Specifying Monitoring and Switching Problems in Context

Mohammed Salifu Yijun Yu Bashar Nuseibeh
Department of Computing, The Open University, UK
{M.Salifu, Y.Yu, B.Nuseibeh}@open.ac.uk

Abstract *Context-aware applications monitor changes in their operating environment and switch their behaviour to keep satisfying their requirements. Therefore, they must be equipped with the capability to detect variations in their operating context and to switch behaviour in response to such variations. However, specifying monitoring and switching in such applications can be difficult due to their dependence on varying contextual properties which need to be made explicit. In this paper, we present a problem-oriented approach to represent and reason about contextual variability and assess its impact on requirements; to elicit and specify concerns facing monitors and switchers, such as initialisation and interference; and to specify monitoring and switching behaviours that can detect changes and adapt in response. We illustrate our approach by applying it to a published case study.*

1. Introduction

Context-aware applications monitor changes in their operating environment and switch their behaviour to continue satisfying their requirements [1, 2]. Therefore, they must be equipped with the capability to detect variations in their operating context and to switch their behaviour in response. *Monitoring requirements* define what applications must do to detect changes in their operating environment that may violate their requirements, while *switching requirements* define what applications must do to restore the satisfaction of such requirements by adapting their behaviour.

We define *contextual variability* as a space of variables whose different values require different application behaviours. Specifying monitoring and switching behaviours in context-aware applications is often complex [3] due to dependency on the states of varying contextual properties of the operating environment. We refer to monitoring and switching behaviours as *context-awareness*.

Context-awareness concerns have been investigated in autonomic computing [4] and adaptive systems [2]. Autonomic software applications manage themselves with minimal human intervention to accommodate a changing operating environment, while adaptive applications deal with more general behaviour adaptation beyond simply minimising human involvement. In both cases, information gathering and analysis on one hand and planning and adapting application behaviour on the other are required [2, 4].

Kramer and Magee [5] have observed that the operating context of applications can be captured and reasoned about at different layers of abstraction. They propose a three-layer conceptual model for reasoning about and analysing requirements for context-aware applications. Software architectural components occupy the lower layer and represent the traditional focus of autonomic applications research [6]. The middle layer covers applications that require explicit change management in the operating environment, either prior to or after an adaptation. The higher layer addresses software goals that may be affected by adaptation and must be assessed for its impact. The higher the abstraction layer, the closer it is to the problem space, and the harder it is to analyse and specify the required context-awareness concerns.

Cheng and Atlee [7] have recognised the need for requirements engineering that considers both normal environmental behaviour as well as other possible threats and hazards of applications' operating environment. Similarly, Sutcliffe et al. [8] have recognised the need to consider physical contextual properties, in addition to individual and group needs and personal characteristics, in specifying software applications.

Monitoring and switching requirements are usually not at the forefront of user requirements. However, Berry et al. [9] argue that the overall success of context-aware applications largely depends on the success at specifying their monitoring and switching behaviours.

Although researchers have recognised the importance of context for effective adaptation [5, 7, 8], the context of applications to be monitored is rarely made explicit. Therefore, there is a need for guidance to (1) represent and reason about contextual variability, (2) elicit and specify concerns facing monitors and switchers, and (3) relate and assess the effect of such concerns on specifying monitoring and switching behaviours.

We propose a problem-oriented approach to address these issues. First, we adapt the problem frames notation [10, 11] to represent and reason about contextual variability using the notion of *variant problems*. Second, we propose four dimensions to eliciting contextual variability – these are variations in quality requirements, related physical contextual properties, associated decision making processes, and types of concerns such as interference. Finally, we derive and specify, using statecharts, monitoring and switching behaviours from problem variants.

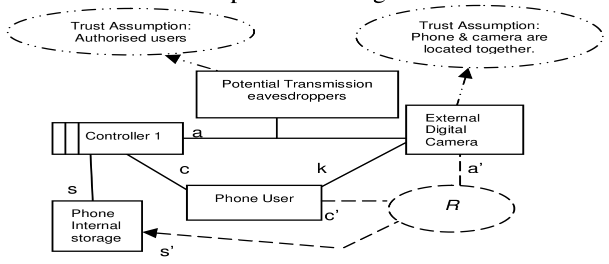
This paper focuses on specifying monitoring and switching behaviours associated with core requirements arising from the changing properties of the operating context. It does not address the more general problem of solutions composition. In order to focus on analysing the impact of contextual variability on monitoring and switching problems, we assume that functional requirements are invariant, while variability of quality requirements is taken into account as one source of contextual variability. Our approach is primarily aimed at software applications for which physical context and its changes are significant. Problem frames explicitly capture the notion of physical context.

The remainder of the paper is structured as follows. We begin with a description of the problem frames notation in Section 2, followed by a presentation of our approach in Section 3. A detailed illustration of our approach using a case study follows in Section 4. We review related work in Section 5. We conclude with summary and an agenda for further work in Section 6.

2. Problem Frames

The Problem Frames approach to requirements engineering provides a basis for analysing software problems and their context [10]. In this approach, a problem comprises three sets of descriptions: (1) a description of the context in which the problem resides in terms of known domain properties of the world, denoted by *W*; (2) a description of the required properties of the world, denoted by *R*; and (3) a description specifying what the machine, or the computer system running the software-to-be, must do to meet the requirements, denoted by *S*. Problem frames are particularly well-suited to analyse context-awareness because they emphasise the need for understanding the physical context of problems.

