Mälardalen University Licentiate Thesis 278

Formally Assured Intelligent Systems for Enhanced Ambient Assisted Living Support

Ashalatha Kunnappilly

MÄLARDALEN UNIVERSITY
SWEDEN

Mälardalen University Press Licentiate Theses
No. 278

FORMALLY ASSURED INTELLIGENT SYSTEMS FOR ENHANCED AMBIENT ASSISTED LIVING SUPPORT

Ashalatha Kunnappilly

2019

School of Innovation, Design and Engineering

ISBN 978-91-7485-425-1
ISSN 1651-9256
Printed by E-Print AB, Stockholm, Sweden

Abstract

Ambient Assisted Living (AAL) solutions are aimed to assist the elderly in their independent and safe living. During the last decade, the AAL field has witnessed a significant development due to advancements in Information and Communication Technologies, Ubiquitous Computing and Internet of Things. However, a closer look at the existing AAL solutions shows that these improvements are used mostly to deliver one or a few functions mainly of the same type (e.g. health monitoring functions). There are comparatively fewer initiatives that integrate different kinds of AAL functionalities, such as fall detection, reminders, fire alarms, etc., besides health monitoring, into a common framework, with intelligent decision-making that can thereby offer enhanced reasoning by combining multiple events.

To address this shortage, in this thesis, we propose two different categories of AAL architecture frameworks onto which different functionalities, chosen based on user preferences, can be integrated. One of them follows a centralized approach, using an intelligent Decision Support System (DSS), and the other, follows a truly distributed approach, involving multiple intelligent agents. The centralized architecture is our initial choice, due to its ease of development by combining multiple functionalities with a centralized DSS that can assess the dependency between multiple events in real time. While easy to develop, our centralized solution suffers from the well-known single point of failure, which we remove by adding a redundant DSS. Nevertheless, the scalability, flexibility, multiple user accesses, and potential self-healing capability of the centralized solution are hard to achieve, therefore we also propose a distributed, agent-based architecture as a second solution, to provide the community with two different AAL solutions that can be applied depending on needs and available resources. Both solutions are to be used in safety-critical applications, therefore their design-time assurance, that is, providing a guarantee that they meet

functional requirements and deliver the needed quality-of-service, is beneficial.

Our first solution is a generic architecture that follows the design of many commercial AAL solutions with sensors, a data collector, DSS, security and privacy, database (DB) systems, user interfaces (UI), and cloud computing support. We represent this architecture in the Architecture Analysis and Design Language (AADL) via a set of component patterns that we propose. The advantage of using patterns is that they are easily re-usable when building specific AAL architectures. Our patterns describe the behavior of the components in the Behavioral Annex of AADL, and the error behavior in AADL's Error Annex. We also show various instantiations of our generic model that can be developed based on user requirements. To formally assure these solutions against functional, timing and reliability requirements, we show how we can employ exhaustive model checking using the state-of-art model checker, UPPAAL, and also statistical model-checking techniques with UPPAAL SMC, an extension of the UPPAAL model checker for stochastic systems, which can be employed in cases when exhaustive verification does not scale. The second proposed architecture is an agent-based architecture for AAL systems, where agents are intelligent entities capable of communicating with each other in order to decide on an action to take. Therefore, the decision support is now distributed among agents and can be used by multiple users distributed across multiple locations. Due to the fact that this solution requires describing agents and their interaction, the existing core AADL does not suffice as an architectural framework. Hence, we propose an extension to the core AADL language - The Agent Annex, with formal semantics as Stochastic Transition Systems, which allows us to specify probabilistic, non-deterministic and real-time AAL system behaviors. In order to formally assure our multi-agent system, we employ the state-of-art probabilistic model checker PRISM, which allows us to perform probabilistic yet exhaustive verification.

As a final contribution, we also present a small-scale validation of an architecture of the first category, with end users from three countries (Romania, Poland, Denmark). This work has been carried out with partners from the mentioned countries.

Our work in this thesis paves the way towards the development of user-centered, intelligent ambient assisted living solutions with ensured quality of service.

Sammanfattning

Ambient Assisted Living (AAL) lösningar är riktade för att assistera äldre till ett självständigt och säkert leverne. AAL har under det senaste årtiondet fått ett stort uppsving, mycket tack vare framsteg inom informations- och kommunikationsteknologier, Ubiquitous Computing och Internet of Things (IoT). En närmare granskning av nuvarande AAL lösningar visar dock på att dessa framsteg främst levererar endast en eller ett fåtal funktioner, oftast av samma typ, t.ex. (funktioner för att bevaka hälsa). Det finns jämförelsevis mycket färre initiativ som integrerar olika sorters AAL funktioner som falldetektering, påminnelser, brandlarm etc., förutom hälsobevakning till ett gemensamt ramverk som har intelligent beslutsfattande och därmed bättre förutsättning att kombinera flera olika händelser.

I denna avhandling föreslår vi två olika kategorier av AAL ramverksarkitekturer som implementerar användaranpassade funktionaliteter för att adressera ovanstående problem. Den ena kategorin har en centraliserad approach och använder intelligent Decision Support System (DSS). Den andra kategorin har en distribuerad approach och innefattar flera intelligenta agenter. Den centraliserade arkitekturen är vårt förstahandsval på grund av den enkelheten att utveckla genom att kombinera flera funktionaliteter med ett centraliserat DSS som kan utvärdera beroendes mellan flera händelser I real-tid. Genom att addera ytterligare ett redundant DSS har vi även uteslutit den välkända Single Point of Failure problematiken. Skalbarhet, flexibilitet, självläkande förmåga samt åtkomst för flera användare hos vår centraliserade lösning är svårt att uppnå, vilket är anledningen till att vi även presenterar en distribuerad, agentbaserad arkitektur som andrahandslösning som används vid behov. Båda dessa lösningar kommer att användas i säkerhetskritiska applikationer. Lösningarnas designtidsförsäkran, det vill säga att garantin att de uppfylla kan de funktionella krav som ställs samt leverans av nödvändig servicekvalitet är där-

för fördelaktig.

Vår första lösning är en generisk arkitektur, utformad enligt andra kommersiella AAL-lösningar med sensorer, datasamlare, DSS, säkerhet och integritet, databas (DB) system, användargränssnitt (UI) och Cloud Computing stöd. Vi specificerar Architecture Analysis and Design Language (AADL) via en uppsättning av komponentmönster som vi föreslår. Fördelen med att använda mönster är att de lätt återanvänds när man bygger specifika AAL-arkitekturer. Våra mönster beskriver beteendet hos komponenterna i AADLs beteendeannex och felbeteendet i AADL: s felannex., vi visar även olika instanser av vår generiska modell som kan utvecklas utifrån användarnas krav. Genom att använda hjälp av den toppmoderna modellkontrollen UPPAAL försäkrar vi även att dessa lösningar tillmötesgår de funktionella, tidsmässiga och tillförlitliga kraven. Vi använder även en statistisk modellkontrollsteknik genom UPPAAL SMC vilket är en förlängning av UPPAAL modell checker för stokastiska system som används i de fall då en uttömmande verifiering inte är möjlig. Vår andra arkitektur är en agent-baserad arkitektur för AAL-system, där agenter är intelligenta enheter kommunicerar med varandra för att komma fram till beslut om nödvändiga åtgärder. Beslutsfattandet fördelas nu istället mellan agenter och kan användas av flera användare fördelade på flera platser. Denna lösning kräver dock en beskrivning av agenter samt deras interaktion vilket innebär att den befintliga kärnan AADL inte räcker som enda ramverk. Därför föreslår vi en utvidgning till det centrala AADL-språket - Agent Annexet, som har en formell semantik likt Stochastic Transition Systems, vilket gör att vi kan specificera probabilistiska, icke-deterministiska och real-tids system beteenden inom AAL. Vi använder den toppmoderna probabilistiska modellkontrollen PRISM, som gör det möjligt för oss att utföra en probabilistisk, men uttömmande verifiering av vårt multi-agent system.

Slutligen presenterar vi också en mindre omfattande validering av en arkitektur i den första kategorin, med slutanvändare från tre länder (Rumänien, Polen, Danmark). Detta arbete har utförts med partner från de nämnda länderna.

Vårt arbete i denna avhandling banar väg mot utveckling av användarcentrerade, intelligent ambient-assisted lösningar med garanti för servicekvalitet.

To my husband, Kiran

Acknowledgements

It is with immense gratitude that I write this section. First of all, I would like to sincerely thank my supervisors- Associate Professor Cristina Seceleanu, and Professor Maria Linden for their support, guidance and patience. Thank you for believing in me and giving me an opportunity to undertake PhD studies. Also, special thanks to Dr. Raluca Marinescu for her supervision during her Postdoc employment at MDH. Without all of your guidance and support, this thesis would not have been possible.

Next, I would like to thank all the professors and lecturers at the university for the knowledge they shared. It was a pleasure being with you all and learning new things. Many thanks to my fellow PhD students and the staff at the department. I really enjoyed the time spent with you guys! I would especially like to thank my office mates - Predrag and Nesredin for all the wonderful times we had. I will really miss you guys. Next, I would like to thank the rest of my amazing group - Simin and Rong, you guys are amazing and thanks for all the help and support. I know it is impossible to mention all the people, but this section would be completely meaningless if I don't mention some names. Aida- thanks a lot for all the strong advises and support you have given me. It really means a lot. Gita - thanks for being a wonderful friend. My sincere thanks to Leo, Sveta, Hamidur, Gabriel, Sara, Momo, Filip, Sara A. (2), Jakob, Nabar, Mobyen, Shahina, Elena, Lana etc. etc. for all the helps, friendly chats and discussions.

I would like to thank my opponent Associate Professor Elena Troubitsyna, and the grading committee members Professor Einar Broch Jonsen and Associate Professor Antonio Cicchetti for accepting the invite and taking time in reviewing this thesis.

Last but not least, I would like to thank my family and family friends. To the love of my life, Kiran - words wont suffice to describe what you mean to me

and the support and love you have given me in our 5 years of togetherness. Next to my parents- I cannot express what I feel for them, how much I love them, and how proud I am to be their daughter. To my in-laws, Kiran's mom dad- thanks a lot for accepting me as your daughter, for the freedom and support you have given me. It means a lot. Next, to my brother, Kishore- there is no one like you, thank you for the unconditional love. Heartfelt gratitude to all my school friends, bachelor and master college mates and all the friends we have at India and Sweden for the love and support you have given me. I would also like to thank our family friend, Manoj Bhaskar, for motivating me to apply for PhD positions. And finally, above all, to God, for being my inner strength.

Ashalatha Kunnappilly
Västerås, March, 2019

List of publications

Publications included in the licentiate thesis[1]

Paper A *Do we need an integrated framework for Ambient Assisted Living?*. Ashalatha Kunnappilly, Cristina Seceleanu, Maria Lindén. In Proceedings of the 10th International Conference on Ubiquitous Computing and Ambient Intelligence (UCAmI), LNCS, Springer, pages 52-63, November 2016, Canary Islands, Spain.

Paper B *A Novel Integrated Architecture for Ambient Assisted Living Systems*. Ashalatha Kunnappilly, Alexandru Sorici, Imad Alex Awada, Irina Mocanu, Cristina Seceleanu, Adina Madga Florea. In Proceedings of the 40th IEEE Computer Society International Conference on Computers, Software & Applications (COMPSAC), July 2017, Turin, Italy, IEEE Computer Society, pages 465-472.

Paper C *A Model-Checking-Based Framework For Analyzing Ambient Assisted Living Solutions*. Ashalatha Kunnappilly, Raluca Marinescu, Cristina Seceleanu. MRTC Report, Mälardalen Real-Time Research Center, MDH-MRTC-322/2018-1-SE, March, 2019. *NOTE:* This paper is an extended version of the following article: *Assuring Intelligent Ambient Assisted Living Solutions by Statistical Model Checking*. Ashalatha Kunnappilly, Raluca Marinescu, Cristina Seceleanu. In Proceedings of the 8th International Symposium On Leveraging Applications of Formal Methods, Verification and Validation (ISoLA), November 2018, Limassol, Cyprus, Springer, pages 457-476.

[1] The included articles have been reformatted to comply with the licentiate thesis layout.

In Proceedings of the 8th International Symposium On Leveraging Applications of Formal Methods, Verification and Validation (ISoLA), November 2018, Limassol, Cyprus, Springer, pages 457-476.

Paper D *Architecture Modelling and Formal Analysis of Intelligent Multi-Agent Systems*. Ashalatha Kunnappilly, Simin Cai, Raluca Marinescu, Cristina Seceleanu. In Proceedings of the 14th International Conference on Evaluation of Novel Approaches to Software Engineering (ENASE), Crete, Greece, SCITEPRESS, May 2019.

Paper E *An end-user perspective on the CAMI Ambient Assisted Living Framework*. Imad Alex Awada, Oana Cramariuc, Irina Mocanu, Cristina Seceleanu, Ashalatha Kunnappilly, Adina Magda Florea. In Proceedings of the 12th Annual International Technology, Education and Development Conference (INTED), Edulearn, March 2018, Spain.

Additional publications, not included in the licentiate thesis

1. *CAMI - An Integrated Architecture Solution for Improving Quality of Life of the Elderly*. Alexandru Sorici , Imad Alex Awada , Ashalatha Kunnappilly, Irina Mocanu , Oana Cramariuc , Lukasz Malicki , Cristina Seceleanu, Adina Magda Florea. In Proceedings of the 3rd EAI International Conference on IoT Technologies for HealthCare (HealthyIoT), 2016, Springer, LNCS.

2. *Analyzing Ambient Assisted Living Solutions: A Research Perspective*. Ashalatha Kunnappilly, Axel Legay, Tiziana Margaria, Cristina Seceleanu, Bernhard Steffen, Louis-Marie Tranonouez. 12th International Conference on Design and Technology of Integrated Systems in Nanoscale Era (DTIS), 2017, IEEE.

3. *A Formally Assured Intelligent Ecosystem for Enhanced Ambient Assisted Living Support*. Ashalatha Kunnappilly. The 33rd ACM/SIGAPP Symposium On Applied Computing (SAC), Student Research Competition (**Second position**), 2018, ACM.

4. *A Systematic Mapping Study on Real-time Cloud Services*. Jakob Danielsson, Nandinbaatar Tsog, Ashalatha Kunnappilly. IEEE/ACM International Conference on Utility and Cloud Computing Companion (UCC Companion), 2018, IEEE.

Contents

I

Thesis

Chapter 1

Introduction

According to the statistics of the World Population Ageing Report 2015 [1], the world's elderly population is predicted to reach 2.1 billion by 2050, which is more than double of the population of elderly adults in 2015. The ageing society entails coping with increased health-care costs, shortage of caregivers [2], etc. Ambient Assisted Living (AAL) solutions are gaining popularity in this context, as they can assist the elderly in their daily activities and in their independent living, with limited risks. Some examples of assistance are health monitoring, home monitoring, fall detection, communication with caregivers, mobility, providing recommendations, reminders, etc.

AAL systems are real-time safety-critical systems, i.e., not delivering the right functionality at the right time may have consequences that could even lead to the death of the elderly user. For example, most of the AAL systems use sensors to monitor health parameters like pulse, ECG, blood glucose level, blood pressure, etc. In many cases, health parameter deviations are critical and must be notified to the caregiver in due time, and the failure to do so can endanger the life of the elderly. Hence, early design-stage assurance via techniques like model checking can uncover potential errors before their propagation to implementation levels, or simply provide a guarantee that the design meets the specification.

Upon undertaking a survey of existing AAL solutions [3], we find that many of the existing ones have limited support of functionalities, despite the fact that helping an older adult in his/her daily living requires supporting health-related functions, but also home and social-life related functionalities. Although the above holds, one can use various independent systems providing

one or more of such supporting functions. However, there exist potentially critical scenarios that such solutions cannot resolve in due time, which justify the architectural integration of independent solutions. For instance, if an elderly person's home is equipped with an AAL solution that does not support automatic fall detection, the user can purchase it separately, as there exist readily available solutions that detect a fall and raise an alarm. This functionality of a fall detection system remains the same whether it is an independent system or part of an integrated system. The question that follows is: if the separate solutions can perform their functionality without integration, in isolation, then why would one even think of designing an integrated, more complex solution that in addition comes at a higher price? The most obvious reason for a positive answer would be the increased practicality of a single, integrated solution that offers all the needed support vs. more systems each delivering a particular function, which the users need to purchase. However, there exists a more important reason - the fact that the performance of individualized solutions differ dramatically if they are integrated into a coherent framework, versus the case when they are employed in isolation especially in scenarios where critical events might occur simultaneously. In some cases, independent solutions cannot even depict a potential causality between simultaneous critical events, as we exemplify below.

In our **first** contribution [3], we discuss the behaviors of independent and integrated solutions by selecting representative scenarios that we simulate via sequence diagrams, and check their offline schedules against real-time deadlines. As a result, we conclude that certain critical scenarios can be tackled intelligently only by using **integrated** solutions. For instance, let us assume the following scenarios:

- A fall event occurring due to low pulse: In this case, if the fall sensor and the pulse monitoring sensor work independently of each other, no connection can be established between the two events, indicating that the potential reason for the fall is in fact the person's low pulse, which in turn may be critical for diagnosis.

- Simultaneous occurrence of fire and fall events: When both these events occur together, a safe mitigation of the scenario is achieved only when both these events are communicated to caregivers and firefighters, which is not guaranteed by independent systems working side by side. Assuming that the fire alarm communicated to the firefighters is verified for confirmation by a phone call to the user's home, and since the elderly who has fallen may not be able to answer, the fire alarm may be deemed

false and discarded, triggering a potential catastrophe.

