8-2018

Hybrid Cloud Model Checking Using the Interaction Layer of HARMS for Ambient Intelligent Systems

Mauricio Alejandro Gomez Morales
Purdue University

HYBRID CLOUD MODEL CHECKING USING THE INTERACTION LAYER OF HARMS FOR AMBIENT ASSISTIVE LIVING ENVIRONMENTS

A Dissertation

Submitted to the Faculty

of

Purdue University

by

Mauricio Alejandro Gomez Morales

In Partial Fulfillment of the

Requirements for the Degree

of

Doctor of Philosophy

August 2018

Purdue University

West Lafayette, Indiana

THE PURDUE UNIVERSITY GRADUATE SCHOOL
STATEMENT OF DISSERTATION APPROVAL

Dedicated to God, my parents, and my nieces.

Armin and Aura; one has always been my example to follow, the other has taught me to be patient and persistent at the same time.

Alejandra, Jimena, and Daniela who with their creativity and energy have indirectly given me a push to keep fighting.

ACKNOWLEDGMENTS

The goal that I am achieving this time has been a success not only because of my own effort. On the contrary, all this would not have been possible without the help of many people. I feel blessed for that and I would like to specially thank to:

First, I feel greatly thankful with my major professor, Dr. Eric T. Matson. He has dedicated tons of time to guide me throughout all this process. I have witnessed how he truly worries about his students, including me, not only in areas such as professional, research, and studies but in personal matters as well. He is always looking to open opportunities where his students can go making their own path. And after meeting him, I can say that now I have seen a way of how to be a good professor when I get the chance to be part of a faculty team.

Second, I would like to thank Dr. Amirat, Dr. Chibani, Dr. Springer, and Dr. Min, who have given me good insights to improve my dissertation.

Third, thanks to all my family members, Armin, Aura, Armin (Jr.), Margarita, Alejandra, Jimena, Daniela, Fernando, and Olga who have always been cheering me up throughout all this long walked road.

Fourth, I am very happy that God has put all that people that I can call friends, relatives, and labmates who have been there with me. Special gratitude to Nathalie, my best friend, who in spite the distance, the difference of time, and any imaginable issue, there is not a time I cannot recall of listening or reading the right words at the right time from her.

Finally, this dissertation was conducted in Cotutelle research cooperation with the LISSI Lab and the University of Paris-Est, Creteil.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABSTRACT

Gomez, Mauricio A. Ph.D., Purdue University, August 2018. Hybrid Cloud Model Checking Using the Interaction Layer of HARMS for Ambient Assistive Living Environments. Major Professor: Eric T. Matson Professor.

Soon, humans will be co-living and taking advantage of the help of multi-agent systems in a broader way than the present. Such systems will involve machines or devices of any variety, including robots. These kind of solutions will adapt to the special needs of each individual. However, to the concern of this research effort, systems like the ones mentioned above might encounter situations that will not be seen before execution time. It is understood that there are two possible outcomes that could materialize; either keep working without corrective measures, which could lead to an entirely different end or completely stop working. Both results should be avoided, specially in cases where the end user will depend on a high level guidance provided by the system, such as in ambient intelligence applications.

This dissertation worked towards two specific goals. First, to assure that the system will always work, independently of which of the agents performs the different tasks needed to accomplish a bigger objective. Second, to provide initial steps towards autonomous survivable systems which can change their future actions in order to achieve the original final goals. Therefore, the use of the third layer of the HARMS model was proposed to insure the indistinguishability of the actors accomplishing each task and sub-task without regard of the intrinsic complexity of the activity. Additionally, a framework was proposed using model checking methodology during run-time for providing possible solutions to issues encountered in execution time, as a part of the survivability feature of the systems final goals.

CHAPTER 1. INTRODUCTION

Long ago humans stopped having those futuristic thoughts of looking at robots walking, moving, even flying by themselves. In recent times, instead, the research community started developing and designing robots to interact either with humans or with other robots and machines. Many real world problems can only be elucidated by a set of different individuals that work together towards finding and executing specific solutions. Some examples for groups of human activities are the construction of a bridge, surveillance of a wide mountainous area, musical performances, and military missions. Likewise, there are a number of reasons to think that achieving cooperation between any number of humans, machines, and robots will soon be needed to overcome specific problems in the future.

