Analysing Monitoring and Switching Requirements using Constraint Satisfiability

Mohammed Salifu
Yijun Yu
Bashar Nuseibeh

20th March 2008

Department of Computing
Faculty of Mathematics and Computing
The Open University
Walton Hall,
Milton Keynes
MK7 6AA
United Kingdom

http://computing.open.ac.uk
Abstract
Context-aware applications monitor changes in their environment and switch their behaviour in order to continue satisfying requirements. Specifying monitoring and switching in such applications can be difficult due to their dependence on varying environmental properties. Two problems require analysis: the monitoring of environmental properties to assess their impact on continual requirements satisfaction, and the selection of appropriate behaviours that ensure requirements satisfaction. To address these problems, we provide concepts for refining contextual properties which we then use to formulate two theorems for monitoring and switching. These enable us to formally analyse the impact of context on monitoring and switching problems. We have instantiated our general approach by encoding monitoring and switching problems into propositional logic constraints, which we analyse automatically using a standard SAT solver. The approach is applied to an industrial case study to assess its efficacy.

Keywords
Problem descriptions; variant problems; monitoring problems; switching problems; context-awareness; constraint satisfiability

1. Introduction
Context-aware applications monitor changes in their operating environment and switch their behaviour in order to continue to satisfy requirements [19, 21]. Therefore, they must be equipped with the capability to monitor environmental changes and to adapt (switch) their behaviour. Monitoring requirements define what context-aware applications must do to detect changes in their operating environment that lead to requirements violations, while switching requirements define what they must do to adapt their behaviour to restore the satisfaction of such requirements.

Specifying monitoring and switching in context-aware applications can be difficult due to their dependence on varying environmental properties which, if not systematically analysed, may lead to unforeseen requirements violations. There is a need to analyse the problems of: (1) monitoring all variables in the changing environment to assess their impact on continual requirements satisfaction; and (2) that of selecting an appropriate behaviour to restore requirements satisfaction when they are violated.

Kramer and Magee [16] have suggested representing higher level software application requirements of context-aware applications as goals, in order to provide alternative ways of achieving system behaviour during run-time. Cheng and Atlee [3] have recognised the need for requirement engineering that considers not only normal environmental behaviour but also other possible threats and hazards of the operating environment. These two issues are critical in the case of developing context-aware applications due to their dependency on the operating environment and the difficulty of eliciting alternative system behaviours for such environment.

Although monitoring and switching have been recognised in Monitoring-Analyzing-Planning-Executing (MAPE) adaptive mechanisms [15, 24] of traditional autonomic and ubiquitous computing research [1, 15], the impact of varying environmental properties has not been investigated. For example, although monitoring and switching behaviours are often considered [1, 8, 9, 33], analysing the impact of varying contextual properties within the problem space remains unaddressed.

In earlier work [25, 26], we proposed a problem-oriented approach for analysing the relationships between contextual properties and both the satisfaction of requirements and the activities of monitoring and switching. By contextual properties, we mean the key properties of the environment of applications that may or should have an impact on its execution. Our approach is focused on analysing individual problems in different contexts [25] and how they are used to characterise monitoring and switching problems [26]. Although, we formalised relationships between individual problems, and did so systematically, our analysis of the impact of contextual properties on monitoring and switching behaviour was informal and manual. The contribution of this paper is a more formal analysis of the impact of varying contextual properties on monitoring and switching behaviour. We characterise concepts for refining contextual properties that we then use to formulate and prove two theorems for monitoring and switching. These theorems define the necessary impact criteria for monitoring a contextual variable, and for switching application behaviour to address a different problem. The necessary criteria refer to the effect on the satisfaction of requirements by either the variable changing states or the application switching behaviour.

We have instantiated our general approach by encoding monitoring and switching problems as propositional logic constraints, which we then analyse using our automated analysis tool based on a standard SAT solver (SAT4J [17]). We begin validating our approach by applying it to an industrial case study, which surfaces hidden assumptions about applications’ operating environment that may lead to unforeseen requirements violations, and identifies possible mitigation activities.

The remainder of the paper is structured as follows: we begin in Section 2 with a discussion of related work in variability, monitoring and switching problems analysis. This is followed by an overview and discussion of our (systematic but informal) approach to problem analysis of context-awareness in Section 3, upon which the more formal approach of this paper is based. We present a description of our approach in Section 4. Our automated analysis, which uses an instantiation of the general approach, is discussed in Section 5. Section 6 presents the application of the
approach to a hypothetical device mobility problem and a real stock item movement problem in the logistics domain. We conclude in Section 7 with an agenda for further work.

2. Related Work
We briefly discuss two areas that have a direct bearing on our research: representations of variation points and dependency relations between variants; and approaches to context-awareness.

2.1 Variability and Dependencies
Variations in requirements are generally regarded as variations in the intentions of stakeholders in their use of an end product [2]. This type of variability has often been modelled and analysed using feature diagrams [20, 28], which capture user-visible functionalities. However, Liaskos et al. [18] have observed that variability in requirements may be exacerbated by contextual variability which they refer to as background (or unintentional) variability. They argue that feature diagrams do not take contextual variability into consideration and are therefore unsuitable for representing and reasoning about variability of systems where contextual changes are common place. In earlier work, we therefore proposed the use of problem descriptions to capture different problems for different contexts [25].

2.2 Monitoring and Switching
The need to monitor application software operating environments to assess the continual satisfaction of requirements has been recognised by Fickas and Feather [7]. The primary focus of their work is on the assumptions about the operating environment whose failure is likely to invalidate the requirements. They have proposed the use of two kinds of parameters to store monitored and controlled variables respectively [6]. Monitored parameters provide the means of detecting changes that cause assumptions failures and controlled parameters provide a means to adapt application software behaviour. In addition, linear temporal logic (used, for example in KAOS [5]) may be used to express these parameters and to show their interdependency. Robinson [22] has also recognised the need to monitor application software and to assess their continual satisfaction of higher level goals. The primary objective of this approach is to detect inconsistency in goals resulting from changing operating environment and to take corrective measures to restore them. Robinson has proposed the `framework [23] for defining the core requirements and the monitoring behaviours with associated software development tools. The focus of these approaches is on monitoring requirements satisfaction based on the outcome of application execution. Therefore, unforeseen requirements violations caused by changes in the physical context may remain unnoticed.

Zhang and Cheng [33] have proposed general switching semantics without giving details on how such semantics relate to physical contextual changes. Hence, we have proposed the systematic analysis of the impact of context on requirements satisfaction [25] and the use of context monitoring and application behaviour switching as mitigation activities [26].

SAT solvers have been used by Wang et al. [31] for the automated analysis of monitors for internal system state for fault detection and diagnostics. However, this is limited by its inability to detect requirements violations caused by contextual changes. Also, switching is not address in their approach.

Although, we formalised relationships between individual problems [25], our analysis of the impact of contextual properties on monitoring and switching behaviour [26] was informal and manual. This makes tool support and automated analysis of monitors and switchers difficult. The contribution of this paper is a more formal analysis of the impact of contextual properties on monitoring and switching behaviour.

3. Problem Analysis of Context-Awareness
We have adopted a problem-oriented approach based on Jackson’s Problem Frames (PFs) [14] in order to represent monitoring and switching problems. In this section we provide an overview of our approach upon which we base our formal analysis in later sections. We begin the section with a brief account of the PF approach and illustrate its associated notation as we present our approach.

