one day compared to over a week). This gain was primarily due to CADS* reducing significantly the potential for developer error through automating or aiding the development of the self-* system artifacts, and their integration into a fully operational solution.

4 Conclusion

We introduced the prototype computer-aided development tool CADS*, and briefly described how its use in two case studies sped up the development of self-* systems compared to implementing equivalent systems manually. Future work includes augmenting CADS* with the ability to aid users in the specification of valid, non-conflicting system objectives, and to validate the tool further by exposing it to developers with limited expertise in self-* system development.

Acknowledgement This work was partly supported by the UK Engineering and Physical Sciences Research Council grant EP/F001096/1.

References

1. R. Calinescu. Implementation of a generic autonomic framework. In *Proc. 4th Intl. Conf. Autonomic and Autonomous Systems*, pages 124–129, 2008.
2. R. Calinescu. General-purpose autonomic computing. In M. Denko et al., editors, *Autonomic Computing and Networking*. Springer, April 2009.
3. R. Calinescu and M. Kwiatkowska. Software engineering techniques for the development of systems of systems. In *Proc. 15th Monterey Workshop on Foundations of Computer Software*, pages 86–93, 2008.
4. J. O. Kephart and D. M. Chess. The vision of autonomic computing. *IEEE Computer Journal*, 36(1):41–50, January 2003.
5. M. Kwiatkowska et al. Quantitative analysis with the probabilistic model checker PRISM. *Electronic Notes in Theoretical Computer Science*, 153(2):5–31, 2005.
6. M. Kwiatkowska et al. Stochastic model checking. In M. Bernardo and J. Hillston, editors, *Formal Methods for the Design of Computer, Communication and Software Systems: Performance Evaluation*, pages 220–270. Springer, 2007.
7. Microsoft Corporation. Xml schema definition tool (xsd.exe), 2007. http://msdn2.microsoft.com/en-us/library/x6c1kb0s(VS.80).aspx.
8. M. Parashar and S. Hariri. *Autonomic Computing: Concepts, Infrastructure & Applications*. CRC Press, 2006.
9. PRISM Case Studies. http://www.prismmodelchecker.org/casestudies.