Justified by the above, we establish the fact that the need of integrated AAL solutions that cater for various types of functions is veridical [3]. The next challenge is to develop such systems that can integrate multiple functionalities and deliver them correctly. When AAL solutions are integrated such that they cover a wide variety of functionalities [4], out of which many are safety critical, ensuring the correctness of the system behavior by verifying the functional and quality-of-service (QoS) attributes of the system at the design stage is beneficial. In this thesis, we propose two integrated solutions for AAL systems: a) An architecture with centralized artificial intelligence (AI)-based decision support, and b) An architecture with distributed decision-making using multiple intelligent agents that cooperate with each other. We also show the correctness of the proposed solutions at design level.

The integration of various functionalities can be easily accomplished if there exists a centralized decision maker that all the various devices communicate to, such that different events can be combined in real-time. This prompts us to our **second** contribution as a *centralized* integrated AAL solution that we describe in the Architecture Analysis and Design Language (AADL) [5, 6]. We design our centralized architecture as a generic model that follows the AAL architecture design in the literature by: (i) integrating multiple sensors, data collector unit, decision-support systems, cloud computing facilities, communication gateways and user-interfaces, and (ii) incorporating redundancy to the decision-support systems (local and cloud) to tackle the single point of failure in centralized systems, hence increasing the system's fault tolerance [3, 6]. When modeling this architecture in AADL, we follow a pattern-based modeling approach that facilitates the models' reuse. By using AADL's Behavior Annex (BA), we specify the AI support, combining context modeling, fuzzy logic, case-based reasoning and rule-based reasoning. The combination of the various AI techniques also strengthens the decision-making support of the AAL architecture. Using the AADL model, we perform initial analysis like latency analysis, schedulability, and resource analysis, within the OSATE platform [7].

The generic architecture model can also be customized to address different user requirements and preferences. In this thesis, we show three different instantiated versions of the generic model, that is, a minimal configuration, an intermediate configuration and a complex configuration, modelled in AADL. We give formal semantics to the “semi-formal" AADL modeling constructs of the type used in our work, in the framework of stochastic timed automata [8]. In order to formally analyze the system against various functional and quality-

of-service (QoS) attributes, we show exhaustive verification of the minimal configuration using the UPPAAL model checker and statistical model checking of complex configuration using UPPAAL SMC [9]. The reason for employing statistical model checking is twofold: a) exhaustive model-checking might not scale for large complex systems, and (ii) we model the failure probabilities of various components, and hence the choice of reasoning statistically is justified. Our modeling and verification approach facilitate reuse via a pattern-based modeling infrastructure, covers AI support, and is able to cover a larger set of properties for verification , as compared to existing approaches to AAL system formal modeling and analysis [10, 11]. In addition, most of the commercially available AAL solutions lack a documented proof of correctness [5]. However, our first solution has the same disadvantages as all centralized solutions, that is: (i) redundancy overheads due to ensuring fault tolerance, and (ii) limited scalability and adaptivity.

Our **third** contribution and second architectural solution [12] follows the upcoming trend of using distributed architectures for designing AAL systems, as they provide autonomy, scalability, adaptability and fault-tolerance, in addition to the fact that it servers multiple users at the same time. Hence, we propose a *distributed agent-based AAL solution*, as the second category of architectures that support the design of AAL systems. However, such systems usually possess additional overhead encountered during agent synchronizations for collective decision-making and data consistency maintenance. This overhead can sometimes hamper the real-time behavior of the system. To address this, we investigate how we can use these systems for developing integrated solutions that ensure a safe trade off between autonomous behavior and consistency overheads. This is a challenging requirement since agents are inter-dependent, and have only a limited view of the environment. Concretely, the agent-based solution should ensure a consistent view of the environment, in terms of processed data and events, as well as an inter-agent communication overhead that should not result in breaching the real-time system demands.

Our agent-based architecture consists of independent agents that cater for a particular functionality, respectively, for e.g., a health monitoring agent detects health parameter variations and raise a notification to caregiver. Our architecture supports interactions between different categories of agents. In this thesis, we consider only 2 agent categories: a) simple reflex agents, with reasoning based on if-then-else rules, and b) self-learning intelligent agents, embedded with AI learning algorithms, like Reinforcement Learning [13]. In order for the agents to cooperate in real-time, each agent maintains the dependencies it can have with other agents. For example, if a health-monitoring agent detects

that there is a high pulse, it would need to cooperate with an activity agent to determine the user activity, a high pulse during an exercise session is normal and no notifications should be generated. Hence, the activity agent is included in the dependency list of the health monitoring agent. For formally modeling the agent-based architecture, existing architecture languages such as AADL cannot specify autonomy, adaptability, self-healing, self-learning etc., as these behaviours are usually non-deterministic, probabilistic and have real-time constraints. To describe the agents and the system's architecture, we propose an extension to AADL specification language as a sub-language called *Agent annex*, and define its semantics described in terms of Stochastic Transition Systems [14].

As the **fourth** and final contribution, we also present some initial validation of the centralized architecture by testing some of the implemented functionalities with end-users and in the laboratory [15]. In this contribution, we present an implemented version of the architecture of the first category. The functionalities chosen for implementation are based on user surveys undertaken by the end user organizations within the project. We show the validation results with respect to functionalities like health monitoring (i.e. blood pressure, heart rate, blood glucose, weight, blood oxygenation), fall detection, supervised physical exercises and vocal interactions.

1.1 Thesis Overview

The thesis is divided into two major parts. The first part is an overall summary of the thesis, organized as follows. In Chapter 2, we give a short overview of the preliminaries; in Chapter 3, we describe the research method used for conducting the research and producing the research results described in the thesis. Chapter 4 introduces the research goals of the thesis. In Chapter 5, we briefly describe the contributions of the thesis, and map them to the corresponding research goals, respectively. The overview and comparison to the related work is given in Chapter 6, after which we conclude the first part of the thesis and present the directions for future work in Chapter 7.

The second part of the thesis is given as a collection of publications that encompass all the thesis contributions. The included papers are:

Paper A. *Do we need an integrated framework for Ambient Assisted Living?*. Ashalatha Kunnappilly, Cristina Seceleanu, Maria Lindén. In Proceedings of the 10th International Conference on Ubiquitous Computing and Ambient

Intelligence (UCAmI), LNCS, Springer, pages 52-63, November 2016, Canary Islands, Spain .

Abstract. The significant increase of ageing population calls for solutions that help the elderly to live an independent, healthy and low risk life, but also ensure their social interaction. The improvements in Information and Communication Technologies (ICT) and Ambient Assisted Living (AAL) have resulted in the development of equipment that supports ubiquitous computing, ubiquitous communication and intelligent user interfaces. The smart home technologies, assisted robotics, sensors for health monitoring and e-health solutions are some examples in this category. Despite such growth in these individualized technologies, there are only few solutions that provide integrated AAL frameworks that interconnect all of these technologies. In this paper, we discuss the necessity to opt for an integrated solution in AAL. To support the study we describe real life scenarios that help us justify the need for integrated solutions over individualized ones. Our analysis points to the clear conclusion that an integrated solution for AAL outperforms the individualized ones.

Contributions. I was the main contributor to this work and the main driver for the paper. I performed a literature review of the SOA and SOP of existing AAL solutions and identified that there are very few AAL solutions that are fully integrated w.r.t functionalities chosen based on a multi-national survey conducted in the same research project (CAMI EU project) by end-user organizations. I also performed an analysis of timing requirements for integrated and non-integrated AAL solutions in certain critical scenarios via sequence diagrams and offline schedules. I was helped by the second author to formulate the scenarios and to select the tools for analysis. The second and third authors also provided constructive feedback for the paper.

Paper B. *A Novel Integrated Architecture for Ambient Assisted Living Systems*.Ashalatha Kunnappilly, Alexandru Sorici, Imad Alex Awada, Irina Mocanu, Cristina Seceleanu, Adina Madga Florea. In Proceedings of the 40th IEEE Computer Society International Conference on Computers, Software & Applications (COMPSAC), July 2017, Turin, Italy, IEEE Computer Society, pages 465-472.

Abstract. The increase in life expectancy and the slumping birth rates across the world result in lengthening the average age of the society. There-

fore, we are in need of techniques that will assist the elderly in their daily life, while preventing their social isolation. The recent developments in Ambient Intelligence and Information and Communication Technologies have facilitated a technological revolution in the field of Ambient Assisted Living. At present, there are many technologies on the market that support the independent life of older adults, requiring less assistance from family and caregivers, yet most of them offer isolated services, such as health monitoring, reminders etc; moreover none of current solutions incorporates the integration of various functionalities and user preferences or are formally analyzed for their functionality and quality-of-service attributes, a much needed endeavor in order to ensure safe mitigations of potential critical scenarios. In this paper, we propose a novel architectural solution that integrates necessary functions of an AAL system seamlessly, based on user preferences. To enable the first level of the architecture's analysis, we model our system in Architecture Analysis and Design Language, and carry out its simulation for analyzing the end-to-end data-flow latency, resource budgets and system safety.

Contributions. I was the main driver for the paper. My technical contributions include the AADL analysis of selected architectures from the literature, proposing a novel architecture solution with local and cloud processing as a platform for seamless integration of various AAL functionalities chosen based on Paper 1, evaluation of the proposed architecture in AADL for latency, resource budgets and failure. The other authors contributed with ideas on architecture design, and comments on the paper.

Paper C. *A Model-Checking-Based Framework For Analyzing Ambient Assisted Living Solutions*. Ashalatha Kunnappilly, Raluca Marinescu, Cristina Seceleanu, MRTC Technical Report, Mälardalen Real-Time Research Centre, Mälardalen University, Mälardalen University Press, MDH-MRTC-322/2018-1-SE, March 2019. *NOTE:* This publication is an extended version of the article: *Assuring Intelligent Ambient Assisted Living Solutions by Statistical Model Checking*. Ashalatha Kunnappilly, Raluca Marinescu, Cristina Seceleanu. In Proceedings of the 8th International Symposium On Leveraging Applications of Formal Methods, Verification and Validation (ISoLA), November 2018, Limassol, Cyprus, Springer, pages 457-476.

Abstract. Since modern ambient assisted living solutions integrate a multitude of assisted-living functionalities within a common design framework, some are safety-critical, it is desirable that these systems are analyzed

already at their design stage to detect possible errors. To achieve this, one needs suitable architectures that support the seamless design of the integrated assisted-living functions, as well as capabilities for the formal modeling and analysis of the architecture. In this paper, we attempt to address this need, by proposing a generic integrated ambient assisted living system architecture, consisting of sensors, data-collector, local and cloud processing schemes, and an intelligent decision support system, which can be easily extended to suite specific architecture categories. Our solution is customizable, therefore, we show three instantiations of the generic model, as simple, intermediate and complex configuration, respectively, and show how to analyze the first and third categories by model checking. Our approach starts by specifying the architecture, using an architecture description language, in our case, the Architecture Analysis and Design Language that can also account for the probabilistic behavior of such systems. To enable formal analysis, we describe the semantics of the simple and complex categories as stochastic timed automata. The former we model check exhaustively with UPPAAL, whereas for the latter we employ statistical model checking using UPPAAL SMC, the statistical extension of UPPAAL, for scalability reasons. Our work paves the way for the development formally-assured future ambient assisted living solutions.

Contributions. I was the main driver of the paper. My technical contributions include: (i) a generalized architecture framework for AAL systems with centralized decision support, (ii) the pattern-based design of the system architecture in AADL, (iii) the design of an Intelligent Decision Support System combining context modeling, rule-based reasoning, fuzzy logic and case-based reasoning and its pattern-based model, (iii) formal semantics of the AADL patterns, and (iv) verification of the functional specification, and QoS of the system, including its DSS. The other two authors provided ideas w.r.t the modeling and verification of the AAL system, as well as feedback for the paper.

Paper D. *Architecture Modelling and Formal Analysis of Intelligent Multi-Agent Systems*. Ashalatha Kunnappilly, Simin Cai, Raluca Marinescu, Cristina Seceleanu. Accepted in 14th International Conference on Evaluation of Novel Approaches to Software Engineering (ENASE 2019), Crete, Greece, May 2019.

Abstract. Modern cyber-physical systems usually assume a certain degree of autonomy. Such systems, like Ambient Assisted Living systems

aimed at assisting elderly people in their daily life, often need to perform safety-critical functions, for instance, fall detection, health deviation monitoring, communication to caregivers, etc. In many cases, the system users have distributed locations, as well as different needs that need to be serviced at the same time. These features call for adaptive, scalable and fault-tolerant system design solutions, which are well embodied by multi-agent architectures. Analyzing such complex architectures at design phase, to verify if an abstraction of the system satisfies all the critical requirements is beneficial. In this paper, we start from an agent-based architecture for ambient assisted living systems, inspired from the literature, which we model in the popular Architecture Description and Design Language. Since the latter lacks the ability to specify autonomous agent behaviours, which are often non-deterministic or probabilistic, we extend the architectural language with a sub-language called Agent Annex, which we formally encode as a Stochastic Transition System. This contribution allows us to specify behaviours of agents involved in agent-based architectures of cyber-physical systems, which we show how to exhaustively verify with the state-of-art model checker PRISM. As a final step, we apply our framework on a distributed ambient assisted living system, whose critical requirements we verify with PRISM.

Contributions. I was the main driver for the paper. I designed the intelligent multi-agent system architecture for Ambient Assisted Living Systems (intelligence is incorporated by employing self-learning techniques in agents using Reinforcement Learning). Also, I proposed the extension to the existing AADL modeling framework for modeling agent behaviours- the Agent Annex and formulated its semantics. Further, I also developed the PTA model of the corresponding AADL model that could be model-checked via PRISM. Simin Cai helped in writing the introduction and preliminaries of PRISM and also helped in formatting diagrams, other sections and also gave valuable suggestions and feedback in developing the PTA model and its verification in PRISM. The other authors provided valuable comments and feedback both on the approach and on the final version of the paper.

Paper E. *An end-user perspective on the CAMI Ambient Assisted Living Framework*.mad Alex Awada, Oana Cramariuc, Irina Mocanu, Cristina Seceleanu, Ashalatha Kunnappilly, Adina Magda Florea. In Proceedings of the 12th Annual International Technology, Education and Development Conference (INTED), Edulearn, March 2018, Spain

Abstract. In this paper, we present the outcomes and conclusions obtained by involving seniors from three countries (Denmark, Poland and Romania) in an innovative project funded under the European Ambient Assisted Living (ALL) program. CAMI stands for “Companion with Autonomously Mobile Interface" in “Artificially intelligent ecosystem for self-management and sustainable quality of life in AAL". The CAMI solution enables flexible, scalable and individualised services that support elderly to self-manage their daily life and prolong their involvement in the society (sharing knowledge, continue working, etc). This also allows their informal caregivers (family and friends) to continue working and participating in society while caring for their loved ones. The solution is designed as an innovative architecture that allows for individualized, intelligent self-management which can be tailored to an individual’s preferences and needs. A user-centered approach has ranked health monitoring, computer supervised physical exercises and voice based interaction among the top favoured CAMI functionalities. Respondents from three countries (Poland, Romania and Denmark) participated in a multinational survey and a conjoint analysis study.

Contributions. The second author was the main driver of the work. I contributed to the paper by proposing a smaller implementable version of the CAMI architecture proposed in Paper 2, took part in DSS implementation and in testing the fall detection functionality using Vibby sensors in laboratory. I have also contributed to the writing and reviewing of the paper, along with the other co-authors.

Chapter 2

Preliminaries

In this chapter, we introduce the preliminary concepts that are used throughout the thesis. First, in Section 2.1 we present the Architecture Analysis and Design Language. Next, in Section 2.2 we give an overview of agents and multi-agent systems. In Section 2.3, we present an overview of the formal modeling, verification and analysis techniques and tools used in the thesis.

2.1 Architecture Analysis and Design Language

AADL [16] is a textual and graphical language in which one can model and analyze a real-time system's hardware and software architecture as hierarchies of components at various levels of abstraction. There are three categories of component abstractions in AADL: Application Software (*Process, Data, Subprogram, Thread,* and *Thread Group*, etc.), Execution Platform (*Device, Bus, Processor, Memory,* etc.), and general composite components (*System* and *Abstract*). *System* components are the top-level components. A *process* component contains a set of *thread* components that define the dynamic behavior of the process. AADL component categories like *Application Software*, *Execution Platform* and *System* are used to represent the run-time architecture of the system, however a more generalized representation is possible by specifying it as *abstract*.

A component in AADL can be defined by its *type* and *implementation*: the component type declaration defines the interface of the component and its externally observable attributes, whereas the component implementation defines

its internal structure. AADL allows possible component interactions via *ports/features*, *shared data*, *subprograms*, and *parameter connections*. In AADL, the input/output ports can be defined as: *event ports*, *data ports*, and *event-data ports*. Based on the component interactions, explicit *control flows* and *data flows* can be defined across the interfaces of AADL components by specifying the components as *flow source*, *flow path* or *flow sink*. The components can also be associated with various *properties*, like the *period* and *execution time* and the *dispatch protocol*. The *dispatch protocol* specifies if the component trigger is *periodic* or *aperiodic*.

The functional and error behavior of a component are described by the *Behavior Annex (BA)* [17] and the *Error Annex (EA)* [18] respectively, which model behaviors as transition systems. The BA state machine interacts with the component interface and represents the system behavior. Given finite sets of states and state variables, the behavior of a component is defined by a set of state transitions of the form $s \xrightarrow{guard,\ actions} s'$, where s, s' are *states*, *guard* is a boolean condition on the values of state variables or presence of events/data in the component's input ports, and *actions* are performed over the transition and may update state variables, or generate new outputs. Similarly, the EA models the error behavior of a component as transitions between states triggered by error events. It is also possible to represent the different types of errors, recovery paradigms, probability distribution associated with the error states and events, and also specify error flows and propagations within the component, and between various components.