3.1 Background: the world and the machine
Goals [30], use cases and scenarios [4] are often used to represent and analyse requirements. While these approaches have concepts for capturing the context of software applications, they do not distinguish between physical context and other types of contexts [29]. Also, they do not make an explicit differentiation between context, specification, and requirements. One approach that does make such a distinction is Jackson’s Problem Frames (PF) approach. Given that the focus of our research is on analysing the impact of physical contextual properties on requirements satisfaction and context-awareness, we have found that the PF approach to problem description appropriate. It enables us as to capture physical contextual properties and to separate requirements from specifications and context.

Jackson proposes three descriptions of a problem: (1) a description of the context in which the problem resides in terms of known domain properties of the world, denoted as the world \( W \); (b) a description of the required properties of the world, denoted as the requirement \( R \); and (c) a description specifying what the computer system running the software-to-be must do to meet the requirements, denoted as the specification \( S \). These three descriptions are related by the formula [11, 32]:

\[ W, S \models R \]  

(1)

where the symbol “\( \models \)” denotes entailment; that is, the satisfaction of \( S \) in \( W \) entails that of \( R \).

3.2 Overview of Our Approach
Given initial requirements, we use domain knowledge about the varying operating environment to identify a set of contextual variables as possible sources of variations. Contextual variability refers to changes in the elicited variables that may cause requirements satisfaction violations. We derive different problems for different contexts from which we then derive problems and context to derive monitoring and switching problems. The analysis at this stage is systematic but informal and focused on the problem of monitoring individual variables and switching between two variants. Following this, an informal sketch of the context-awareness problem is derived, which is subsequently refined following the more formal analysis (Section 4). The output from the formal analysis is a monitoring and switching regime.
which is used in the refinement. Appendix A gives further high level overview of our approach.

### 3.2.1 Deriving Variant Problems

Variability is a space of alternatives confined by all possible values of a set of variables. In the problem space, variability in stakeholder goals can be represented by combinations of OR-decompositions, variability in product lines by optional and alternative features, and variability in problem diagrams by so-called variant problems. The notion of variability can be further extended for context-aware applications.

The core requirements of context-aware applications are defined as the requirements that the system needs to satisfy in every significant context. In a varying operating environment, contextual variability is defined as a subspace of variability that may result in different behaviours such that the core requirements of the application are maintained. For example, when two mobile devices are used in some locations, to secure image exchanges, a dashed oval represents the requirement (e.g., Secured image application are maintained. For example, when two mobile devices are used in some other locations. On the other hand, the variability in image formats may not demand any different exchange behaviour. Figures 1 and 2 show problem descriptions of secured and unsecured locations, needed to satisfy the requirements in each case. A rectangle in problem diagrams represents a physical domain of the operating environment whose properties are relevant to the problem (e.g., External Digital Camera, Phone Internal Storage and Potential Transmission Eavesdroppers).

**Figure 1 A Secured Environment Variant Problem Description**

A dashed oval represents the requirement (e.g., Secured image exchanges, R1), and a rectangle with a double stripe represents the machine domain whose specification is required (e.g., SecLoc Controller). A solid line connecting two domains presents a set of shared phenomena (properties) between the two domains. For example, a:SC !(RequestTransmission, TerminatesTransmission) indicates that the details of secured image transmissions are shared between the domains External Digital Camera (EDC) and the SecLoc Controller (SC). The infix ‘!’ specifies that SC decides when image transmissions take place, whilst EDC only takes part but cannot initiate transmission requests. A dashed line between the requirement and a domain (e.g., EDC) denotes that the requirement references a phenomenon of EDC, and a dashed arrow between the requirement and a domain (e.g., Phone Internal Storage (PIS)) denotes that the requirement constrains a phenomenon of PIS: when an image transmission ends, a new image file is expected to be saved on the phone’s internal storage.

**Figure 2 An Unsecured Environment Variant Problem Description**

We define contextual variants as problems that share their core requirements in R, induced by contextual properties. Therefore, Figures 1 and 2 represent contextual variant problems. They differ in the sense that Figure 2 has an additional concern of carrying out encryptions and decryptions which introduces the new phenomena a-secure. From this point on, we use the term variant or variant problem to denote a contextual variant problem.

**3.2.2 Deriving Monitoring Problems**

Contextual variables inducing variant problems (e.g., location changes between secured and unsecured locations) must be monitored to detect changes that may violate requirements satisfaction. However, it is not always possible to monitor variables directly; for instance, if a security requirement is introducing a contextual change requiring different variant problems, then an initial analysis may identify ‘eavesdropper’ as a variable to be monitored. Further domain analysis may reveal a mapping relation between an eavesdropper and a location, which leads to the location being monitored instead. In this case, the problem of observing an eavesdropper, which is a less observable variable, is transformed into that of observing location, which is more observable. In such a case, location can be treated as an observable equivalent of eavesdropper.

**Figure 3 A Location Monitoring Problem Description**

For such transformations to be valid, we need to show that by observing location, one can assess the satisfaction of R in the real world. A possible way of achieving this is to use trust assumptions [10] about the physical world and the observed phenomena, which is the approach we have taken [26]. If a change in trust assumption breaks the transformation, a new validation problem is introduced; the specification of which restores the transformation.
In this sense, the transformation problem should be handled recursively as a context-aware problem. Figure 3 shows a location problem description. $R_m$ represents the location monitoring requirements. At this level of problem description, we only focus on the sub-problem of location monitoring and the continual fidelity of monitoring location instead of eavesdropper. The problem of when to monitor not only location but all other contextual variables and the impact of dependencies between variables on the overall monitoring problem is what is addressed in this paper (in Section 4).

3.2.3 Deriving Switching Problems

When a context change results in the violation of requirements, the software application must respond by changing its behaviour to restore it. Switching problem analysis aims at assessing the impact of contextual properties on when and how switching should be done. For instance, while we can switch from SecLoc Controller (Figure 1) to that of UnSecLoc Controller (Figure 2) at anytime, the reverse may not be possible. Assuming encryption must be fully completed before transmission begins; then when location changes from unsecured to secured location during image transmission, it will be too late to switch if encryption is completed. Therefore, the transmission will be completed by the UnSecLoc Controller. However, if encryption was yet to begin, then the application needed to switch to the SecLoc Controller in order to satisfy the efficiency sub-requirement. Figures 4 and 5 show the switching problem descriptions: from UnSecLoc Controller to SecLoc Controller and vice versa. Note the changes in phenomena in the two problem descriptions. That is, the problem of switching from one to the other differs in the concerns to be addressed.

3.2.4 Constructing Context-Aware Problems

Next, we consider the context-aware problem, whose specification ensures that the proper variant is being used in every context. Using input from the monitoring specification, a context-aware application transfers control to designated variant specifications to ensure continual satisfaction of requirements. The context-aware application problem description is as shown in Figure 6. Note that while $R_{S1}$ focuses on the problem of switching from UnSecLoc to SecLoc and $R_{S2}$ the other way round, $R_{SA}$ must satisfying both $R_{S1}$ and $R_{S2}$. This is in addition to ensuring that all switching must be done in the correct context.

Considering Figure 6, while we can tell that it is the monitor that decides when to notify the switcher for a context change that leads to requirements violations, we cannot directly observe the effect of the switcher in selecting the required variant. To achieve this, we use executable state machines to capture dynamic problems.