We represent problem descriptions by *problem diagrams*, such as that shown in Figure 1. A rectangle in a problem diagram represents a physical domain of the problem world (e.g., Phone User, External Digital Camera, Potential Transmission eavesdroppers and Phone Internal Storage). A dashed oval with an outgoing arrow represents the requirement R and one with an ingoing arrow, with two dots, represents trust assumptions about the domain it references. Trust assumptions are an extension to the standard problem frames notation proposed [12]. A rectangle with a double stripe is a machine domain whose specification is required (e.g., Controller 1). A solid line connecting two or more domains represents a set of phenomena shared by the connected domains, such as events and states. For example, the shared phenomenon a indicates the details of transmission are shared between the domains External Digital Camera (EDC) and the Controller 1 (C1). The infix '!' indicates that C1 can manipulate the transmission, whilst EDC can only take part in it. The dashed line between the requirement R and the Phone User (PU) denotes that R references the property of PU while the dashed line with an arrow head between R and Phone Internal Storage (PIS) denotes that R constrains the property of PIS. This means that when a transmission is received, its data must be saved in the phone's storage.


Trust Assumption: Authorised users
Trust Assumption: Phone & camera are located together.
Potential Transmission eavesdroppers
Controller 1
External Digital Camera
Phone User
Phone Internal storage
R
a
c
k
s
a'
c'
s'

a: C1 !{RequestTransmission, TerminatesTransmission}
c: PU !{StartTransmission, StopTranmission}
k: PU !{ConfirmsStartedTransmission, ConfirmsCompletedTranmission}
s: PIS !{receivespicture, savespicture}

Figure 1: An initial problem in secure environment; representing equation (1a)

A problem frame is a known pattern of problems with well-understood structures and concerns. The problem diagram in Figure 1 is an instance of a basic type of problem known as *commanded behaviour*. In Figure 1, the domain EDC is the commanded domain, and C1 is the software to allow a phone user to issues transmission requests.

Although many problems, when decomposed, are expected to fit one of a small number of basic frames, there are usually additional concerns. A *variant frame* represents a new problem pattern that closely matches a basic problem frame but differs because of the presence or absence of a problem domain or control pattern in the existing problem frame. Therefore, variant problems – the instances of variant frames – differ from the basic problem diagrams [10].

An example variant frame is called a *connection variant* [10], which introduces a new domain to connect the existing domains in a basic problem frame diagram. Figure 3 shows a problem diagram that is similar to the one in Figure 1, with the addition of a connection domain Encryption / Decryption between the External Digital Camera and the Controller 2. The new diagram signifies the fact that all transmissions between the camera and the controller will be encrypted and decrypted respectively. Since this problem diagram shares the main concern of the problem diagram in Figure 1, we regard this new problem diagram as a variant of the original problem diagram.

In problem frames, the machine *S* is a physical domain which cannot be specified without the specifications of other physical domains in *W*. Meanwhile, *R* must be expressed as constraints in terms of the properties of *W* [13]. Also, the machine *S* in a solved sub-problem becomes part of the context domain *W* in the parent problem.

To capture behaviour in a problem, we describe physical domains *W* and *S* as parallel state machines, and represent the requirements within these state machines as final states, reachable when they are satisfiable, or as error states, reachable if they can be invalidated. We also describe shared phenomena among domains and requirements as events or guard conditions on corresponding transitions in these parallel state machines. Sub-problem state machines are modelled as super-states within the state machines of the overall problem. This motivates us to use hierarchical state machines, or statecharts [14], to represent the results of problem compositions.

3. Context-Awareness Problems Analysis

Our approach comprises three main activities: (i) the derivation of context variant problems for different context situations; (ii) the derivation of monitoring problems to detect changes in context; and (iii) the composition of context variant and monitoring problem diagrams and the derivation of the behaviour switching specification of the composed problem diagram. The focus of this paper is on the last two activities. Detailed discussion of the first activity is in our earlier publication [11].

3.1 Context-Awareness Variability

We propose four dimensions to consider when eliciting contextual variability and related concerns.

The first dimension concerns quality requirements that may induce variation in their satisfaction in different contexts. For example, satisfying security requirements may require encryption in one location and no encryption in another.

The second dimension concerns physical phenomena whose variations determine the satisfaction of the quality requirement. For example, different locations' physical phenomena may constrain security requirements differently.

The third dimension arises from variation in applying a decision-making process to the quality requirement. For example, for security requirements, the use of trust assumptions [12] may introduce different arguments that can be counter-argued in different contexts.

The fourth dimension is the identification of known classes of concerns associated with variant problems frames. For example, a connector variant frame raises an initialisation concern due to the enabling and disabling of the connecting domain. Every problem diagram has a frame concern [10], which is to achieve the desired effect *R* on the operation environment *W*. In addition, other concerns that may prevent the realisation of *R*, such as the soundness of commands issued by an operator in commanded behaviour problem frames, are elicited. In such cases, the specification must ensure that each operator command is validated for its soundness and reject those deemed unsound.

We suggest that failure to investigate all the relevant dimensions of contextual variability in a given project may lead to incomplete understanding of the contextual variations, and thus may lead to unpredictable errors in specifications.

3.2 Context Variant Problems

Problem descriptions based on the problem frames notation can be described by the expression:

$$W, S \vdash R \qquad (1a)$$

where *R* is the requirements, *W* is the context that *R* is concerned with and *S* is the specifications needed to achieve *R*. The symbol "$\vdash$" denotes entailment, that is, the satisfaction of *W* and *S* entails that of *R*. The "," is the separator for context and specifications. As the problem is further refined, the context can include some specifications from the sub-problems.

In order to express the impact of context-awareness, we use the same notation to describe all sub-problems. Given $W, S \vdash R$ which assumes a certain context *W*, we construct a specification *S* for *R*. Figure 1 represents such a problem description. Throughout this paper, *S* is defined using state machines, as shown in Figure 2.

Using domain knowledge about the problem context, we identify a set of variables $\{v_1, v_2 \ldots v_n\}$ as possible sources of contextual variations. In other words, the context is parameterised by different values of these variables as W (v_1, v_2,…, v_n). For instance, in our case study, the presence or absence of threats represents different values of the security variable.

