Workshop Proceedings of the 11th International Conference on Intelligent Environments
D. Preuveneers (Ed.)

doi:10.3233/978-1-61499-530-2-70

Change Impact Analysis for Context-Aware Applications in Intelligent Environments

Davy PREUVENEERS [1] and Wouter JOOSEN
iMinds-DistriNet, KU Leuven, Belgium

Abstract. As software systems for context-aware applications and intelligent environments become increasingly complex and adaptive, the need to understand and predict the impact of changes grows. Such changes may manifest themselves (1) as alterations in the way users behave, (2) as software customizations to handle new requirements, and (3) as variations on the dynamic context in which intelligent environment systems are deployed and operate. Such internal and external changes may be anticipated or unforeseen in nature. With reliability being a key concern for intelligent environments, we revisit in this work the state-of-practice on *change impact analysis* (CIA) − a well-known methodology in software engineering − and investigate to what extent it can be applied and enhanced to contribute to the development of more reliable context-aware adaptive applications to increase the confidence in intelligent environment systems.

Keywords. Change impact analysis, context, rapid decisions, reliability

1. Introduction

Over the past decade, intelligent environments have grown in complexity as distributed software and hardware platforms almost autonomously and collectively operate to transparently and non-intrusively support users and address their needs during their activities of daily living. It is inevitable that some software components undergo changes as they are customized to correct errors or to address new requirements. Such modifications may trigger subsequent changes in other components and the potential consequences of these side-effects may not always be clear upfront or desired. That is why the objective of *change impact analysis* [1] in software development is to provide an accurate understanding of the implications of changes, such that stakeholders can plan and make better informed decisions.

Change impact analysis is a well-established methodology in the software evolution domain to manage change, usually at the level of code modifications. However, for intelligent environments and context-aware applications, the notion

[1]Corresponding Author: Department of Computer Science, Celestijnenlaan 200A, B-3001, Heverlee, Belgium; Email: davy.preuveneers.cs.kuleuven.be

of *change* imposes more far-reaching challenges. Contrary to applications that are context agnostic, the situational awareness of intelligent software systems causes them to change not only after modifications in the program code (i.e. *internal* changes), but also when the context of the user or the operational circumstances of the intelligent system (i.e. *external* changes) evolves. The key problem that we aim to highlight in this work is the fact that state-of-practice impact analysis methodologies are not sufficiently equipped to reason upon the impact of (un)planned changes, especially for dynamically adapting software systems such as those that are frequently found in intelligent environments.

Recent work [2,3] has explored model checking as an approach to verify consistency, safety and reliability properties of context-aware applications and intelligent environments. Consistency checking for a context-aware system in a steady state is already a non-trivial endeavor. We now step it up a notch in the sense that we aim for reliable *evolving* intelligent environments and context-aware applications that are *in transition*. This means we must ascertain that enacting change in complex intelligent distributed software systems causes these systems to evolve from one consistent reliable state to the next with as little undesired side effects as possible. It is clear that analyzing the impact of changes now becomes even more arduous. These changes can be both internal and external in nature as they are triggered by different stakeholders (i.e. from software developers at design time to end-users at runtime). For context agnostic systems, the software developer has a good understanding of the intended operational circumstances, and changes can be *planned* at design time or during deployment. For applications with behavior largely driven by external influences such as intelligent environment applications, changes can no longer be planned but must be *anticipated*. This is why we claim impact analysis evolves from a pure static design time concern towards a dynamic runtime concern. Indeed, changes may manifest themselves (1) as alterations in the way users behave or interact with the system, (2) as software customizations to handle new requirements, and (3) as variations in the dynamic context in which intelligent environment systems are deployed and operate. In this work, we revisit the state-of-practice on change impact analysis in the software engineering domain and investigate to what extent it can be applied and enhanced to contribute to the development of more reliable context-aware adaptive applications and intelligent environment systems.

In section 2 we provide an overview of related work in the domain of impact analysis and change management. To illustrate the complexity of change impact analysis for context-aware applications in intelligent environments, we provide a motivating example in section 3. Section 4 provides a taxonomy of changes that a more encompassing impact analysis methodology fore reliable intelligent environments should support. In section 5, we elaborate on our initial endeavors in this area. We reflect back on our work in section 6 before concluding with our final thoughts and suggestions for future work in section 7.

