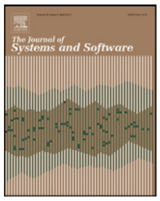
The Journal of Systems and Software 85 (2012) 2829–2839

Contents lists available at SciVerse ScienceDirect

The Journal of Systems and Software

journal homepage: www.elsevier.com/locate/jss



Analysing monitoring and switching problems for adaptive systems

Mohammed Salifu[a], Yijun Yu[a,*], Arosha K. Bandara[a], Bashar Nuseibeh[a,b]

[a] Department of Computing, The Open University, UK
[b] Lero, University of Limerick, Ireland

ARTICLE INFO

Article history:
Received 30 April 2011
Received in revised form 15 July 2012
Accepted 17 July 2012
Available online 31 July 2012

Keywords:
Monitoring
Switching
Problem description
Requirements engineering
Self-adaptive systems

ABSTRACT

In the field of pervasive and ubiquitous computing, context-aware adaptive systems need to monitor changes in their environment in order to detect violations of requirements and switch their behaviour in order to continue satisfying requirements. In a complex and rapidly changing environment, identifying what to monitor and deciding when and how to switch behaviours effectively is difficult and error prone. The goal of our research is to provide systematic and, where possible, automated support for the software engineer developing such adaptive systems.

In this paper, we investigate the necessary and sufficient conditions for both monitoring and switching in order to adapt the system behaviours as the problem context varies. Necessary and sufficient conditions provide complementary safeguards to ensure that not too much and not too little monitoring and switching are carried out. Our approach encodes monitoring and switching problems into propositional logic constraints in order for these conditions to be analysed automatically using a standard SAT solver.

We demonstrate our approach by analysing a mobile phone system problem. We analysed requirements violations caused by changes in the system's operating environment. By providing necessary and sufficient monitoring and switching capabilities to the system, particular requirements violations were avoided.

1. Introduction

In pervasive and ubiquitous computing (Bettini et al., 2010), many software systems are required to be context-aware and self-adaptive (Weyns et al., 2012). They need to monitor the changes in their environment in order to detect violations of their core requirements; they also need to switch their behaviour in order to continue satisfying those requirements. *Monitoring* requirements define the activities needed to detect changes in operating environments that lead to requirements violations. *Switching* requirements define activities needed to adapt system behaviour to restore the satisfaction of such requirements. In general, *context-aware* requirements define the self-adaptive activities needed to compose monitoring and switching activities in order to maintain the satisfaction of the original requirements.

Monitoring and switching have been recognised in research on adaptive mechanisms of autonomic, ubiquitous and self-managing software systems (Abowd, 1999; Georgiadis et al., 2002; Grimm et al., 2004; Zhang and Cheng, 2006a) such as in the MAPE (Monitoring, Analysing, Planning, Executing) loop proposed for autonomic computing (Abowd, 1999; Kephart and Chess, 2003). Current research on context-aware systems, however, focuses mostly on issues such as user-device interaction design, systems architectures and implementation (Abowd, 1999). An early analysis of the impact of contextual changes on requirements satisfaction and the analysis of the requirements of monitoring and switching problems have not been fully explored. Furthermore, in a complex and rapidly changing environment, identifying what to monitor and deciding when and how to switch behaviours effectively can be difficult and error-prone.

Therefore, an open research question addressed in this paper is, *given requirements that adaptive software must monitor contexts and switch its solution alternatives, what are the respective necessary and sufficient conditions for the monitoring and switching actions?* The goal of our work is to investigate necessary and sufficient conditions for both monitoring and switching in order to adapt the system behaviours as the context varies. Necessary and sufficient conditions provide complementary safeguards to ensure that neither too much and nor too little monitoring and switching are carried out.

Our theoretical framework to derive these conditions for the context-awareness requirements can reduce monitoring overhead by ensuring that fewer contextual variables are monitored. It can also switch the system to a solution that can offer better quality requirements to meet the preferences of a given tradeoff. Unnecessary switching is also avoided, especially when the available solutions do not provide any enhancement of the quality requirements as defined in the tradeoff. Overall, our problem-oriented approach enables a software engineer to: (1) represent and reason

* Corresponding author.
E-mail address: y.yu@open.ac.uk (Y. Yu).

0164-1212/$ – see front matter © 2012 Elsevier Inc. All rights reserved.
http://dx.doi.org/10.1016/j.jss.2012.07.062

about changes in the physical environment of software systems and assess their impact on requirements satisfaction; (2) specify monitors and switchers that can detect changes and adapt in response to requirements satisfaction violation; and (3) ensure that the conditions for monitors and switchers are both necessary and sufficient.

According to Kramer and Magee (2007), such an early analysis of problems is necessary to both the development and validation of self-managing systems. Self-management is an inherent characteristic of context-aware self-adaptive systems (Hinchey and Sterritt, 2006). Also, the explicit analysis of systems' operating environment is fundamental to assessing the continuous satisfaction of requirements (Cheng and Atlee, 2007). Fig. 1 presents Kramer et al.'s view of self-adaptive systems as a three-layered architecture, in which the top layer concerns the *goals* of the adaptation, the middle layer concerns the *plans* of the adaptation, and the bottom layer concerns the *components* that need to be reconfigured to support the adaptation. Our work focuses on the top two layers of the architecture that address the problem space, representing the relations among contextual information (W), requirements (R) and the specifications of the required solution (S) as a logic entailment $W, S \vdash R$.