An *abstract* component allow us to defer from the run-time architecture of the system. The need for this generic model stems from the fact that in real-world applications like AAL, it is difficult to assign run-time semantics to components before the design matures. These generic component categories can be parameterized, and can be refined later in the design process through the "extends" capability of AADL. AADL allows us to archive these components and reuse them. For this, we partition them into two public packages in AADL, namely *component library* and *reference architecture* [19]. A *component library* creates a repository of component types and implementations with simple hierarchy. It can be established via two packages: (i) *Interfaces Library* comprising generic components like sensors, actuators and user-interfaces (UI), and (ii) *Controller Library* that includes the control logic. The *Reference architecture* creates a repository of components of complex hierarchy, e.g. the top-level system architecture.

The AADL core language is designed to be extensible and can be extended via *user-defined properties* and *annex sub-languages*. *User-defined properties*

are relatively simpler extensions, when compared to sub-languages, and can be associated with modeling elements as simple values, for instance, integers or strings. However, *sub-languages* allow more complex structures to be added to an AADL model. A sub-language can be standardized and published as an *AADL annex*. Several such annexes have been defined, for example, the *behavior annex* to model the component's behaviour, and the *error annex* for modeling the error behaviour of the system. *Annex sub-languages* are included into AADL specifications as *annex libraries* or *annex sub-clauses*. An annex library is used to define classifiers defined in an anonymous namespace, or in a public or private part of a package. *Annex sub-clauses* are inserted into component types and component implementations and can reference the classifiers declared in the annex library. In AADL, annexes are considered to be separate from the core AADL, i.e., if we remove all the annex libraries, sub-clauses, and annex-related property associations, the resulting model is a valid core AADL model. Moreover, the different annexes are assumed to be independent of each other.

2.2 Multi-Agent Systems

Although we can define agents from different perspectives, in this thesis we define agents as entities that can sense the environment via sensors and act on the environment via actuators. As described in the literature [20] [21], an agent can be characterized by the following properties:

- **Autonomy**: Agents are autonomous entities, that is, they are capable of taking an independent action without human intervention to meet their respective functionalities.
- **Cooperation:** Cooperation is an inherent property of agents, where each agent has only a limited view of the environment and needs to cooperate with the other agents by exchanging information in order to make a decision. In our work, we consider agent cooperation via a dependency relation that each agent has with other agents and we refer to it by the term *agent dependencies*. Agent dependencies can be dynamic or static, i.e, they may or may not change in time. We assume that the agents establish the cooperation (communication) via message-passing.
- **Responsiveness:** By responsiveness, we mean that the agents should be able to perceive the changes in the environment over time and take timely actions accordingly.

- **Learning:** This is the property of some intelligent agents who learn over the course of their interactions with the environment and thereby produce an increased performance over time. Specific learning algorithms like supervised learning, unsupervised learning, or reinforcement learning can be utilized for imbibing learning into the agents. In this paper, we consider rule-based reasoning algorithms (which are simple if-then-else rules) and reinforcement learning [22] techniques.

- **Adaptivity:** Agent systems should be adaptive, i.e., even if an agent fails, the other agents should be able to carry out their respective functionalities. In our architecture, this is employed by deploying multiple redundant agents such that if an agent fails, another one can take over.

- **Mobility:** Some agents (e.g., robots) are capable of moving from one location to another and we indicate this via the mobility property. At the moment, we consider only stationary agents.

Among the different types of agents available, we consider only Reflex agents and Learning agents. The salient features of these two agent types are discussed briefly below.

1. **Reflex agents:** Reflex agents are based on condition-action rules to decide on the action it should enforce on the environment based on sensor data. They are the fastest, but fail completely if the environment is not fully observable.

2. **Learning agents:** These agents have mechanisms to improve their actions over time. They monitor their actions each time via a performance element and at any point of time, suggests an action that will improve the system performance. In our work, we incorporate learning via reinforcement learning.

A network of agents is referred to as a *Multi-Agent System (MAS).* In a MAS, different types of agents interact with each other to achieve a common or a conflicting goal [23]. The interaction between the agents can be direct or indirect. In direct communication, agents can send messages directly to each other and are responsible for their own coordination, however communication cost and implementation complexity (in case of large MAS) are the major disadvantages of direct communication. An alternative approach is to use mediator for communication between the agents (indirect communication). In this thesis, we use

a combination of direct and indirect communication to achieve the agent interactions in our MAS. The mediator (in our case, *a tracker*) is responsible for maintaining agent locations and addresses and thereby establishing the MAS coordination. Once an agent establishes a coordination to another agent via the tracker to achieve a goal (via dependency relation), the further communications are handled directly. In case the tracker fails, the direct communication between the agents are established for achieving the necessary coordination and data consistency.

2.3 Formal Modeling and Verification by Model Checking

Formal Modeling and Verification relies on a set of mathematical techniques that are used to rigorously prove the correctness of a system model expressed in some formal notation. Formal verification techniques are deemed to deliver a higher degree of assurance when compared to other verification techniques such as simulation and testing.

One of the most popular formal verification techniques, *model checking*, an automated technique that checks a finite-state abstract system model in a systematic and exhaustive manner, to prove whether it satisfies a given property modeled in logic. Model checking if fully automated and is performed by a verifier tool called *model checker*. The core of model checking is the verification algorithm, performed by the model checker. The input to the model checker is a system model expressed in a formal notation and a set of formally specified logical properties. For verification of qualitative properties (that admit a yes/no answer) there are two possible outcomes of the model checking procedure. If the model conforms to a given property, the model checker returns a positive answer. For reachability and some liveness properties (e.g., something good will eventually happen) the model checker returns a witness trace in case of fulfillment. Then, the model checking activity can be continued for the rest of system properties. When a safety property is not satisfied, the model checker generates a counter example, which is usually a path (error trace) to the state that violates the property.

Due to its systematic approach and the exhaustiveness of the state space exploration, the model checking procedure can handle models with state spaces up to a certain size, above which there is not enough memory to store new states. This is known as the state space explosion problem.

In this thesis, we apply model checking on architectures of ambient as-

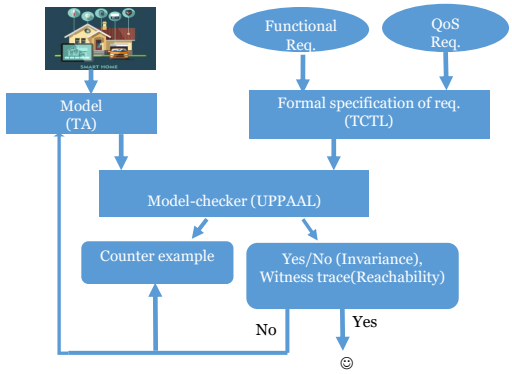


Figure 2.1: Model checking procedure.

sisted living system that we propose, and show our approach on minimal as well as more complex configurations. For minimal architectures, the exhaustive model-checking method scales. The exhaustive model checking, employing the model checker UPPAAL, is shown in Figure 2.1. For complex architectures that integrate multiple AAL functionalities and have several component connections and users, the exhaustive model checking is likely not to scale. Therefore, we resort to a special type of model checking called *statistical model checking* (SMC), which offers the guarantee that a model satisfies a given property up to some probability, based on a finite number of model simulations. A high-level overview of the SMC, usally employed by statistical model checkers like UPPAAL SMC, is given in Figure 2.2. SMC uses a series of simulation-based techniques to answer two types of questions: i) *Qualitative*: is the probability of a given property being satisfied by random system executions greater or equal than some threshold? and ii) *Quantitative:* what is the probability that a random system execution satisfies a given property? The qualitative properties are also referred to as *hypothesis testing*, while the quantitative are called *probability estimation*. In both cases, the answer provided by the procedure will be correct up to a certain level of confidence. Since statistical model checking is less memory intensive than traditional model checking, it can be used to statistically verify models with infinite state spaces. Even though the technique is less precise than the exact model checking, it still solves the verification problem in a rigorous and efficient way.

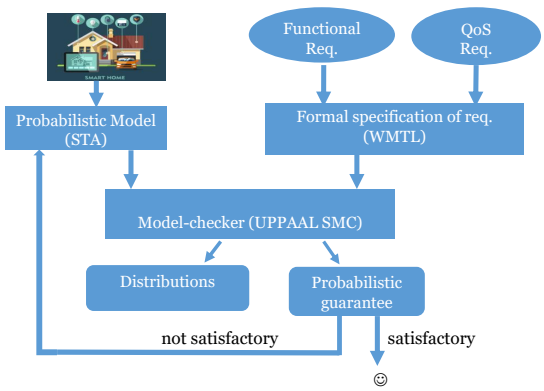


Figure 2.2: Statistical model checking procedure.

In our work, we use the model checkers - UPPAAL [24] and PRISM [25] for exhaustive model-checking and UPPAAL Statistical Model Checker (SMC) [8] for statistical model-checking of complex models. In the following subsections, we briefly overview the timed automata, probabilistic timed automata and stochastic timed automata frameworks, and the temporal logics used for specification of the system properties in the respective model checkers.

2.3.1 Formal Modeling Frameworks

Timed Automata. Timed automata (TA) [26] formalism is an extension of finite-state automata with a set or real-valued variables called *clocks*, suitable for modeling the behavior of real-time systems. The clocks are non-negative variables that grow at a fixed rate with the passage of time, and can be reset to zero.

The semantics of TA is defined as a *timed transition system* $(S, \rightarrow)$, where S is a set of states and $\rightarrow$ is a transition relation that defines how the system evolves from one state to another. A state in the system is a pair (l, v), where l is the location and the v is the valuation of the clocks. A timed automaton can proceed, that is, move to a new state, by performing either a *discrete* or a

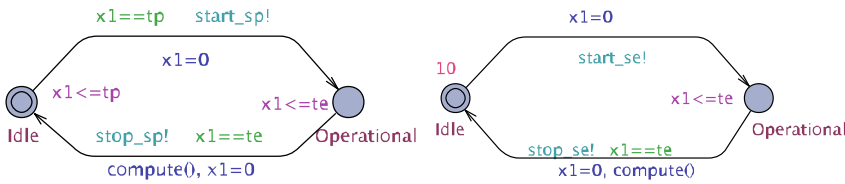
delay transition. By executing a *discrete* transition the automaton transitions from one location into another without any time delay, whereas by executing a *delay* transition the automaton stays in the same location while time passes.

A system can be modeled as a set of communicating components. Let $A_1, A_2, ...A_n$ be a set of timed automata each corresponding to an individual component in the system. A *network* of timed automata (NTA) is simply a parallel composition $A_1 \| A_2 \| \cdots \| A_n$ of a finite number of timed automata.

Next, we present the timed automata variants used by the UPPAAL model checker, UPPAAL SMC model checker, and PRISM model checker by means of examples.

UPPAAL Timed Automata. UPPAAL TA [24] extends TA with discrete variables as well as other modeling features, like urgent and committed locations, synchronization channels, etc. A real-time system can be modeled as a network of TA composed via the parallel composition operator ($\|$), which allows an individual automaton to carry out internal actions, while pairs of automata can perform handshake synchronization. UPPAAL model checker [24] provides exhaustive model checking for UPPAAL TA. The formal definitions of its syntax and semantics can be found in our Paper C [6].

The automaton in Figure 2.3a shows an example of an ordinary UPPAAL TA that models the behavior of a periodic sensor executing some computational routine (`compute()`) that maps inputs into outputs. It has two locations: `Idle` and `Operational`, out of which `Idle` is marked to be the initial one, denoted by two concentric circles. `Idle` is decorated with an invariant `x1` $\leq$ `tp`, denoting that the automaton is allowed to stay in that location as long as the value of the clock variable (`x1`) is smaller or equal to the value of the period (`tp`). The edge from `Idle` to the `Operational` location is decorated with the guard (`x1==tp`). It also has an `update` action, in this particular case being a reset of the clock variable `x1` and a `synchronization action` (`start_sp!`) to synchronize the start of the sensor with the rest of the system. The `Operational` location is decorated with an invariant `x1` $\leq$ `te`, denoting that the automaton is allowed to stay in that location as long as the value of the clock variable is smaller or equal to the value of the execution time (`te`). The `Operational` location represents the operational mode of the automaton and has a transition decorated with a *guard* expression `x1 == te`. On the same edge two update actions are performed, namely executing the computational routine that produces output from the execution (`compute()`), and reset of the clock variable. The computational routine is encoded as a `C` function.



(a) Timed Automata (b) Stochastic Timed Automata

Figure 2.3: Illustrative scenario of UPPAAL TA and UPPAAL SMC TA

UPPAAL SMC Timed Automata. UPPAAL SMC TA [9], referred to as stochastic timed automata (STA) is a formalism defined as the stochastic interpretation of the TA and refines it with: (i) probabilistic choices between multiple enabled transition, where the output *probability* function γ may be defined by the user, and (ii) probability distributions for non-deterministic time delays, where the *delay density function* μ is a uniform distribution for time-bounded delays or an exponential distribution with user-defined rates for cases of unbounded delays.

UPPAAL Statistical Model Checker (UPPAAL SMC) [9] provides statistical model checking for STA. A model in UPPAAL SMC consists of a network of interacting STA (NSTA) that communicate via broadcast channels and shared variables. In the network, the automata repeatedly race against each other, that is, they independently and stochastically decide how much to delay before delivering the output, and what output to broadcast at that moment, with the "winner" being the component that chooses the minimum delay.

Figure 2.3b shows an example of a timed automaton with stochastic semantics, in our case, a sensor that operates aperiodically. The automaton is composed of the same two locations (`Idle` - the initial one, and `Operational`) like the example we had in Figure 2.3a. To model the aperiodic behavior of the component, instead of an invariant, the `Idle` location is decorated with a *rate of exponential*. The distribution parameter λ is the user-defined parameter in the delay function that calculates the probability of the automaton leaving the `Idle` location at each simulation step, given as: $Pr(\text{leaving after } t) = 1 - e^{-\lambda t}$. The greater the value of λ, the smaller is the probability that the automaton stays in the location.

In this thesis, we use NSTA formalism to model our centralized AAL architecture exhibiting random failures.

PRISM Probabilistic Timed Automata. Probabilistic Timed Automata (PTA) as modelled by PRISM model checker [25] supports formalism to model systems that exhibit probabilistic, non-deterministic and real-time characteristics.

Listing 2.1 represents an excerpt of the PTA model of a periodic sensor. The variable `s1` represents the location, `s1=0` represent the initial location `Idle`, and `s1=1` represent the location `Operational` and `s1=2` represent the `Failed` location indicating the sensor failure. `x1` is the `Clock` variable. The `invariant` in location s1=0 represent the periodic activation of the sensor and that in location s1=1 represent the sensor execution time. The `transitions` of the system indicate that the sensor gets periodically activated and it has a probability 0.999 to go to `Operational state (s1=1)` and 0.111 probability to go to `Failed state (s1=2)`

Listing 2.1: PTA Model in PRSIM of a periodic sensor

```
pta
module sensor
  s1: [0..2] init 0; //states 0-Idle,1-Operational,2-Fail
  x1: clock;
  invariant
   (s1=1 => x1<=2) & (s1=0 => x1<=1)
  endinvariant
  [1]s1 =0 & x1=1-> 0.999:(s1'=1)
    & (x1'=0) + 0.001:(s1'=2) &(x1'=0)
endmodule
```

A *system* is defined as a network of modules via parallel composition: $Sys = PTA_1||...||PTA_n$. A global state is the valuation of all variables of all modules. A module can both read and write its own local variables, but only has read access to the local variables of other modules. Synchronized transitions of modules are identified by the commands with the same labels.

In this thesis, we use PTA models to represent our AAL multi-agent system.

2.3.2 Model-checking Tools

UPPAAL

UPPAAL [24] is an integrated development environment for modeling, simulation and verification of real-time systems. It has been developed as a joint research effort by the Uppsala University and Aalborg University. The tool has been first released in 1995 and since has been constantly updated with new features. The properties to be verified by model checking the resulting network of timed automata are specified in a decidable subset of (Timed)

Computation Tree Logic ((T)CTL) [27], and checked by the UPPAAL model checker. UPPAAL supports verification of liveness and safety properties [24]. The ((T)CTL) queries that we verify in this thesis are of the form: i) **Reachability**: $E\Diamond p$ means that there exists a path where p is satisfied by at least one state of the path, and (ii) **Time bounded Leads to**: $p \rightsquigarrow_{\leq t} q$, which means that whenever p holds, q must hold within at most t time units thereafter.

UPPAAL SMC

UPPAAL SMC [9] is an extension of UPPAAL tool that supports the model-checking of timed-automata networks with stochastic semantics. Unlike the exhaustive model-checking performed by UPPAAL, SMC performs statistical model-checking. The SMC algorithms are less memory intensive, and do not suffer from the state space explosion problem. UPPAAL SMC uses an extension of weighted metric temporal logic (WMTL) [28] to provide probability evaluation $Pr(*_{x \leq C} \phi)$, where $*$ stands for $\Diamond(eventually)$ or $\Box(always)$, which calculates the probability that ϕ is satisfied within cost $x \leq C$, but also hypothesis testing and probability comparison.

