Formalisms for Use Cases in Ubiquitous Computing

Richard E. Gunstone
Computing and Informatics, School of Design, Engineering and Computing
Bournemouth University, Poole, Dorset, United Kingdom
rgunstone@bournemouth.ac.uk

***Abstract*—Use cases have achieved widespread adoption in software requirements capture and representation in software system analysis and design. Ubiquitous Computing, a paradigm attributed to the late Mark Weiser, offers new challenges in the representation of user requirements. Such systems are likely to require a high degree of user-centricity if they are to meet sometimes demanding requirements of the modern computer user. This paper presents a review of relevant use case methodologies, principally those that augment or replace the Unified Modelling Language metamodel for use cases.**

***Index Terms*—use cases; software; ubiquitous computing**

I. Introduction

Use cases have achieved widespread adoption in software requirements capture and representation in contemporary software engineering and they are frequently integrated into business processes. The requirements engineering process is widely recognised as being crucial in the process of building a software system, by building a specification through iterative processes of elicitation, specification, and validation; and crucially this should ideally also integrate multiple viewpoints to foster objectivity [1].

A widely recognised shift in the usage of computing technology, focusing more on computing as a communications tool rather than a collection of discrete computational processing systems, leads toward a close relationship between user needs and software systems. This represents a change in the focus of attention, away from the computer system itself toward support for the activities of the user. *Ubiquitous computing*, and the software systems used to achieve it, shares this shift in emphasis toward user- and not system-centricity.

The concept of ubiquitous computing is acknowledged as beginning with the work of Mark Weiser at Xerox [2]. Weiser can be considered something of a visionary, describing a future paradigm of computing that eschewed then-accepted approaches in favour of a highly distributed, interactive and pervasive world. Similarly, we view ubiquitous computing as a move toward an environment where technology diffuses into the background and where software systems are used that adapt to user needs autonomously. We consider the current understanding of ubiquitous computing to be an evolution of Weiser's original design, reflecting some aspects but not all. Contemporary ubiquitous computing examples are wide and varied, but they tend to require relatively advanced functionality in networks (including Cloud computing), context-awareness, and interaction. A prevalent instance of ubiquitous computing to date has been the emergence of *converged devices* such as smartphones and tablet-class computing, and these devices are indicative of a widespread trend toward more ubiquitous computing architectures.

The use case approach to software engineering (as proposed by Jacobson) would appear to fit many of the requirements of ubiquitous computing. It encapsulates user requirements of a system in an easily-understood formalism, and caters for multiple viewpoints. However, limitations have been identified in several studies since its introduction.

The conventional use case representation is based on the concept of scenarios or collections of use cases that represent flows of events [1]. There are several prominent characteristics. The first is that a use case should consist of a *description* [3], which is typically unstructured or semi-structured text. The use case should also contain a *sequence of actions* (or transactions) performed by the system [3, 4, 5]. In addition, for the use case concept to be valid such actions should *lead to an observable result* [3, 4, 5] thereby excluding incomplete or multiple sequences [5]. Finally, the observable result should be of *value* to an actor [3, 4, 5]. Use cases can be represented using a graphical notation.

Use cases serve as projections of future system usage and as projected visions of interactions with a designed system [6]. They are considered by some authors to be scalable to large and complex systems, as they can be improved and expanded incrementally with little or no loss of prior information. Use cases can also be used in such a way as to facilitate requirements traceability throughout design and implementation [1]. Use cases typically make use of natural language, and because of this they are recognised as being a good medium through which requirements can be elicited and recorded, and because of their representation they also avoid the problems associated with purely narrative approaches, while at the same time permitting partial specifications [7]. Taking a user's viewpoint is of significant value when validating the 'adequacy of requirements' [7], and use cases can help facilitate such a process. Use cases are also considered useful for introducing abstraction into the requirements capture and design processes, to allow the system to be understood from structural, behavioural and interactive perspectives [8]. As a consequence of these characteristics, they can support a more effective elicitation process and consequently enable an agreement on the views of the users involved [9].