![Figure 4 A Switching Problem Description – UnSecLoc Controller to SecLoc Controller](image)

![Figure 5 A Switching Problem Description – SecLoc Controller to UnSecLoc Controller](image)

![Figure 6 A Context-Aware Problem Description](image)

4. Formal Analysis of Context-Awareness

We now present our approach for formally analysing the impact of context on monitoring and switching. We begin by detailing the
general approach, followed by its encoding into SAT constraints to facilitate automated analysis. Throughout the section, we continue to use our running example from the mobile device domain to illustrate the approach.

Two variables are said to be dependent, when the value of one variable is constrained by the value of the other. In order to analyse the relationship between alternatives in contextual variability, we need to explicitly express dependencies among contextual variables, namely, contextual dependencies. An example of such contextual dependencies is given by the trust assumptions made about varying locations and varying user permissions: all users at a secured location are authorised to access the data.

Given contextual dependencies, we partition the contextual variability space (i.e. variable domain properties) into a set of subspaces. Each subspace deals with a sub-context constrained by values of its dependant variables. After partitioning, the sub-context can be analysed and reasoned about individually. Referring to our example, a contextual dependency can be expressed as: if the location is secured, a sub-context contains authorised users only; otherwise, two partitions are created for authorised and unauthorised users respectively.

Different parts of the core requirements such as security and performance exhibit different sensitivity to changes in partitioned spaces. For instance, while the sub-space of secured location is being considered, the performance requirement is not affected since there is no encryption. Therefore, we say performance is insensitive in such a sub-space. However, when the sub-space of unsecured location is considered, both performance and security will be sensitive since the size of the image being encrypted and transmitted affects the performance level; while lack of encryption violates the security requirement.

In order to manage variations in making decisions and in prioritising core requirements that are sensitive to contextual changes, the dependencies between sub-contexts have to be viewed and compared from different perspectives. For example, from the security perspective, location of the mobile devices may be the primary contextual variable. While from the performance perspective, image size and encryption/decryption overhead are two dominant contextual variables. One has to make trade-offs between security and performance requirements to derive a switching regime that can satisfy prioritised requirements in all contexts.

To summarise, we express contextual dependencies in terms of partitions of contextual variables and analyse them from different perspectives.

4.1 Analysing the Impact of Contextual Dependency on Monitoring Problems

From the properties of the operating context, we obtain a set of context variables that may result in different behaviour such that the core requirements of the system are maintained. In other words, these contextual variables must be monitored in order to inform the system to take certain actions. Because monitoring may add an overhead to the existing system, one needs to reduce the contextual variables to be monitored, wherever possible. Contextual dependencies are a source for such analysis – the number of monitored contextual variables can be reduced whilst still sufficient to detect the changes that may violate core requirements. Since sufficient monitoring conditions depend on the current context, it is dependent on contextual variability. For example, whilst a mobile device is in a secured location, one need to detect changes to unsecured locations only; but when it moves to an insecure location, changes in both the location and the size of image transmission need to be monitored. Using the context dependency information, some variables are never monitored if their status can be inferred from those of other monitored variables.

In some cases in order to resolve non-determinism under a given context (i.e. more than one possible action), the monitoring process must collect more or less contextual information. For example, the aforementioned condition “Encrypts anytime when the location is insecure” resolves the conflict inefficiently. If performance is also considered important at an insecure location, one can monitor two more contextual variables such as “status of listener” and “image size” and so to decide to have no encryption when the image size is large and all listeners are authorised. Therefore, analysing contextual dependency between partitions from different perspectives enables us to: (a) identify variables that must be monitored; (b) avoid monitoring all variables all the time; and (c) initialise the monitored variables according to their assigned weights (the detail of how the weights may be calculated is discussed in Section 5.3).

4.2 Analysing the Impact of Contextual Dependency on Switching Problems

Contextual dependencies and variability are the source for deriving switching conditions. Local switching conditions found in individual perspectives must be brought into a wider context to find potential conflicts. Such conflicts are resolved by considering the priorities in the individual perspectives. The contextual variability of our running example can be partitioned into two perspectives of context-sensitive quality requirements, security and performance. For security, “location” is monitored with a local switching condition to “encrypt message” when the location is insecure. For performance, “avoid encryption” is the local switching condition. In either case, the local switching condition satisfies the local (perspective) requirement(s). However, when brought into a wider context, there is a conflict to decide whether or not to do encryption when the location is insecure. In resolving such a conflict, we have to prioritise, to decide which perspective is more important. When, for example, security has higher priority than performance, the switching condition has to be updated to “encrypt when the location is insecure”.

We classify the satisfaction level of quality requirements into one of four grades: fully satisfied (FS), partially satisfied (PS), partially denied (PD) and fully denied (FD). We then assess the impact of contextual changes on the satisfaction level of core requirements. Given that the impact of contextual changes varies from one perspective to another, we carry out a trade-off analysis such that not all contextual changes will result in switching behaviours. For instance, given that security must be fully

---

1 The satisfaction of a quality requirement can be represented either qualitatively into a finite set of labels (e.g., FS, PS, PD, FD for softgoals), or quantitatively into a finite set of disjoint ranges. In either case, one can map them into a finite set of propositions.
satisfied and performance must never be fully denied in the running example, we can set a trade-off that allows for variation in performance satisfaction between PD and FS and, as long as security is FS, there is no switching response.

However, irrespective of any trade-off, the constraints of the context will ultimately dictate if switching is possible or not. For instance, once encryption is completed but transmission is yet to begin, a location change from unsecured to secured will not result in switching behaviour as it is too late to do so; even though the trade-off setting suggests such an action.

Therefore, analysing the impact of contextual dependency, we are able to express the constraints of the context in specifying the overall switching behaviour. Also, given dependencies of context-sensitive requirements on the changes in context, we use trade-off to minimise the overall switching behaviour while ensuring that the core requirements are continually being satisfied.

4.3 Monitoring and Switching Behaviours

Using the concepts characterised thus far, we formulate two theorems for monitoring and switching, which define the necessary impact criteria for monitoring a contextual variable, and for switching application behaviour to address a different problem. These theorems guide the formal analysis of the problem of monitoring all relevant environmental variables and that of selecting an appropriate behaviour for a given context.

**Theorem 1** (Monitoring condition): If the trade-off between core requirements is satisfiable by any value of a variable, then the variable need not be monitored. Alternatively, an observable variable needs to be monitored if and only if its value satisfies the core requirements under certain contexts and not in others.

**Theorem 2** (Switching condition): Following a contextual change, a switching of specifications is valid if and only if the current specification does not satisfy core requirements and the replacement does.

The outline proofs of these two theorems are given in appendix B. Their consequences are that, given variant specifications and knowledge about the contextual dependency on monitoring and switching, we are able to encode a constraint satisfiability formula, the solution of which produces a specification for context-aware behaviour. This provides the basis for automated analysis of the impact of varying context on monitoring and switching, and a formal basis for verifying the resulting context-aware behaviour through simulation. Appendix C gives a high level conceptual model of the context-aware concepts introduced in this section.

5. Automated Analysis Support

This section uses the concepts prepared in the previous section to encode a constraint satisfiability problem. The constraint problem is then analysed using our automated analysis tool using a standard solver. Encodings of the inputs to the tool are discussed first, followed by how they are used to derive monitoring and switching behaviour. The overall architecture of the tool is shown in Figure 7, which we use to guide the discussion in this section.