PRISM

PRISM [25] is a probabilistic model checker that allows model-checking of systems with random or probabilistic behaviour. Although PRSIM supports model checking of various categories of probablistic models like discrete-time Markov chains (DTMCs), continuous-time Markov chains (CTMCs), Markov decision processes (MDPs), probabilistic automata (PAs) and probabilistic timed automata (PTAs), we employ it only for the PTA models discussed in the previous section. In PRISM, a PTA is represented by a *module*. The property specification language of PRISM for PTA is based on Probabilistic Computation Tree Logic (PCTL) [29]. The model checker can verify whether the probability of a path property pp is within a bound b, which is specified as: $P\, b\, [\, pp\,]$. Here, b can be any of $>= p$, $> p$, $<= p$ or $< p$, where p is a double within [0,1]. A path property pp is a formula that evaluates to either true or false for a single path in the model, in which one can apply the following operators: X (next), U (until), F (eventually), G (always), W (weak until), R (release). PRISM can also compute the minimum and maximum probabilities of a path property, in the form of: $Pmin =?\, [\, pp\,]$, and $Pmax =?\, [\, pp\,]$, respectively.

Unlike the statistical model-checking employed in UPPAAL SMC, which yields approximate results, PRISM supports exhaustive model-checking of

probabilistic models and hence can yield concrete results. However, it suffers state space explosion problem while model-checking of complex models. Moreover, PRISM does not have a simulator implemented for PTA models which makes the debugging of complex models extremely difficult. Finally, it should be remarked that in contrast to UPPAAL, PRISM does not have a graphical representation of its models, which are completely defined in a textual way.

Chapter 3

Research Methodology

In this chapter, we present the research methodology that describes the various steps followed to address our research goal. In Computer Science, a research process is described by four iterative steps: (i) formulating the research problem, (ii) proposing the solution, (iii) implementing the solution, and (iv) validation [30]. Solving a research problem is an iterative process, allowing feedbacks between stages.

The overview of the research methodology used in this thesis is described in Figure 3.1. As shown, the main steps are as follows: (i) identifying the research problem, (ii) formulating the overall research goal, (iii) formulating research questions that address the goal, (iv) proposing and implementing solutions to tackle the research questions, and (v) validating the proposed solutions on relevant use cases and users.

As a first step, we identify the research problem. This is done by performing an extensive literature survey [31] to identify the state-of-art (SOA) and the state-of- practice (SOP) in the area of research. The literature survey involves identifying and reading potential publications in the form of journals, conferences, workshops, PhD dissertations, peer reviewed reports related to the domain. In this thesis, we have conducted an extensive literature survey by the so-called *critical analysis of literature* method [32], in the field of AAL. The literature study has also involved analysis of certain user scenarios. With the study, we have identified the potential research problems in the field of AAL, the major ones being the lack of fully integrated systems and rudimentary AI decision support of existing systems, limited user involvement, and lack of formal assurance of the existing systems. This has helped us to formulate

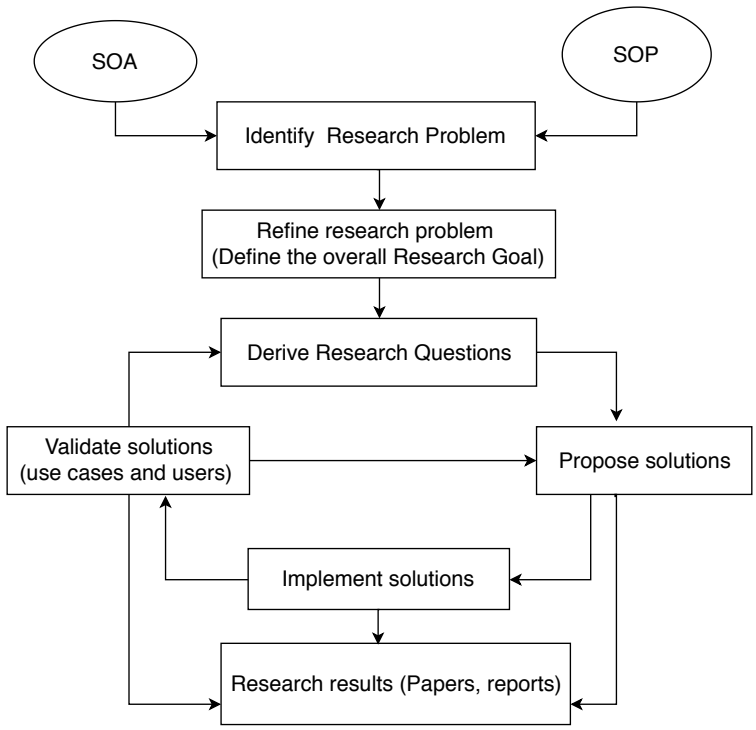


Figure 3.1: Our research process.

our research goal, focusing on the development of integrated AAL solutions with enhanced AI reasoning, sufficient user involvement and formal assurance. Once the research goal has been established, to narrow down the focus, we have identified a set of tailored research questions that need to be answered in order to meet our overall goal. The research questions are presented in Chapter 4.2 . After this step, we have moved to addressing the smaller research questions by developing solutions, presenting the achieved research results and comparing these research results with the research questions. In developing our solutions we have drawn ideas from the related work. Our research results include proposing architecture frameworks for intelligent AAL systems and their formal specification and verification. In this thesis, we use a combination of various research methods, i.e., *proof of concept*, *inductive method*, *model-building* and *simulations of models*, *analysis*, *examples* [32, 33, 34]. For

formulating the AAL architectures, we have used the *proof of concept* research method. While modeling the AAL architectures, we have used *model building* and *simulation of models* approaches.

The last stage of our research process is validation. For formally verifying the properties of AAL architectures, we have used the *analysis method* of validation. However for analyzing various case-studies to evaluate the effectiveness of existing solutions, we have used the *inductive research method*. We have also used *case studies* to validate our proposed architecture with end-users.

Chapter 4

Research Problem

In this chapter, we define the research problem, the overall goal and the research questions of the thesis. In Chapter 4.1 we describe the research problem, after which in Chapter 4.2 we define the overall research goal based on the actual state of practice and state of the art. To narrow the over-arching goal, we define in the same section research questions that help us to structure our research and relate the results to the problem.

4.1 Problem Definition

The world population is rapidly increasing across the world [1]. This demographic trend is followed by new challenges in the society, like increasing number of diseases, increased health-care costs, shortage of caregivers, etc. Besides, it is also witnessed that almost 89% of elderly adults like to live within the comfort of their own homes [2]. Such facts have motivated the research community to focus on the so-called "Ambient Assisted Living" paradigm, which aims to develop intelligent assisting solutions that help the elderly in their safe and independent living, while ensuring that they are not socially isolated.

We have carried out a survey of the state-of-the-art and state-of-practice with respect to AAL solutions, which results in the clear conclusion that there exists a potential research gap in the design and development of a user-tested solution for AAL that integrates various relevant functionalities (e.g. health monitoring, smart home, reminders, fall detection, telepresence etc.) but also

caters for the possible critical situations in a timely manner [3]. Assuming that particular critical events might occur simultaneously, we have analyzed the behaviors of existing individualized or partially-integrated solutions working side by side and concluded that they are not able to tackle particular critical, concurrent events, in real time. One example of a critical scenario is the occurrence of fire and fall events simultaneously, which according to our study cannot be safely resolved by employing independent systems. In such scenarios, a safe resolution can be achieved only when the occurrences of both events are communicated to both caregivers and firefighters, who can then further communicate and prioritize their actions accordingly. If the firefighters are informed only of the fire event and not of the person's fall too, and assuming that answering a telephone call is the way to confirm that the event is veridical, they might deem the fire alarm false and decide not to take action, which can possibly result in loss of life.

Moreover, most of the existing AAL solutions are not necessarily backed by user-acceptance studies, and there is no formal-analysis-based evidence of their functional and timing correctness, which given the connected and distributed nature of most AAL systems is no trivial job. Given the above motivation and challenges, our overall thesis goal is to propose intelligent, integrated and user-centered AAL solutions with ensured functional and extra-functional requirements.

4.2 Research Goals

Based on the above discussed problems, we formulate the overall research goal of the thesis as follows:

Overall Research Goal. *Facilitate the integrated support for achieving self-management of the elderly people by using intelligent ambient assisted living solutions with ensured quality-of-service.*

The overall goal aims to develop integrated solutions for enhancing the support given to elderly adults living independently in their homes and provide a formal assurance to such assisted living frameworks at the design stage of development. Nevertheless, it is obvious that the overall goal is highly abstract and broad. To narrow down the goal and to be able to measure the contributions, we divide it into three research questions in the following.

In order to gain insight into the existing AAL systems and discover their potential challenges, we formulate the first research question as follows:

Research Question 1. *What are the main characteristics, strengths and limitations of existing AAL solutions?*

To answer this research question, we conduct a literature survey of existing AAL solutions and establish a need for an integrated, flexible, and user-centric AAL solution that should include health monitoring, home management, as well as communication and socialization. This leads to our second research question that focuses on how to design such a solution with ensured QoS. This research question is divided into two sub-questions. Research Question 2a focuses on designing intelligent AAL solutions that integrate various health and home management functions. Once the AAL solutions are designed, it is crucial to ensure their QoS and timely response, hence we formulate the Research Question 2b on how to guarantee the function and the considered QoS of the proposed solutions.

Research Question 2. *How can we achieve an intelligent AAL architecture with ensured behavior, timeliness and reliability, which integrates various health and home management functions?*

Research Question 2a. *How do we design architecture frameworks that allow for the integration of multiple functionalities to achieve intelligent decision making in real time?*

Research Question 2b. *How can we employ formal specification and verification technologies to provide certain level of assurance to the integrated solutions with respect to functional correctness and other QoS attributes?*

The outcome of addressing Research Question 2 should be an approach or framework for developing formally assured integrated AAL systems, starting from the architecture design, their specifications and formal verification that can provide guarantees for the critical functional and quality-of-service requirements at early stages of development.

As a final step, we need to validate our proposed solutions on real-life scenarios with representative users. Thus we formulate our Research Question 3 as follows.

Research Question 3. *How do we validate the suitability of the proposed AAL solutions with respect to various AAL scenarios?*

The outcome of addressing Research Question 3 should be the validation or testing of the proposed AAL system in laboratory settings and with real-users and collecting their feedback.

Chapter 5

Thesis Contributions

In this chapter, we give an overview of the research results and contributions that address the research questions defined in Chapter 4.2. The main contributions of the thesis are on four fronts: i) a literature survey of the existing AAL solutions to identify their pros and cons; ii) an integrated architecture design for AAL systems with a centralized decision support, and a formal analysis framework for such systems, based on model checking the architectural specifications and behavior, using UPPAAL and UPPPAL SMC; iii) an agent-based architecture for AAL systems and a method for its formal analysis using PRISM, and iv) a small-scale validation of the proposed architecture of the proposed centralized solution with end users.

5.1 Literature Survey of Existing AAL Solutions

In our first contribution, we address the motivation and background of our study, and thereby tackle RQ 1. The literature study that we undertake compares some AAL solutions against the required functionalities of an AAL system. The functionalities for analysis are chosen based on a multi-national survey with participants from Poland, Denmark and Romania, carried out in the EU project CAMI [1]. The AAL functions that we consider are as follows: (1) health monitoring, (2) fall detection, (3) communication and socialization, (4) support for supervised physical exercises, (5) personalized intelligent and dynamic program management, (6) robotics platform support, (7) intelligent

[1] http://www.camiproject.eu/

Table 5.1 Functionalities supported by various AAL frameworks

AAL Platforms	1	2	3	4	5	6	7	8	9	10
inCASA	✓	✓	✓	✗	✗	✗	✓	✗	✗	✓
iCarer	✗	✗	✓	✗	✓	✗	✓	✗	✗	✓
Persona	✗	✗	✓	✗	✓	✗	✓	✓	✗	✓
Reaction	✓	✗	✓	✗	✓	✗	✓	✗	✗	✗
UniversAAL	✓	✓	✓	✗	✓	✗	✓	✗	✗	✗
iDorm	✗	✗	✗	✗	✓	✗	✗	✗	✗	✓
Robocare	✗	✓	✗	✗	✓	✓	✓	✓	✗	✓
Aware Home	✗	✗	✓	✗	✓	✗	✗	✗	✗	✓
Mav Home	✗	✗	✗	✗	✓	✗	✓	✗	✗	✓
CASAS	✗	✗	✗	✗	✓	✗	✗	✗	✗	✓
GiraffPlus	✓	✓	✓	✗	✓	✓	✓	✗	✗	✓
MobiServ	✗	✓	✓	✓	✓	✓	✗	✓	✗	✓

personal assistant that takes orders, gives advice and reminders, (8) support for vocal interface, (9) mobility assistance, and (10) home and environment management. Table 5.1 shows the functionalities supported by some of the existing AAL frameworks. As shown, none of the existing frameworks support a complete integration of all the functionalities. To establish the need for an integrated architecture for AAL systems, we analyze the timing requirements of integrated and non-integrated AAL solutions in certain critical scenarios, involving simultaneous occurrence of fire and fall events. The analysis is carried out via sequence diagram simulations in Visual Paradigm [35] and by computing their offline schedules. At the end of the analysis, we conclude that there are potential critical scenarios, that can only be tackled by fully integrated AAL solutions.

As a first step targeted towards the design of integrated architectures in AAL, we present a feature diagram representation [36] of functions of an integrated AAL system. We also analyze the existing architecture frameworks that support integration of multiple functionalities [37, 38]. Our analysis considers two types of existing architecture solutions: one with a centralized decision maker [37] and the other with distributed decision making entities, i.e., agents [38]. We employ the Architecture Analysis and Design Language (AADL) to model both architectural variants, and the AADL's analysis capabilities to compare the latencies of the selected frameworks. Our analysis points to the

conclusion that centralized architectures are straightforward solutions to integrate multiple AAL functionalities in real time (due to the lower communication and consistency overheads accounted during the latency analysis, when compared to distributed architectures). Nevertheless, the agent-based solution with distributed decision making bears the advantage of better scalability and adaptivity.

These contributions are addressed in Paper A [3] and Paper B [5], and answer to RQ 1.

5.2 A Centralized Integrated Architecture for Ambient Assisted Living and a Framework for its Formal Assurance

The second contribution of our work is a novel integrated architectural framework for AAL, with a centralized decision support system (DSS), and its formal assurance framework.

Centralized Architecture for integrated AAL Functions. We develop the centralized architecture as a generic model that can be easily instantiated to suit different system requirements. The generic architecture model is inspired by commercial AAL systems with various sensors for home and health monitoring, a data collector, DSS, security and privacy, database (DB) systems, user interfaces (UI), and cloud computing support. The architecture is presented in Fig. 5.1 and supports four different sensor categories: a) Sensor_A: Wearable sensors that send information as data (W_data), e.g., sensors measuring health parameters like pulse, ECG, etc.; b) Sensor_B: Non-wearable sensors measuring ambient parameters and health parameters (NW_data), e.g., camera sensors, motion sensors, etc.; c) Sensor_C: Wearable sensors that detect events (W_event), e.g., fall sensors; d) Sensor_D: Non-wearable sensors detecting events (NW_event), e.g., fire sensors. The data from the sensors are collected by the Data Collector unit. The latter does some data processing and assigns labels and priorities to the incoming data. The Data Collector sends the data to the message queue in the Local Controller, where it gets sorted according to its priority such that when the DSS processes the first element in the queue, it processes the message with the highest priority. Our architecture has both local and cloud-based processing in order to ensure fault tolerance with respect to DSS. The components of the architecture can interact via various commu-

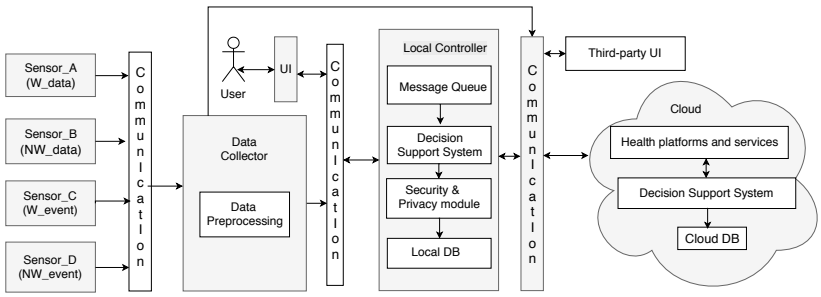


Figure 5.1: A centralized integrated architecture for AAL

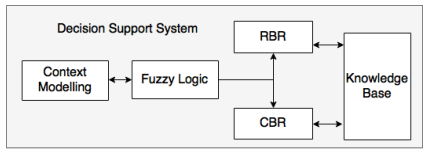


Figure 5.2: An intelligent context-aware Decision Support for AAL Systems

nication protocols. Two distinct features of the architecture, as compared to other AAL architectures, are: (i) the presence of both local and cloud-based processing schemes, and (ii) the continuity of services, even in the absence of Internet (due to the local processing of data).

