CatchML - A Language for Modeling and Verification of Context-Aware Exception Handling Behaviour

Rafael de Lima[1], **Lincoln S. Rocha**[1,2], **Rossana M. C. Andrade**[1,2], **Valéria Lelli**[1,2]

[1]Group of Computer Networks, Software Engineering and Systems (GREat)

[2]Department of Computer Science – Federal University of Ceará (UFC)
Fortaleza, CE – Brazil

{rafaellima, lincoln, rossana, valerialelli}@great.ufc.br

Abstract. *The context-aware exception handling (CAEH) is an error recovery technique employed to improve the ubiquitous software robustness. The design of CAEH is a difficult and error-prone task. The erroneous specification of such conditions represents a critical design fault that can lead the CAEH mechanism to behave erroneously or improperly at runtime. To deal with this problem, we propose a domain-specific language for modeling CAEH, called CatchML, using a high-level interface to make the design of CAEH models simpler and more intuitive. The CatchML language is integrated into a tool to allow designers to perform automatic model verifications by looking at the errors directly in the specification code. We conducted a case study on a sample system called UbiParking with nine volunteers. The results showed that the CatchML language is easy to model the context-aware exception handling and also allowed the participants to quickly locate the injected design faults.*

1. Introduction

Ubiquitous Computing (UbiComp) is a paradigm that performs computation in a multitude of devices embedded in the user environment (e.g., smartphones and smartwatches) that are put together to form an unobtrusive, continual, and reliable connectivity for information exchanging, resource sharing, and service provision [Weiser 1991]. Even when dealing with more complex issues of the Internet of Things (IoT) paradigm, we still have to deal with features of UbiComp that remain in IoT systems such as context-awareness and adaptability [Andrade et al. 2017]. The context-aware ubiquitous systems are deployed in and can interact with the user physical environment [Patikirikorala et al. 2012]. Such systems are designed to behave autonomously, perceiving the environment's changes and reacting according, adapting its structure and behavior and performing automatic tasks to accomplish their goals to meet the user's needs [Loke 2009]. Since such systems have a broad domain (e.g., smart home, healthcare and disaster response) and replace, autonomously, the user role in the decision-making process, they must be reliable to fulfill their function and robust to deal with abnormal situations, avoiding occasional failures [Chetan et al. 2005].

Exception handling is a well-known forward or recovery technique employed to improve the software robustness [Parnas and Würges 1976]. The **context-aware exception handling** (CAEH) is an exception handling variant technique in which the context and the context-awareness are used in the exception handling mechanism to deal with abnormal situations, named **contextual exceptions** or **semantic exceptions** [Damasceno et al. 2006,

Kulkarni and Tripathi 2010, Mercadal et al. 2010, Beder and de Araújo 2011, Cho and Helal 2011, Rocha and Andrade 2012, Cho and Helal 2012]. In such approach, the context is used to detect contextual exceptions, select the proper handlers and perform them, putting the system back in a safe state. When a contextual exception occurrence is detected, the CAEH mechanism deviates the system's standard control flow to the context-aware exceptional control flow.

The design of CAEH is a complex and error-prone activity [Cho and Helal 2011, Cho and Helal 2012]. Indeed, designers may introduce accidentally faults during the CAEH design, leading the CAEH mechanism to behave erroneously or improperly at runtime. To deal with this problem, Lincoln et al. [Rocha et al. 2013] have proposed a method for model checking context-aware exception handling design, called CAEHV (Context-Aware Exception Handling Verification). Such method defines a set of abstractions and concepts that allow designers to model the CAEH behavior and verify if a specific set of design faults occurs in the CAEH design. The CAEHV method was implemented as a Java API called JCAEHV that requires more programming knowledge for designers.

In this scenario, we propose the *CatchML*[1], a DSL (Domain-Specific Language) to help designers to model the CAEH behaviour and to automatically verify its consistency efficiently and easy to use. This languague is integrated in a development environment that enables the verification of the CAEH models and provides an effective feedback mechanism to directly locate the design faults into the CAEH model. To assess our approach, we have conducted an evaluation with nine volunteers that had to model the CAEH design of a ubiquitous system using *CatchML*. The experiment results show evidences that *CatchML* language is ease to use and also helps the participants to correctly locate several design faults.

The remaining of this paper is then organized as follows: Section 2 provides the background; Section 3 introduces the *CatchML* language and its evaluation is described in Section 4; related work in Section 5 and our conclusions and research agenda in Section 6.