Shifting focus from design activities to earlier requirements analysis, we build on problem-oriented approaches (Jureta et al., 2009; Jackson, 2001; Sutcliffe and Maiden, 1998) to describe three kinds of artefacts in adaptation: requirements goals, switching plans and contextual components that may impact system behaviour (i.e. business logic). We also adopt the annotation-based approach in Problem Frames for UML (Côté et al., 2011) to introduce stereotypes on states of UML statecharts for eliciting context variables and annotate the transitions of the statecharts for contextual dependencies. Then we use a standard SAT solver for automated analysis.

The paper is an extension and validation of our earlier work on the use of a problem-oriented approach to analyse the requirements for monitoring and switching problems under varying contexts (Salifu et al., 2007). The novelty of the extension is an additional assessment of necessity and sufficiency of the derived monitoring and switching behaviours. By analysing a mobile phone system problem, we demonstrate how such a framework can be applied to analysis of its context-awareness and self-adaptive requirements, and measured the improvement of the proposed approach on the simulated monitoring and switching behaviour of the self-adaptive system combining the Java implementation with SAT solvers.

The remainder of the paper is structured as follows: we begin with a description of related work in Section 2 that presents fundamental concepts and their application to our work and related work on monitoring and switching. Section 3 presents our approach in two parts – a general description and its application. In Section 4 we provide a detailed illustration of our approach's application to a case study to show its usefulness in avoiding requirements violations caused by contextual changes. We conclude with a discussion of our approach in Section 5 and some final remarks in Section 6.

2. Related work

2.1. Representation of the context-awareness requirements

In their four-variable monitoring model, Peters and Parnas refer to the monitoring activities that focus on internal state change as *software monitors* and those focusing on the external environmental states change as *system monitors* (Peters and Parnas, 2002). For context-awareness we are interested in system monitors that detect the changes requiring variation in software behaviour that are caused by environmental changes. Zave and Jackson (1997) describe software problems as consisting of three kinds of artefacts – contextual information about the physical world (W), requirements (R) and the specifications of the required solution (S), which are logically related as the entailment $W, S \vdash R$. That is, the use of specification S in context W ensures the satisfaction of the requirements R. Detailed analysis of contexts is important for analysing monitoring and switching problems. It is necessary to distinguish physical and other types of contexts to specify system monitoring and switching (Sutcliffe and Maiden, 1998).

Ali et al. (2010) represent context information as a set of variable constraints organised in a refinement hierarchy similar to that of goal models, and combine the analysis of goal satisfactions with the contextual constraints. While goal-based approaches emphasise the refinement of R and arguably S, they are rather weak in the refinement of W (Hall et al., 2007) because (a) the development of goal-trees is driven by goal refinements; and (b) leaf level goals and agents that represent some part of S and W are terminal, which make their further refinement implausible. Mapping the contextual variables to the goal refinement is a task requiring analysts to check $m \times n$ possible relations where m is the number of refinements in the goal model, and n is the number of context variables. On the other hand, the Problem Frames approach (Jackson, 2001) describes both optative and indicative properties of W and relates them to the requirements R and the solutions S. Therefore, we choose this representation of contextual properties for analysing the impact of W on both satisfaction of requirements R and context-awareness of S. Comparing to Ali et al. (2010) we do not require extra mappings between goals and contexts since they are already present as a by-product of the problem context diagrams.

2.2. Requirements-driven analysis

Current monitoring approaches on requirements failure diagnostics focus on execution failures. Example work on failure diagnostics is the ReqMon approach (Robinson, 2002, 2005) that claims to exhaustively investigate all failure sources. Other work on failure diagnostics includes the work of Wang et al. (2009) that identifies and defines possible failure points thereby restricting their diagnostic analysis space at run-time. What these approaches have in common is their focus on the execution failure as the triggering event for diagnostics. This is insufficient in the case of context-awareness (Feather et al., 1998; Fickas and Feather, 1995) because the software execution may only seem to satisfy the requirement according to invalid assumptions about the environment. Whittle et al. have developed a declarative style requirements language for specifying both the variables to be monitored and components for adaptation (Whittle et al., 2009). In comparison, their language expresses the degraded requirements in logic rules using fuzzy operators, while our approach makes use of partial fulfilment labels to encode the different degrees of satisfaction of quality requirements as crisp proposition literals, such as Fully Satisfied (FS), Partially Satisfied (PS), Partially Denied (PD), and Fully Denied (FD). In both languages, since the adaptation is declarative, an exhaustive specification of different adaptation behaviours is not required. Furthermore, uncertainties in the system context are explicitly specified and dynamically managed during system execution. The dynamic management of uncertainties may be used to diagnose and prevent systems from failures in unexpected situations. The advantages of avoiding the fuzzy operators, in our case, help the runtime decisions based on the propositions to be more deterministic, while some degree of uncertainty is tolerated by the rules with partial satisfaction and denial encoding of propositions.

Concerning switching activities, existing approaches (Zhang and Cheng, 2006b,a) primarily focus on the collaboration of multiple threads at runtime (Brown et al., 2006; Cheng et al., 2006; Janik