### 5.1 Encoding the SAT Problem

Our constraints satisfaction problem is denoted by a propositional formula $\Phi$. Using knowledge derived from problem descriptions, $\Phi_{\text{context}}$ encodes the contextual variability, problem variant specifications, context-sensitive requirements, and their dependencies. After domain knowledge is specified, the context-aware application can be configured using different variant specifications and different tradeoffs of context-sensitive requirements; these are encoded respectively in $\Phi_{\text{switch}}$ and $\Phi_{\text{trade-off}}$. The formula $\Phi$ can be written in conjunctive normal form (CNF), which consists of the following components:

$$\Phi = \Phi_{\text{context}}(V, X, Q) \land \Phi_{\text{variat}}(X) \land \Phi_{\text{trade-off}}(Q)$$

where $\Phi_{\text{context}}(V, X, Q) = \Phi_{\text{context}}(V) \land \Phi_{\text{req}}(Q) \land \Phi_{\text{specs}}(X)$

$\land \Phi_{\text{specs}}(V \land X \land Q)$

A detailed discussion of each component of (2.) and their input parameters now follows.

#### Contextual Variability

$\Phi_{\text{variat}}(V)$ is the conjunction of partitioned disjoint values of every element in the set of contextual variables $V$:

$$\Phi_{\text{variat}}(V) = \bigwedge_{v \in V} \Phi_{\text{variat}}(v)$$

We can encode $\Phi_{\text{variat}}$, for individual contextual variable $v$ using domain properties. Using our running example, a variable listener can be either authorised or unauthorised but not both, written in CNF as follows:

$$\Phi_{\text{variat}}(\text{Listener}) = ($$

$$\text{Listener==Authorised} \lor \text{Listener==Unauthorised}) \land$$

$$\text{(! Listener==Authorised) \land}$$

$$\text{( Listener==Authorised \land \text{Listener==Unauthorised})}$$

Other variables can be encoded similarly.

For brevity, in the remaining encoded expressions, we use $\text{VariableValue}$ to represent a proposition that binds a variable to a value: $\text{Variable = Value}$.

#### Core requirements

$\Phi_{\text{req}}(Q)$ is the conjunction of rules that express different satisfaction levels of each element in the set of context-sensitive requirements $Q$:

$$\Phi_{\text{req}}(Q) = \bigwedge_{q \in Q} \Phi_{\text{req}}(q)$$

In our running example, $Q = \{\text{security, performance}\}$, both element have four disjoint levels of satisfaction and can only take one value at a time. We can encode $\Phi_{\text{req}}(\text{performance})$ in CNF as follows:

$$\Phi_{\text{performance}} = (\text{PerfFS} \lor \text{PerfFS} \lor \text{PerfFP} \lor \text{PerfFD}) \land (\text{!PerfFS} \lor$$

$$\text{!PerfPS} \lor \text{!PerfFP} \lor \text{!PerfFD}) \land (\text{PerfPS} \lor \text{PerfFP} \lor \text{PerfFD}) \land (\text{PerfFS} \lor \text{PerfPS} \lor \text{PerfFP} \lor \text{PerfFD})$$

One can similarly encode different level of satisfaction for security requirement.

#### Variant specifications

$\Phi_{\text{specs}}(X)$ represents disjoint set of variant problems $X$. Each variant in $X$ addresses a different perspective.

$$\Phi_{\text{specs}}(X) = \bigwedge_{x \in X} \Phi_{\text{specs}}(x)$$
In our running example, \( \Phi_{\text{dep}} = \{ \text{encryption} \} \), we can discern two disjoint variants: one with encryption (ON) and the other without (OFF). Thus, \( \Phi_{\text{dep}}(\text{encryption}) \) can be expressed in terms of ON/OFF.

\[ \Phi_{\text{dep}}(\text{encryption}) = (\text{EncryptionON} \lor \text{EncryptionOFF}) \land \\
(\lnot \text{EncryptionON} \lor \text{EncryptionOFF}) \land \\
(\text{EncryptionON} \lor \lnot \text{EncryptionOFF}) \]

Variant specification switches. \( \Phi_{\text{switch}}(X) \) indicates the values of problem variants \( X \) that are currently used in the specifications.

\[ \Phi_{\text{switch}}(X) = \bigwedge_{x \in X} \Phi_{\text{switch}}(x) \] (2.4)

For example, if encryption is currently ON, then:

\[ \Phi_{\text{switch}(\text{encryption})} = \text{EncryptionON} \]

Perspectives. \( \Phi_{\text{perspective}}(V \land X \land Q) \) is the conjunction of rules that express the implications of contextual variables \( V \) and/or variant problems \( X \) to the satisfaction of context-sensitive requirements in \( Q \).

\[ \Phi_{\text{perspective}}(V \land X \land Q) = \bigwedge_{d \in V \land X \land Q} \Phi_{\text{perspective}}(d) \] (2.5)

As an example for \( \Phi_{\text{perspective}} \), consider the relationships between different values for the listener and transmission variables in a security perspective, expressed in CNF as:

\[ \Phi_{\text{Listener,security}} = \lnot \text{ListenerUnauthorised} \lor \text{TransmissionUnsecured} \lor \text{SecurityFD} \]

\[ \Phi_{\text{Listener,security}} = \lnot \text{ListenerUnauthorised} \lor \text{TransmissionSecured} \lor \text{SecurityFS} \]

This simply states that, when a listener is unauthorised and the transmission is unsecured, the security requirement is fully denied. It is however fully satisfied when the listener is unauthorised and the transmission is secured.

Trade-off relations. \( \Phi_{\text{trade-off}}(Q) \) encodes the trade-off relations between context-sensitive requirements \( Q \) represented in \( \Phi_{\text{perspective}} \).

\[ \Phi_{\text{trade-off}}(Q) = \bigwedge_{v \in Q} \Phi_{\text{trade-off}}(v) \] (2.6)

Each quality requirement trade-off is specified by a certain threshold level of satisfaction. As an example of \( \Phi_{\text{trade-off}} \), consider the trade-off conditions “Security must always be fully satisfied but performance cannot be fully denied”, expressed as:

\[ \Phi_{\text{trade-off}}(\text{Security}) = \text{SecurityFS} \]

\[ \Phi_{\text{trade-off}}(\text{Performance}) = \text{PerfFD} \]

Contextual Dependencies. \( \Phi_{\text{dep}}(V \land X) \) indicates dependencies of values between a contextual variable in \( V \) and a variable or a problem variant in \( V \land X \).

\[ \Phi_{\text{dep}}(V \land X) = \bigwedge_{x \in V \land X} \Phi_{\text{dep}}(x, y) \] (2.7)

Here we use implication operator to capture dependency \( \Phi_{\text{dep}}(x, y) \) explicitly as “\( x \Rightarrow y \)” or equivalently “\( \lnot x \lor y \)” in CNF.

As an example, consider the following dependency: “All listeners are authorised in a secured location”, expressed as:

\[ \Phi_{\text{dep}}(\text{SecurityFD}, \text{ListenerUnauthorised}) = \\
\text{LocationSecured implies ListenerAuthorised} \]

This is transformed to its CNF equivalent as:

\[ \lnot \text{LocationSecured} \lor \text{ListenerAuthorised} \]

Satisfiable contexts (SATContext). We have implemented an encoder to transform the \( \Phi \) in (2) into an input to a SAT solver, and a decoder to extract context from a solution \( \mu \) of the SAT solver into value combination of contextual variables. A solution from the SAT solver is a list of every atomic proposition (literals \( L \)) of the form either positive \( L \) or negative \( \lnot L \). According to our encoding, all these positive literals are in the form of \( \text{VariableValue} \).