2. Exception Handling

2.1. Definition

Exception handling is a well-known forward error recovery technique employed to improve the software robustness present in mainstream programming languages (e.g., Java and C#) [Cacho et al. 2014]. An exception describes an erroneous or abnormal situation in which the normal flow of the system execution cannot proceed [Kienzle 2008]. If an exception occurrence is detected at runtime, the system's normal control flow is deviated to the exceptional control flow that makes an additional computation to deal with this occurrence [Knudsen 1987].

The distributed, concurrent, and mobile nature of context-aware ubiquitous systems impose limits to the classic exception-handling models. The contextual exception handling requires a context-aware exceptional control flow, consisting in notification-based reactions that involves several software elements [Damasceno et al. 2006, Kulkarni and Tripathi 2010, Mercadal et al. 2010].

[1] `http://rafalimaz.github.io/CatchML`

2.2. CAEHV - Context-Aware Exception Handling Verification

CAEHV is a method for model checking context-aware exception handling design proposed by [Rocha et al. 2013]. We summarize its properties as follows:

Detection Liveness states that, for each context state in the model, there should be at least one state in which each contextual exception is detected. The violation of this property indicates that exist contextual exceptions that cannot be detected. Such kind of design fault is called *dead exception*.

Catch Liveness states that, for each contextual exception raised, there is at least one handling case enabled to catch such exception. Its violation indicates that exist context states in the model that can raised contextual exception, but cannot be caught and, as a consequence, cannot be handled. Such kind of design fault is called *null handling*.

Handler Liveness states that, for each context state that raised a contextual exception, there is at least one context state in which each handling case is selected to handle this contextual exception. Its violation indicates that exist handle cases defined in a handling scope of a given contextual exception that never will be selected. Such kind of design fault is called *dead handler*.

Handling Stability states that, for each contextual exception that have been handled, the context state of the exceptional control-flow resumption cannot be one in which the same contextual exception can be raised again. Its violation indicates that the context-aware exceptional control-flow is in loop. Such kind of design fault is called *cyclic handling*.

Reachability states that, for each contextual exception raised and caught, it is possible to perform all handling measures and the control-flow can be resumed. Its violation indicates that the specified handling measures cannot conduct the system to a state where the resumed of control flow be possible. Such kind of design fault is called *resume impossible*.

3. The CatchML DSL

This section presents the domain-specific language *CatchML* that is an alternative solution for modeling and verification of context-aware exception handling (CAEH) design. The goal is to provide a high-level interface that allows designers to specify and to verify automatically the consistency of their CAEH models. The verification mechanism is integrated with the JCAEHV tool [Rocha et al. 2013]. An overview of the process is illustrated in Figure 1. Initially, the designer specifies the CAEH model of a system using the notations and constructors provided by the *CatchML* language. During the specification, the designer can make calls to the model checker by using the JCAEHV API. For each new call, the model specification is interpreted by the JCAEHV tool and the verification is performed. Then, the report provided by this verification is interpreted to generate a more efficient faults report. Thus, the faults are displayed directly in the specification code, making them easier to understand and to fix.

In the next subsections, we present the development phases followed to define the *CatchML* language.

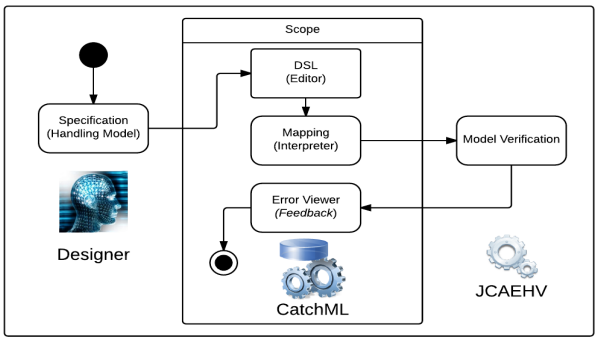

Figure 1. Overview of the process used by *CatchML*

3.1. Domain Study

The goal of this phase was to identify and describe the key concepts and abstractions relevant to the CAEH domain, and understand the relation between them. We used the concepts related to CAEHV [Rocha et al. 2013], together with other essential ones [Damasceno et al. 2006, Kulkarni and Tripathi 2010, Beder and de Araújo 2011, Cho and Helal 2012] to define the grammar of the *CatchML* language.