2. Related work

In this section, we briefly discuss relevant related work on change impact analysis in the software evolution domain. While already almost two decades old, Arnold's

book [1] on software change impact analysis highlights many of the important topics, including the need for *traceability* between requirements and design elements of the software architecture, and the representation of *dependencies* that determine the consequences of changes. More recently, Lenhert [4] presented a taxonomy of impact analysis methods. Based on a broad literature review, he presents a multitude of criteria − ranging from the scope of analysis, the granularity of changes and impact, up to the availability of tool support − to classify and compare various impact analysis approaches.

A sound representation of dependencies is instrumental to analyze and measure the impact of changes. German et al. [5] proposed the concept of change impact graphs to determine the impact prior to the actual enactment of code changes. Their method recursively defines the dependency graph $G(f)$ of a function f based on the other functions that can be reached by f. Other works, such as those by Lock et al. [6] and by Tang et al. [7], focus on how to use probabilistic causal relationships to asses how changing requirements and design decisions may affect design elements of a software architecture. These kinds of works go beyond rule based inference in the analysis phase and, for example, leverage Bayesian Belief Networks (BBN) to quantify the likelihood of change impact.

Briand et al. [8] proposed an UML-based method for impact analysis. Their model-based methodology is used to predict the cost and the complexity of changes in order to decide whether to actually implement these changes in a next iteration of a software release. It first starts with a consistency validation phase of the UML models of the system. The impact analysis itself is then carried out between two different versions of the UML model. Their framework offers a set of change detection rules with respect to a given change taxonomy. Model elements that directly or indirectly undergo changes are formally represented by impact analysis rules defined in the Object Constraint Language (OCL).

When software companies have to deal with customers with individual requirements and expectations for specific features − also not that uncommon when deploying context-aware systems adapted to the needs of individuals in an intelligent environment − they usually address such concerns by adopting a Software Product Line (SPL) [9] development methodology to manage common and variable features within a software product line. Both Díaz [10] and Angerer [11] proposed an impact analysis method for derived variants within a software product line or product line architecture. The objective is to measure the impact of customized variants when merging modified features back into the original SPL. The key contribution of these works is that they offer variability-aware program analysis without having to analyze each and every variant of the software product line independently. The combinatorial explosion of feature combinations in the product line (i.e. customization towards the clients or end-users) would make this an intractable impact analysis task.

An in-depth survey is beyond the scope of this work, but it is clear that most of the research on this topic is on program code and design model analysis methods in the software evolution domain. The change impact analysis phase is still carried out at design time, i.e. prior to the actual code modifications. The same is true even if the software engineering methodology pursues an iterative development life cycle to support customization per client. In this work, we borrow

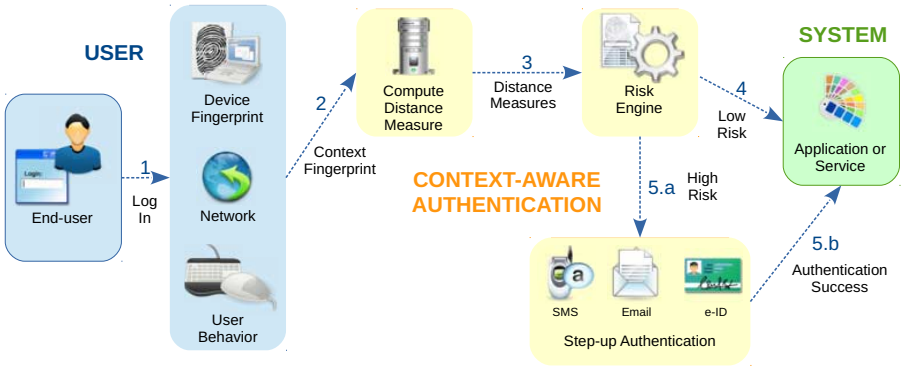

Figure 1. A motivating example on user-friendly context-aware authentication: the context of an individual is used for identification and to quantify risk. If needed, the intelligent authentication system will trigger stronger methods of authentication when the situation demands it.

concepts from the state-of-the-art on dependency graphs, application models, and variability. We investigate to what extend these works can be enhanced for change impact analysis where change is not only a design time concern.