In order to find all contexts, we negate the positive literals for every contextual variable and conjunctively join these negative propositions with \( \Phi \) to invoke SAT solver again. By excluding the solutions generated one at a time, we can exhaustively find all context solutions satisfying \( \Phi \). These solutions are stored in a map \( \text{SATContext} \) that contains the bindings of variables to the satisfiability problem.

5.2 Eliciting Monitored Variables

We select contextual variables that must be monitored during run-time to assess the satisfaction of \( \Phi \) by testing for the satisfaction of (2) (i.e., \( \Phi \land \mu = 1 \) \lor (not \( \Phi \land \mu = 0 \)) where \( i \) and \( j \) represent different values of a variable \( v \)). Also, we refine \( \Phi \) by removing all redundant encodings elicited. To ensure completeness, a SAT solver is invoked to test satisfiability of \( \Phi \) each time a redundant condition is removed. It is worth noting that this is done offline, to improve run-time efficiency.

5.3 Deriving Monitoring and Switching Behaviour

Using theorems 1 and 2 described in section 4.3, and the encoded input in (2), we have developed a number of algorithms that implement the first five components in the third row of the tool architecture in Figure 7. Appendix D gives further details of these algorithms. Taking into account the encoded contextual dependency and trade-off constraints, our tool uses a standard solver to derive all satisfiable configurations of (2). Also, as part of the initialisation of monitored variables, we assign weights to each monitored variable that we then use to determine the monitoring sequence of variables (increasing in order of weights). The weight is calculated as the total number of satisfiable configurations, in which state changes in each variable have an effect on requirements satisfaction. The
underlying premise is that, the higher the weight, the lower its impact on requirements violations and therefore the lower monitoring need.

5.4 Simulating Context-Aware Behaviour

The presentation here is focused on how Simulate Context-Aware Behaviour is implemented using two procedures, which we use to simulate monitoring and switching activities in support of validation.

In order to avoid re-computing the satisfiability of $\Phi_{context} \land \Phi_{monitored} \land \Phi_{tradeoff}$ which is more restricted than $\Phi_{context} \land \Phi_{tradeoff}$ in (2), we can filter out the already monitored ones, where $\Phi_{monitored}$ holds from the satisfied solutions to (2). As a result, we create a look up table $SATcontext$ for each monitored context $\Phi_{monitored} \land \Phi_{context}$ to tell which contextual variables in $V_m$ need to be monitored further.

A high level description of the online monitoring procedures is as given below, with the input specified as follows:

- $V_m$ contains a list of all monitored contextual variables
- $X$ contains a list of all variant problems
- $\Phi_{switches}$ contains bindings of current variant problem
- $\Phi_{monitored}$ contains bindings of currently monitored variables
- $\Phi_{context}$ contains bindings of currently monitored variables and contextual dependency information
- $\Phi_{tradeoff}$ contains all possible trade-offs of context-sensitive requirements
- $SATcontext$ contains results of SAT solver that map a combination of bindings to contextual variables, problem variants and context-sensitive requirements to a Boolean satisfaction.

Procedure SimulateMonitoring

$$\Phi_{switches} = \bigwedge_{V_m} \Phi_{monitored} \land \Phi_{context} \land \Phi_{tradeoff}, SATContext$$

BEGIN

$\Phi_{switches} = \bigwedge_{V_m} \Phi_{monitored} \land \Phi_{context} \land \Phi_{tradeoff}, SATContext$

WHILE (true) DO

currentSATContext = $SATcontext \cap \{\text{random}(\{SATcontext\})\}$

Extract $\Phi_{monitored}, \Phi_{context}, \Phi_{switches}$ and $\Phi_{tradeoff}$ from currentSATContext

FOR EACH $VAR$ IN $V_m$ DO

IF there is a $VAL$ such that:

$\Phi_{monitored} \land \Phi_{context} \land \Phi_{tradeoff}$ AND

currentSATContext $\cap \{\text{random}(\{SATcontext\})\}$

THEN

// Let $VAR = VAL$ be monitored for a variable $VAR$

$\Phi_{monitored} = \Phi_{monitored} \land \Phi_{monitored} \land \Phi_{context} \land \Phi_{tradeoff}$

END FOR

SwitchingProcedure ($\Phi_{monitored}, \Phi_{context}, \Phi_{switches}, \Phi_{tradeoff}, SATContext$) // Algorithm 11

END WHILE

END

In this procedure, $\Phi_{tradeoff}$ is only used for the Self-Managing Switching procedure that we describe in the following subsections. Here the Boolean argument adaptive indicates whether adaptive switching is used.

Taking into account all the components of (2), a high level adaptive switching procedure produced by the supporting tool is given below. $\Phi_{monitored}, \Phi_{switches}, \Phi_{tradeoff}$ are bindings defined previously in the SimulateMonitoring procedure.

Procedure SimulateSwitching ($\Phi_{monitored}, \Phi_{context}, \Phi_{switches}, \Phi_{tradeoff}, SATContext$)

BEGIN

NeedSwitch = $\neg SATcontext \cap \{\text{random}(\{SATcontext\})\}$

IF $\neg$ NeedSwitch THEN

RETURN $\Phi_{switches}$ // no need to switch

ELSE

BestTradeoff = $\Phi_{tradeoff}$

BestSwitches = $\Phi_{switches}$

FOR EACH binding of switches $\Phi_{switches}$ DO

IF $\Phi_{switches}$ implies BestTradeoff AND

$\neg SATcontext \cap \{\text{random}(\{SATcontext\})\}$

THEN

BestTradeoff = $\Phi_{switches}$

BestSwitches = $\Phi_{switches}$

END IF

END FOR

RETURN BestSwitches

END

Using the above two online procedures, the context-aware specification ensures that only sufficient variables are monitored and non-essential switching is removed.

6. Case Studies

We have applied our approach to two case studies: a device mobility problem and products’ movements in a logistics problem. While the device mobility problem was used to develop the approach and its validation using simulation, the logistics problem was used to assess its industrial relevance and applicability. Supported by questionnaires, we also used the logistics problem to validate the context-aware concepts we introduced. We are bound by contractual non-disclosure obligations to limit the information we can publish on the logistics problem. Therefore, the discussion of it is more general in comparison to that of the device mobility problem.

In Figures 1 to 6, we showed how the details of individual variants, monitoring and switching problems are analysed using our running example. Therefore, in this section we only present the results of our automated analysis tool and show how we integrated it with the Telelogic Rhapsody application modeller. However, the full implementation is available for download [27]. The overall requirement $R$ for the mobile application problem is stated as:

A secured and efficient transfer of images from a digital camera to a mobile phone’s storage is required. All transmissions must be secured while remaining as efficient as possible.

We elicited a total of six variables, each taking one of two possible values. These were: Location, Eavesdropper, Transmission-status, Image-size, Startup and Organisation-type. Even though Organisation-type was added, the tool discovered that it should never be monitored and therefore did not elicit it. Altogether, we encoded 31 propositions into 479 CNF inputs for SAT4J.

8
Besides eliciting the variables to be monitored and the conditions for monitoring and switching, the simulation procedures produced relations between all contextual changes and the variant specifications needed in each. This was saved in an output file which we use to control the statechart descriptions shown in Figure 8.

The composed statechart is shown on the left hand side of Figure 8, in which the highlighted states (i.e., darker colour) show active states. The upper right screen shows the current context change. The caption CURRENT CONTEXT lists the current variables partition and the required specification needed in such a partition, which corresponds to the output from our tool. Appendix E gives further details about the tool’s output. The lower right screen shows a GUI interface which we use to control contextual changes while we observe its effect on the statechart model, which must reflect changes in the upper right screen.