Rocha et al. [Rocha et al. 2013] state that it is important to observe some design decisions considered for modeling the adaptive behavior in a rigorous way. Such decisions include: (i) defining the form of abstract away the real values of context data; (ii) eliminating semantic inconsistencies in the context information; (iii) establish a valid state space to be explored; and (iv) prevent the occurring of inconsistent transitions between states. Thus, the following abstractions are adopted from the CAEHV method: *contextual propositions*, *semantic constraints*, *transition constraints*, *states of context*, *contextual exceptions*, *handling cases* and *handling scopes*.

The aforementioned abstractions provide a standard way to represent behavioral aspects regarding: the definition and detection of exceptional context; the grouping, selection and implementation of handling measures; and the resumption of the control flow. Furthermore, they have also been cited in other research studies [Damasceno et al. 2006, Kulkarni and Tripathi 2010, Filho et al. 2014] that address context-aware exception handling. We leverage the CAEHV method to provide a high-level solution for modeling and verification context-aware exception handling.

3.2. Design

The main goal of this phase was to transform a model specified in the *CatchML* language into a JCAEHV specification model.

The abstract syntax of the *CatchML* language[2] is a data structure that stores key information on a program. In such case, details about the concrete syntax notation such as keywords, symbols, comments and layout information are not stored. We have defined the language grammar in the EBNF format using Xtext, a framework that derives the abstract syntax as an instance of an Ecore metamodel, which is part of the Eclipse Modeling

[2] `http://rafalimaz.github.io/CatchML/images/catchml_xtextsyntaxgraph.png`

Framework (EMF)[3]. Although the abstract syntax and metamodel have arisen historically in different ways, they are typically considered as synonymous [Voelter et al. 2013]. Consequently, the formalisms to define abstract syntax trees are conceptually similar to the target metamodel. The metamodel of *CatchML* language[4] describes the entities that comprising a set of abstractions to express a context-aware exception model.

3.3. Implementation

The work proposed by Lincoln et al. [Rocha et al. 2013] provides the JCAEHV, a Java API for CAEH design and verification. Unfortunately, this tool requires designers to know many details of a general purpose programming language. These details could be abstracted away in order to make it easy and efficient to design and verify CAEH models. Our solution is a direct mapping of the *CatchML* language model to the Java API provided by JCAEHV to facilitate the CAEH design task and maintain the same level of functionality.

We also provide an automated integration with JCAEVH model verifier that allows its execution in an efficient way. Also, we present the semantic inconsistencies as a code snippet directly in the source specification. For instance, in the *Ubiparking* example (see Section 3.4), a design fault that violates the property *Detection Liveness* was injected in the fire exception scenario. This fault is detected as a *Dead Exception* and was showed as a code snippet in the specification by the *CatchML* verifier mechanism as shown in Figure 2.

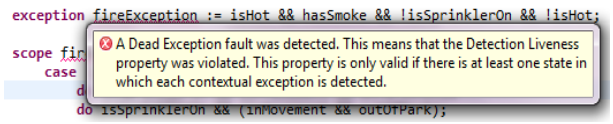

Figure 2. Design faults are shown directly in the specification model

3.4. CatchML Example

To illustrate how to specify a CAEH model using the *CatchML* language, we model a context-aware system called *UbiParking* (Listing 1). This system uses the contextual information from sensors to notify users about parking space availability. Moreover, there are refrigeration and ventilation control modules that also use the contextual information about the parking temperature provided by sensors. At last, there is a fire module that uses information collected by smoke detectors and sprinklers.

From this specification, it is possible to define two exceptional scenarios. The first scenario occurs when the system detects smoke presence, the temperature is high and the sprinklers are turned on (*fire* exception). Such situation characterizes fire and the system must behave according to the contextual information. For example, if the car is at the park place when the fire occurs, the car must go to the parking exit and then leave the parking. Or, if the car is at the parking exit, it only needs to leave the parking. The second scenario occurs when a car try to park in a place without free space (*no space* exception). In this scenario, if the car is in movement and at the parking place, it must go to the parking exit.