3. A motivating example on user-friendly context-aware authentication

In this section, we will briefly introduce a motivating example to illustrate the impact of change in a non-trivial context-aware application. The application builds upon previous work [12] on the use of context information to offer an intelligent platform for authentication that enables a more user friendly alternative to the intrusive and cumbersome login/password style of authentication.

3.1. Context-aware step-up authentication

Without going into the technical details of our previous work, we provide below a step-wise description of how the user-friendly context-aware authentication works. See Figure 1 for a conceptual overview of the system. The context-aware authentication compares context fingerprints. These fingerprints are in essence an aggregation of various context attributes that collectively can distinguish users or entities from one another.

1. The authentication platform uses and combines different types of context to distinguish and identify a user (time, location, device, behavior, etc.)
2. The different context parameters are aggregated from various context sources to establish a global context fingerprint of the user.
3. The context-aware authentication system compares the current context fingerprint against previous fingerprints of a known individual.
4. Depending on the matching score of the context fingerprint, the authentication system may be confident enough that it has identified the user (without having to ask for credentials in a fairly intrusive way).
5. Otherwise, if the matching score is deemed too low (or alternatively the risk score for mis-identification too high), the system may ask the users to identify themselves with a strong form of authentication.

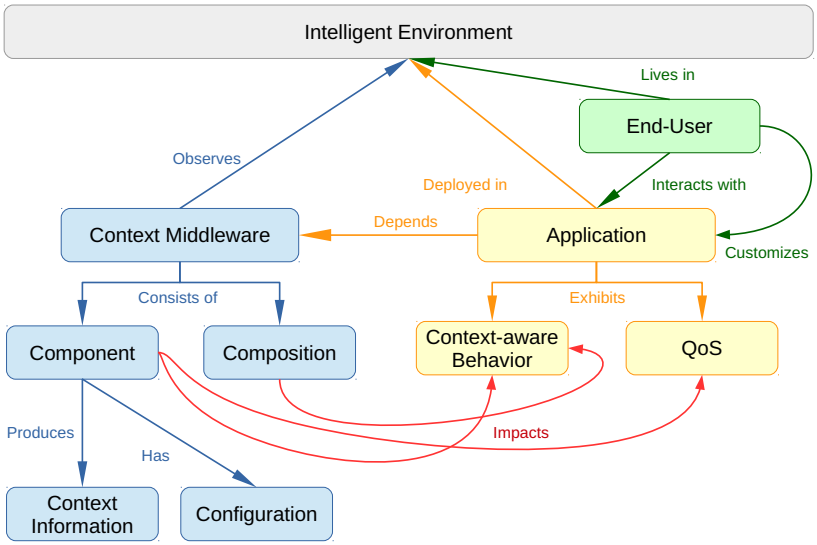

Figure 2. Impact analysis on context-aware applications in intelligent environments.

The overall objective of using context for authentication is to get rid of weak or hard to remember complex passwords. DARPA's Active Authentication program [13] is pursuing similar ways for a continuous authentication based on context and user traits that can be observed through how people interact with the world (e.g. keystroke patterns, mouse and eye movements, cognitive aspects). These techniques have been demonstrated successfully, but long-term stability of behavior patterns is a concern. Our work specifically focuses on the impact of changes when modifying the context sources that characterize human behavior.

3.2. Generalization to context-aware applications in intelligent environments

The above user-friendly authentication system is merely an example of an advanced context-aware application. While it might not be the prototypical example for intelligent environments, it shares a lot of similarities with applications that are more characteristic for these areas, such as ambient assisted and independent living scenarios. We will therefore now generalize the main characteristics of the application, as depicted in Figure 2, to illustrate that it covers a broad range of context-aware applications:

- Context information is exploited to create an understanding of the user, such as his or her identity, current location and situation, activities, etc.
- The context-aware behavior of an intelligent software system is often adapted and customized to the preferences and needs of the individual.
- The quality of the context (e.g. accuracy) and the quality of service of the context-aware behavior (e.g. real-time reaction) play an important role.
- The acquisition and management of context is often decoupled from the business logic (applications and services) that depends on it.

It is clear that in such an ecosystem various changes can have an unforeseen impact on the overall reliability of a context-aware application in an intelligent environment. Often the internal and external dependencies have not been made

explicit, which makes it challenging to analyze the impact of change and hence the reliability of the system at design time.