The initial contexts W in (1a) correspond to the default assignment of these variables. Next, we vary the contextual variables to access their impact on the "context-unaware" problem.

We introduce an operator ';' to compose changes of physical domains on top of the conventional use of "," that delimits W and S in problem frame semantics. The "," separator appears in the formula only once.

A variation $W_v(v_i)$, $1 \leq i \leq n$, may cause requirement R to be no longer satisfied by the specification S; this is expressed as:

$$Not\ (W;W_v, S \vdash R) \qquad (1b)$$

In order to ensure the satisfaction of requirements when this change occurs, a variant specification $S_\Delta \neq \{\ \}$ for this context needs to be derived to restore R, such that:

$$W;W_v,\ S;S_\Delta \vdash R \qquad (1c)$$

While eliciting members of $W;W_v$, whenever $W;W_v$, $S \vdash R$ still holds, then we say the contextual variation does not invalidate the core requirements R, and hence $S_\Delta = \{\ \}$. In Figure 3, one such W_v = the presence of unauthorised *eavesdroppers* and S_Δ = specification of the encryption/decryption domain.

Although introducing S_Δ can restore R under the new context $W;W_v$, in general there is no guarantee that it can still meet the requirements under the original context W: in other words, $W, S;S_\Delta \vdash R$ does not always hold.

We repeat the above steps until all W_v satisfying (1b) and (1c) are found. All such W_v are parallel composed in W_V where $V = \{v_1, v_2 \dots v_n\}$. We then describe the detection of context changes that may invalidate core requirements R as a new problem:

$$W;W_V,\ S_V \vdash R_V \qquad (1d)$$

where R_V represents the requirement of detecting contextual changes $W;W_V$ invalidating R. In (1c), S_Δ represents variant specifications to restore R when the context change $W;W_v$ occurs. Here in (1d), S_V represents the specification to satisfy R_V, i.e., to detect the changes in (1b) that invalidate R.

3.3 Monitoring Problems

It is not always possible to monitor variables in W_V directly. For instance, if security is the variable introducing the contextual change satisfying (1b), an initial analysis may identify eavesdropper as a variable to be monitored in W_V. Further domain analysis may reveal a mapping relation between an eavesdropper and a location, which leads to the location being monitored instead. In such cases, all W_v in W_V are transformed into W_M. For example, location (in W_M) is considered to be an observable equivalent of eavesdropper (in W_V). Similarly, R_V will be transformed into R_M. The context change detection is defined as a monitoring problem, expressed as:

$$(W_M, S_M \vdash R_M)\ \textit{if and only if}\ (W;W_V,\ S_V \vdash R_V) \quad (2)$$

where W_M represents the physical domain being monitored which determine the values of the context variable, S_M and R_M represent the monitoring specification and requirement respectively. Figure 4 shows the monitoring problem in our case study in which W_M Location Receiver, Threat Location Mapping and Threat Report are not present in Figures 1 and 3. S_M is embedded in Figure 6. R_M may address additional concerns beyond context change detection, such as interference.

For the transformation of W_V in (1d) to W_M in (2) to be valid, we need to demonstrate that by observing W_M one can assess the satisfaction of R in the real world. A possible way of achieving this is to use structured argumentation about trust assumptions in the physical world and the observed phenomena [12]. For example, given that all transmission eavesdroppers at secured locations are authorised, one can justifiably monitor location (W_m) and not eavesdropper (W_v) (see Figure 4). Also, we assume W_m is relatively more observable than W_v.

3.4 Switching Problems

Next, we consider the switching problem, to ensure that the correct variant is being used in different contexts. Using input from the monitoring problem, a switcher transfers control to designated variant specifications to ensure the continual satisfaction of requirements. This raises some concerns dealing with contextual variability in monitoring, as well as with switching concerns, such as interference, initialisation, and synchronisation. The switching problem can be expressed as:

$$W_s, S_s \vdash R_s \qquad (3a)$$

where W_s, S_s and R_s respectively represent domains, specifications, and requirements for switching problems.

Combining the monitoring and switching sub-problems together, the resulting context-aware problem can be expressed as:

$$W;W_M;S;S_\Delta;S_M,\ \ S_s \vdash R_s \qquad (3b)$$

where $W;W_M;S;S_\Delta;S_M$ together represent W_s in (3a) and R_s entails R in (1a). Hence, we have obtained a context-aware specification by putting together $S;S_\Delta;S_m$ and S_s. Figures 5 and 6 depict the switching problem in our case study, and its specification, respectively.

We have chosen to represent problem descriptions in (1a) to (3b) using state machines, so that their behaviour can be observed. When problem domains are inherently concurrent, their state machines are put in parallel while synchronisations among them are handled by sending/receiving shared events. Even though the use of statecharts provides us with a way to elicit some concerns (for example, the use of shared events or operations in parallel states to investigate synchronisation and interference concerns), reasoning about and assessing the satisfaction of quality requirements using statecharts remains difficult. For example, it requires encoding of such requirements into discrete values with well defined functions.

4. A Case Study

We have already seen illustrative examples taken from our case study. We now elaborate on these to provide further illustration and demonstration of the utility of our approach.

Our case study description is as follows. Software is to be specified to control the transmission of pictures from an external digital camera (Concord EyeQ [15]) into a mobile phone's storage (Nokia 9500 [16]) under the commands of a phone user. A requirement R of this problem is stated as:

A secure and efficient transfer of pictures is required from a digital camera to the mobile phone's storage.

4.1 The Context-Aware Variability

In R, we first identify two quality requirements, Security and Performance. We investigate possible variations in their satisfaction in different contexts (in accordance with the first dimension of contextual variability). Transferring unsecured pictures in certain locations may satisfy the security requirement while others may not. To satisfy the security requirements in all contexts requires encryption of the transferred pictures. However, the performance requirements are better achieved when encryption is carried out only in unsecured locations. Therefore, both Security and Performance induce variations in different contexts in this case study.