[3] http://www.eclipse.org/modeling/emf

[4] http://rafalimaz.github.io/CatchML/images/metamodel.png

```
1  system UbiParking {
2   prop atParkPlace, inMovement, atParkEntrance, atParkExit, hasSmoke, hasSpace,
        isSprinklerOn, isHot;
3   situation outOfPark := !atParkEntrance && !atParkPlace && !atParkExit;
4   sconstraint allPlacesDisjoined := xor(atParkEntrance, atParkPlace, atParkExit,
        outOfPark);
5   tconstraint t1 := pre outOfPark post !atParkPlace && !atParkExit;
6   tconstraint t2 := pre atParkEntrance post !atParkExit;
7   tconstraint t3 := pre atParkPlace post !atParkEntrance && !outOfPark;
8   exception fireException := isHot && hasSmoke && !isSprinklerOn && !isHot;
9   scope fireScope handle (fireException) {
10    case (inMovement && atParkPlace) {
11     do isSprinklerOn && (inMovement && atParkExit);
12     do isSprinklerOn && (inMovement && outOfPark);
13    }
14    case (inMovement && atParkExit) {
15     do isSprinklerOn && (inMovement && outOfPark);
16    }
17  }
18  exception noSpaceException := inMovement && atParkPlace && !hasSpace;
19  scope noSpaceScope handle (noSpaceException) {
20    case (inMovement && atParkPlace) {
21     do inMovement && atParkExit;
22    }
23  }
24 }
```

Listing 1. *UbiParking* specification in the CatchML language

4. Evaluation

This section presents the study conducted to evaluate the *CatchML* language. Our evaluation has the following objectives: 1. demonstrate that our language supports the modelling and verification of context-aware exception as a simple and intuitive alternative; and 2. assess whether or not our language provides an effective feedback to help designers to quickly locate the existent faults in the specification model.

4.1. Research Questions

The research questions and the metrics that we investigate are described as follows.

RQ1: How easy is to model the context-aware exception handling using the *CatchML* language?

Metric 1: The *average time* spent to specify a context-aware exception handling model; and

Metric 2: The *number of nonconformities* found in the model;

Metric 3: The amount of times (*executions*) that the participants run the *CatchML* verifier in Task 01;

RQ2: How easy is to collect feedback about the design faults into a CAEH model using the *CatchML* verifier?

To answer this research question, we should calculate the *average time* spent by the participants to analyze and remove the design faults. We also perform a qualitative evaluation (QE) of the *CatchML* verifier. For this QE, we elaborated a questionnaire to investigate the comprehension level of the participants regarding the design faults and the graphical user interface of the *CatchML* verifier. The questions are:

QE1: What are the kinds of *errors identified* without using the verifier? The participants should be able to identify the errors that injected the design faults in the specification.

QE2: What are the kinds of *violated properties* found into the specification? The participants should identify the violated properties displayed on the *CatchML* editor.

QE3: What are the kinds of *design faults* found into the specification? The participants should look at the design faults found by the verifier.

QE4: What are the kinds of *errors removed* with success into the specification? The participants should be able to fix correctly the design faults.

4.2. Methodology

We have conducted the evaluation with nine volunteers: three undergraduate and six graduate students of Computer Science at Federal University of Ceará. To balance the knowledge on the study domain, all participants first must attend to a training session on the CAEH modeling and verification. Next, they use the *CatchML* language and the environment to model the CAEH of the *UbiParking* by performing two modeling tasks:

1) Task 01[5] to check their understanding when using the *CatchML* language. To perform this task, the participants should receive a textual description of an exception handling model. Then, they should use the *CatchML* language to model the specification. Next, they should analyze the specification, i.e., search design faults by using the verifier. This task finishes when the participants fix the specification and no more errors are found in their model or when they give up to remove the faults.

2) Task 02[6] to analyse two (2) faulty CAEH specifications. To perform the Task 02, the participants choose one faulty specification to manually look for design faults. The participants are supported by the verification mechanism provided by the *CatchML* editor.

4.3. Results and Discussion

The results and discussion are described as follows.

RQ1: Ease of use. Table 1 shows the results for Task 01. The average time spent by the participants to complete the task was 33.11 minutes. The average amount of executions was seven. The average number of nonconformities was approximately one. These results may be an evidence that our language is easy to use and useful to help participants in modeling and verifying the CAEH.

RQ2: Comprehension level. Table 2 gives an overview of the results for Task 02. For each participant, we check whether he or she answered correctly the qualitative questions. We observed that most participants have identified correctly the logical errors without the support of the *CatchML* editor. Regarding the *feedback* provided by the *CatchML* verifier, we verified that most volunteers answered correctly the types of violated properties and the types of design faults pointed out by them. We also observed that the participants leverage the *CatchML* verifier to remove the design faults. Regarding the time to complete the task, the participants spent an average time of 15 minutes to analyze and remove the design faults.