When multi-agent systems become pervasive and integrated within human society, they will encounter dynamic environments. Concepts that have been applied to control these environment variations like reliability and adaptability may not be enough. Survivability of a system, on the other hand, assures not only to keep working as reliability and reacting as adaptability but also assures the fulfillment of the original set of final goals. A survivable technique is based on what is observed in the natural behaviour of living mechanisms that react to inconvenient situations that they have not seen before. Reactions may vary from species to species, however, the survival instinct is present in many of them. For example, a bird may break a wing in the middle of a flight. The normal reaction is to try to counterpart the lack of pushing force in one of the sides by testing other strategies such as flapping faster with the other wing as a merely act to stay alive. In this document, experiments are conducted to measure the indistinguishability. A survivability feature is proposed as a mechanism for any multi-agent system. The tests were focused in Ambient Intelligence (AmI) environments using the Humans, software Agents, Robots, Machines, and Sensors (HARMS) model.

1.1 Problem Statement

Although the field of multi-agent systems, including robotics is still in its early stages, researchers must insure that solutions involving machines and humans are resilient. In other words, a system of this kind needs to be capable enough to autonomously interact with humans or other machines and continue working even when unexpected situations appear. Methods and procedures vastly used for identifying possible errors of systems during design time are verification and validation [1]. Much research has been done on the verification and validation of individual systems. Online verification for standalone systems study is still undergoing. Moreover, problems become more complex-intensive, resource-demanding, costly, and error-prone when the number of agents interacting increases. Perhaps, given that the concept of multi-agent systems (MAS) is relatively novel and of high complexity, the reasons mentioned above make research efforts in MAS validation and verification in an online manner less possible up to now.

Robots or machines with the ability to correctly react to uncertain situations are needed to let humans feel comfortable working with them. One of the mechanisms for this purpose observed in nature are the self-healing organisms which take actions in advance to avoid undesired future results. However, one of the biggest challenges for self-healing systems in an online verification approach is overcoming the state explosion problem. Such problem is generated when the system is in charge of revising all possible paths that the flow of a system can take. This problem can even grow onto bigger proportions when talking about a multi-agent system.

This dissertation proposes a framework that provides means to enhance the indistinguishability feature with the use of the interaction layer of the HARMS model. Survivability feature, in the other hand, is proposed with a framework taking advantage of model checking during run-time to validate the reach-ability of the systems original final goals.

1.2 Scope

The scope of this work is focused on finding possible solutions to issues encountered in the system during run-time. The strategy is based on the survivability feature that is motivated in the trial and error natural reaction that living creatures apply when they face problems that could lead to unwanted future states. The reach-ability of a goal state is verified online with model checking techniques in possible future states of the system when applying changes to the original model which runs smoothly in perfect conditions.

As mentioned above, the validation process is known to consume excessive time and resources; hence, it begs the question of whether, depending on the resources that each agent has to give answer to such requests, it may be better to execute the validation process within its own resources or offloading it to the cloud.

In this matter, this dissertation will present multiple scenarios where heterogeneous cyber-physical systems are involved and will guide or monitor a human in one specific daily activity such as: going to sleep and cooking. There will also be backup devices that could start working instead of the ones damaged or not working properly. Validation processes will take place over the cloud or in the resources of the specific agent. The validation will be performed using a model checking tool such as: NuSMV [2].

1.3 Significance

The findings of this study will redound to the benefit of society considering the need of building systems that will be able to overcome problems that were not identified before execution time. That lack of identification of those issues implies that systems will not count with the mechanisms to work around those possible threats.

The increasing push to embrace pervasive and immerse cyber-physical systems in humankind lives justifies the need of systems with a survivability feature that enhances the accomplishment of the original goals of such systems. The necessity on accomplishing indistinguishability of the agents that carry on the tasks will also give more reliability to multi-agent systems. For the research community, the findings of this effort will guide to a set of different studies that want to provide solutions that are always looking to assure the well-being of the end users.