4. A taxonomy of changes in intelligent environments

In this section, we will highlight how a variety of changes can affect the reliable operation of an application or service within an intelligent environment. We will not revisit analyzing the impact of typical changes that have already been identified by the state-of-the-art, but mainly focus on those topics that deserve much more attention in contemporary change impact analysis techniques and methodologies.

In the subsections below, we consider the core functionality of the authentication platform in the motivating example of section 3.1 as fixed. Its objective is to identify and authenticate an individual in a non-intrusive way by any context means available.

4.1. Functional changes to contextual dependencies

A functional change in the context middleware includes the addition, removal, replacement or reconfiguration of any context sources. With context sources, we refer to application dependencies to both the type of context information as well as the software or hardware component that provides that context information.

A reason to modify the context middleware of our authentication platform is that some sources no longer provide sufficient entropy, i.e. they provide too little useful information to distinguish individuals from one another based on the context fingerprints alone. For example, a device fingerprint obtained through a web browser may aggregate context attributes such as the screen height and width and the user agent string. However, for mobile devices some context-based device fingerprints may not contain sufficient entropy to distinguish different devices, as illustrated in our previous work [14]. Such problems can be mitigated by adding additional context attributes in the device fingerprint, such as an HTML5 canvas fingerprint taken by a browser to better characterize the device.

Changes in the context middleware can affect the similarity assessment with previous context fingerprints in the application. Some context fingerprints may still make use of old (and now removed) context sources or attributes, causing unexpected mismatches. If this change is not taken into consideration in the computation of the similarity score and the overall risk assessment, previous context fingerprints may be discarded without valid reasons.

> *The change impact analysis method should account for changes in the dependencies on context sources and attributes the application relies on.*

4.2. Non-functional changes to contextual dependencies

Non-functional changes can also occur when components are reconfigured, added to or removed from the distributed software architecture. For example, the sampling rate of a context source or the amount of information it provides may evolve

after a change, leading to a possible performance impact on the application, especially when the latter has to process more or more frequent context updates.

For example, our context-aware authentication platform leverages location information and exploits the fact that an individual cannot be at two different places at the same time. Faster location updates improves the quality of the risk assessment of our authentication platform but at the expense of a higher performance or latency impact in the authentication platform.

Changes in the composition or configuration of the context middleware can affect the context-aware behavior and quality of service of the application.

4.3. Changes by the end-user stakeholder at runtime

Some state-of-practice change impact analysis methodologies support traceability between (non-)functional requirements and design elements in the software architecture. These techniques are targeted towards the developer as the stakeholder of the system that initiated the change at design time. However, context-aware applications often also adapt (and hence *change*) upon request of their users to address their evolving needs and preferences. End-users are also important stakeholders in the system, but the impact of end-user initiated changes is usually not accounted for in the impact analysis methods.

For our authentication platform, a simple change such as changing ownership of a mobile device could have an influence on the way a user is implicitly identified by the system, because the device fingerprint will remain the same but the way one interacts with the device will most likely change.

User initiated changes that are allowed by the system at runtime should be represented as explicit dependencies to be able to analyze their impact.

4.4. Unanticipated contextual changes

When creating complex applications, software architects and developers make a lot of decisions that are related to functional or quality (non-functional) requirements. Some of these decisions are based on expertise, budget constraints, personal experience or other assumptions that are not made explicit in the documentation. For example, the application developers may not account for the fact that context sources may become unavailable or produce erroneous or inconsistent values, hereby jeopardizing the functionality of their applications.

In our authentication application example, the use of location information in the fingerprint is not mandatory. However, if a previous context fingerprint has location information (WiFi or GPS-based), then the application assumes all follow-up context fingerprints will contain the attributes as well to analyze the similarity.

Assumptions about the presence of context sources, as well as the quality of the information they produce, should be explicit context dependencies.