[5] `http://rafalimaz.github.io/CatchML/zip/task1.zip`
[6] `http://rafalimaz.github.io/CatchML/zip/task2.zip`

Table 1. Data results for Task 01

Participant	Time (min)	Nonconformity (#)	Execution (#)
P1	16	0	3
P2	50	4	16
P3	12	0	2
P4	18	0	7
P5	50	1	6
P6	50	3	8
P7	41	2	9
P8	36	0	7
P9	25	1	6
Average	33	1	7

Table 2. Data results for Task 02

Participant	QE1	QE2	QE3	QE4	Time (min)
P1	✓	✗	✓	✓	13
P2	✗	✓	✗	✗	22
P3	✓	✓	✓	✓	7
P4	✓	✗	✗	✓	4
P5	✗	✓	✗	✗	30
P6	✓	✗	✗	✓	25
P7	✗	✓	✓	✓	6
P8	✓	✓	✓	✓	29
P9	✗	✓	✓	✓	5
Average	✓	✓	✓	✓	15

The results from Task 02 demonstrate an evidence that the participants had an acceptable comprehension level of the design faults found by the verification mechanism. So, the conclusion about RQ2 is that we have evidences that the *CatchML* editor has achieved its main goal, which is to provide an intuitive feedback to identify and remove design faults into a CAEH model.

5. Related Work

This section presents related approaches to the *context-aware exception handling*.

Damasceno et al. [Damasceno et al. 2006] list a set of core abstractions and propose a CAEH mechanism for mobile applications. To implement the solution, the authors use the middleware *MoCA* [Sacramento et al. 2004], which enables agents to perceive the context and allows to build mobile and collaborative applications in a publish-subscribe based model for coordination of software agents. The research work brings major contributions regarding the CAEH such as abstractions and concepts that serve as a knowledge base for this domain. However, the work does not provide support for modeling and verifying the context-aware exception behavior as we proposed in this paper.

Cho and Helal [Cho and Helal 2012] propose a new method to detect semantic exceptions through a high level language. This language has eight constructs that solve the problem of expressiveness such as *duration*, *flap*, *repeat*, etc. The authors focus on the detection of contextual exceptions by addressing critical issues related to our work such as gains in expressiveness and exceptions representation. However, the authors do not consider the essential aspects of CAEH, such as handlers selection, exception handling and resumption of control flow, that we have addressed.

Lincoln et al. [Rocha et al. 2013] proposes a model checking method for CAEH, the CAEHV. This model allows to verify the exceptional behavior of a context-aware system. The authors also present a set of behavioral properties (see Section 2.2) that ensures model consistency. The violation of such properties leads to design faults, which can be identified through the CAEHV method. However, the authors do not provide a high level interface to specify and verify CAEH models as we proposed. Indeed, our work reuses their mechanism to provide a high level verification.

Pryss and Reichert [Pryss and Reichert 2017] have developed a framework for providing an exception handling service during the execution of context-aware mobile activities. Unlike our work, the framework does not define a domain-specific language to handle the exceptions during the design-time.

6. Conclusions and Future Work

In this work, we have proposed a domain-specific language for context-aware exception handling modeling and verification, called *CatchML*. This language has been integrated with the JCAEHV tool to locate design faults and aims to make it even easier to found and reduce the design faults that may be introduced in the model specification. In addition, an alternative presentation of the design faults reported by JCAEHV is provided to facilitate the understanding and correction of the model. We also conducted an evaluation to verify the feasibility of our language. This study provides an evidence that *CatchML* language is feasible to help designers to model context-aware exception handling.

Based on our results, we believe that *CatchML* can increase the productivity of designers in modeling the context-aware exception handling of ubiquitous systems and avoid the introduction of specific types of design faults. As next steps, we consider to deal with (i) composition of adaptive and exceptional models - this work has focused only in modeling the context-aware exception handling behavior. One possible direction of research is to investigate a way to compose both behaviour models and understand how they interact. and (ii) concurrent exceptions modeling and exceptions propagation - based on the study of context-aware exception handling domain, one of the characteristic that a designer should be able to model is how the system will behave in the case of simultaneous occurrence of more than one exception.

References

Andrade, R. M. C., Carvalho, R. M., de Araújo, I. L., Oliveira, K. M., and Maia, M. E. F. (2017). What changes from ubiquitous computing to internet of things in interaction evaluation? In Streitz, N. and Markopoulos, P., editors, *Distributed, Ambient and Pervasive Interactions*, pages 3–21, Cham. Springer International Publishing.