The core of our system is the **intelligent context-aware DSS**, shown in Fig.5.2. The novelty of our DSS stems from the combination of various AI algorithms, like rule-based reasoning (RBR), fuzzy logic, and case-based reasoning(CBR) with context reasoning for efficient decision-making. The context-reasoning module of the DSS is capable to react to the rapidly changing contexts. We use fuzzy reasoning to identify deviations in health parameters. For instance, the normal pulse range of a person is between 60-100 heart beats per minute. If a person has a pulse measure of 59.5 or 100.5, it should be still considered normal and should not raise any pulse deviation alarms. However, the normal boolean logic, cannot handle the scenario, which can only calssify 59.5 and 120.5 as abnormal pulse values. Hence, we employ fuzzy reasoning with RBR, where a crisp classification can be avoided. With fuzzy reasoning, we can provide degree of memberships, i.e, a pulse value of 59.5 can be considered 97% normal and 3 % abnormal, making the reasoning more effective. The inference engine is a combination of RBR and CBR. RBR is based on

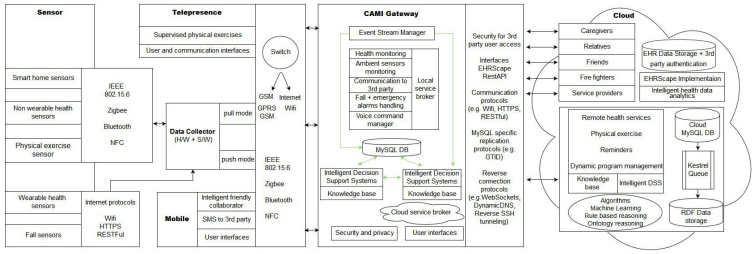


Figure 5.3: The CAMI architecture

simple *if-then-else* rules, which works effectively in cases of well-defined domain knowledge. For instance, " if a fire is detected, then alert the firefighter". However, RBR systems fails completely if there is no specific rule to handle a scenario and in situations where we need to learn and adapt. Such a situation would be, for instance, making a personalized medical recommendation for a user, which may vary from person to person. In this case, CBR reasoning, which can adapt and learn from previously-solved cases can be better suited. In our DSS, we allow RBR to handle the context first, followed by the CBR. The case base of the CBR system is built with successful RBR outputs.

In this thesis, we show three different instantiated configurations of the generic architecture model and their DSS as follows:

- A minimal configuration that uses two sensors - a pulse monitoring sensor wearable and fall sensor wearable, a data collector, a mobile phone UI, and a cloud controller with a DSS that uses limited context modeling (based on available sensor data) and Rule-based Reasoning with fuzzy logic.

- An intermediate configuration that comprises of a physical exercise sensor, a fall sensor, a set of health monitoring sensors measuring pulse and blood pressure, and a set of home sensors that identify the user position and movement. It has a local controller with inbuilt data collection functionality, which forwards the data to the cloud controller. The cloud controller has a DSS with context modeling, fuzzy logic and RBR.

- A complex configuration architecture (Fig.5.3) developed as part of the European Project, CAMI [2]. The CAMI architecture is based on microservices, and has a clean and robust skeleton, onto which several plugin

[2] http://www.aal-europe.eu/projects/cami/

modules can be coupled ensuring modularity and reuse. The major CAMI architecture components are: (i) Sensor unit with multiple sensors, (ii) Data collector unit that collects data from the sensors, (iii) Gateway with intelligent Decision-Support Systems (DSS) for data processing, its local back up, and a DB for storage of data, (iv) Robotic telepresence unit for communication with caregivers, friends, family etc., (v) Mobile phone unit for generation of reminders, and (vi) Cloud Services for redundant data processing (Cloud DSS) and storage (Cloud DB). The CAMI AAL architecture, as a contribution by itself, is presented in Paper B, where we also show its AADL modeling and the initial architecture analysis supported in AADL models developed in OSATE tool, like latency, resource budget and failure.

Modeling the Centralized Architecture in AADL and its Fornal Assurance Framework. For specifying the architecture and its properties, we choose the architecture modeling framework, AADL (Architecture Analysis and Design Language). The generalized AAL architecture along with the DSS is modeled as patterns in AADL (AADL patterns are abstract components). We describe two categories of component patterns in AADL: a) *Atomic Component (AC) Pattern* that do not have hierarchies in terms of sub-components with port interfaces, and b) *Composite Component (CC) Patterns* that has hierarchies with sub-components with port interfaces. We represent the behaviors of the components in the Behavior Annex (BA) of AADL, and the component failures in the Error Annex (EA) of AADL. The EA modeling allows us to efficiently represent the failure and recovery events, which occur via certain probabilistic distributions.

We also show how the AADL patterns can be extended to suit the requirements of the different architecture instantiations mentioned above. For instance, an excerpt of an AADL model of the RBR component (AC) of the DSS (which is a CC), with its interface, BA and EA is shown in Listing 5.1 (The RBR component is common for all the three architecture configurations.) Lines 1-20 define the interface of the RBR component and specify its features, flows, properties and sub-components (in this case, the various data sub-components that do not have port interfaces). The Behaviour Annex (BA) model of the RBR shows a representative scenario of an abnormal high pulse alert generated to the caregiver, and modelled as state transition system in lines 21-28. Lines 29-49 show the error behaviour of the RBR component, modeled as a state transition system in the Error Annex (EA) of AADL. It also shows the failure events causing transient and permanent failures, recovery event (*reset*),

and their distributions.

Listing 5.1: An excerpt from the RBR component in AADL for CAMI

```
1  ——RBR (Component Type +Implementation)———
2  abstract RBR
3  features
4  input: in event data port;
5  output: out event data port;
6  flows
7  Fl : flow path input –> output;
8  properties
9  Dispatch_Protocol => Aperiodic;
10 property_eventgeneration::AperiodicEventGeneration=>1.0;
11 property eventgeneration ::Distribution=> Exponential;
12 property_failure_recovery::FailureRecoveryRate=>1.0;
13 property_failure_recovery::Distribution=> Exponential;
14 Compute_Execution_Time =>1s..1s;
15 end RBR;
16 abstract implementation RBR.impl
17 fuzzy_out_pulse:data  fuzzified_data_pulse;
18 DA: data ADL;
19 u_profile: data user;
20 end RBR.impl
21 —BA—
22 states
23 Waiting: initial complete final state;
24 Operational: state;
25 transitions
26 Waiting –[on dispatch input]–>Operational
27 {if (fuzzyo_pulse=high and DA!= exercising and u_prof =cardiac_patient)
28 {output:= not_caregiver_highpulse}
29 —EA—
30 states
31 Waiting: initial state;
32 Failed_Transient: state;
33 Failed_Permanent:state;
34 LReset: state;
35 Failed_ep:state;
36 events
37 Reset: recover event;
38 TF: error event;
39 PF: error event;
40 Transitions
41 t1: Waiting –[PF]–>Failed_Permanent
42 t2: Waiting –[TF]–>Failed_Transient;
43 t3: Failed_Transient –[Reset]–> {LReset with 0.9,
44 Failed_Permanent with 0.1};
45 t4: LReset–[]–>{Waiting with 0.8, Failed_Permanent with 0.2}
46 properties
47 EMV2::DurationDistribution => [Duration => 1s..2s; applies to Reset;
48 EMV2::OccurrenceDistribution =>[ProbabilityValue => 0.9;
49 Distribution => Fixed;] applies to Reset;
```

To facilitate formal verification, we propose a formal semantics to these AAL specific architecture patterns as a Network of Stochastic Timed Automata (NSTA) that can be model-checked using the UPPAAL model checker or its

statistical extension UPPAAL SMC [6]. An AADL component that we employ in this thesis can be defined as a tuple:

$$AADL_{\text{Comp}} = \langle Comp_{\text{type}}, Comp_{\text{imp}}, EA, BA \rangle, \tag{5.1}$$

where $Comp_{\text{type}}$ represents the component type, $Comp_{\text{imp}}$ represents the component implementation, BA the behavioral annex specification, and EA the error annex. The above definition holds for an AC, but for a CC, we model only the EA showing its composite error behaviour that indicates how the failure of its sub-components affects the CC failure. No separate BA is modeled for CC as its behaviour is already encoded by its AC sub-components. We exemplify the generic definition on the RBR and DSS components in a nutshell. The complete semantics is given in Paper C [6].

$$RBR_{\text{AADL}} = \langle Comp_{\text{type RBR}}, Comp_{\text{imp RBR}}, EA_{\text{RBR}}, BA_{\text{RBR}} \rangle \tag{5.2}$$

On the other hand, the DSS component is a CC defined by its type, implementation and EA, as follows:

$$DSS_{\text{AADL}} = \langle Comp_{\text{type DSS}}, Comp_{\text{imp DSS}}, EA_{\text{DSS}} \rangle \tag{5.3}$$

Formal encoding of AADL components as NSTA. An AADL atomic component (AC), formally defined by the tuple: $AC = \langle Comp_{\text{typeAC}}, Comp_{\text{implAC}}, EA_{\text{AC}}, BA_{\text{AC}} \rangle$ is encoded as an NSTA as follows: $AC \rightsquigarrow NSTA_{AC} = AC_{\text{iSTA}} || AC_{\text{aSTA}}$, where AC_{iSTA} is the so-called "Interface STA" of AC, which corresponds to $Comp_{\text{typeAC}}$ and $Comp_{\text{implAC}}$, whereas AC_{aSTA} is the "Behavioral STA" that encodes the EA and BA of an AC.

Similarly, an AADL Composite Component (CC), formally defined by the tuple: $CC = \langle Comp_{\text{typeCC}}, Comp_{\text{implCC}}, EA_{\text{CC}} \rangle$ is also a network of two synchronized STA, $CC_{\text{NSTA}} = CC_{\text{iSTA}} || CC_{\text{aSTA}}$, where CC_{iSTA} is the "interface" STA of the CC component, and CC_{aSTA} is the "annex" STA that encodes the information from the error annex in AADL. A nutshell of the AADL encoding as NSTA model is presented in Table 5.2, with the detailed description of the encoding rules to be found in Paper C [6].

We take the example of the RBR component and present its semantic encoding as an NSTA model. Let us assume an RBR component defined by Equation 5.2. We define the formal encoding of RBR as the following network of synchronized STA: $RBR_{\text{NSTA}} = RBR_{\text{iSTA}} || RBR_{\text{aSTA}}$, where RBR_{iSTA} is the "interface" STA of the RBR component and RBR_{aSTA} is the "annex" STA that encodes both the behavior and the error annex information.

Table 5.2 Encoding of AADL Component as STA

AADL comp	**STA**
$Comp_{\text{type}}, Comp_{\text{imp}}$	STA
EA, BA	STA
T_{p}	$Invariant + Guard$
T_{e}	$Invariant + Guard$
Ports	$Variables$ + $Synchronization$
$Data$	$Variable$
A sub-component	STA
EA + BA states	*Locations*
EA + BA transitions	$Edges$
Error events	$Variables$
Distribution of error events	γ
Distribution of aperiodic events	μ

Figure 5.4 depicts the NSTA of the RBR based on the encoding rules.

In Paper C [6], we also show the feasibility of using exhaustive model checking with UPPAAL and simulation-based model checking with UPPAAL SMC. For small models, like that of minimal configuration model, exhaustive verification scales. However, for complex models like CAMI, the only feasible option is to use simulation-based model-checking, that is, statistical model checking that returns a probabilistic guarantee of fulfilling a particular requirement.

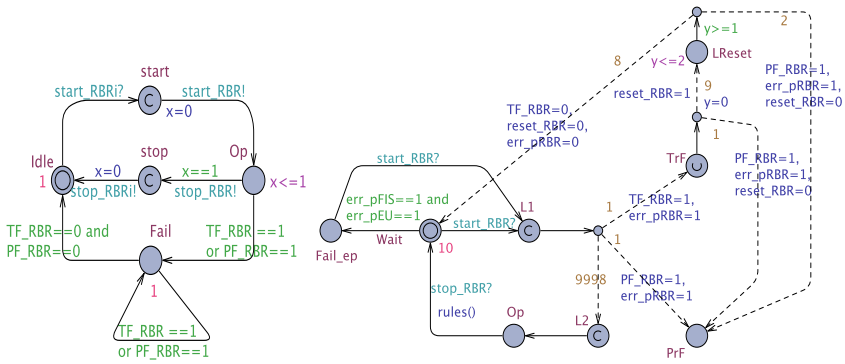
For the CAMI architecture, which is the most complex configuration obtained by instantiating our generic architecture of Fig 5.3, we show the verification of a set of functional and QoS requirements presented.

R1: If the fire sensor detects a fire, then the DSS sends a notification to the firefighters, within 20 s.

R2: If a fall is detected by the wearable or the camera sensor, then the DSS sends a notification to the caregiver, within 20 s.

R3: If there is a pulse data deviation indicating high pulse, the DA is "not exercising", and the user has a disease history of a cardiac patient, then the DSS sends a notification to the caregiver, within 20 s.

R4: If fire and fall are detected simultaneously by the respective sensors, then the DSS should detect the presence of the simultaneous events and send notifications to both the firefighters and the caregiver indicating the presence of both



(a) RBR_{iSTA} (b) RBR_{aSTA}

Figure 5.4: The NSTA for the RBR

events, within 20 s.

R5: The decisions taken by the local DSS are updated in the cloud DSS such that they are eventually synchronized. This requirement relates to the data-consistency requirement of CAMI.

R6: If the local DSS fails, then the cloud DSS eventually becomes active. It corresponds to the fault-tolerance aspect of the CAMI system.

The verification results, by employing UPPAAL SMC, are tabulated in Table 5.3. To check that our CAMI DSS meets its requirements, we employ monitor STA that monitor the sensor values, the respective DSS output, and the corresponding clock. The verification results show that the system satisfies all the functional requirements (R1 to R4) with high probabilities (close to 1) and with high confidence. Requirements R5 and R6 are related the QoS attributes of the CAMI architecture. R5 checks the data consistency of Local DSS and Cloud DSS and requires that the RBR outputs of the local DSS get stored in the case-base of the cloud DSS. This requirement is satisfied with a high probability of [0.99975, 1] and high confidence of 0.998. Query R6 models the fault-tolerance requirement of CAMI. We see from Table 5.3 that the probability of cloud DSS to become activated (R4) is [0.01, 0.04]; this is because it gets activated only when the local DSS has failed and the failure probability of local DSS is between [0.01, 0.04] for a simulation over 1000 time units. However, if the local DSS has failed, we see that the probability of cloud DSS getting activated is very high [0.99975, 1] with a confidence of

0.998, which satisfies our requirement. Most of the requirements are verified with queries that contain terms of the form $A\ imply\ B$, therefore a pre-check of each corresponding “A" being reachable is first carried out.

This contribution is a first attempt to analyze formally an integrated AAL design, including its AI support (context modelling with fuzzy logic and RBR). The described contributions are addressed in Paper B [5] and Paper C [6] and answer Research Questions 2a and 2b.

Table 5.3 SMC analysis results for CAMI Architecture

Req.	Query	Result	Runs
R1	$Pr[<= 1000]([](M\_fire.fire\_alarm == 1)\ imply\ (se\_nw.fire == 1$ and $M\_fire.s1 <= 20)))$	Pr [0.99975,1] confidence 0.998	3868
	$Pr[<= 1000](<>\ (M\_fire.fire\_alarm == 1))$	Pr [0.99975,1] confidence 0.998	4901
R2	$Pr[<= 1000]([](M\_fall.fall\_not == 7)\ imply\ ((se\_w.fall == 1$ or $sd\_nw.data\_val == 1)\ and (M\_fall.s1 <= 20))))$	Pr [0.99975,1] confidence 0.998	3868
	$Pr[<= 1000](<>\ (M\_fall.fall\_not == 7))$	Pr [0.99975,1] confidence 0.998	4901
R3	$Pr[<= 1000]([](M\_pulse.pulse\_not == 3)\ imply\ (110 <= sd\_w.data\_val <= 300\ and\ M\_pulse.FIS\_out == 3\ and\ ADL == 1\ and\ upro.disease\_history == 3$ and $M\_pulse.s1 <= 20))$	Pr [0.99975,1] confidence 0.998	3868
	$Pr[<= 1000](<>\ (M\_pulse.pulse\_not == 3))$	Pr [0.99975,1] confidence 0.998	3868
R4	$Pr[<= 1000]([](\ M\_firefall.fire\_not == 2\ and\ M\_firefall.fall\_not == 2\ imply\ ((se\_w.fall == 1$ or $sd\_nw.data\_val == 1)\ and\ se\_nw.fire == 1$ and $M\_firefall.s1 <= 20))$	Pr [0.99975,1] confidence 0.998	3868
	$Pr[<= 1000](<>\ (Pr[<= 100](<>\ (M\_firefall.fall\_not == 2$ and $M\_firefall.fire\_not == 2))$	Pr [0.99975,1] confidence 0.998	7905
R5	$Pr[<= 1000]([](M\_consistency.stop\ imply\ (RBR\_om == iCBRCC_m)))$	Pr [0.99975,1] confidence 0.998	3868
	$Pr[<= 1000](<>\ (M\_consistency.stop))$	Pr [0.99975,1] confidence 0.998	5777
R6	$Pr[<= 1000]([](INT\_CC.DSSCC\ imply\ PF\_DSS == 1))$	Pr [0.99975,1] confidence 0.998	3868
	$Pr[<= 1000](<>\ (INT\_CC.DSSCC))$	Pr [0.01,0.04] confidence 0.998	2885

5.3 A Multi-agent-based Integrated Architecture for Ambient Assisted Living and its Modeling and Analysis Framework

Driven by the fact that the scalability, adaptability and service accessibility to multiple users are also important concerns that need equal attention as integration and real-timeliness in AAL systems, as the third thesis contribution, we propose an alternative to the previous centralized solution, as a distributed agent-based architecture for AAL systems, which is inspired from the existing architectures in literature. We also propose a modeling and analysis framework for such solutions. The modeling framework uses the core AADL language that we extend with a sub-language called "Agent Annex". The Agent Annex sub-language is formulated by extending the core AADL language class diagrams. The semantics of Agent Annex is defined as Stochastic Transition Systems [14], which can effectively capture the non-deterministic, probabilistic and real-time behaviours of AAL systems. In order to formally verify the system for its requirements, we use the PRISM model checker [25], where the system architecture is modeled as a parallel composition of probabilistic timed automata (PTA) modules. These contributions are be presented in Paper D [12].