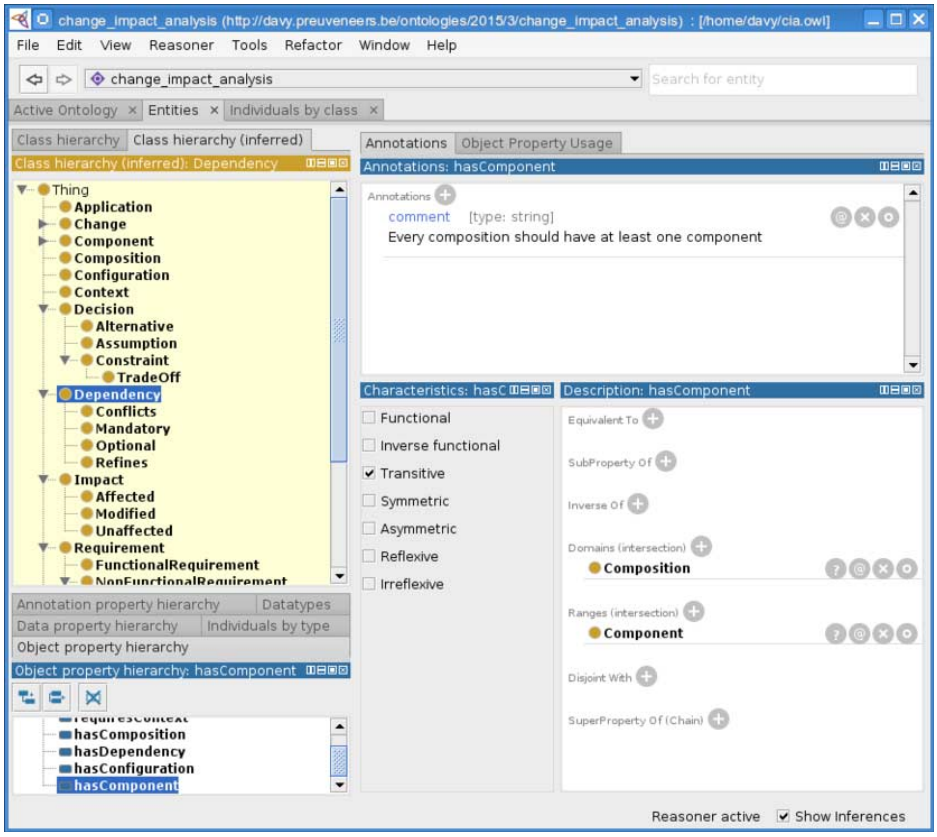

Figure 3. Using ontologies to formally model dependencies in context-aware applications.

5. Towards change impact analysis for context-aware applications

In this section, we will discuss our first steps towards an adapted methodology for change impact analysis for context-aware applications in an intelligent environment. We will highlight where we build on previous work and best practices from the literature.

5.1. Ontology-based modeling and annotation of dependencies as a graph

In our approach, we also model the knowledge we have about the intelligent system, including its design (components and connectors), constraints, assumptions, conflicts, as well as decisions, alternatives and the rationale driving the design and the dependencies among design time and runtime decisions.

Rather than modeling a system in UML notation, we adopted an ontology-based approach that offers us more formal semantics and reasoning capabilities out of the box. Ontologies (in OWL format and edited with the Protégé tool[2] as depicted in Figure 3) allow modeling of dependencies as transitive properties to analyze direct and indirect impacts. It allows us to reason about whether a change means a component is modified (direct change), remains unaffected, or is affected (indirect change) because one of its dependencies has changed. The

[2]http://protege.stanford.edu/

disadvantage is that ontologies do not offer the means to quantitatively measure the impact of changes.

5.2. Traceability of requirements and contextual variability of the system

Conceptually we model classes and properties in an ontology. As a result, the ontology represents a meta-model for change impact analysis. If we want to apply this meta-model to, for example, our context-aware authentication platform, we have to instantiate the relevant classes and properties. This includes instances (or individuals) for at least the following classes and properties:

1. Components within the application
2. Requirements, and components that fulfill them
3. Optional and mandatory context dependencies
4. Design assumptions and constraints

Once we have modeled the application with sufficient detail, we can analyze the impact of changes.

5.3. Representing change and impact within the ontology

Representing change in a context-aware application would boil down to having two ontology instantiations, and have tool support to automatically detect and discover the differences between both model instantiations. Protégé offers tool support to view the differences between two ontologies. However, to simplify analyzing and reasoning upon change and have better traceability between the changes and the entities that are impacted, we opted to directly model changes as a delta in the original ontology model instance.