Rhapsody provides a mechanism to link statechart models to standard Java applications, a facility we have used to integrate our tool into Rhapsody. We show variant specifications as XOR sub-states embedded in parallel state machines, among which only one represents the current variant where the other represent flags of the other mutually exclusive variants. By controlling each variant through a corresponding event that transits from flag states to the state of the variant, we can stack up any number of variant specifications.

Also, we simulate contextual changes in the Java applications, but show its effect in terms of the monitor and switcher behaviours and their effect on varying behaviours. This allows us to manage scalability. The number of variables and their states has no impact on the statechart model where they are hidden. Also, by explicitly showing the validation specification as a state machine, we can continually validate the model even when new variants are being added by executing a recomplied simulation model.

We also applied our approach to an industrial logistics problem. The overall requirement for this domain was expressed as:

A software application is needed to control the movement of items from a distribution centre to retail centres, where the items are sold to customers. The software must ensure that items are available at retail centres at all times and at minimal cost wherever possible. Items may be moved from long term storage locations belonging to a distribution centre, moved from other distribution centres, or ordered from a supplier to a given distribution centre before delivery to retail centres. Each retail centre is allocated to a distribution centre from which it is allowed to receive items.

In all, we elicited a total of 18 variables requiring 4 variant problem descriptions: a variant for when items are available in the distribution centre for delivery to retail centres, a variant for when items are moved from an extended storage location to distribution centres before delivery to retail centres, a variant for when items are moved from one distribution centre to another distribution centre before delivery to retail centres, and a variant for when items are first ordered into a distribution centre before delivery to retail centres. Altogether, we encoded 61 propositions into 1361 CNF formulae as initial input to the SAT solver.

The application of the approach to analyse context-aware problems for these two domains revealed similar contextual variability and dependencies. However, while activities in the logistics domain were generally time bound, those in the mobile device domain were not. Despite these distinctions, our approach was still able to analyse the impact of varying context in both domains. This is because, while there is an increase in the number of variables and states in the logistics domain, the nature of the impact of context on monitoring and switching problems in both applications domains remains similar. While working on the logistics problem, we uncovered a hidden assumption in the forecasting model used in an application for automated ordering of items, which had caused problems of under- or over-ordering of items.

7. Conclusions and Further Work
We have provided an approach for the formal analysis of the impact of varying contextual properties on monitoring and switching behaviour. To achieve this, we characterised concepts for refining contextual properties that we then used to formulate (and prove) two theorems for monitoring and switching. These theorems define the necessary impact criteria for monitoring a contextual variable, and for switching application behaviour to address a different problem. The necessary criteria refer to the effect on the satisfaction of requirements by either the variable changing states or the application switching behaviour. We instantiated our general approach by encoding monitoring and switching problems into propositional logic constraints, which we then analyse using our automated analysis tool,
based on a standard SAT solver (SAT4J [17]). The approach was applied to an industrial case study, which suggested its usefulness in bringing to light hidden assumptions about applications operating environment that may lead to unforeseen requirements violations; and in identifying possible mitigation activities using monitoring and switching problems.

We believe that our approach is ready for application to other problem areas. We are particularly interested in exploring the logistics application sector further, and to begin work on applying it to the financial services sectors such as mobile banking.

Future work is aimed at relating context-aware problems to solution structures, where the body of research in self-managing systems is growing [13]. Also, we are seeking to develop tool support for the derivation of individual (variant) problems, which we currently perform manually.

**Acknowledgements**

We thank the EPSRC for their financial support. We also thank our colleagues Thein Than Tun, Charles Haley, Arosha Bandara, Armstrong Nihalatsi, Robin Laney and Michael Jackson for many useful discussions and feedback on earlier drafts of this paper.

**References**


Appendix A: Our Context-Awareness Approach Overview

We elicit 
contextual dependencies and contextual trade-off
from Describe Requirements. Analyse Individual Problems is carried out using problem descriptions in PF, statechart and process models. PF problem descriptions are manually verified for notation compliance. However, problem descriptions in statechart and process models are verified using Rhapsody and IBM WebSphere Business Process Modeller, respectively in Process/Statechart Simulator & Model Checker. Abstracting away individual problem details, we derive the problem of monitoring all variables and selecting the appropriate variant problem into a constraint satisfiability problem. This is represented using first order propositional logic statements which we analyse in Analyse CA Problem.

In essence, while the informal problem analysis focuses on the problem of monitoring an individual environmental property or behaving in a particular way in Analyse Individual Problems, the more formal problems analysis focuses on the problem of monitoring all relevant environmental variables and that of selecting an appropriate behaviour for a given context in Analyse CA Problem. In the latter's case, the formalism makes it possible to provide tool support in Automated CA Analysis, which is needed due to the complexity of the context-awareness problem space.

Appendix B: Proofs of Monitoring and Switching Theorems

**Theorem 1’** (Monitoring condition) Given a trade-off $\Phi_{\text{makeoff}}$ specified constrained by $\Phi_{\text{context}}$ in (2), a contextual variable VAR needs to be monitored if and only if there is a value VAL such that (VAR==VAL) ^ $\Phi_{\text{context}}$ is satisfiable and (VAR==VAL) ^ $\Phi_{\text{tradeoff}}$ is unsatisfiable.

**Proof.** (If) Suppose VAR==VAL ^ $\Phi_{\text{context}}$ ^ $\Phi_{\text{makeoff}}$ is unsatisfiable and VAR==VAL ^ $\Phi_{\text{context}}$ is satisfiable, then VAR=VAL is one possible context under which $\Phi_{\text{makeoff}}$ holds. Therefore one must monitor VAR to avoid invalidation of context-sensitive requirements.

(Only if) Suppose one has monitored the value of VAR. Then if VAR==VAL is monitored, one knows by the domain knowledge, VAR==VAL satisfies the context constraints, therefore (VAR==VAL) ^ $\Phi_{\text{context}}$. Furthermore, if VAR==VAL is found invalidating the trade-offs, then (VAR==VAL) ^ $\Phi_{\text{context}}$ ^ $\Phi_{\text{makeoff}}$ is not satisfiable. Q.E.D.

**Corollary 1.** (Invariant variables) Under all possible contexts, if the variable VAR==VAL always holds, we do not need to monitor VAR.

**Corollary 2.** (Invariant context-sensitive requirements) Under all possible contexts, if no matter what value the variable VAR is bound, the trade-offs of context-sensitive requirements are always satisfied at the same level, we do not need to monitor VAR.

These corollaries give heuristics that prevent us from monitoring certain variables. Note the second heuristic is dependent on the trade-offs setting of context-sensitive requirements.

**Theorem 2’** (Switching condition) Let $\Phi_{\text{monitored}}$ be the bindings of currently monitored variables. A binding of a variant specification $\Phi_{\text{switches}}$ is a valid switch from the original binding $\Phi_{\text{switches}}$ if $\Phi_{\text{switches}}$ ^ $\Phi_{\text{context}}$ ^ $\Phi_{\text{monitored}}$ ^ $\Phi_{\text{makeoff}}$ is unsatisfiable and $\Phi_{\text{switches}}$ ^ $\Phi_{\text{context}}$ ^ $\Phi_{\text{monitored}}$ ^ $\Phi_{\text{makeoff}}$ is satisfiable.