The multi-agent AAL architecture that we propose in this thesis (Fig. 5.5) has some important advantages if compared to our centralized AAL solution, in terms of fault-tolerance, scalability, adaptability and accessibility for multiple users. Due to these obvious reasons (although difficult to develop and maintain), distributed architectures are preferred in the AAL community compared to their the centralized counter-parts. However, in our first contribution in which we analyze the existing the agent-based architectures [5], we conclude that the distributed agent-based architectures suffer from an increased overhead for the collective decision making while handling multiple functionalities. Therefore, in our solution, we attempt to reduce such overhead by letting the agents cooperate to ensure intelligent decision making. In our solution, each agent has a particular function, that is, the fire agent deals with detecting fire and raising a notification to the firefighter, the pulse agent detects the pulse deviations and alerts the caregiver, etc. To be able to cooperate efficiently, each agent is equipped with a list of possible dependencies that it can have with other agents. The dependencies can vary dynamically. Each agent in the system is represented with two dependency levels, level 1 and 2, respectively. Dependency Level 1 denotes the list of agents that a particular agent has to cooperate

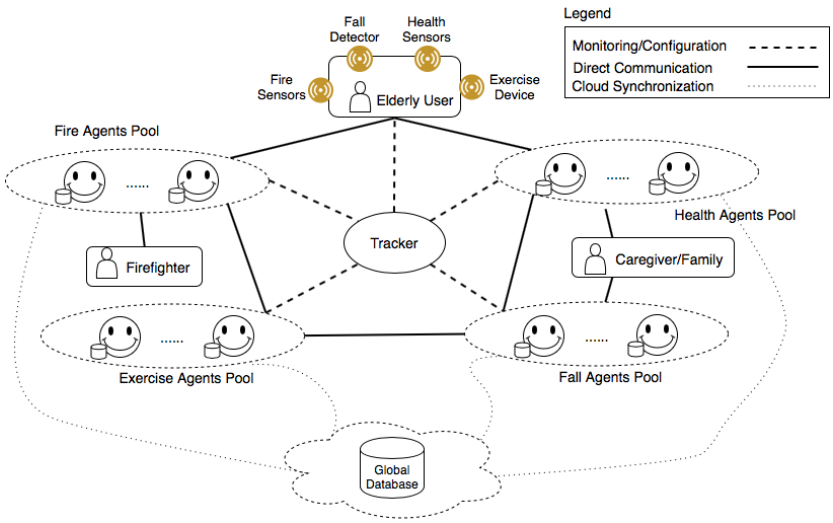


Figure 5.5: A Multi-agent system architecture for AAL systems

with to make an intelligent decision. For instance, in order for the fire agent to notify the user of a fire in the kitchen, it has to cooperate with location agent to identify where the user is. Dependency Level 2 shows the external dependencies that an agent can have in terms of simultaneous occurrence of two events, which does not affect its initial autonomous action directly. For instance, a fire can occur along with a fall. The agents can later synchronize with second-level dependent agents and later compensate for the action taken. In this case, a fire agent compensates its initial action by informing the firefighters that the user has fallen, besides sending the fire notification, such that the firefighters take an immediate action without waiting for a fire event confirmation from the user.

Our multi-agent system (MAS) enables interaction with multiple agents belonging to different categories, ranging from simple reflex agents to complex intelligent agents. In our architecture, we consider only two agent categories: a) *reflex agents* with simple *if-then-else* rules, and b) *intelligent agents* with *reinforcement learning (RL)* [13]. The AAL system described as a MAS contains exercise agents, fall agents, health agents and fire agents (Fig. 5.5). The exercise agent is modelled as an intelligent agent with RL and all other agents are modelled as reflex agents.

Listing 5.2: An excerpt of the fire agent modeling in AADL (interface)

```
1  abstract Fire_Agent1
2  features
3   BA1: requires bus access ACP;
4   BA2: requires bus access SA_comm1;
5  properties
6   Dispatch_Protocol => Periodic;
7   Period => 1 ms;
8   Compute_Execution_time => 2ms..2ms;
9  end agent1;
10 bus ACP ... end ACP;
11 system agent_system ... end agent_system;
12 system implementation agent_system.impl
13 subcomponents
14  A1: abstract Fire_Agent1;
15  Agent_Comm_Protocol: bus ACP;
16 connections
17  BAsys1: bus access Agent_Comm_Protocol <->A1.BA1;
18 end agent_system.impl;
```

We model the proposed architecture in AADL, which we extend with an *Agent Annex* sub-language for specifying agent properties. For simplicity, we show the modeling and analysis on an abstracted version of our MAS architecture consisting of a pulse agent, a fire agent, a fall agent, and an exercise agent each with a redundant copy. These agents cooperate via message-passing. Each agent can accept 2 connections simultaneously from users. We show the case where this system is used by 2 users simultaneously- Jim and Mary.

In the following, we briefly describe our AADL core model and illustrate its Agent Annex. We consider the example of a fire agent as a representative of reflex agents, whereas in Paper D [12], we look at an exercise agent representing the intelligent agents category. The fire agent has the functionality of notifying the firefighter in case of a fire event (with its behavior encoded as if-then-else rules). Listing 5.2 shows an excerpt of the AADL model of a fire agent in our MAS, and a bus component; the Agent Communication Protocol (ACP) models the communication protocol between multiple agents. We assume that the communication protocols defined here work via shared variables. The communication between the agents via tracker is assumed instantaneous, however direct communication between the agents always encounters a delay. The Agent component is modeled as an abstract component in AADL (Lines 1-9), which can be later refined towards a particular hardware or software, based on the application. We also show a system-level representation (Lines 12-18) with its sub-components and their connections defining the communication.

For extending the core AADL with Agent Annex, we first extend the core AADL meta-model [39], specified as UML2 class diagrams with classes for specifying agent behaviours. The meta-model extension is presented in our

Paper D [12]. The *Agent Model Annex* is formulated by extending the AADL abstract classes, *Annex Library* and *Annex Subclause*. The *Annex Library* is used to declare *classifiers* of our Agent Annex in packages. The Annex Library concepts are attached to an AADL model by *using Annex subclause* within a component type or component implementation declaration.

Listing 5.3 shows the syntax of the Annex sub-clause of fire agent attached to its component implementation. It defines a probabilistic transition system with 3 states - $Idle$, $Operational$, and $Fail$, and a clock variable x, and boolean variables $fire$ and the $alarm$. $Idle$ represents the initial state. It also defines a probabilistic transition from state $Idle$. The transition is enabled periodically based on the component's period and it has a probability of 0.999 to reach the state $Operational$, and 0.001 probability to reach the state $Fail$. If the agent reaches the $Operational$ state, it raises a fire alarm in response to the fire event, and takes a transition back to $Idle$, once the component has consumed its execution time (defined by $exec\_time$).

Listing 5.3: An example of Agent Model Annex Subclause attached to Fire Agent

```
1  system implementation fire.agent
2    subcomponents
3    fire_sensor: device f_sensor;
4    annex Agent_Model {**
5      states
6      Idle, Operational, Fail;
7      Idle: initial state;
8      transitions
9      [] state=Idle & x=Period ->0.999:(state '=Operational & x'=0)
10     + 0.001:(state '=Fail);
11     [] state=Idle & x=Period & fire=1 ->0.999:(state '=Operational
12     & x'=0 & fire_alarm '=1)+ 0.001:(state '=Fail &fire_alarm '=0);
13     [] state=Operational & x=exec_time ->state '=Idle & fire_alarm '=0;
14     variables
15     clock x;
16     bool fire;
17     bool fire_alarm;
18   **};
19 end fire.agent;
```

Formal Encoding and Analysis of MAS. An AADL component is defined by the following tuple:

$$AADL_{\text{Comp}} = \langle Comp_{\text{type}}, Comp_{\text{imp}}, AA \rangle, \tag{5.4}$$

where $Comp_{\text{type}}$ is the component type, $Comp_{\text{imp}}$ represents the component implementation, and AA, the agent annex specification [3]. The AA is formally encoded as a tuple:

$$AA = \langle Var, Init, Tt \rangle, \tag{5.5}$$

where: Var represents the set of variables defined in AA, inclusive of state labels, and others representing the sensor data and events, as well as clock variables for specifying the real-time behaviour; $Init$ is the assertion over Var denoting the set of initial states; Tt is the set of state transitions, defined accordingly to follow PRISM's input language syntax defined in Chapter 2.

We also provide a formal encoding of our multi-agent system, including its Agent Annex, in terms of a Stochastic Transition System (STS) [14] as follows:

The AADL component defined by Equation 5.4 is formally encoded as an STS. The MAS architecture is represented as a parallel composition of all the STS modules: $MAS = ||_{i=0}^{n} STS_{modulesi}$, *where* n *is the number of AADL components of the system, excluding data components and bus components, if defined in the system (as variables in the AADL component using them).*

For instance, the fire agent is formally encoded as an STS module, where:

- $V : \{(s1 = 0, fire = 0, fire\_alarm = 0, x = 0), (s1 = 0, fire = 0, fire\_alarm = 0, x = 1), (s1 = 1, fire = 1, fire\_alarm = 1, x = 0), (s1 = 1, fire = 1, fire\_alarm = 1, x = 1), (s1 = 1, fire = 1, fire\_alarm = 1, x = 2), (s1 = 2, fire = 1, fire\_alarm = 0, x = 0)\}$
- $\theta : s \models (s1 = 0 \wedge fire = 0 \wedge fire\_alarm = 0 \wedge x = 0)$
- T is defined by the set of transitions as follows:
  $T : \{\tau 1 : \{(s1 = 0 \wedge fire = 0 \wedge fire\_alarm = 0 \wedge x = period) \longrightarrow (s1' = 0 \wedge fire' = 0 \wedge fire\_alarm' = 0 \wedge x' = 0), P = 1\},$
  $\tau 2 : \{(s1 = 0 \wedge fire = 1 \wedge fire\_alarm = 0 \wedge x = period) \longrightarrow (s1' = 1 \wedge fire' = 1 \wedge fire\_alarm' = 1 \wedge x' = 0), P = 0.999 \cup (s1' = 2 \wedge fire' = 1 \wedge fire\_alarm' = 0 \wedge x' = 0), P = 0.001\},$
  $\tau 3 : \{(s1 = 1 \wedge fire = 1 \wedge fire\_alarm = 1 \wedge x = 2) \longrightarrow (s1' = 0 \wedge fire' = 0 \wedge fire\_alarm' = 0 \wedge x' = 0), P = 1\}\}$

Similarly, all other AADL components are encoded as STS modules, respectively.

[3] Although Agent Annex is specifically tailored to represent agent behaviours, it can also specify the behaviours of other components, like the standard Behaviour Annex.

Listing 5.4: PRISM Model of a Fire Agent

```
1  pta
2  module Fire_agent1
3    s1: [0..2] init 0; //states 0-Idle,1-Operational,2-Fail
4    fire: [0..1] init 0; fire_alert0: [0..1] init 0;
5    x1: clock;
6    invariant
7     (s1=1 => x1<=2) & (s1=0 => x1<=1)
8    endinvariant
9    [1] s1=0 & fire=0 &x1=1-> (s1'=0) &(x1'=0) &(fire_alert0'=0);
10   [2]s1 =0 & fire=1   & x1=1-> 0.999:(s1'=1) & (fire_alert0'=1)
11     & (x1'=0) + 0.001:(s1'=2) &(x1'=0) & (fire_alert0'=0);
12   [3]s1=1  & x1=2  -> (s1'=0) & (x1'=0) &(fire_alert0'=0);
13 endmodule
```

The next step is the formal analysis of the architecture using PRISM model-checker. The STS modules are encoded as PTA modules in PRISM. Listing 5.4 shows the encoding of the fire agent in PRISM. The encoding of STS as PTA modules used by PRISM is straightforward, hence we change only the syntax to match the PRISM input language. The Fire Agent PTA module uses a variable $s1$ to represent the state: $s1 = 0$ (Idle), $s1 = 1$ (Operational), $s1 = 2$ (Fail). There are variables to represent the fire event (*fire: [0..1]*), and raised fall alert (*fire_alert0[0..1]*), where *0* indicates that the event has not occurred, whereas *1* indicates the opposite. Variable $x1$ is a clock variable. The invariants associated with the states (Lines 6-8) depend on the component properties like dispatch protocol and execution time (that are defined at the interface of the AADL component's model). If the component is periodic, the state $Idle$ has the invariant $x1 \leq Period$ associated with it. Similarly, the invariant of state $Operational$ is $x1 \leq Exec\_time$. The transitions (Lines denoted by labels 1-3) follow the transitions definition of the Agent Annex specification of the Fire Agent.

After specifying the architecture as a parallel composition of the PTA modules [12], we verify the the following system requirements with PRISM.

- R1: If a fall occurs due to low pulse, then raise an alert to caregiver indicating *fall due to low pulse* within 20 s.
- R2: If a fire and a fall event occur simultaneously, then raise an alert to both caregiver and firefighter indicating the issue, within 20 s.
- R3: The exercise session is scheduled only if the health agent indicates a normal pulse.

Table 5.4 Verification results

Req.	Query	Result
R1	$filter(forall, fall\_user1 = 1 \& pulse\_user1 \leq 50 \ \& tracker\_fail = 0 \rightarrow P \geq 0.999\ [F((pulse\_alert0\_u1 = 3 \ \| pulse\_alert1\_u1 = 3 \| \ pulse\_alert2\_u1 = 3 \| pulse\_alert3\_u1 = 3)\ \&\ (y \leq 10) \& (fall\_fail = 0)\ \& (pulse\_fail = 0))]$	satisfied
R2	$filter(forall, fall\_user2 = 1 \& fire\_user2 = 1 \& tracker\_fail = 0 \rightarrow P \geq 0.999\ [F((firefall\_alert0\_u2 = 2 \ \| firefall\_alert1\_u2 = 2 \| \ firefall\_alert2\_u2 = 2 \| firefall\_alert3\_u2 = 2) \& (y \leq 10)\ \& (fall\_fail = 0) \ \& (fire\_fail = 0))]$	satisfied
R3	$filter(forall, cal\_notexc\_user1 = 1 \& tracker\_fail = 0 \ \& (pulse\_user1 \geq 60\ \& pulse\_user1 \leq 120) \rightarrow P \geq 0.999\ [F(exc\_sch\_u1 = 1))]$	satisfied
R4	$filter(forall, exc\_sch\_u1 = 1 \& u1\_disease\_history = 1 \ \& u1\_pref = 2 \rightarrow P \geq 0.999\ [F(exc\_u1\_int1 = 2)]$	satisfied
R5	$filter(forall, exc\_sch\_u1 = 1 \& interval = 1\ \& y \leq 5 \ \& pulse\_user1 \geq 200 \rightarrow P \geq 0.999\ [F(exc\_u1\_int2 = 1)]$	satisfied
R6	$filter(forall, fall\_user2 = 1 \& fire\_user2 = 1 \ \& tracker\_fail = 1 \rightarrow P \geq 0.999\ [F((fall\_alert0\_u2 = 2 \ \| fall\_alert1\_u2 = 2 \| \ fall\_alert2\_u2 = 2 \| fall\_alert3\_u2 = 2) \ \& (y \leq 20)\ \&\ (fall\_fail = 0) \& (fire\_fail = 0))]$	satisfied
R7	$filter(forall, fall\_user2 = 1 \& tracker\_fail = 0 \& \ fail1\_fall = 1\ \& fall2\_fall = 0 \rightarrow P \geq 0.999\ [F((fall\_alert2\_u2 = 1 \| fall\_alert3\_u2 = 1)\ \& y \leq 20)]$	satisfied

- R4: The initially suggested exercise is based on user preferences and health condition.
- R5: If any health abnormality is detected in the first sub-session of the exercise, a different set of exercises of lower intensity is prescribed. It should be noted that R1-R5 are safety-critical requirements.

In addition, the system has quality-of-service (QoS) requirements as follows:

- R6: If the tracker fails, the system continues its functionality.
- R7: If one of the agent fails, its function is carried out by the back-up.

The verification results are tabulated in Table 5.4. The requirements are formulated as PCTL queries and the model-checking method is *Digital Clocks*.

Since PRISM, by default, returns the value for the (single) initial state of the model while model checking, we employ *filters* to verify our properties over all states. Requirement $R1$ ensures that if a fall event occurs due to a low pulse for $user1$ (Jim), and the tracker is operational, then the tracker initiates the communication between the respective fall and pulse agents associated with user Jim (the request can be assigned to any of the agent sockets depending on availability), and the probability that one of them sends an alert to caregiver indicating that there is "fall due to low pulse" is greater than 0.999 provided that at least one of the sockets of each agent is functional. Assuming that the communication via tracker takes less time, the requirement is satisfied within 10 time units. Similarly, for $R2$, we verify for $user2$ (Mary) that in case of fire and fall events occurring simultaneously, an alert indicating both events is raised and sent within 10 time units, provided that the tracker has not failed. In case of $R3$, $R4$ and $R5$, we verify the functionality of the exercise agent serving Jim. By $R3$, we establish that the exercise session is scheduled only if the corresponding health agent indicates that the user's pulse level is normal. $R4$ indicates that the initial exercise category is chosen based on user preferences and health condition. By verifying $R5$, we show that if a high pulse deviation occurs during the exercise sub-session, a low intensity exercise is chosen in the next sub-session, irrespective of user preferences. In $R6$, we illustrate a similar function as in $R2$, but assuming that the tracker has failed. In this case, the functionality is met by direct communication between the agents, which takes more time than the communication via tracker (it is shown that this requirement is satisfied within 20 time units). Next, in $R7$, we assume a fall event of $user2$, and one failed fall agent; then, a fall alert is raised and sent to the caregiver by either one of the redundant fall agents. PRISM shows that this requirement is satisfied within 20 time units.

These contributions are contained in Paper D and address Research Questions 2a and 2b.