While use cases offer many opportunities, several of the key benefits offered through their use can also be considered drawbacks. As has been noted, they derive much of their flexibility from their use of natural language, however in lacking formal lack formal syntax and semantics [1, 9] they may permit too much scope, introducing the potential for ambiguity or misinterpretation, and in in the case of plain narrative text this can lead to serious quality problems [7]. They also have limited support for structuring and managing large use-case models [9], which can have consequences for all parts of the development process. Similarly, while there has been recent efforts to represent checks on the quality of use case descriptions [10, 11], the use of text to describe the use case does not necessarily guarantee that the complete process is specified. Use case may also promote a highly localised perspective that can obscure business logic [5]. Finally, in their application, use cases permit the definition of system behaviour from the perspective of many different stakeholders, and, while such flexibility can be considered a benefit in some respects, it is also thought to lead to conflicting functional requirements [12] and contradiction.

In trying to find suitable augmentation or replacement of the methodology for the reasons described in Section I, a number of concepts including some applicable to ubiquitous computing, can be considered desirable. Lee et al. propose several that we propose are very relevant for ubiquitous computing and slightly adapt [1]:

- *clarity* (combining Lee et al.'s *comprehensibility* and *unambiguity*)–a replacement or augmented formalism should continue the theme of ease-of-understanding that Jacobson's original method provided, and also provide a method that is clear and unambiguous to all stakeholders and analysts.
- *scalability*–any change should scale well to the use case elicitation process, providing representational schemes to ensure abstraction is permitted.
- *partiality*–a proposed formalism should accommodate often incomplete information (and to not require a complete representation in order to function).
- *goal-based*–it is desirable for a use case methodology to include some kind of formalisation of user goals as part of the representation to accommodate the goals of the user.

These criteria provide a useful way of analysing the range of formalisms that have emerged since the original use case methdology was proposed. Ubiquitous computing has a strong association with privacy concerns, one aspect not included in the above summary. Incorporating privacy (including Non Functional Requirements - NFRs) from the initial requirements capture process is another important consideration, and we discuss this in more detail in Section III.

We propose the move towards more widespread use of ubiquitous computing technologies interactions between user and system will become focused on the activities of the users in their environment rather than with the system that may be used at a particular time. Therefore, we consider a representational scheme that can accommodate goals as being an important requirement as a means to partly, or fully, address this trend.

The remainder of this paper is structured as follows: In Section II we review a number of extensions that have emerged regarding use cases, with a particular focus on the aspects most relevant to ubiquitous computing. The paper is finished in Section III, where conclusions are made.

II. Review

A number of extensions have emerged that could be regarded as incremental refinements. Lee and Xue [9] propose a goal-based approach (GDUC) to specifying use cases, one that permits the representation of NFRs and consideration of interactions between requirements. NFR representation is a frequent desirable property in the engineering of complex systems, however the use case approach is not geared to representing NFRs easily. Moreover, ineffectively dealing with NFRs is thought to have led to a number of failures in software development (see [13] for a review and an overarching process to elicit NFRs). In GDUC for a given actor-specific and functional goal, it is classified with respect to their competence (complete or partial satisfaction is needed), view (actor- or system-specific), content (functional or non-functional), yielding a faceted break-down into *«extend»* extension use cases and extension goals. The representation used in GDUC allows for NFRs to be represented and also permits associations between indirectly associated goals.

Another process is that by Tan et al. [14] where Data Flow Diagrams (DFDs) are utilised in an augmented form to transform parts of an analysis model such as processes and data flows into an Object Oriented (OO) design and implementation in terms of classes, class attributes, and so on. Their aim is to harness DFDs for use-case realisation. In this approach, DFDs are enhanced to cater for candidate class operations being reflected in processes in the diagram, candidate class attributes, arguments and return values being represented, and control flow being fully specified without recourse to additional specification. The authors propose that by incorporating their augmented DFDs into the requirements analysis stage, by representing use-cases, a comprehensive method can than be used to translate use-cases into OO design, and in turn to implementation.