5.4. Verify consistency before and after change

Ontology inference engines (such as the HermiT 1.3.8.3 reasoner in Protégé 5) are based on complex description logics, rather than the less complex rule engines. The advantage of using description logic reasoners is that analyzing subsumption is very straightforward. This allows to infer whether some entities in the ontology become semantically equivalent with the *owl:Nothing* OWL class identifier. The predefined class extension of *owl:Thing* is the set of all class instances, whereas *owl:Nothing* is always the empty set. If an entity is equivalent with *nothing*, it basically means that it does not exist. For optional components, it means the component is gone. For mandatory ones, we can conclude there must be a consistency issue. Furthermore, advanced tools can explain why ontologies may be inconsistent. We exploit this capability to detect whether *change* instances in our ontology trigger inconsistencies in the ontology instance of our application.

5.5. Analyze design decisions and propagation of changes

The additional advantage of semantically modeling an application with ontologies, compared to UML, is that implicit or indirect relationships between the design elements in the architecture, can be automatically inferred. Reflexive, transitive

Property	Value
# classes	176
# object properties	39
# individuals	42
DL expressivity	AL+

Table 1. Change impact ontology metrics

Change	Time (avg.)
Add context fingerprint	531 msec
Replace location component	876 msec
Change device ownership	796 msec
Impact faulty component	628 msec

Table 2. Change impact analysis performance

or symmetric properties are powerful characteristics in the semantic definition of a dependency relationship when analyzing the propagation of changes.

When a design *assumption* or *trade-off* formally states that an artifact (e.g. a component or context attribute) should not change, but the meta-properties of the relationships infer an impact, unexpected side effects can be detected. Whether this is a big concern, is something that should be investigated by the software architect or developer. When a violation is detected against a design *constraint*, then the proposed change would affect the consistency of our application.

6. Discussion and experience with motivating example

Our approach is still rough around the edges, and thus far, has been validated on one use case being the context-aware authentication application in section 3.1. In the previous sections, we highlighted benefits of our technique. In this section, we mainly focus on the shortcomings we hope to address in future work.

6.1. Quantitative assessment of change impact

A first concern is the fact that semantic ontology reasoners are very good at analyzing the impact of changes, but these tools cannot *measure* the impact or even the likelihood of an impact. Our approach gives a fairly black and white picture. It does not allow us to prioritize changes with respect to their impact.

When our tool infers there is a possibility of an impact, whereas our personal experience or expertise as a developer tells us otherwise, we must model these design assumptions into the application ontology instance.

6.2. Automating test case generation with change boundaries

Many model checker tools are able to automatically produce counter examples to illustrate what kind of steps are needed to let the system evolve to a state that is inconsistent with some predefined safety or reliability properties.

Unfortunately, this is not possible for our change impact analysis approach. In theory, the number of changes that can be applied on a system are endless, and as such there will always be changes that will jeopardize the consistency and reliability of our application.

What is needed is either a catalogue of common changes, or way to set the boundaries for the kind of changes that can be expected or that developers believe should not harm the reliability of the system. The automatic generation of these kind of test case is currently not supported.

6.3. Performance of the analysis

Table 1 lists various characteristics of our change impact ontology for the authentication platform. It provides some statistics on the number the classes, instances and relationships among them, as well as a characterization of the complexity of the change impact ontology. Table 2 lists 4 simple experiments and the average execution time of 5 runs to analyze the impact of each change:

- **Add context fingerprint**: Add the HTML5 canvas device fingerprint
- **Replace location component**: Replace the GPS with the WiFi component
- **Change device ownership**: Measure similarity of stakeholders
- **Impact faulty component**: Infer all dependent components

This example is still fairly small and simple with very reasonable performance results. However, preliminary experiments with artificial ontologies with more than 1000 axioms showed that the analysis of a single change − adding a context feature in the authentication framework − can be carried out in less than a second on a Dell Optiplex 7010 desktop machine, with 16GB of memory and an Intel Core i7-3770 quad-core CPU running at 3.40 GHz. We have not carried out extensive experiments, but plan to enhance the analysis tool with a SPARQL query front-end such that benchmarking can be automated and produce more reliable performance results.

6.4. Integration with other model checking tools

Our change impact analysis technique focuses on a key specific aspect that may harm the reliability of a context-aware application or an intelligent environment. There are various concerns that our technique does not detect, and for which other tools are much more adequate.

Ideally, we should find a way to interlink all these reliability analysis tools in a common test suite such that developers can analyze all of them with a simple click on a button, or schedule them as part of a continuous integration and automated test execution environment such as Jenkins.