5.4 Validation with End Users

As a final contribution, we present some initial validation that we have performed on the CAMI architecture (contribution 2), thereby addressing the Research Question 3 formulated in Chapter 4. The research contribution is presented in Paper E. In the analysis presented in the paper, we show the response of 105 senior citizens (55-75 years old) from Romania, Poland and Denmark regarding the CAMI functionalities. With the responses, we give priorities to

health monitoring, fall detection, supervised physical exercises and vocal interaction. We formulate a smaller implemented version of the initial CAMI architecture, presented in Paper B [5]. We also show an initial analysis results of these implemented functionalities by carrying out a set of tests in laboratories by involving different users.

Health-monitoring functionalities: In CAMI, we offer a set of health monitoring functionalities that allow us to monitor blood pressure, heart rate, blood glucose, etc. In addition, CAMI also employs fall detection sensors to identify falls of the elderly and raise timely alerts. Among the respondents, 59% consider the graphic display of various health measurements (e.g. blood pressure, heart rate, oxygen levels) as an interesting feature. The ability to share health measurements with various doctors is considered useful by 60% of the respondents. CAMI's fall detection functionality employing Vibby wearable is tested in laboratory by employing multiple users.

Physical exercise monitoring: CAMI system implements physical exercise monitoring using two avatars: the training avatar and the avatar of the user. The training avatar performs different physical exercises that the user must reproduce. We also test the suitability of exercise avatars in laboratory with multiple users.

Vocal interaction: CAMI's vocal interaction module comprises of modules for automatic speech recognition , natural language understanding, dialog management, natural language generation, and text to speech synthesis. In order to test our vocal interaction functionality, we devise a variety of text inputs for the user to interact with the CAMI system and test the system responsiveness.

A plan for extensive field trails in user homes has been devised and the questionnaires has been set up, and this will be accounted in our future work.

Chapter 6

Related Work

In recent years, there has been a lot of research in the field of AAL, due to to the need for supporting an increased elderly population [4]. In this section, we describe some of the prominent AAL solutions with respect to their software architecture models and and compare the formal approaches where they exist with our solutions.

6.1 Software Architecture Models for AAL

A literature study on existing AAL architectures shows that there are certain architecture types that address the construction of integrative AAL applications (i.e., those that focus on creating a holistic user experience, not just the development of a specific functionality such as health data management or social interaction) [40]. We have classified them into the following architecture types: Multi-Agent Systems (MAS), Cloud-based systems, and Internet-of-Things (IoT) centric.

Agent-based architectures: These are the most commonly used architectures for AAL applications due to their flexibility, autonomy, adaptability, better response and service continuity due to the distributed nature [41, 42]. The agents are autonomous processing entities and can be local and/or cloud based. Some examples of healthcare frameworks that rely on a distributed agent architecture are proposed by Pez et al.[43], Sernani et. al [44], and Tapia et. al[38]. However, the agent based architectures also have some drawbacks, for instance, restricted communication protocols for agent communication and the delay

overhead in taking a collective decision [41].

Cloud-based AAL solutions: There are many AAL solutions that leverage the potential of cloud computing for context modeling [45, 46, 47] intelligent decision making, and use it as a data store [37].

Although cloud-based solutions are scalable, cost-effective, reusable, adaptable, and extendable, the sole processing with cloud cannot guarantee strict hard real-time properties, and the system fails completely in the absence of Internet.

IoT architectures: IoT technology is now getting widely utilized in the field of AAL owing to its technological advancements. The IoT concept of communication (i) between smart objects, (ii) smart objects and people, and (iii) among people themselves, are widely exploited in the field of AAL, thereby providing connectivity, context-awareness and adaptivity [48]. There are also approaches to integrate the autonomous behavior of agent-based systems with IoT technology [49, 50, 51].

Although AAL systems based on IoT offer high flexibility, adaptability, the system depends only on the availability of the Internet for operation, which can lead to a complete failure of such systems in places where Internet connectivity is meager.

In our work, we propose two architecture solutions using the concepts from these existing architecture solutions:

- A centralized solution which uses the cloud for data processing and storage (Paper B [5], Paper C [6]).

  Our architecture is designed with local processing along with cloud processing. In order to overcome the lack of real timeliness property of cloud, we allow hard real-time functionalities to be handled by the local processor. We also overcome the sole dependency of Internet for the operation of our system by a switch which can select using either GSM and Internet communication options.

- A distributed architecture solution with agents and cloud support [Paper D [12]].

  In this architecture, we use cloud as a data store, and for the long-term processing of the stored data. Our architecture eliminates the disadvantages of existing agent systems by providing an efficient way to deal with agent cooperation for intelligent decision making, in real-time.

The backbone of any AAL solution lies in its ambient intelligence utilized for context awareness and intelligent decision making [52]. Many studies have

progressed in this aspect; some of them utilize AI solutions like case-based reasoning [53, 52], fuzzy-logic-based reasoning [54, 55] etc. Based on our preliminary studies [3, 5], we conclude that there is room for improving existing AAL solutions in terms of flexibility and continuity of use, range of provided services, as well as incorporating user preferences into the design, for higher acceptance. In addition, by incorporating multiple AI intelligence algorithms to tackle different scenarios, one can effectively improve the intelligence offered in such solutions. In this thesis, we also propose a DSS architecture combining multiple AI techniques to support enhanced reasoning (Paper C [6]) .

6.1.1 Formal Modeling and Analysis of AAL Systems

Since AAL systems are complex safety-critical systems, which are dynamic and subjected to an unpredictable environment, it is essential that their behavior is analyzed by using formal techniques, to provide assurance that they meet their requirements.

The use of architecture description languages to specify AAL systems has not been exercised previously, yet this is a common approach in automotive or automation systems. In order to specify agent-based systems, there are other approaches for their modeling, especially based on logic-based formalisms, although not so commonly used in the AAL domain. Some of the most popular ones are AUML [56], extended DESCARTES [57], GAIA [58], SLABS [59], CASL[60], DESIRE [61], dMARS [62], agent-based G-net model [63], concurrent METATEM [64] etc. AUML (Agent Unified Modeling Language) [56] is one of the most widely employed modeling framework for agent-based systems in industry. The advantages include provision of simple graphical design tools that enable non-mathematical designers to use it efficiently. However, AUML specifications are often graphical and lack formal semantics. Therefore, one cannot verify AUML designs formally to guarantee certain assurance, which is a very important factor to be considered in the design of safety-critical systems. The specification language called extended DESCARTES [57], provides an executable specification language for BDI (Belief-Desire-Intention) agents, based on Hoare logic. The language is an extension of the DESCARTES specification language [65] developed for specifying real-time systems. However, the language is targeted to specify only closed-loop BDI agents, and lacks expressions for the self-learning of autonomous agents. GAIA [58] is one of the initial approaches for agent-oriented design and analysis. However, the lack of a formal specification language, and the inability

to model dynamic and open systems are its major drawbacks. SLABS (Specification Language for agent Based Systems)[59] is one of the popular agent specification languages that specifies the notion of agents, environment and multi-agent systems, and communication between agents. However, SLABS lacks an executable framework like DESCARTES and fails to express high-level agent properties like self-learning.

CASL (Cognitive Agent Specification Language) [60] specifies agents with mental attributes, knowledge, beliefs, and goals. For the formal specification, it considers the action theory defined by situation calculus. However, limited expressiveness due to the employed modeling notations, and difficulty in specifying complex multi-agent systems are the major drawbacks of CASL. DESIRE [61] is a modeling framework developed to specify multi-agent systems, which allows user to specify various intra-agent and inter-agent functionalities. However, DESIRE lacks a formal language to represent the agents, and does not specify various agent properties, such as beliefs, desires, intentions, commitments etc. dMARS (Distributed Multi-Agent Reasoning System) [62] is based on a Procedural Reasoning System (PRS) [66] for modeling BDI agents. However, the approach restricts only to BDI agents, does not support agent properties such as agent roles, interactions and message passing and it does not provide tool support for executing agent specifications.

Another type of approach relies on using traditional software engineering formalisms. One such approach proposed by Luck and d'Inverno [67] using the Z-specification language. However, the usage of Z makes the specifications of MAS complex. Moreover, certain aspects of MAS, like reactive behaviour are difficult to specify using Z. In addition, these specifications are not executable so simulation and prototyping are not possible [68]. Hilaire et al. [68] have used a multi-formalism-based approach using Object Z and statecharts, which provides expressiveness to specify agent reactiveness and supports prototyping by simulation. However, the specifications are way too complex to be easily comprehensible. Moreover, the Object Z specifications are not executable. Another interesting work by authors in [69], uses Event-B specifications to specify goal-oriented resilient MAS. The approach is not as complex as Z-language, and can specify different abstractions, and in addition, has a tool support Rodin to develop the specifications, hence can be viewed as complementary to our approach due to the deductive approach for verification.

Few works have considered the specification and formal analysis of agent behavior in architecture description languages [70]. However, it uses complex formal semantics that hinder their usability and extension.

In comparison, we propose solutions that are able to specify AAL sys-

tem architectures that possess autonomy, non-determinism, probabilistic and real-time behaviour. In our work, we choose AADL as modeling framework due to the fact that it is a popular architecture modeling framework used with a practical appeal, also providing tool support [19]. In addition to the rich semantics provided by the language to specify real-time embedded systems, and its mechanisms to carry out initial architecture analysis (schedulability, latency, resource utilization, error analysis, etc.), the language is also extensible with user-defined properties and annex sub-languages. In the AADL modeling framework that we propose in Paper D [12] to specify MAS, we present an annex extension to the core AADL language that can specify the non-deterministic, probabilistic, real-time behaviours of agents along with agent learning, reactiveness, and system fault-tolerance. Although we have currently modeled the specifications of a small-scale MAS architecture comprising of simple reflex agents and other self-learning agents utilizing reinforcement learning, due to the extensibility offered by AADL, we can also extend our proposed sub-language to represent any agent types, and their properties. Moreover, there are also many standardized annexes like the Behaviour Annex (BA) and Error Annex (EA) that are integrated to the core AADL, which can also be utilized for specifying the behaviour of systems according to the needs. We show the usage of AADL's EA and BA for specifying the behaviour of our centralized AAL architecture (Paper C [6]).

For the purpose of formal analysis, we employ model-checking. Unlike theorem proving approaches for the structured formal development of systems [71], model-checking is an automated approach, although it has limitations with respect to state space explosion in case of large models and is less expressive compared to theorem-provers.

To enable model checking our architectural models, we transform the AADL model to timed automata constructs. There have also been approaches to formally verify AADL designs in other domains. The transformation approach from AADL to TA or variants has been already addressed by related work [72, 73, 74]. Although these approaches are automated verification techniques, there is a lack of focus on abstract components/patterns with stochastic properties (like our approach in Paper C [6]). In addition, these approaches also suffer from state-space explosion, therefore they might not scale well with complex AAL designs. Nevertheless, there is interesting research that deals with stochastic properties and statistical model checking for the analysis of extended AADL models. One such example is the work of Bruintjes et al. [75], where the authors have used an SMC approach for timed reachability analysis of extended AADL designs. Although our approach in Paper C [6] also focuses

on linear systems, it is different from the mentioned work in the fact that we focus on abstract components, and also introduce BA modeling for capturing the functional behavior of our modules, specifically for modeling the behavior of intelligent DSS. In their work, Bruintjes et al. use the SLIM Language, which is strongly based on AADL and is specific to avionics and automotive industry, including the error behavior and modes. However, we use the AADL core language with its standardized annex sets (EA and BA) for the architecture specification, thereby enabling the representation of the functional and error behavior with the architecture model. The abstract component-based modeling also brings exensiblity and reusability to our approach. Moreover, the authors only consider the event occurrences or delay variations using uniform or exponential distributions, whereas by employing our user-defined properties, we can also specify other distributions. Furthermore, the approach of Bruintjes et al. only deals with evaluation of time-bounded queries, however we also evaluate properties like reliability, data consistency, etc., besides timeliness. Another interesting work [76], possibly carried out in parallel with our work, employs statistical model checking using UPPAAL SMC to evaluate the performance of nonlinear hybrid models with uncertainty modeled in extended AADL. Although the approach is not specific to the AAL domain, it is promising to specify complex CPS systems considering uncertainties from the physical environment. Unlike our model which uses STA, the authors use Priced Timed Automata (PTA) models. In our work in Paper D [12], we also propose the extension of AADL with the Agent Annex specification, whose semantics we define in terms of Stochastic Transition Systems, that can explicitly capture the non-determinism, probabilistic and real-time behavior of AAL systems. We also specify the formal encoding of our AAL specific AADL constructs as PTA and show exhaustive verification in PRISM.

In the AAL domain, we do not have any evidence for any AAL architectures existing on the market being formally assured, although there has been some research in this direction. Parente et al. provide a list of various formal methods that can be used for AAL systems [77]. Rodrigues et al. perform a dependability analysis of AAL architectures using UML and PRISM [11]. Other interesting works use temporal reasoning [10, 78] and Markov Decision Processes to formally verify the reliability of AAL systems [79]. However, the analysis presented in these approaches address only simple scenarios and are not used to analyze complex behaviors of integrated AAL systems and their decision making capabilities during critical scenarios, unlike the work presented in this thesis, in Paper C [6] and Paper D [12].

Chapter 7

Conclusions and Future Work

In this thesis, we have presented the first research steps towards increasing the support offered to the elderly by providing assured intelligent AAL solutions that integrate most of the functionalities that the users would need to be helped in their daily lives.

First of all, we have surveyed the SOA and SOP of existing solutions and identified that most of them are fragmented, less user-friendly, and lack assurance of their functionality and QoS. We have evaluated the performance of AAL systems by combining functionalities of individualized solutions and identified that they are not sufficient to tackle potential critical scenarios involving multiple events, which need combined analysis for enhanced and safe reasoning.

As a second contribution, we have proposed a generic model of an integrated architecture solution for AAL, with a centralized “brain" (intelligent DSS) and its formal assurance framework. This architecture can be chosen as the integration framework for future AAL systems, if major concerns are the ease of development and maintenance. To carry out architecture analysis, we have represented it at pattern-level using AADL. These representations can be easily instantiated to form specific architecture types. In this thesis, we present three different configurations of the generic model - a simple model, an intermediate model and a complex model. To provide formal verification for the AAL systems, we have formally encoded the AADL model into a Network of

Stochastic Timed Automata (NSTA). We also show the formal modeling and analysis of the simple and complex configuration (CAMI architecture). In case of the simple configuration, the formal model represented as NSTA is verified with the model-checking tool UPPAAL, whereas for the complex configuration, exhaustive model-checking does not scale and hence we use the statistical model-checking tool, UPPAAL SMC, to ensure functional behavior with timeliness, consistency and fault-tolerance. The approach presented paves the way for the development of formally assured future intelligent AAL solutions that integrate multiple functionalities and it can be applied at earlier design stages to capture potential errors that can propagate across the development stages, which may result in significant re-engineering costs. Our architecture description framework (AADL) has a commercially available tool support, OSATE [80] for automated modeling, and provides some preliminary architecture level analysis. It also allows us to model the behavior of the architecture components via behavior annex and encode the probabilities of failures of various components, via the error annex. However, AADL also has its limitations of expressing complex behaviors of algorithms such as CBR, which we have omitted in this work. The analysis approach which we use is exhaustive model checking and stochastic model checking, that is automated via commercial tools called UPPAAL [24], and its extension, UPPAAL SMC [9], respectively. The verification results are specific to our architecture instantiations, however one can use the approach to verify any set of requirements for various architecture types defined by the generic architectural model. It is worth mentioning that the results are derived assuming high reliability of individual architecture components and considering specific values for the periods and execution times. However, taking into account the wide variety of available sensors and other components, we can easily adapt the values to account for requirements of any specific architecture.

Our third contribution is another architecture of integrated AAL system architecture, following a distributed approach with multiple intelligent agents and its formal modeling and analysis framework. If fault-tolerance, scalability, adaptability, and simultaneous access to multiple users are the major considerations, then the second solution outweighs the first-one. For representing our agent-based architecture, we use the same AADL modeling framework, as our first solution. Although MAS specifications based on logics and domain specific languages do exist and are popular, they are mostly limited to specification of properties at the agent level and also do not have tool support (see Chapter 6). AADL, on the other hand, allows us to focus on the component level (here *agents*) and also at the system level (*MAS architecture*) and can effectively

model agents' real-time characteristics. However, since the core AADL and its integrated annexes lack expressiveness to specify the behaviour of multi-agent systems that are non-deterministic, probabilistic and real-time, we have proposed a sub-language extension to AADL, named *Agent Annex*. Our modeling framework of MAS is the core AADL language and Agent Annex. The semantics of the modeling framework is encoded as a Stochastic Transition System. We have also proposed a formal analysis framework for a small-scale MAS architecture that possesses four agent categories and their back-ups using the PRISM model checker that supports exhaustive model-checking of probabilistic models. To enable model-checking in PRISM, the AADL model of the system, including its Agent Annex encoded as Stochastic Transition Systems (STS) is represented as Probabilistic Timed Automata (PTA) in PRISM. Also, since our approach is exhaustive, we provide comprehensive guarantees to the functional and QoS attributes of the system.

As a final contribution, we have also shown some initial validation of the architecture of the first category with respect to various functionalities like fall detection, health monitoring, voice interaction and supervised physical exercises.