Glinz has proposed an approach that combines a structured textual representation and a statechart-based structure [7]. In this method there is a clear distinction between events produced by the actor and the responses of the system, and there are simple structuring constructs (such as *if*, *go to step*, etc.) to add further detail. Activity flows between scenarios can be achieved by arranging the scenarios as a directed graph, where edges have explanatory text describing the conditions required for that flow of activity. Glinz's statechart methodology appears to offer greater clarity compared to other approaches, and we return to this in our observations section.

Similar to Glinz's approach, Nebut et al. [15] propose a formalisation of the use case corpus by capitalising on pre- and post-conditions, to make contracts executable by using requirement-level logical expressions. This is a logical evolution of the use case approach and gives rise to a sequential structuring. In the case of Nebut et al.'s work, this is done with the goal of generating tests from a formalisation (in particular the detection of faults in embedded software). Logical expressions used in the conditions of use cases are constructed from Boolean logic operators: *conjunction*, *disjunction*, and *negation*, in addition to quantifiers ∀ *(forall)* and ∃ *(exists)*, and *implication* in post-conditions. The authors use the formalism to generate all possible orderings of use cases to form a 'transition system', and once constructed this allows a variety of structural and logical tests to be performed, exercising paths in the transition system. Tests are implemented as sequence diagrams, representing nominal or exceptional scenarios associated with use cases. The use of a formalised representation has particular benefits in terms of analysis (c.f. Lee et al.) however the authors note the difficulty it may impose on the requirements analyst, who is forced to be rigorous and to clearly specify conditions.

An interesting continuation of augmentation to the UML® metamodel is the work of Dias et al. [16] that introduces *Use Case Fragments* (UCFs) into the development process of use cases. They identify that successful use cases should include several elements, including the basic flow and sequence of steps, alternate flows, information exchange details, and also business rules associated with the interactions encapsulated in the model. Addressing the perceived requirements of both students and novice professionals, they consider the use of a catalogue of recurring use case fragments that can be composited quickly to address the majority of use cases needed in typical information systems. Such fragments are noted as having a similar organisation to software patterns, with each addressing the abstract concept such as *select one element from a set of existent elements*. A UCF template contains the fragment name, sub-goals, purpose, basic flow, alternative flows, input and output details, and rule details. In order to have an applied UCF reflect the situation where it is being used, there are customisation points in the UCF where business terms are substituted. Some of the advantages noted by Dias et al. include the UCF acting as a facilitator, supporting novice requirements professionals in developing use cases, improving writing speed and supporting the specification process.

Sutcliffe [6] introduces the scenario-based requirements engineering approach, of relevance to ubiquitous computing. In common with the original use case approach, this process involves use case elicitation from users and gives formatting guidelines. The final two stages in this process involve scenario generation (from a use case specification) and scenario validation. The process makes use of a reusable library of requirements and associated application classes, influencing factors, exception types, requirements specification(s), and validation frames to support method stages. The use case is modelled as a collection of actions with rules that govern connectivity between actions, and these actions lead to the attainment of a goal. Use cases are organised into a hierarchy to allow for refinement of higher-level use cases. Several action link types are available, including strict sequencing, part sequencing, and inclusion, offering a range of ways links can be established. Validation functionality is possible, to check conformance of the use case models using a schema, and if suitable abstraction is present the use case model can be linked to related systems allowing identification of abstract application classes. This work provides a series of functions that would aid the development of ubiquitous computing systems, particularly where there are common application functions across use cases. An example could be route finding and geo-social networking, which both require geo-location services through the Global Positioning System (GPS), Assisted GPS, Wi-Fi® and IEEE® 802.11 Positioning Systems, etc.