7. Conclusion

In this work, we discussed the state-of-practice on change impact analysis in the software evolution domain, and we investigated to what extent it can be applied and enhanced to contribute to the development of more reliable context-aware adaptive applications. The contribution of our work was mainly focused on external dependencies that influence the context-aware behavior and quality of service of an application. Additionally, we semantically modeled a variety of dependencies, such that ontology reasoners can make implicit relationships explicit so that the impact propagation can account for these hidden dependencies.

As future work, we will further explore how we can add more quantitative data to our analysis framework such that we can measure the likelihood and size of the impact. Additionally, a formal representation of bounded changes would allow us to automate the generation of test cases to automatically analyze which permitted changes would jeopardize the reliability of our applications.

Acknowledgment

This research is partially funded by the Research Fund KU Leuven.

References

[1] R. S. Arnold, *Software Change Impact Analysis*, IEEE Computer Society Press, Los Alamitos, CA, USA, 1996.

[2] D. Preuveneers and Y. Berbers, Consistency in context-aware behavior: a model checking approach, In J. A. Botía, H. R. Schmidtke, T. Nakashima, M. R. Al-Mulla, J. C. Augusto, A. Aztiria, M. Ball, V. Callaghan, D. J. Cook, J. Dooley, J. O'Donoghue, S. Egerton, P. A. Haya, M. J. Hornos, E. Morales, J. C. Orozco, O. Portillo-Rodríguez, A. R. González, O. Sandoval, P. Tripicchio, M. Wang, and V. Zamudio, editors, *Workshop Proceedings of the 8th International Conference on Intelligent Environments, Guanajuato, México, June 26-29, 2012*, volume 13 of *Ambient Intelligence and Smart Environments*, pages 401–412, IOS Press, 2012.

[3] J. C. Augusto and M. J. Hornos, Software simulation and verification to increase the reliability of intelligent environments, *Advances in Engineering Software*, **58**(0):18 – 34, 2013.

[4] S. Lehnert, A taxonomy for software change impact analysis, In *Proceedings of the 12th International Workshop on Principles of Software Evolution and the 7th Annual ERCIM Workshop on Software Evolution*, IWPSE-EVOL '11, pages 41–50, New York, NY, USA, 2011, ACM.

[5] D. M. German, A. E. Hassan, and G. Robles, Change impact graphs: Determining the impact of prior codechanges, *Information and Software Technology*, **51**(10):1394 – 1408, 2009, Source Code Analysis and Manipulation, (SCAM) 2008.

[6] S. Lock and G. Kotonya, An integrated, probabilistic framework for requirement change impact analysis, *Australasian J. of Inf. Systems*, **6**(2), 1999.

[7] A. Tang, A. Nicholson, Y. Jin, and J. Han, Using bayesian belief networks for change impact analysis in architecture design, *J. Syst. Softw.*, **80**(1):127–148, January 2007.

[8] L. Briand, Y. Labiche, and L. O'Sullivan, Impact analysis and change management of uml models, In *Software Maintenance, 2003. ICSM 2003. Proceedings. International Conference on*, pages 256–265, Sept 2003.

[9] K. Pohl, G. Böckle, and F. J. v. d. Linden, *Software Product Line Engineering: Foundations, Principles and Techniques*, Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2005.

[10] J. Daz, J. Prez, J. Garbajosa, and A. Wolf, Change impact analysis in product-line architectures, In I. Crnkovic, V. Gruhn, and M. Book, editors, *Software Architecture*, volume 6903 of *Lecture Notes in Computer Science*, pages 114–129, Springer Berlin Heidelberg, 2011.

[11] F. Angerer, Variability-aware change impact analysis of multi-language product lines, In *Proceedings of the 29th ACM/IEEE International Conference on Automated Software Engineering*, ASE '14, pages 903–906, New York, NY, USA, 2014, ACM.

[12] D. Preuveneers and W. Joosen, Smartauth: Dynamic context fingerprinting for continuous user authentication, In *Proceedings of the 2015 ACM Symposium on Applied Computing (SAC), Salamanca, Spain*, April 2015.

[13] R. P. Guidorizzi, Security: Active authentication, *IT Professional*, **15**(4):4–7, 2013.

[14] J. Spooren, D. Preuveneers, and W. Joosen, Mobile device fingerprinting considered harmful for risk-based authentication, In *Proceedings of the 2015 ACM European Workshop on System Security, Bordeaux, France*, April 2015.