Second, we analyse the physical phenomena location which determines the possible satisfaction of both Security and Performance. This is in line with the second dimension of contextual variability.

Third, for security, we make our trust assumptions explicit; for example, different trust assumptions may hold, depending on whether an eavesdropper is authorised to listen to the transmission. Trust assumptions are represented in Figure 1 by ovals pointed to by the dashed arrows. For Performance, we apply the 90/10 rule – this is the ratio of time consumed by encryption/decryption activities to that of transmission to determine whether or not encryption is required in a secured location. The application of these two decision procedures enabled us to satisfy the conditions of the third dimension of contextual variability.

We will account for context variations associated with different frame concerns (as required by the fourth dimension) when concrete problem frames are introduced in the following sub-sections.

4.2 Representing Context Variant Problems

We begin by assuming that our operating environment is secure, that is, all users are authorised and the devices are located together. The picture transfer problem can be described in Figure 1, which corresponds to (1a). This problem fits the commanded behaviour frame: the requirement is to control a physical domain under the commands of an operator. Here the Digital Camera is the controlled domain; the Phone user is the operator. These, together with Phone Internal Storage and Potential Transmission Eavesdroppers domains represent W. Our R addresses the primary concern of the commanded behaviour frame, which is to achieve the required result in every case; the specification of Controller 1 is (S) of the required controller.

Considering the fourth dimension of contextual variability, the encryption in the digital camera and the decryption in the phone raise an overrun concern [10]. That is, if there is a mismatch between the camera's encryption speed and that of the phone's decryption speed then this could result in loss of transmitted data. To prevent this, we identify the following concern which effectively transforms and expands the scope of R.

Concern 1: *The encryption and decryption speeds of the digital camera and the phone, respectively, must be such that no transmission data is lost.*

The trust assumptions in Figure 1 highlight the reliance of the specification on the physical environment in satisfying R. The All users are authorised assumption does not hold in a situation where unauthorised users exist, such as eavesdroppers who can listen to the transmission. The Phone and camera are located together assumption does not hold when the two mobile devices have different cell addresses not apparent to the phone user.

We show the specification S of this problem in Figure 2, which characterises the problem of Figure 1 in four concurrent state machines. The top two show the specification of the controller machine (S) and the camera (W). To facilitate the validation of the secured transfer requirements, the Authorised Access Threat Checking state machine at the right-bottom, which corresponds to (1b), shows an Unsecured state can only be reached when an unauthorised eavesdropper can have access to data during the transmission in an unsecured location. This state, indicated by the left-bottom state machine, is assumed impossible due to the trust assumption (only authorised employee can access the location).

However, when one executes the above state machine, a violation of the requirements is detected when the unfortunate event evChangeToUnsecure, which corresponds to $W;W_v$, occurs during transmission. When this happens, the unsecured state is reached. Currently, the identification of $W;W_V$ and validation of R is done manually, aided by the executable state machines models of the problem descriptions.

Withdrawing the assumption of All users are authorised because they are in a secured location, the requirement R can be invalidated by the same specification. Hence, we analyse the variant problem for unsecured locations, where any eavesdropping is assumed unauthorised ($W;W_v$), and find a *Connection* variant frame suitable to specify the variant solution, corresponding to (1c), by introducing an Encryption/Decryption domain to block the eavesdropping of unsecured transmissions. Figure 3 depicts the resulting variant problem. The trust assumptions here play similar roles as they did in Figure 1. Since Figure 3 is a variant of Figure 1, their state machine specifications are also similar. $S;S_\Delta$ in Figure 3 differs from S in Figure 1 due to the presence of the Encryption/Decryption domain in Figure 3.

Also, since the problem description in Figure 3 is a variant of the description shown in Figure 1, all states in Figure 2 are relevant states in the state machines of Figure 3, which are shown in Figure 6 for brevity. Inside the nested statecharts of Controller and Digital Camera, the decrypting and encrypting sub-states are inserted respectively between the transitions from Receiving to Saving and between the transitions from Idle to Transmitting. These additional states represent the Encryption/ Decryption domain introduced in Figure 3.

Figure 2 explicitly shows the left hand side of the entailment $\vdash$ relation, that is, W, W_m, S_m and S. We use an unreachable state in S_m to represent and check the satisfaction of (1d). The same representation is made of the switching problem in Figures 5 which is specified in Figure 6; showing $W;W_m; S;S_\Delta;S_m$ and S.

The statechart specifies the behaviour of Controller, that of the Digital Camera and the Simulated Eavesdroppers (W). The validating specification in S_m in Figure 6 validates the requirements in Figures 1 and 3.

By analysing the underlying Performance requirements, the original simpler variant specification delivers higher speed than the variant one because the time-consuming encryption/decryption is not always appropriate when the environment is secured. Therefore we need to consider context-awareness to maintain our functional and quality requirements for both secured and unsecured locations.

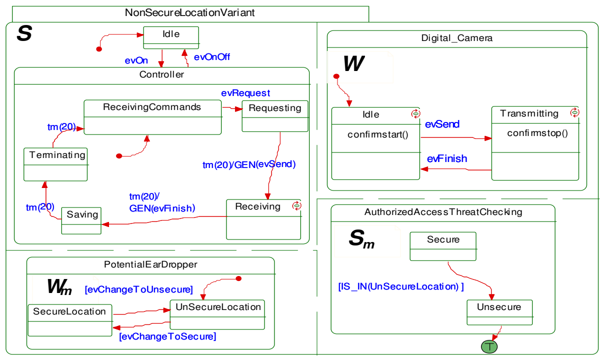


Figure 2: A statechart specification of the problem description in Figure 1, representing *S* and *S_m* in (1a) & (1b) respectively