A justification for many of the research proposals that advocate a replacement formalism in the UML metamodel is that the use case method does not lend itself well to formal analysis [1]. In comparison, a formalised model would permit an analysis of the dependencies and inconsistencies (or flaws) in the model. Several research studies have proposed replacing the UML metamodel for use cases to achieve well-defined representations suitable for analysis and resolving imprecision, which we mention in passing.

The Petri net formalism is a technique that has been used to model software systems since the 1970s [17]. Devised as a means of modelling discrete-event systems, Petri nets are well-suited modelling software concepts and in particular for modelling concurrency, with no inherent requirement for synchronicity, and offer (through their asynchronous features) an abstraction above the flow of time, to present events in terms of their partial ordering [17]. They have been applied in a number of ways to enhance the use case approach. Xu & He [18] apply the concept of the Place Transition variant of the Petri net ((PT, PrT, P/T net) to the generation of test requirements, in the context of aspect-oriented use cases. In contrast, Lee et al. propose Constraints-Based Modular Petri Nets (CMPNs) (see [1] for mathematical definitions), which are suggested as improving on the drawbacks of P/T nets and coloured Petri nets. CMPNs are argued as addressing several shortcomings with other approaches, such as being cumbersome where incremental modification is required, to limited analysis techniques on interaction and dependency, and in the case of Finite State Automatons (FSAs) high state space complexity [1].

Replacement formalisms are geared toward a more radical change to the representational schemed used for use cases, and while offering benefits in terms of machine analysis it does present potential drawbacks. Such drawbacks can include a loss of clarity, and potentially more work on the part of the use case analyst.

III. Observations

Use cases have achieved widespread use in software engineering problems, and more widely as a generic term for user needs from products and services. In this paper we have identified the principal benefits of the original use case method, shortcomings and new approaches. For ubiquitous computing, the adoption of use cases provides an opportunity to ensure any systems developed are more closely aligned to user requirements.

This paper has identified several evolutions to the original use case representation, and these can be broadly classified as either *augmentation* or *replacement* (though we note this is not a clear separation and in reality some degree of overlap exists). Replacement formalisms tend to use a more formal representation to permit machine analysis. Regards this latter point and our earlier observations in Section I, we note Glinz, that while formal representations are useful for analysis and can achieve high levels of precision, this comes at the expense of readability and effort to write the scenarios [7]. This shortcoming in comprehensibility of some approaches detracts from the comprehensibility of the original use case method (a characteristic identified in Section I as worthy of preservation).

Taking these concepts further requires the development of an extension to the UML metamodel that facilitates the desirable attributes needed for ubiquitous computing, in particular representing user goals. It is also necessary to construct and refine a suitable requirements example for ubiquitous computing that illustrates these requirements, and use this to validate the modifications to the UML metamodel. One example is the the class of situation awareness applications for ubiquitous computing (c.f. [19]) that provides a basis for an illustrative set of use cases from a user perspective.

One aspect not examined in depth, but noted in the introduction, is that of privacy. It is desirable particularly for ubiquitous computing systems to consider how privacy requirements on the parts of users can be represented and implemented early-on in the development process.

Ubiquitous computing takes privacy considerations for contemporary computing much further, because such systems are intrinsically based upon the collection and processing of information about their users, the environment, their property, actions, and so on. This information is collected all or most of the time, senses information types that are not readily accessible in contemporary computing paradigms (such as video camera feeds of rooms, contents of fridges via Radio Frequency Identification, etc.), and is extensively processed to yield new useful information that can aid the activities of its users. On a practical level moving toward building such complex systems inevitably requires specific technical developments to the system design to ensure privacy can be protected or enhanced. This necessitates the implementation of technical measures–Privacy Enhancing Technology (PET), discussed in a complementary paper [20].

References

[1] W. J. Lee, S. D. Cha, and Y. R. Kwon, "Integration and Analysis of Use Cases Using Modular Petri Nets in Requirements Engineering," *IEEE Transactions on Software Engineering*, vol. 24, pp. 1115–1130, December 1998.

[2] M. Weiser, "Some Computer Science Issues in Ubiquitous Computing," *Commun. ACM*, vol. 36, pp. 75–84, July 1993.