1.4 Research Questions

Can the implementation of capability model based approaches in multi-agent systems, such as the HARMS model, help to insure the indistinguishability of the agents executing the different tasks to accomplish bigger final goals of a system?

How can off-loading computationally complex processing activities, such as parallel model checking to the cloud, be beneficial to reduce the time to provide a possible solution within the implementation of survivability of systems final goals feature for multi-agent systems using the HARMS model?

1.5 Assumptions

The assumptions for this study include:

- First, the system is a set of heterogeneous actors
- Second, within the devices there were or were not enough resources to execute the validation process.
- Third, the test environment included a program that let the system make decisions over the pass of time in the attempt to avoid in future states malfunctioning.

- Fourth, the proposed solution will be able to be applied to real life scenarios of an ambient intelligence environment. Environments where one or more of the actors stopped working properly while executing the instruction, received and accepted previously, to perform in a collaborative way.

1.6 Limitations

The limitations for this study include:

- First, the solution contemplates that the problems will appear in the interaction or lower layers of the HARMS model. This, because one of the agents will stop working properly. For example, before or when the interaction is taken place.
- Second, a model checking solution is the validation tool that will be used in order to detect a possible path that leads to the desired final goal.
- Third, it will be assumed that one or more agents will not have enough resources to run the validation tool used to verify if the modification in the paths guide the system to the desired final goals. Consequently, the process of model checking will be executed in a parallel way, instead of a sequential manner. That situation will lead the system to be able to compute the time it takes to execute model checking processing within the components of the agent and also the time for execution in the cloud.

1.7 Delimitations

The delimitations for this study include:

- First, the security mechanisms involved in any of the processes of the scenarios, such as offloading the load-charge of model checking processing to the cloud, are not included in this study.

- Second, the validation of the human part will take into consideration that the human follows as much as possible the instructions received by the actors that integrate the system, such as a robot.

- Third, the environment to make real scenario tests is going to be ambient intelligence, where a robot will help guiding a person to do a normal daily activity, such as guiding an elderly person go to sleep.

- Fourth, all the network connectivity will be assured to be working properly all the time that the system is running. This is delimited in order to assure that the system encounters only problems generated in the interaction layer of the system.

1.8 Summary

This chapter provided the problem statement, scope, significance, research questions, assumptions, limitations, delimitations, and definitions for the research project. The reminder of this dissertation document is organized as follows: The next chapter provides a review of the literature relevant to this dissertation. Chapter 3, discusses the HARMS model in terms of how it is a tool that helps insuring the indistinguishability of multi-agent systems. It also includes the full description of how the interaction layer was implemented based in the capability model. Following that, in chapter 4 it is detailed the way that indistinguishability and survivability features are applied using HARMS model in applications of ambient intelligence. Finally, in chapter 5 are mentioned the contributions, conclusions and are also drawn what are the future paths that following work could take based on what it is presented in this dissertation.

CHAPTER 2. REVIEW OF RELEVANT LITERATURE

This chapter provides a review of the literature relevant to the topics concerning this dissertation.

2.1 Distributed Artificial Intelligence and Multi-Agent Systems

Distributed Artificial Intelligence (DAI) systems were conceived as a group of intelligent entities, called agents, that interact by cooperation, by coexistence or by competition [3]. Researchers have focused on a ramification of DAI which is called cooperative distributed problem solving (CDPS). CDPS covers the study of agents with different capabilities that work in a collaborative way to solve problems that are not possible to be deciphered by any of them alone [4]. CDPS solutions can be measured by two different ways, cohesion and coordination [5]. A cohesive way to evaluate the successfulness of a system as a unit where all agents work towards a goal. The second way, coordination, is when the different agents work together to avoid uncanny situations. Wooldridge [6] argues that issues concerning CDPS comprehends: a)dividing a problem to distribute it between different agents. b)synthesizing a solution in sub-problem results. c)optimization of activities to maximize coherence metric. d)coordination of activities amongst agents avoiding unhelpful interactions and exploit success.