Beder, D. M. and de Araújo, R. B. (2011). Towards the definition of a context-aware exception handling mechanism. In *Dependable Computing Workshops (LADCW), 2011 Fifth Latin-American Symposium on*, pages 25–28. IEEE.

Cacho, N., César, T., Filipe, T., Soares, E., Cassio, A., Souza, R., Garcia, I., Barbosa, E. A., and Garcia, A. (2014). Trading robustness for maintainability: An empirical study of evolving c# programs. In *Proc. of the ICSE'14*, pages 584–595.

Chetan, S., Ranganathan, A., and Campbell, R. (2005). Towards fault tolerance pervasive computing. *Technology and Society Magazine, IEEE*, 24(1):38–44.

Cho, E.-S. and Helal, S. (2011). A situation-based exception detection mechanism for safety in pervasive systems. In *2011 IEEE/IPSJ 11th International Symposium on Applications and the Internet (SAINT)*, pages 196 –201. IEEE.

Cho, E.-S. and Helal, S. (2012). Toward efficient detection of semantic exceptions in context-aware systems. In *Proc. of the 9th International Conference on Ubiquitous Intelligence Computing and 9th International Conference on Autonomic Trusted Computing (UIC-ATC)*, pages 826 –831.

Damasceno, K., Cacho, N., Garcia, A., Romanovsky, A., and Lucena, C. (2006). Context-aware exception handling in mobile agent systems: the moca case. In *Proc. of the International workshop on software engineering for large-scale multi-agent systems*, pages 37–44. ACM.

Filho, C. A. B. d. Q., Andrade, R. M. C., Rocha, L. S., Braga, R. B., and Oliveira, C. T. d. (2014). Conext-u: A context-aware exception handling mechanism for task-based ubiquitous systems. In *Proc. of 28th International Conference on Advanced Information Networking and Applications Workshops*, WAINA '14, pages 127–132, Washington, DC, USA. IEEE Computer Society.

Kienzle, J. (2008). On exceptions and the software development life cycle. In *Proceedings of the 4th International Workshop on Exception Handling*, WEH'08, pages 32–38, New York, NY, USA. ACM Press.

Knudsen, J. L. (1987). Better exception-handling in block-structured systems. *IEEE Software*, 4(3):40–49.

Kulkarni, D. and Tripathi, A. (2010). A framework for programming robust context-aware applications. *IEEE Transaction Software Engineering*, 36(2):184–197.

Loke, S. W. (2009). Building taskable spaces over ubiquitous services. *IEEE Pervasive Computing*, 8(4):72–78.

Mercadal, J., Enard, Q., Consel, C., and Loriant, N. (2010). A domain-specific approach to architecturing error handling in pervasive computing. In *Proceedings of OOPSLA '10*, pages 47–61, New York, NY, USA. ACM.

Parnas, D. L. and Würges, H. (1976). Response to undesired events in software systems. In *Proc. of ICSE'76*, pages 437–446, Los Alamitos, CA, USA. IEEE Computer Society.

Patikirikorala, T., Colman, A., Han, J., and Wang, L. (2012). A systematic survey on the design of self-adaptive software systems using control engineering approaches. In *Proceedings of the SEAMS'12*, pages 33–42, Piscataway, NJ, USA. IEEE Press.

Pryss, R. and Reichert, M. (2017). Context-based prevention and handling of exceptions for human-centric mobile services. In *2017 IEEE International Conference on AI Mobile Services (AIMS)*, pages 100–104.

Rocha, L., Andrade, R., and Garcia, A. (2013). A method for model checking context-aware exception handling. In *Software Engineering (SBES), 2013 27th Brazilian Symposium on*, pages 59–68.

Rocha, L. S. and Andrade, R. M. C. (2012). Towards a formal model to reason about context-aware exception handling. In *Exception Handling (WEH), 2012 5th International Workshop on*, pages 27 –33.

Sacramento, V., Endler, M., Rubinsztejn, H. K., Lima, L. S., Goncalves, K., Nascimento, F. N., and Bueno, G. A. (2004). Moca: A middleware for developing collaborative applications for mobile users. *Distributed Systems Online, IEEE*, 5(10):2–2.

Voelter, M., Benz, S., Dietrich, C., Engelmann, B., Kats, L., Helander, M., Visser, E., and Wachsmuth, G. (2013). *DSL Engineering - Designing, Implementing and Using Domain-Specific Languages*. dslbook.org.

Weiser, M. (1991). The computer for the 21st century. *Scientific American*, 265(3):94–104.