Future Work. There are several directions for future research endeavors to fill in the voids in the current state of the framework proposed for analyzing our AAL architectures. The most immediate future work is to provide tool support for our model-transformations, that is, to automate the AADL to NSTA and PTA transformations, respectively. We also plan to integrate the Agent Annex sublanguage to the core AADL. In addition, since the PRISM-based formal analysis framework of MAS is hard to scale in case of a larger system with many and different kinds of agents, we intend to show the feasibility of using other model-checkers, like UPPAAL SMC instead. Although this approach can generate only probabilistic guarantees, it is one of the best suited approach to formally analyze complex cyber-physical systems that do not scale with exhaustive verification. Finally, we also envision to perform a comparison of the two architecture solutions that we have proposed in terms of their formal modeling and analysis framework. We also intend to continue with the system validation of CAMI architecture in real-user scenarios.

Bibliography

[1] Department of Economic and Social Affairs Population Division. World Population Ageing 2015. Technical report, United Nations, NewYork, 11 2015.

[2] Parisa Rashidi and Alex Mihailidis. A survey on ambient-assisted living tools for older adults. *IEEE journal of biomedical and health informatics*, 17(3):579–590, 2013.

[3] Ashalatha Kunnappilly, Cristina Seceleanu, and Maria Lindén. Do We Need an Integrated Framework for Ambient Assisted Living? In *Ubiquitous Computing and Ambient Intelligence: 10th International Conference, UCAmI 2016, San Bartolomé de Tirajana, Gran Canaria, Spain, November 29–December 2, 2016, Part II 10*, pages 52–63. Springer, 2016.

[4] Ruijiao Li, Bowen Lu, and Klaus D McDonald-Maier. Cognitive assisted living ambient system: A survey. *Digital Communications and Networks*, 1(4):229–252, 2015.

[5] Ashalatha Kunnappilly, Alexandru Sorici, Imad Alex Awada, Irina Mocanu, Cristina Seceleanu, and Adina Madga Florea. A Novel Integrated Architecture for Ambient Assisted Living Systems. In *Computer Software and Applications Conference (COMPSAC), 2017 IEEE 41st Annual*, volume 1, pages 465–472. IEEE, 2017.

[6] Ashalatha Kunnappilly, Raluca Marinescu, and Cristina Seceleanu. A Model-Checking-Based Framework For Analyzing Ambient Assisted Living Solutions. Mälardalen Real-Time Research Centre, Mälardalen University, March 2019.

[7] Peter Feiler. Open Source AADL tool environment (OSATE). In *AADL Workshop, Paris*, pages 1–40, 2004.

[8] Alexandre David, Dehui Du, Kim G Larsen, Axel Legay, Marius Mikučionis, Danny Bøgsted Poulsen, and Sean Sedwards. Statistical model checking for stochastic hybrid systems. *Electronic Proceedings in Theoretical Computer Science*, pages 122–136, 2012.

[9] Alexandre David, Kim G Larsen, Axel Legay, Marius Mikučionis, and Danny Bøgsted Poulsen. Uppaal SMC tutorial. *International Journal on Software Tools for Technology Transfer*, 17(4):397–415, 2015.

[10] Juan C Augusto and Chris D Nugent. The use of temporal reasoning and management of complex events in smart homes. In *Proceedings of the 16th European Conference on Artificial Intelligence*, pages 778–782. IOS Press, 2004.

[11] Genaína Nunes Rodrigues, Vander Alves, Renato Silveira, and Luiz A Laranjeira. Dependability analysis in the ambient assisted living domain: An exploratory case study. *Journal of Systems and Software*, 85(1):112–131, 2012.

[12] Ashalatha Kunnappilly, Simin Cai, Cristina Seceleanu, and Raluca Marinescu. Architecture modelling and formal analysis of intelligent multi-agent systems. In *14th International Conference on Evaluation of Novel Approaches to Software Engineering*, May 2019.

[13] Richard S Sutton, Andrew G Barto, et al. Introduction to reinforcement learning. 135, 1998.

[14] Luca De Alfaro. Stochastic transition systems. In *International Conference on Concurrency Theory*, pages 423–438. Springer, 1998.

[15] Imad Alex Awada, Oana Cramariuc, Irina Mocanu, Cristina Seceleanu, Ashalatha Kunnappilly, and Adina Magda Florea. An end- user perspective on the cami ambient and assisted living project. In *12th annual International Technology, Education and Development Conference*, March 2018.

[16] Peter H Feiler, Bruce Lewis, Steve Vestal, and Ed Colbert. An overview of the SAE architecture analysis & design language (AADL) standard: a basis for model-based architecture-driven embedded systems engineering. In *Architecture Description Languages*, pages 3–15. Springer, 2005.

[17] RB Frana, J-P Bodeveix, Mamoun Filali, and J-F Rolland. The AADL behaviour annex–experiments and roadmap. In *Engineering Complex Computer Systems, 2007. 12th IEEE International Conference on*, pages 377–382. IEEE, 2007.

[18] Julien Delange and Peter Feiler. Architecture fault modeling with the AADL error-model annex. In *Software Engineering and Advanced Applications (SEAA), 2014 40th EUROMICRO Conference on*, pages 361–368. IEEE, 2014.

[19] *Model-based engineering with AADL: an introduction to the SAE architecture analysis & design language*.

[20] Michael Wooldridge and Nicholas R Jennings. Intelligent agents: Theory and practice. *The knowledge engineering review*, 10(2):115–152, 1995.

[21] Michael Wooldridge. *An introduction to multiagent systems*. John Wiley & Sons, 2009.

[22] Long-Ji Lin. Self-improving reactive agents based on reinforcement learning, planning and teaching. *Machine learning*, 8(3-4):293–321, 1992.

[23] Chih-Han Yu, Justin Werfel, and Radhika Nagpal. Collective decision-making in multi-agent systems by implicit leadership. In *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems: volume 3-Volume 3*, pages 1189–1196. International Foundation for Autonomous Agents and Multiagent Systems, 2010.

[24] Kim G Larsen, Paul Pettersson, and Wang Yi. UPPAAL in a nutshell. *International journal on software tools for technology transfer*, 1(1-2):134–152, 1997.

[25] Marta Kwiatkowska, Gethin Norman, and David Parker. PRISM: Probabilistic symbolic model checker. In *International Conference on Modelling Techniques and Tools for Computer Performance Evaluation*, pages 200–204. Springer, 2002.

[26] Rajeev Alur and David L Dill. A theory of timed automata. *Theoretical computer science*, 126(2):183–235, 1994.

[27] Rajeev Alur, Costas Courcoubetis, and David Dill. Model-checking for real-time systems. In *Logic in Computer Science, 1990. LICS'90, Proceedings., Fifth Annual IEEE Symposium*, pages 414–425. IEEE, 1990.

[28] Peter E Bulychev, Alexandre David, Kim G Larsen, Axel Legay, Guangyuan Li, and Danny Bøgsted Poulsen. Rewrite-Based Statistical Model Checking of WMTL. *RV*, 7687:260–275, 2012.

[29] Hans Hansson and Bengt Jonsson. A logic for reasoning about time and reliability. *Formal aspects of computing*, 6(5):512–535, 1994.

[30] Hilary J Holz, Anne Applin, Bruria Haberman, Donald Joyce, Helen Purchase, and Catherine Reed. Research methods in computing: what are they, and how should we teach them? In *ACM SIGCSE Bulletin*, volume 38, pages 96–114. ACM, 2006.

[31] Jane Webster and Richard T Watson. Analyzing the past to prepare for the future: Writing a literature review. *MIS quarterly*, pages xiii–xxiii, 2002.

[32] Wayne Goddard and Stuart Melville. *Research methodology: An introduction*. Juta and Company Ltd, 2004.

[33] Dawn G Gregg, Uday R Kulkarni, and Ajay S Vinzé. Understanding the philosophical underpinnings of software engineering research in information systems. *Information Systems Frontiers*, 3(2):169–183, 2001.

[34] Mary Shaw. What makes good research in software engineering? *International Journal on Software Tools for Technology Transfer*, 4(1):1–7, 2002.

[35] Visual Paradigm. Visual paradigm for uml. *Visual Paradigm for UML-UML tool for software application development*, page 72, 2013.

[36] Don Batory. Feature models, grammars, and propositional formulas. In *International Conference on Software Product Lines*, pages 7–20. Springer, 2005.

[37] Mobyen Uddin Ahmed, Mats Björkman, and Maria Lindén. A generic system-level framework for self-serve health monitoring system through internet of things (iot). *Studies in health technology and informatics*, 211:305–307, 2015.

[38] Dante I Tapia, Sara Rodrıguez, and Juan M Corchado. A distributed ambient intelligence based multi-agent system for Alzheimer health care. In *Pervasive Computing*, pages 181–199. Springer, 2009.

[39] PA USA Society of Automotive Engineers, Warrendale. AE-AS5506/1, SAE Architecture Analysis and Design Language (AADL) Annex Volume 1, Annex C: AADL Meta-Model and Interchange Formats, 2006.

[40] Martin Becker. Software architecture trends and promising technology for ambient assisted living systems. In *Dagstuhl Seminar Proceedings*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2008.

[41] David Isern, David Sánchez, and Antonio Moreno. Agents applied in health care: A review. *International journal of medical informatics*, 79(3):145–166, 2010.

[42] John Nealon and Antonio Moreno. Agent-based applications in health care. *Applications of software agent technology in the health care domain*, pages 3–18, 2003.

[43] Juan De Paz, Sara Rodríguez, Javier Bajo, Juan Corchado, and Emilio Corchado. OVACARE: A multi-agent system for assistance and health care. *Knowledge-Based and Intelligent Information and Engineering Systems*, pages 318–327, 2010.

[44] Paolo Sernani, Andrea Claudi, Luca Palazzo, Gianluca Dolcini, and Aldo Franco Dragoni. Home care expert systems for ambient assisted living: A multi-agent approach. In *Proceedings of the Workshop on The Challenge of Ageing Society: Technological Roles and Opportunities for Artificial Intelligence, Turin, Italy*, volume 6, 2013.

[45] Elarbi Badidi and Larbi Esmahi. A cloud-based approach for context information provisioning. *arXiv preprint arXiv:1105.2213*, 2011.

[46] Alisa Devlic and Klintskog Erik. Context retrieval and distribution in a mobile distributed environment. In *Third Workshop on Context Awareness for Proactive Systems (CAPS 2007)*, 2007.

[47] Abdur Forkan, Ibrahim Khalil, and Zahir Tari. CoCaMAAL: A cloud-oriented context-aware middleware in ambient assisted living. *Future Generation Computer Systems*, 35:114–127, 2014.

[48] Angelika Dohr, Robert Modre-Osprian, Mario Drobics, Dieter Hayn, and Günter Schreier. The Internet of Things for Ambient Assisted Living. *ITNG*, 10:804–809, 2010.

[49] Giancarlo Fortino, Antonio Guerrieri, and Wilma Russo. Agent-oriented smart objects development. In *Computer Supported Cooperative Work in Design (CSCWD), 2012 IEEE 16th International Conference on*, pages 907–912. IEEE, 2012.

[50] Peter Leong and Liming Lu. Multiagent web for the Internet of Things. In *Information Science and Applications (ICISA), 2014 International Conference on*, pages 1–4. IEEE, 2014.

[51] Teemu Leppänen, Jukka Riekki, Meirong Liu, Erkki Harjula, and Timo Ojala. Mobile agents-based smart objects for the internet of things. In *Internet of Things Based on Smart Objects*, pages 29–48. Springer, 2014.

[52] Feng Zhou, Jianxin Roger Jiao, Songlin Chen, and Daqing Zhang. A case-driven ambient intelligence system for elderly in-home assistance applications. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 41(2):179–189, 2011.

[53] Eduardo Lupiani, Jose M Juarez, Jose Palma, Christian Serverin Sauer, and Thomas Roth-Berghofer. Using case-based reasoning to detect risk scenarios of elderly people living alone at home. In *International Conference on Case-Based Reasoning*, pages 274–288. Springer, 2014.

[54] Krasimira Kapitanova, Sang H Son, and Kyoung-Don Kang. Using fuzzy logic for robust event detection in wireless sensor networks. *Ad Hoc Networks*, 10(4):709–722, 2012.

[55] Hamid Medjahed, Dan Istrate, Jérôme Boudy, Jean Louis Baldinger, Lamine Bougueroua, Mohamed Achraf Dhouib, and Bernadette Dorizzi. A fuzzy logic approach for remote healthcare monitoring by learning and recognizing human activities of daily living. In *Fuzzy Logic-Emerging Technologies and Applications*. InTech, 2012.

[56] Bernhard Bauer, Jörg P Müller, and James Odell. Agent UML: A formalism for specifying multiagent software systems. *International journal of software engineering and knowledge engineering*, 11(03):207–230, 2001.

[57] Vinitha Hannah Subburaj and Joseph E Urban. A formal specification language for modeling agent systems. In *Informatics and Applications (ICIA), 2013 Second International Conference on*, pages 300–305. IEEE, 2013.

[58] Michael Wooldridge, Nicholas R Jennings, and David Kinny. The Gaia methodology for agent-oriented analysis and design. *Autonomous Agents and multi-agent systems*, 3(3):285–312, 2000.

[59] Hong Zhu. SLABS: A formal specification language for agent-based systems. *International Journal of Software Engineering and Knowledge Engineering*, 11(05):529–558, 2001.

[60] Steven Shapiro, Yves Lespérance, and Hector J Levesque. The cognitive agents specification language and verification environment for multiagent systems. In *Proceedings of the first international joint conference on Autonomous agents and multiagent systems: part 1*, pages 19–26. ACM, 2002.

[61] Frances MT Brazier, Barbara M Dunin-Keplicz, Nick R Jennings, and Jan Treur. DESIRE: Modelling multi-agent systems in a compositional formal framework. *International Journal of Cooperative Information Systems*, 6(01):67–94, 1997.

[62] Mark d'Inverno, Michael Luck, Michael Georgeff, David Kinny, and Michael Wooldridge. The dMARS architecture: A specification of the distributed multi-agent reasoning system. *Autonomous Agents and Multi-Agent Systems*, 9(1-2):5–53, 2004.

[63] Haiping Xu and Sol M Shatz. An agent-based Petri net model with application to seller/buyer design in electronic commerce. In *Autonomous Decentralized Systems, 2001. Proceedings. 5th International Symposium on*, pages 11–18. IEEE, 2001.

[64] Marcelo Finger, Michael Fisher, and Richard Owens. Metatem at work: Modelling reactive systems using executable temporal logic. In *Sixth International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems (IEA/AIE-93)*, 1993.

[65] K-Y Sung and Joseph E Urban. Real-time Descartes: A real-time specification language. In *Distributed Computing Systems, 1992., Proceedings of the Third Workshop on Future Trends of*, pages 79–85. IEEE, 1992.

[66] Michael P Georgeff and Amy L Lansky. Reactive reasoning and planning. In *AAAI*, volume 87, pages 677–682, 1987.

[67] Michael Luck, Mark d'Inverno, et al. A formal framework for agency and autonomy. In *ICMAS*, volume 95, pages 254–260, 1995.

[68] Vincent Hilaire, Abder Koukam, Pablo Gruer, and Jean-Pierre Müller. Formal specification and prototyping of multi-agent systems. In *International Workshop on Engineering Societies in the Agents World*, pages 114–127. Springer, 2000.

[69] Linas Laibinis, Inna Pereverzeva, and Elena Troubitsyna. Formal reasoning about resilient goal-oriented multi-agent systems. *Science of Computer Programming*, 148:66–87, 2017.

[70] Flavio Oquendo. π-ADL: an Architecture Description Language based on the higher-order typed π-calculus for specifying dynamic and mobile software architectures. *ACM SIGSOFT Software Engineering Notes*, 29(3):1–14, 2004.

[71] Maksym Bortin, Einar Broch Johnsen, and Christoph Lüth. Structured formal development in Isabelle. *Nordic Journal of Computing*, 13(1/2):2, 2006.

[72] Loïc Besnard, Thierry Gautier, Paul Le Guernic, Clément Guy, Jean-Pierre Talpin, Brian Larson, and Etienne Borde. Formal semantics of behavior specifications in the architecture analysis and design language standard. In *Cyber-Physical System Design from an Architecture Analysis Viewpoint*, pages 53–79. Springer, 2017.

[73] Mohamed Elkamel Hamdane, Allaoui Chaoui, and Martin Strecker. From AADL to timed automaton-A verification approach. *International Journal of Software Engineering and Its Applications*, 7(4), 2013.

[74] Andreas Johnsen, Kristina Lundqvist, Paul Pettersson, and Omar Jaradat. Automated verification of AADL-specifications using UPPAAL. In *High-Assurance Systems Engineering (HASE), 2012 IEEE 14th International Symposium on*, pages 130–138. IEEE, 2012.

[75] Harold Bruintjes, Joost-Pieter Katoen, and David Lesens. A statistical approach for timed reachability in AADL models. In *Dependable Systems and Networks (DSN), 45th Annual IEEE/IFIP International Conference on*, pages 81–88. IEEE, 2015.

[76] Yongxiang Bao, Mingsong Chen, Qi Zhu, Tongquan Wei, Frederic Mallet, and Tingliang Zhou. Quantitative performance evaluation of uncertainty-aware hybrid AADL designs using statistical model checking. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 36(12):1989–2002, 2017.

[77] Guido Parente, Christopher D Nugent, Xin Hong, Mark P Donnelly, Liming Chen, and Enrico Vicario. Formal modeling techniques for ambient assisted living. *Ageing International*, 36(2):192–216, 2011.

[78] using temporal logic and model checking in automated recognition of human activities for ambient-assisted living.

[79] Yan Liu, Lin Gui, and Yang Liu. MDP-based reliability analysis of an ambient assisted living system. In *International Symposium on Formal Methods*, pages 688–702. Springer, 2014.

[80] OSATE-Open Source AADL Test Environment. `http://osate.github.io/`. Accessed: 2018-05-15.