[3] D. Leffingwell and D. Widrig, *Managing Software Requirements: A Use Case Approach.* Addison-Wesley Professional, 2 ed., May 2003. Print ISBN-10: 0-321-12247-X; Print ISBN-13: 978-0-321-12247-6.

[4] I. Jacobson, M. Griss, and P. Jonsson, *Software Reuse: Architecture, Process and Organisation for Business Success.* Reading, MA: Addison-Wesley/ACM Press, 1997.

[5] A. J. H. Simons, "Use Cases Considered Harmful," tech. rep., University of Sheffield, 1999.

[6] A. G. Sutcliffe, N. A. M. Maiden, S. Minocha, and D. Manuel, "Supporting Scenario-Based Requirements Engineering," *IEEE Transactions on Software Engineering*, vol. 24, pp. 1072–1088, December 1998.

[7] M. Glinz, "Improving the Quality of Requirements with Scenarios," in *Proceedings of the Second World Congress for Software Quality (2WCSQ)*, (Yokohama), pp. 55–60, September 2000.

[8] Object Management Group, "Introduction to OMG's Unified Modelling Language," July 2005. http://www.omg.org/gettingstarted/what_is_uml.htm, Last accessed June 2011.

[9] J. Lee and N.-L. Xue, "Analyzing User Requirements by Use Cases: A Goal-Driven Approach," *IEEE Software*, pp. 92–101, July/August 1999.

[10] K. Phalp, J. Vincent, and K. Cox, "Assessing the Quality of Use Case Descriptions," *Software Quality Journal*, vol. 15, pp. 69–97, 2007.

[11] K. Phalp, A. Adlem, S. Jeary, J. Vincent, and J. Kanyaru, "The Role of Comprehension In Requirements and Implications for Use Case Descriptions," *Software Quality Journal*, 2010.

[12] J. H. Hausmann and R. Heckel, "Detection of Conflicting Functional Requirements in a Use Case-Driven Approach - A static analysis technique based on graph transformation," in *ICSE 2002*, pp. 105–115, ACM Press, 2002.

[13] L. M. Cysneiros and J. C. S. do Prado Leite, "Non-functional Requirements: From Elicitation to Conceptual Models," *IEEE Transactions on Software Engineering*, vol. 30, pp. 328–350, May 2004.

[14] H. B. K. Tan, Y. Yang, and L. Bian, "Systematic Transformation of Functional Analysis Model into OO Design and Implementation," *IEEE Transactions on Software Engineering*, vol. 32, pp. 111–135, February 2006.

[15] C. Nebut, F. Fleurey, Y. L. Traon, and J.-M. Jezequel, "Automatic Test Generation: A Use Case Driven Approach," *IEEE Transactions on Software Engineering*, vol. 32, pp. 140–155, March 2006.

[16] F. G. Dias, E. A. Schmitz, M. L. M. Campos, A. L. Correa, and A. J. Alencar, "Elaboration of Use Case Specifications: an Approach Based on Use Case Fragments," in *Proceedings of SAC08*, (Fortaleza, Ceara, Brazil), pp. 614–618, ACM, March 2008.

[17] J. L. Peterson, "Petri Nets," *Computing Surveys*, vol. 9, pp. 223–252, September 1977.

[18] D. Xu and X. He, "Generation of Test Requirements from Aspectual Use Cases," in *Proceedings of WTAOP07 Workshop*, (Vancouver, British Columbia, Canada), pp. 17–22, ACM, March 2007.

[19] S. Minamimoto, S. Fujii, H. Yamaguchi, and T. Higashino, "Map estimation using GPS-equipped mobile wireless nodes," *Pervasive and Mobile Computing*, vol. 6, pp. 623–641, 2010.

[20] R. E. Gunstone, "Integrating Privacy During Requirements Capture for Ubiquitous Computing," in *Proceedings of the First International Conference on Social Eco-Informatics*, (Barcelona), 2011.