Verification of Smart Homes Specifications which are based on ECA Rules

Juan C. Augusto

School of Computing and Mathematics,
University of Ulster at Jordanstown, UK

Abstract. Smart homes implementations are usually based on Active Databases (ADBs). A core concept of ADBs is the concept of Event-Condition-Action (ECA) rules allowing the system to react to specific events occurring in contexts of interest and advising on an the actions that should be taken in those situations. Although research in ADBs has been conducted for quite a few years, still no standard verification framework has emerged yet from the area. In this paper we consider some options to verify specifications of Smart Homes based on ADB-related concepts.

1 Introduction

Smart homes implementations are usually based on Active Databases (ADBs) [7]. A core concept of ADBs is the concept of Event-Condition-Action (ECA) rules allowing the system to react to specific events occurring in contexts of interest and advising on an the actions that should be taken in those situations. Although research in ADBs has been conducted for quite a few years, still no standard verification framework has emerged yet from the area. In this paper we consider some options to verify specifications of Smart Homes based on ADB-related concepts.

ECA rules have the following general format: `ON` *event* `IF`*condition* `DO` *action*. The ECA rule specification language we consider (see [1] for BNF definition) has been formally defined [5] and its interpreter has been implemented and tested (see [6]). Galton's proposal considers a set of operators that is mainly focused on events, states and their interaction. We use this as the basis for the specification of ON and IF clauses. Primitive/complex events and calendar-related functions can be used to refer to time constraints within the ECA rules.

The information generated from the sensors within the house and the interfaces embedded into common domestic appliances can be processed by the ADB manager to provide assistance to the person in a number of different ways; prevention of dangerous situations (e.g., leaving food over the cooker for a long time), comfort (e.g., regulation of temperature according to seasons), security (e.g., detecting intruders in the vicinity) and health (e.g., contacting medical personnel after using a self monitoring device).

2 ECA Rules Verification

Our system deals with a wide range of situations of interest including safety and health related issues. That is an extra motivation to ensure correctness of the ECA rules as

C. Augusto J. (2005).
Verification of Smart Homes Specifications which are based on ECA Rules.
In *Proceedings of the 3rd International Workshop on Modelling, Simulation, Verification and Validation of Enterprise Information Systems*, pages 118-120
DOI: 10.5220/0002574701180120

a whole. There have been some reports on similar attempts in the literature, see for example [4], by using planning systems. While we address some of the properties considered in [4], here we offer an alternative solution to the problem that is closer to our framework and the methods and tools used in Software Engineering for verification and validation of software [2].

We are mainly focused on checking semantic correctness of the set of ECA rules as a whole. We consider here a list of properties whose verification can be automised in our framework and are important to obtain a more reliable set of ECA rules: a) *Consistency:* the actions triggered at each step should not be inconsistent. A separate check is performed to consider consistency of the incoming actions compared with the existing DB (but this second consistency check is undertaken at run time) (see [1]). b) *Triggerability:* this checks if each ON clause specification is such that it is always/sometimes potentially triggerable. c) *Joint applicability:* this checks if triggerability is possible for any subset. Due to the computational complexity of this goal N-Joint applicability can be explored meaning that only possible combinations of up to N rules are explored for joint applicability. d) *Rule coverage:* this checks if there are pairs (R1, R2) of rules such that the triggering conditions of R1 is a subset of the triggering conditions of R2. e) *Rule cascading:* this checks if there are any forced sequences of rule activations imposed by design. f) *Postcondition satisfaction:* this checks if a given postcondition is achieved after activation of a rule. An example illustrating this situations can be seen in section 6.2 of [1].

We use an interpreter of the language proposed in [3] where a fragment of MTL, called *MTL-programs* is mapped to a constraint logic programming framework which admits efficient satisfiability checking of the constraints generated. Complexity is in the worse case linear in the number of variables involved. On the other hand, MTL offers a language which is expressive enough for us to map the language of our ECA rules into and to write the queries that allow us to check the properties described previously. We further restrict the expressiveness of *MTL-programs* by discarding the past fragment.

Due to space restrictions we do not provide full details of this theoretical framework. Here we instantiate the general framework offered in [3] to a specific case study and temporal domain so formulas given below will refer to meaningful predicates used in the ECA rules and metrics will be referred to calendar dates. Although the initial framework considers an unbounded temporal structure we have natural initial and final times given by the time where the system started and the present, respectively. Also, before starting the verification process, the ECA rules have been translated and labelled so that a rule `ON E IF C DO A` will produce an *MTL-program-F*-based clause $do(\text{RuleID}, A) \leftarrow E \wedge C$. So that the rule will be identified by a unique RuleID.
Actions inconsistency, returns the IDs of rules which are conflicting in any sense that can be specified by using a predicate *conflictingActions*:

$$\Box(checkInconsistency(RuleID1, RuleID2) \leftarrow \\ do(RuleID1, A1) \wedge do(RuleID2, A2) \wedge conflictingActions(A1, A2))$$

Further details on how to deal with the other properties can be seen in section 6.3 of [1].

3 Conclusions

The notions of ADBs and ECA rules mixed with Temporal Reasoning can provide a very useful framework to implement monitoring software for Smart Homes. There are many potential applications of the concept of Smart Homes which are beneficial to society. We based our presentation in one which is focused on health issues but the concept of smart buildings, or smart environments, can be applied with advantages in many ways (e.g., schools, train stations, etc.).

Despite the importance of the possible applications of Smart Homes and of the concept of ADBs, there is no standard methodology and tools which can be easily applied to verify the correctness of a set of ECA rules. Although some previous work has been conducted we think there is still much to do in order to provide the development teams an environment where to develop software for this kind of applications in a more systematic and safe manner.

References

1. Juan Carlos Augusto and Chris Nugent. The use of temporal reasoning and management of complex events in smart homes. Technical report, School of Computing and Mathematics, University of Ulster, UK, 2004. (`http://www.infj.ulst.ac.uk/~jcaug/TR-AugustoNugent2004a.pdf`).
2. B. Berard, M. Bidoit, A. Finkel, F. Laroussinie, A. Petit, L. Petrucci, Ph. Schnoebelen, and P. McKenzie. *Systems and Software Verification (Model Checking Techniques and Tools)*. Springer Verlag, 1999.
3. Christoph Brzoska. Temporal logic programming with metric and past operators. In *Executable and Temporal Logics, IJCAI'93 Satellite Workshop*, pages 21–39. Springer Verlag, 1993.
4. P. Fraternali, E. Teniente, and T. Urpi. Validating Active Rules by Planning. In *Proceedings of the 3rd International Workshop on Rules in Database Systems*, volume 1312, pages 181–196. Springer, 1997.
5. A. Galton. Eventualities. In Vila, van Beek, Boddy, Fisher, Gabbay, Galton, and Morris, editors, *The Handbook of Time and Temporal Reasoning in Artificial Intelligence*. MIT Press, 2004. (to be published).
6. Rodolfo Gómez, Juan Carlos Augusto, and Antony Galton. Testing an Event Specification Language. In *Proceedings of the 13th. Int. Conf. of Software Engineering and Knowledge Engineering (SEKE 2001)*, pages 341–346, Bs.As., Argentina, 2001.
7. N.W. Paton and O. Diaz. Active Database Systems. *ACM Computing Surveys*, 31(1):63–103, 1999.