**Proof.** If $\Phi_{\text{switches}}$ ^ $\Phi_{\text{context}}$ ^ $\Phi_{\text{monitored}}$ ^ $\Phi_{\text{makeoff}}$ is unsatisfiable and $\Phi_{\text{switches}}$ ^ $\Phi_{\text{context}}$ ^ $\Phi_{\text{monitored}}$ ^ $\Phi_{\text{makeoff}}$ is satisfiable, then one has to switch according to trade-offs. Q.E.D.
Appendix C: A Conceptual Model for our Characterised Context-Aware Concepts

Note: Partition is analogous to phenomena in problem diagrams; the shaded concepts represent newly introduced concepts for context-aware property analysis.
Appendix D: Algorithms of our automated analysis tool

Algorithm 1. DeriveMonitoringSwitchingConditionsAndSimulatingProcedures \((V, X, Req, S, P, Q, D, Cost, T)\)

**INPUT**

\[ V = \{v_1, v_2, ..., v_n\}, \text{where } n \geq 1 \] // a set of contextual variables

\[ X = \{x_1, x_2, ..., x_l\}, \text{where } l \geq 1 \] // a set of problem variants

\[ Req = \{r_1, ..., r_m\}, \text{where } m \geq 1 \] // a set of context-sensitive requirements

\[ S \subset V \times N, \text{where } N \text{ is the natural numbers set} \] // discrete states of contextual variables

\[ P \subset X \times N \] // discrete switches of problem variants

\[ Q \subset Req \times N^2 \] // discrete satisfaction levels of context-sensitive requirements

\[ D \subset S \times (S \cup P \cup Q) \setminus \{(s, s) \mid s \in S\} \] // dependencies between states of S, P and Q

\[ Cost \subset V \times R \text{ where } R \text{ is the set of real numbers} \] // cost of monitoring for each contextual variable

\[ T \subset Req \times N, \text{where for each } r \in Req, Q(r) \geq T(r) \geq 1 \] // trade-off setting for quality requirements

**BEGIN**

\[ S(V_m) = \{S(v) \mid v \in V_m\} \] // aggregate all states of monitored contextual variables

\[ D_C = \{(s_1, s_2) \mid (s_1, s_2) \in D \land s_1, s_2 \in S(V_m)\} \] // dependency among contextual variables

\[ V_m = ElicitMonitoredVariables(V, X, Req, S, P, Q, D, T) \] // Algorithm 2

\[ \Phi = RemoveRedundantRules(\Phi, C(V_m)) \] // Algorithm 3

\[ C(V_m) = NecessaryMonitoringConditions(V_m, D, Req, Q, S) \] // Algorithm 4

\[ V'_m = RankMonitoredVariables(\Phi, V_m, C(V_m), Cost) \] // Algorithm 5

\[ t = |V'_m| \] // the number of monitored context variables

**FOR EACH** \(x \in X\) and \(p \in \{1..P(x)\}\) **DO** // deriving switching conditions

\[ C(x=p) = \{(s_1, ..., s_t) \mid (\land_{i=1}^t v_i = s_i) \land \Phi \land (x=p)\}\]

**ENDDO**

\[ SATcontext = SATcontextGeneration(V'_m, X, S, P, \Phi) \] // Algorithm 7

\[ SimulateMonitoringSwitchingProcedure(V'_m, X, SATcontext) \] // Algorithm 10

**END**

Algorithm 2. ElicitMonitoredVariables \((V, X, Req, S, P, Q, D, T)\)

**INPUT:** \(V, X, Req, S, P, Q, D, T\) // same as defined in Algorithm 1

**OUTPUT:** \(V_m\) // list of variables that may be monitored

**BEGIN**

\[ V_m = \emptyset \] // Initialise the set of monitored contextual variables to the empty set

\[ \Phi = EncodeVariabilityAndDependency(V, X, Req, S, P, Q, D, T) \] // Algorithm 1

**FOR EACH** \(v \in V\) **DO**

**FOR EACH** \(i, j \in \{1..S(v)\} \land i \neq j\) **DO**

// construct validation condition using the Monitoring theorem

\[ R_9 = (\Phi \land v = i) \lor (not \Phi \land v = j) \] // invoke SAT solver

**IF** satisfiable(R9) **THEN**

\[ V_m = V_m \cup \{v\} \] // add the contextual variable to be monitored:

**BREAK**

**ENDIF**

**ENDDO**

**RETURN** \(V_m\)

**END**

Algorithm 3. RemoveRedundantRules \((\Phi, C(V_m))\)

---

We use \{FS, PS, PD, FD\} as 4 level of satisfaction for quality requirements and associate them to a number: FS=4, PS=3, PD=2, FD=1.
Algorithm 4. NecessaryMonitoringConditions(V_m, D_C, Req, Q, S)

INPUT V_m ⊂ V // a set of contextual variables to be monitored
D_C ⊂ S × (S \ I) // Dependency among contextual variables
D, Req, Q, S // same as defined in the Algorithm 1
OUTPUT C(V_m) ⊂ V_m × ((S ∪ P ∪ Q) × Boolean) // conditions for necessary monitoring of V_m

BEGIN
C(v_i) = true for every v_i ∈ V_m // Initialise
FOR EACH i, j ∈ [1 .. |V_m|] | i ≠ j DO
   FOR EACH k ∈ S(v_i) DO
      IF | { m | m ∈ S(v_j) ∧ (k, m) ∈ D_C } | = 1 THEN
         // Condition set by the monitoring theorem
         C(v_j) = C(v_j) ∧ (v_i ≠ k)
      ELSE IF (k, q) ∈ D where r ∈ Req ∧ (r, q) ∈ Q THEN
         // Condition set by the monitoring theorem
         C(v_j) = C(v_j) ∧ (v_i ≠ k ∨ r ≠ q)
      ENDIF
   ENDDO
ENDDO
RETURN C(V_m)
END

Algorithm 5. RankMonitoredVariables(Φ, V_m, C(V_m), Cost)

INPUT Φ ⊂ ((S ∪ P ∪ Q) × Boolean) // variability and dependency propositions
V_m ⊂ V // a set of contextual variables to be monitored
C(V_m) ⊂ V_m × ((S ∪ P ∪ Q) × Boolean) // condition for adaptive monitoring of V_m
Cost ⊂ V × R where R is the set of real numbers // cost of monitoring for each variable where available

OUTPUT V_m' ⊂ V_m // a list of contextual variables to be monitored V_m sorted in the ascending order by weight

BEGIN
FOR EACH v ∈ V_m DO
   n = CountSatisfiableMonitoringConfigurations(Φ, C(v)) // Algorithm 6
   w(v) = n Cost(v) // assign weights to monitored variables
ENDDO
V_m' = sort V_m in ascending order of w
RETURN V_m'
END

Algorithm 6. CountSatisfiableMonitoringConfigurations(Φ, C(v))

INPUT Φ ⊂ ((S ∪ P ∪ Q) × Boolean) // variability and dependency propositions

BEGIN
FOR EACH v ∈ V_m DO
   n = CountSatisfiableMonitoringConfigurations(Φ, C(v)) // Algorithm 6
   w(v) = n Cost(v) // assign weights to monitored variables
ENDDO
RETURN V_m'
END
$C(v) \subset ((S \cup P \cup Q) \times \text{Boolean})$ // condition for adaptive monitoring of $V_m$

OUTPUT \quad \begin{align*} n \in N & \quad \text{// number of satisfiable configurations for } C(V_m) \end{align*}