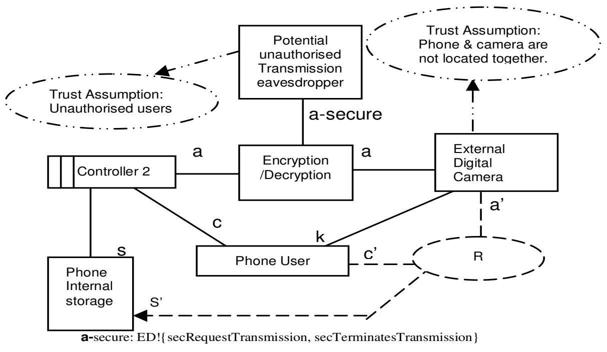

Potential unauthorised Transmission eavesdropper
Trust Assumption: Phone & camera are not located together.
Trust Assumption: Unauthorised users
a-secure
a
a
Controller 2
Encryption /Decryption
External Digital Camera
a'
c
k
s
Phone User
c'
R
Phone Internal storage
S'
a-secure: ED!{secRequestTransmission, secTerminatesTransmission}

Figure 3: A variant problem in non-secure environment; representing equation (1c)

4.4 The Monitoring Problem

Since it is easier to know the location of a person than to know whether a person is a potential eavesdropper, it is preferable if we can monitor locations rather than eavesdroppers. Taking into consideration the trust assumption that only authorised users have access to secured locations, we can determine the validating condition in (1d) by monitoring the location (W_M) of the mobile devices and triggering an event when they change from a secured location into an unsecured one. This corresponds to the derivation and validation of equation (2). The monitoring problem is shown in Figure 4 and its specification embedded in Figure 6.

This monitoring problem fits the information frame [10]: the requirement is to give accurate status information about a real-world domain with the primary concern being the relation between the requirement phenomena and the specification phenomena. Here, the contextual variable we previously identified Location is the real-world domain and its status in the form of Threat Report the output. The specification is to report if a given location is secured or unsecured using pre-analysed location-threat mappings Location Threat Mapping. Thus, the monitoring requirement (R_m) can be stated as:

Using the inputs from the Location Receiver *and that of the* Location-Threat Mapping*, generate a* Threat Report *showing if the current location is a secured environment or not.*

Taking into account the fourth dimension of contextual variability and properties of the Location Threat Mapping domain in the monitoring problem (Figure 4), a consistency concern arises. That is, we need to investigate the possibility of updating the location threat mapping database in the middle of a transmission. If this is found to be probable, then we need to specify how the monitoring problem must handle such a situation. Therefore, we identify and elicit the following concern as:

Concern 2: *The updating of the location threat mapping database is permissible if the location being updated is not the current operating location; and only if current location is being changed from unsecured to secured. Otherwise, the updating is forbidden.*

The elicitation of concern 2 effectively refines and expands the scope of R_m. Also, it is worth noting that the Encryption/Decryption domain represents a solution to the sub-problem of decryption/encryption which we do not need to solve as the solution is given. If this was not the case, then a sub-problem would have to be introduced to carry out the securing of the transmission channel. Problem frames treat solution machines of sub-problems as given domains when used in other problem descriptions.

4.5 The Switching Problem

We now consider switching among variant specifications in a composed context-aware specification. So far, we have one monitoring sub-problem (Figure 4) and two variant sub-problems (Figure 1 and 3). The resulting problem diagram is shown in Figure 5 and its statechart representation in Figure 6. In addition to the original context domains in Figures 1, 3 and 4, the respective specifications Controller 1, Controller 2, and Context Sensor are shown as domains (i.e. rectangles without the two vertical lines) in line with problem frame semantics. Figure 6 represents the statechart for the problem diagram in Figure 5. It shows all sub-problem specifications S, S_Δ, S_m and their contexts and the specification for the context switcher S_s. The expression [(IS_IN(unSecureMode) AND IS_IN(UnSecureLocation)) OR (IS_IN(CSuspend) = IS_IN(DSuspend))] is the condition to check the invalidation of the requirement. Here IS_IN(X) is tool-specific[1], evaluated to true when the super-state enters the state X.

The switching requirement (R_s) is stated as: *Ensure Controller1 executes in a secured location, and Controller2 executes in an unsecured location; using the Threat report of the context sensor to determine which controller is required in every location.*

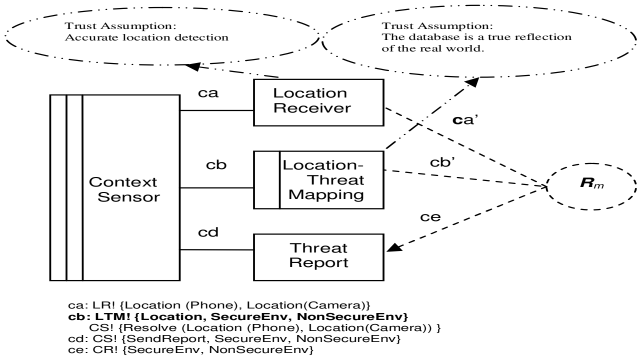

Trust Assumption: Accurate location detection
Trust Assumption: The database is a true reflection of the real world.
ca
Location Receiver
ca'
cb
Location-Threat Mapping
cb'
Context Sensor
R_m
ce
cd
Threat Report
ca: LR! {Location (Phone), Location(Camera)}
cb: LTM! {Location, SecureEnv, NonSecureEnv}
CS! {Resolve (Location (Phone), Location(Camera)) }
cd: CS! {SendReport, SecureEnv, NonSecureEnv}
ce: CR! {SecureEnv, NonSecureEnv}

Figure 4: Threat existence Sensor Problem; representing equation (2)

Investigating the fourth dimension of contextual variability, a possible concern is context change in the middle of a picture transfer, which requires us to consider interference to picture transmissions. To

[1] We used the *Rhapsody* tool set [http://www.ilogix.com/].

account for this, we elicit the following concern, which effectively expands the scope of R_s.

