Fast Abstract: Assessing software designs by simulation

Alfredo Capozucca
Laboratory for Advanced Software Systems
University of Luxembourg
6, rue Richard Coudenhove-Kalergi, Luxembourg
alfredo.capozucca@uni.lu

Abstract

This paper presents an approach to assess the design of a software system which is modelled using the Coordinated Atomic Actions (CAA) paradigm. The approach relies on simulation to make possible the assessment of the design. By simulating the design, engineers can assess its adherence to the requirements elicited in previous phases of the software development life cycle.

1. Introduction and Motivation

The Coordinated Atomic Actions paradigm [5] was defined to enclose the concepts of *conversations*, *transaction processing*, and *exception handling* around the notion of *atomic action*. This paradigm has proven [1] to be useful when designing distributed software systems meant to hold dependability-related requirements.

However, despite of relying on a paradigm like CAA, which considers dependability as a first-class citizen, ensuring that a particular design satisfies the requirements (either related with dependability or not) is a challenging task that engineers must face with.

The proposal then is to make use of simulation as means to assess the correctness of a CAA-oriented design model with respect to a predefined set of requirements. The approach then consists of simulating a CAA-oriented model M_{CAA} under certain initial conditions s_{init} specified as desired by the engineer. The simulation of M_{CAA} initialised in s_{init} produces results that determine how such a model behaves under the given conditions.

The engineer then, based on the results produced by the simulation, will decide whether the current model M_{CAA} satisfies the elicited requirements. In case of discrepancies then the engineer may conclude that the model M_{CAA} is incorrect. This conclusion is made under the hypothesis that the requirements are correct. The engineer then, after concluded that the design model is incorrect, will have to modify such a model and rerun the simulation. This process has to be done until the engineer concludes that the model M_{CAA} satisfies the elicited requirements.

The hypothesis on the correctness of the elicited requirements is grounded on the assumption of using the Dependability-Oriented Requirements Engineering Process (DREP) [4] to elicit the requirements. Moreover, it has been found empirical evidence [2] of good alignment between DREP as requirement elicitation process and CAA as design paradigm.

2. tCAA

The presented software design assessment approach targets CAA-oriented models. Thus, one of the engineers' key activities is to create such models. To facilitate this activity, a domain specific language (DSL) called *tCAA* has been defined (the *t* stands for *time*). The aim at defining this DSL is to allow engineers to perform the modelling using only terms of the CAA paradigm.

The *tCAA* DSL includes all the extensions [3] that have been proposed since the CAA paradigm was introduced for the very first time. In particular, the DSL includes new time-related extensions. These extensions were proposed because real-time software systems are part of the solution space targeted by the CAA paradigm: real-time software systems are either inherent or imposed concurrent and very often have dependability requirements [2].

3. Simulation environment

A programmer usually does not wait until he considers the program is correct to execute it. Actually, he executes

[1] For a full list of facts see `http://homepages.cs.ncl.ac.uk/alexander.romanovsky/home.formal/caa.html`.

[2] Experiments carried out during the practical sessions of the *Dependably Systems* masters course (summer 2011 and 2012) at the University of Luxembourg.

the program many times before concluding it is correct. This iterative execution of the program lets the programmer gain confidence and ensure that things are going as expected. The purpose of using simulation is to achieve the same functionalities available for a programmer when coding, but at the level of the design phase. Therefore, by simulation, an engineer should be allowed to exercise a CAA-oriented model to understand how it behaves under certain conditions. The goal is to create a design whose behaviour satisfies the elicited requirements.

3.1. DEVS

The Discrete Event System Specification (DEVS) [6] formalism is considered the most general conceptual framework to model discrete event systems. Any system that performs a finite number of changes in a finite interval of time can be modelled with this formalism.

However, beside the generality of the formalism, the main reasons that make DEVS interesting to be used as simulation framework for CAA-oriented models are:

- the possibility to represent discrete time systems: this feature is crucial as CAA-oriented models might have to satisfy time-related requirements,
- the representation of complex systems by the composition of atomic DEVS: the core of the CAA paradigm is a block (called CAA) that not only groups a set of operations spread over several process, but also provides effective fault containment and controlled access to shared transactional resources. Complex software systems are designed by coupling different CAAs. Thus, both DEVS and the CAA paradigm share the same structuring principles,
- the existence of software tools with advanced simulation features [1]: there is not need to perform any development to achieve the simulation of DEVS models. Moreover, the existence of these tools is clear evidence that the formalism is well founded and spread.

The last point deserves further clarification. To exploit any of the existing tools to simulate DEVS, a CAA-oriented model M_{CAA} has to be "translated" to a DEVS model M_{DEVS}. This translation then will not only allow engineers to use any of the existing simulation tools associated to DEVS, but also provide a clear semantic definition of the *tCAA* DSL modelling language. Thus, this translation will bring both usefulness and formality to *tCAA* DSL.

4. Toolset

Both the *tCAA* DSL and the transformation that allows obtaining a DEVS model must be supported by tools. These tools must provide means to ensure the easy definition of CAA-oriented models, as well as their simulation. It has been decided to use the Eclipse ecosystem as implementation platform. The main reason for this choice is the extensive offer both to develop graphical and textual editors and manipulate models. Thus, a toolset integrated within the Eclipse environment would support the work of the engineers when modelling and simulating CAA-oriented designs. Notice, that the required process to obtain the simulation of a CAA-oriented model is hidden to the engineers as the translation of CAA-oriented models to DEVS is achieved automatically by applying model transformation principles and reusing any of the existing DEVS simulators. Therefore, engineers are not required to have any prior knowledge on DEVS to exploit the presented modelling approach and its associated tools.

5. Conclusions

The clear advantages of the presented software design assessment approach are: (1) self-contained analysis of CAA-oriented models: engineers do not need to rely on third-party formalisms to analyse their models (i.e. both the modelling as the assessment in done in terms of the CAA paradigm), and (2) early analysis of a model: the simulation of a (not necessary complete) model lets engineers get early feedback about the decisions made (rather than using simulation to assess the model, it is used to drive the definition or enhance of the model).

References

[1] F. Bergero and E. Kofman. Powerdevs: a tool for hybrid system modeling and real-time simulation. *Simulation*, 87(1-2):113–132, Jan. 2011.

[2] A. Burns and A. J. Wellings. *Real-Time Systems and Programming Languages: ADA 95, Real-Time Java, and Real-Time POSIX*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2001.

[3] A. Capozucca. *DT4BP: A Business Process Modelling Language for Dependable Time-Constrained Business Processes*. PhD thesis, University of Luxembourg (Thesis PhD-FSTC-34-2010), December 2010.

[4] S. Mustafiz. *Dependability-oriented model-driven requirements engineering for reactive systems*. PhD thesis, Montreal, Quebec, Canada, 2010.

[5] J. Xu, B. Randell, A. Romanovsky, C. M. F. Rubira, R. J. Stroud, and Z. Wu. Fault Tolerance in Concurrent Object-Oriented Software through Coordinated Error Recovery. *Proceedings of the 25 International Symposium on Fault-Tolerant Computing*, pages 499–508, 1995.

[6] B. P. Zeigler, T. G. Kim, and H. Praehofer. *Theory of Modeling and Simulation*. Academic Press, Inc., Orlando, FL, USA, 2nd edition, 2000.