BEGIN
\begin{align*}
n &= 0 \\
R10 &= (\Phi \cap C(v)) \quad \text{// rule (10) to be checked} \\
\textbf{WHILE} & \quad \Phi_S \equiv \text{satisfiable} (R10) \quad \text{DO} \\
\Phi_S &= \land v \in V_m, 1 \leq s \leq S(v) \quad (v = s) \quad \text{such that } \Phi_S \text{ entails } \Phi_S \\
& \quad \text{// decode result of SAT solver} \\
n &= n + 1 \\
R10 &= (\Phi \cap \neg \Phi_S)
\end{align*}
END

RETURN $n$

END

Algorithm 7. SATContextGeneration($V_m$, $X$, $S$, $P$, $\Phi$)
INPUT \quad $V_m \subset V$ // an ordered list of subset of $V$ to be monitored, $S$, $X$, $P$, $\Phi$ // same as defined in Algorithm 1.
OUTPUT: SATContext

BEGIN
\begin{align*}
\Phi(\Phi) &= \{ \} \quad \text{// initialise as an empty set} \\
R8 &= \Phi \\
\textbf{WHILE} & \quad \Phi_S \equiv \text{satisfiable} (R8) \quad \text{DO} \\
\Phi_S &= \land v \in V_m, 1 \leq s \leq S(v) \quad (v = s) \land \land x \in X, 1 \leq p \leq P(x) \quad (x = p) \quad \text{such that } \Phi_S \text{ entails } \Phi \\
& \quad \text{// decode satisfiable result} \\
\Phi(\Phi) &= S(\Phi) \cup \{ \Phi_S \} \\
R8 &= R8 \cap \neg \Phi_S
\end{align*}
END

SATContext = $S(\Phi)$
RETURN SATContext

END

Algorithm 8. EncodeVariabilityAndDependency ($V$, $X$, $Req$, $S$, $P$, $Q$, $D$, $T$)
INPUT \quad $V$, $X$, $Req$, $S$, $P$, $Q$, $D$, $T$ // same as defined in Algorithm 1
OUTPUT \quad \begin{align*} \Phi \subset ((S \cup P \cup Q) \times \text{Boolean}) & \quad \text{// variability and dependency propositions} \end{align*}

BEGIN
\begin{align*}
\Phi &= \text{true} \quad // \text{initialise the overall rule} \\
\textbf{FOR EACH} v \in V \quad & \quad \text{// encode contextual variables} \\
\quad \Phi &= \Phi \land \text{EncodeDisjointDiscreteValues} (v, S(v)) \quad // \text{Algorithm 9} \\
\textbf{ENDDO} \\
\textbf{FOR EACH} x \in X \quad & \quad \text{// encode problem variants} \\
\quad \Phi &= \Phi \land \text{EncodeDisjointDiscreteValues} (x, P(x)) \quad // \text{Algorithm 9} \\
\textbf{ENDDO} \\
\textbf{FOR EACH} r \in Req \quad & \quad \text{// encode quality requirements} \\
\quad \Phi &= \Phi \land \text{EncodeDisjointDiscreteValues} (r, Q(r)) \quad // \text{Algorithm 9} \\
\textbf{ENDDO} \\
\textbf{FOR EACH} (s_1, s_2) \in D \quad & \quad \text{// encode dependencies} \\
\quad \Phi &= \Phi \land (\neg s_1 \lor s_2) \\
\textbf{ENDDO} \\
\textbf{FOR EACH} (r, q) \in T \quad & \quad \text{// encode trade-offs} \\
\quad \Phi_i &= \text{true} \\
\quad \textbf{FOR} i = q \text{ TO } Q(r) \quad & \quad \text{DO} \\
\quad \quad \Phi_v &= \Phi_v \lor (r = i) \\
\textbf{ENDDO} \\
\quad \Phi &= \Phi \land \Phi_v \\
\textbf{ENDDO} \\
\text{RETURN } \Phi
\end{align*}

END
Algorithm 9. EncodeDisjointDiscreteValues (v, n)
INPUT n, v is a variable
OUTPUT Φ ⊂ (({v} × N) × B), a propositional logic rule
BEGIN
Φ = true // initialise the rule
Φ₀ = true // initialise the range closure rule
FOR i = 1 TO n DO
Φ₀ = Φ₀ ∨ (v = i) // increment the range closure rule
Φ₀ = true // initialise the disjoint rule
FOR j = 1 TO n DO
IF (i ≠ j) THEN
Φ₀ = Φ₀ ∧ (v ≠ i ∧ v ≠ j) // increment the disjoint rule
ENDIF
ENDDO
Φ = Φ ∧ Φ₀
RETURN Φ
END

Algorithm 10. SimulateMonitoringSwitchingProcedure (Vm, X, SATcontext)
INPUT Vm // a list of all monitored contextual variables
X // a list of all variant problems
SATcontext // same as defined in Algorithm 7
BEGIN
Φswitches = \bigland x ∈ X x=1
WHILE (true) DO
currentSATContext = SATcontext [random(|SATcontext |)] // a random context is selected
// Φmonitored is values of currently monitored part of the context
// Φcontext is values of contextual variables and contextual dependency information
// Φswitches is the current values of switches
// Φtradeoff is the current values of the trade-off setting
Extract Φmonitored, Φcontext, Φswitches and Φtradeoff from currentSATContext
FOR EACH VAR IN Vm DO
IF there is a VAL such that:
currentSATContext [VAR == VAL ∧ Φcontext ∧ Φmonitored ∧ Φswitches] AND
! currentSATContext [VAR == VAL ∧ Φcontext ∧ Φmonitored ∧ Φswitches ∧ Φtradeoff] THEN
// Let VAR=VAL be monitored for a variable VAR
Φmonitored = Φmonitored ∧ VAR==VAL
ENDIF
END FOR
SwitchingProcedure (Φmonitored, Φcontext, Φswitches, Φtradeoff, SATcontext) // Algorithm 11
END WHILE
END

Algorithm 11. SwitchingProcedure (Φmonitored, Φcontext, Φswitches, Φtradeoff, SATcontext)
INPUT Φmonitored, Φcontext, Φswitches, Φtradeoff //same as defined in Algorithm 10
SATcontext // same as defined in Algorithm 8
OUTPUT Φswitches//satisfiable set of switches
BEGIN
NeedSwitch = SATcontext [Φmonitored ∧ Φcontext ∧ Φswitches ∧ Φtradeoff]
IF ! NeedSwitch THEN
RETURN Φswitches // no need to switch
ELSE
BestTradeoff = Φtradeoff
BestSwitches = Φswitches
FOR EACH binding of switches Φ’switches DO
IF (Φ’switches implies BestTradeoff AND
! SATcontext[Φ’switches ∧ Φmonitored ∧ Φcontext ∧ Φ’tradeoff] AND
SATcontext[Φ’switches ∧ Φmonitored ∧ Φcontext ∧ Φ’ tradeoff]) THEN
BestTradeoff = Φ’tradeoff
BestSwitches = Φ’switches
BREAK // Any new satisfying switches is fine
ENDIF
END FOR
RETURN BestSwitches
END
Appendix E: The Output from Our Automated Analysis Tool

Upper screen shows the first output screen: the list of satisfiable configurations; variables to be monitored; and ranked variables; etc.

Lower screen shows the simulated monitoring and switching behaviours: it shows the current list of variables being monitored and the states of all other variables. The sequence of 1s and 0s show the total list of variables that may be monitored. However, while the 1s show the current actively monitored variables, the 0s shows variables that do not need to be monitored in this context. PD and FS represent the satisfaction levels of context sensitive requirements (i.e., performance and security); 44 represents the total cost of the active monitored variables.