Concern 3: *The context switcher must ensure picture transmission without loss during which switching is required.*

Consider phenomena cda and cdb in Figure 5, Enquire (CurrentState (Transmission, Encrypting, Decrypting)) is a refinement of Enquire(CurrentState), which ensures that Concern 1 is catered for; Initialisation, Start, Stop and Resume phenomena account for Concern 3.

This case study has shown that representing, analysing, and reasoning about context-awareness is difficult and complex. Although primarily a manual process, we have illustrated that our approach to representing and reasoning about contextual variation contributes to a deeper understanding of the nature of the variations and enriches the specification of monitoring and switching behaviours. Our approach also enables traceability between requirements and specifications, between the requirements and the variation of problem context, and between core requirements and context-awareness concerns.

5. Related Work

Realising the limitations of solution-oriented approaches to context-awareness, Zhang and Cheng [17] suggest a problem-oriented approach in which they focus on how to derive switching specifications for safety critical systems as discussed in [18, 19]. However, they do not examine how physical context is represented and monitored and what its impact is on such switching specifications.

In an earlier paper [11], we discussed the work of Bachmann and Bass [20] in identifying different sources of variability, such as variation in features and data of applications. Jaring and Bosch [21] have also examined the difference sources of variability and argued that the type of variability depends on dependencies between points of variation. This observation is consistent with the work of Buhne et al. [22] in which they used Use Cases to model requirement variability. However, representations such as use cases [22] and feature diagrams [23] have been observed by Liaskos et al. [19] to be inadequate for expressing intentional and contextual variability. A limitation that all these approaches have in common is that none of them explicitly considers the physical properties of the operating context.

The need to monitor application software's operating environment to assess the continual satisfaction of requirements was recognised by Fickas and Feather [24]. The primary focus of their work is on assumptions about the operating environment whose failure is likely to invalidate the requirements. They proposed the use of two kinds of parameters to store monitored and controlled variables [25]. Monitored parameters provide the means of detecting changes that cause assumptions failures, while controlled parameters provide a means to adapt application software behaviour. In addition, linear temporal logic may be used to express these parameters and to show their interdependency. Robinson [26] has also recognised the need to monitor application software and to assess their continual satisfaction of higher level goals. The primary objective of Robinson's approach is to detect inconsistency in goals resulting from changes in the operating environment, and to take corrective measures to restore them. Robinson describes the $R_{EQ}M_{ON}$ framework [27] for defining the core requirements and the monitoring behaviours, with associated software development tools.

The benefit of explicitly considering the physical context of software applications in the problem space has also been observed by Sutcliffe et al. [8]. However, their work provides a high level of conceptual guidance, without details of how individual problems are represented. Providing these details, our approach can analyse the impact of contextual variations on monitoring and switching problems. It is our view that, the two approaches could complement each other, with Sutcliffe et al.'s approach as a front end to ours.

The primary difference between the related work above [24-26] and ours is that we monitor problem variations in addition to requirements. This requires us to explicitly represent the core requirements *R*, the problem contexts, the specifications and the relevant concerns that might not be apparent in considering the requirements alone.

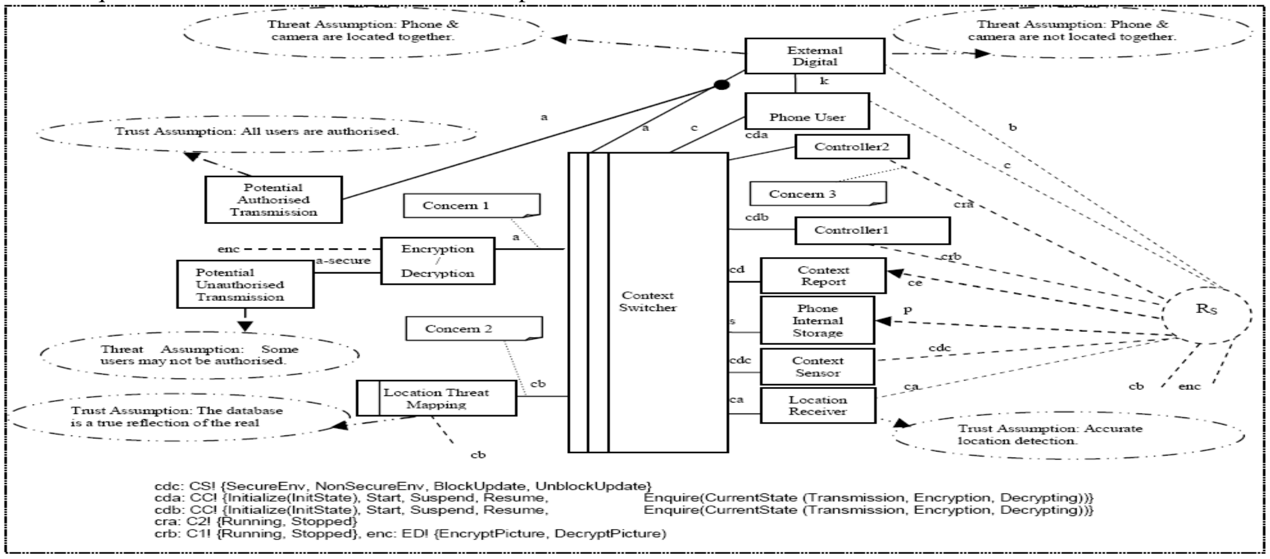
We also reason about contextual variations in four dimensions, which enable us to investigate each underlying quality requirement to see whether it induces any variation. This is necessary for us to define validation criteria that determine when *R* is no longer being satisfied. When *R* is invalid, we modify the specifications to ensure its satisfaction in all contexts. We therefore, see our approach as a finer grain analysis of the context-aware problem space which can also make use of the approaches of Fickas and Feather [25] and that of Robinson [26]. This deeper level of reasoning about context-awareness may contribute to better understanding of contextual variability and its impact on monitoring and switching problems.

The switching problem is a kind of composition problem, similar to what has been described by Laney et al. [28]. In their approach, however, sub-specifications being composed are for different requirements in the same context. In our case, the variant specifications are for the same *R* in different context situations. Therefore, a switcher must be notified when a contextual change that invalidates *R* has been detected. Hence, context monitoring for requirements violations is essential in composing our switching problems.

6. Conclusion and Further Work

The focus of this paper has been on eliciting and reasoning about monitoring and switching concerns induced by varying context – when software switches from one behaviour to another following the detection of requirement violations. We have adapted the problem frames notation to capture and reason about variant problems in context, from which a monitoring problem is derived. These were then used to derive a switching problem. We have also used state machines to capture specifications of problem descriptions so as to analyse their dynamic behaviour and support the elicitation of context-awareness concerns.

An adaptation of our approach will be needed to deal with other aspects of context such as CPU processor speeds or network traffic levels, which are not visible in problem frames and which makes it problematic to project adaptive context into problem diagrams. For software systems where context-awareness is not the primary concern, our analysis may not be necessary.


Threat Assumption: Phone & camera are located together.
External Digital
Threat Assumption: Phone & camera are not located together.
k
Phone User
cda
Trust Assumption: All users are authorised.
Controller2
Potential Authorised Transmission
Concern 1
Concern 3
cdb
Controller1
enc
a-secure
Encryption / Decryption
Potential Unauthorised Transmission
Context Switcher
cd
Context Report
ce
Phone Internal Storage
p
Rs
Concern 2
Threat Assumption: Some users may not be authorised.
cdc
Context Sensor
Location Threat Mapping
cb
ca
Location Receiver
Trust Assumption: The database is a true reflection of the real
Trust Assumption: Accurate location detection.
a
b
c
cra
crb

cdc: CS! {SecureEnv, NonSecureEnv, BlockUpdate, UnblockUpdate}
cda: CC! {Initialize(InitState), Start, Suspend, Resume, Enquire(CurrentState (Transmission, Encryption, Decrypting))}
cdb: CC! {Initialize(InitState), Start, Suspend, Resume, Enquire(CurrentState (Transmission, Encryption, Decrypting))}
cra: C2! {Running, Stopped}
crb: C1! {Running, Stopped}, enc: ED! {EncryptPicture, DecryptPicture)

Figure 5: A switching problem as a composition of sub-problems in Figures 1, 3 and 4; representing (3b)

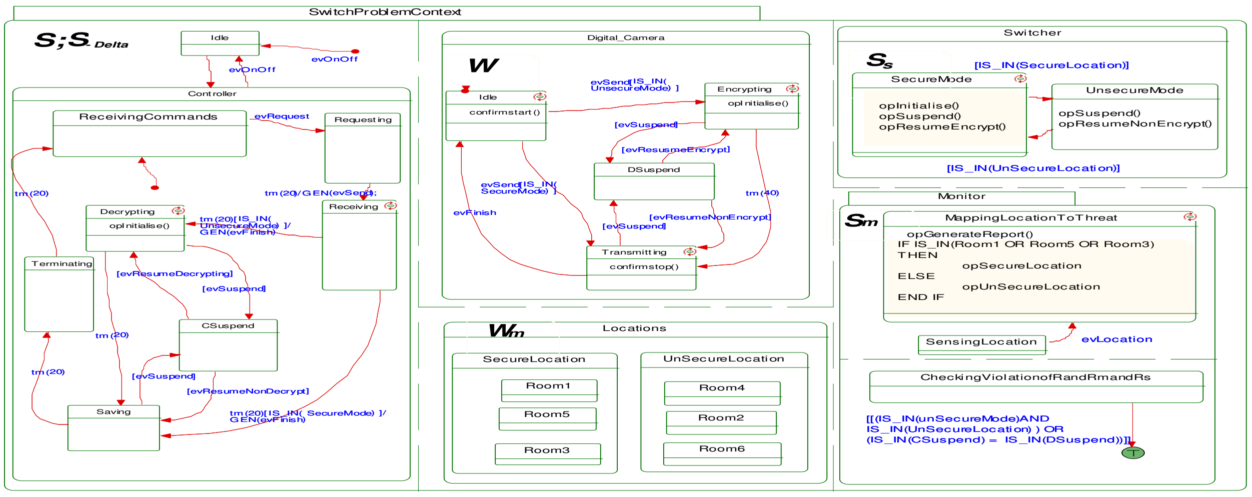


Figure 6: A statechart specification for the switching problem in Figure 5; representing (3b)

Also, since our approach is rooted in the problem space, the primary outputs are variant specifications for varying contextual situations, and specifications for the monitoring and switching problems. This may not be sufficient for dealing with all context-awareness concerns, especially where such concerns are induced by solution structures. Further analysis beyond the specifications may be required in such cases, as the concerns may not be visible in the problem descriptions.

Since monitoring problems are derived from variant problems which together are used to derive switching problems, interference among their problem descriptions may occur. The elicited concerns should eliminate a possible interference between sub-problems that try to update the location-threat mapping database while it is being used by the monitoring problem. This ensures consistency in the monitoring problem and among variant problems during switching. Our approach makes use of composition techniques in dealing with differing concerns; therefore, its use is limited to scenarios where such techniques are applicable. We are currently investigating possible relations between monitoring and switching problems and other context-awareness concerns. However, the wider difficulties of problem composition are beyond the scope of this paper.

A different facet of the problems of monitoring and switching is to enhance efficiency in adaptability. We are investigating a formal encoding of variations in specifications, requirements, and contextual variables, such that they can be reasoned about using dependency and trade-off analyses. Such efficient monitors should adaptively observe a necessary subset of possible variables that are sufficient to detect requirements violations. Similarly, efficient switchers should carry out only necessary responses triggered by contextual and requirements changes.

Though our work assumes stable functional requirements in order to focus our analysis on contextual variability, we are investigating ways variability analysis approaches in product line development, which focus on functional requirements [29, 30] could be improved.

Acknowledgements

We would like thank Jo Atlee and the anonymous referees for tremendously valuable feedback on the submitted paper. We also thank our colleagues Thein Than Tun, Armstrong Nhlabatsi, and Michael Jackson for many useful discussions. Finally, we would like to thank the EPSRC for their financial support.

References

1. Oreizy, P., et al., *An architecture-based approach to self-adaptive software.* Intelligent Systems and Their Applications, IEEE [see also IEEE Intelligent Systems, 1999. **14**(3).
2. Mckinley, P.K., et al., *Composing Adaptive Software.* IEEE Computer, 2004. **37**(7): p. 56-64.
3. Desmet, B., J. Vallejos, and P. Costanza. *Layered design approach for context-aware systems*. in *1st VaMoS 07*. 2007. Limerick, Ireland.
4. Kephart, J.O. and D.M. Chess, *The Vision of Autonomic Computing.* Computer, 2003. **36**(1): p. 41-50.
5. Kramer, J. and J. Magee, *Self-Managed Systems: an Architectural Challenge*, in *ICSE FOSE*. 2007.
6. Georgiadis, I., J. Magee, and J. Kramer. *Self-Organising Software Architectures for Distributed Systems*. in *ACM SIGSOFT Workshop on Self-Healing Systems (WOSS '02)*. 2002. Charleston, South Carolina: ACM.
7. Cheng, B. and J. Atlee. *Research Directions in Requirements Engineering*. in *ICSE FOSE*. 2007.
8. Sutcliffe, A., S. Fickas, and M.M. Sohlberg, *Personal and Contextual Requirements Engineering.* Requirements Engineering, 2005. Proceedings. 13th IEEE International Conference on, 2005: p. 19-30.
9. Berry, D.M., B.H.C. Cheng, and J. Zhang. *The Four Levels of Requirements Engineering for and in Dynamic Adaptive Systems*. in *REFSQ'05*. 2005. Porto, Portugal.
10. Jackson, M., *Problem Frames: Analyzing and structuring software development problems*. 1st ed. 2001b, New York, Oxford: Addison-Wesley. 390.
11. Salifu, M., et al. *Using Problem Descriptions to Represent Variability for Context-Aware Application*. in *1st VaMoS 07*. 2007. Limerick, Ireland: Lero Technical Report.
12. Haley, C.B., et al., *Using trust assumptions with security requirements.* Requirements Engineering, 2006. **11**(2): p. 138-151.
13. Hall, J.G., L. Rapanotti, and M. Jackson, *Problem frame semantics for software development*. 2005, Springer. p. 189-198.
14. Harel, D., *Statecharts: A Visual Formalism For Complex Systems.* Science of Computer Programming, 1987. **8**: p. 231-274.
15. Geeks.com, C., *Concord EyeQ Go Wireless 2MP Bluetooth Digital Camera*. http://www.geeks.com/details.asp?invtid=EYEQ&cat=CAM.
16. Nokia, F., *Enterprise: Developing End-To-End Systems*. 2006, Nokia Forum: Online. p. 1-54.
17. Zhang, J. and B.H.C. Cheng, *Using Temporal Logic to Specify Adaptive Program Semantics.* Architecting Dependable Systems-Journal of Systems and Software (JSS), 2006. **79**(10): p. 1361-1369.
18. Lapouchnian, A., et al. *Requirements-Driven Design of Autonomic Application Software*. in *CASCON2006*. 2006. Canada.
19. Liaskos, S., et al. *On Goal-based Variability Acquisition and Analysis*. in *14th RE*. 2006. Minneapolis/St. Paul, Minnesota, USA,: IEEE CNF.
20. Bachmann, F. and L. Bass. *Managing Variability in Software Architectures*. in *SSR'01*. 2001. Toronto, Ontario, Canada: ACM Press.
21. Jaring, M. and J. Bosch. *A Taxonomy and Hierarchy of Variability Dependencies in Software Product Family Engineering*. in *Proc. of the 28th Annual International Computer Software and Applications Conference (COMPSAC'04)*. 2004: IEEE CNF.
22. Bühne, S., K. Lauenroth, and K. Pohl, *Modelling Requirements Variability across Product Lines.* Proceedings of the 13th RE, 2005: p. 41–50.
23. Schobbens, P.Y., P. Heymans, and J.C. Trigaux. *Feature Diagrams: A Survey and a Formal Semantics*. in *RE'06*. 2006: IEEE Computer Society Washington, DC, USA.
24. Fickas, S. and M. Feather, *Requirements Monitoring in Dynamic Environments.* Proceedings 2nd IEEE Int Symp on RE, 1995: p. 140-147.
25. Feather, M., et al., *Reconciling System Requirements and Runtime Behavior.* Proceedings of the 9th International Workshop on Software Specification and Design, 1998.
26. Robinson, W. and S. Pawlowski, *Managing requirements inconsistency with development goal monitors.* Software Engineering, IEEE Transactions on, 1999. **25**(6): p. 816-835.
27. Robinson, W., *A requirements monitoring framework for enterprise systems.* Requirements Engineering, 2006. **11**(1): p. 17-41.
28. Laney, R., et al. *Composing Requirements Using Problem Frames*. in *Proceedings of the 12th RE*. 2004. Kyoto Japan.: IEEE Computer Society Press.
29. van-Gurp, J., J. Bosch, and M. Svahnberg. *Managing Variability in Software Product Lines*. in *Proceedings of IEEE/IFIP Conference on Software Architecture*. 2000.
30. Ommering, R.V. *Building Product Families with Software Components*. in *ICSE 2002*. 2002. Orlando Florida, USA.