MAS and organization concepts are defined in a very similar way in [7] and [8]. Similar aspects are shared in both of them. Lacking of complete knowledge and all the capabilities needed to solve the problem are common for each of the agents in both kind of systems. Furthermore, data fluctuation is needed to be performed in a decentralized way, global control is impossible, and asynchronous computation is required.

Related to MAS, this work is focused on the aspects of the activity distribution in a multi-agent system using the HARMS model up to the interaction layer. The survivability feature that is needed in such systems is also of vital importance through all this dissertation.

2.1.1 MAS Interaction models

An intelligent agent perceives the environment in which it is placed, interacts with its own environment and takes initiative towards solving a specific problem [9]. Given that the agents interact with their environment, they may be already interacting with one or more agents of the same or different type. However, what makes the difference between intelligent agents and MAS perspective is that the agents will be designed from a multi-agent point of view. The multi-agent environment implies the interaction between those different agents. Different approaches have been proposed which range from very centralized to very de-centralized. Although it is understandable that in a MAS setting all agents will be adding productivity to the complete solution, synergy would not be expected from the simple sum of its component agents.

Submerged in a human society, if it is seen in detail from an organization standpoint, knowledge flow is focused amongts the agents to support the strategic goals of the organization. While individual standpoint focuses more on personal satisfaction, creativity, and development. Organizational knowledge should encompass to the personal interest mentioned above for the individual to decide to take part.

Dignum in [10] expressed her opinion that the society or organization model should provide internal autonomy and collaborative autonomy as requirements. The internal autonomy requirement assures that interaction and structure of the society should be represented differently. Matson in [11] coincides in that, although software agents do not possess the same emotional or physical needs and requirements as humans, they can be adapted to work in a similar way than humans when they are arranged in an organizational environment.

[12], pointed out the problems that researchers find in designing and implementing multi-agent systems are as the following list:

- First, the *domain specification problem* deals with formulation of the problem in a non-ambiguous way.
- Second, the *co-ordination problem* implies a cooperative teamwork that leads to a cohesive and effective overall results.
- Third, the *computational problem* where a computational overload needs to be avoided.
- Fourth, the *implementation problem* comprehends the techniques and tools needed to support a MAS design and implementation.
- Fifth, the *verification problem* in where all formal or practical verification, diagnosis, and possible corrections take place.

This document is focused on the issues between the co-ordination and the computational problems.

Re-organization Process

In a multi-agent system, the re-structural process of an organization may respond to different motivations. Matson in [11] remarks that it must be looking at the interactions and transitions that occur through time to understand why an organization is successful or on the contrary, can be considered imperfect. In other words, when the system detects that it needs to change the structure of the organization, it can be addressed in two different forms.

Dignum, Dignum, and Sonenberg in [13] show two different ways that an organization restructures itself. The same classification is encompassed by Abbas, Shaheen, and Amin [14] who gave specific names to both approaches as follows.

1. Agent-centered MAS (ACMAS) is motivated by a behavioral or agent where internal change happens. Specifically, those changes could be agents joining or leaving or interaction pattern instantiation.

2. Organization centered MAS (OCMAS), is motivated by a structural change which corresponds to an organizational self design or a structural adaptation.

Computational Organization Theory

Computational organization theory (COT) is a new science which combines social science, computer science, and network analysis to represent and design the study of social, organizational, and policy systems as stated by [15].

Models are intended to formalize the understanding of the systems being considered. Hence, a tool adopted within a development methodology for modeling a MAS system is considered an organizational model.

Zambonelli and Omicini [16] discuss that issues in a MAS setting can be analyzed under three different "scales of observation" which are micro, macro and meso scales.

- The micro scale that motivates the artificial intelligence (AI) application into MAS solutions explores nonstandard and extreme development processes.

- The macro scale, because of the multiple agent interactions, entropy and coordination should be considered as specific behavior mechanisms.

- At the meso scale, ways to figure out non-considered phenomena, trust and social intelligence are studied.

Juziuk, Weyns, and Holvoet [17] present an overview of the design pattern classification. The authors of this study saw the need of a classification given that the literature in this regard is either not clear or biased to a specific catalog of patterns. The design pattern classification for Multi-agent Systems is presented in Figure 2.1.

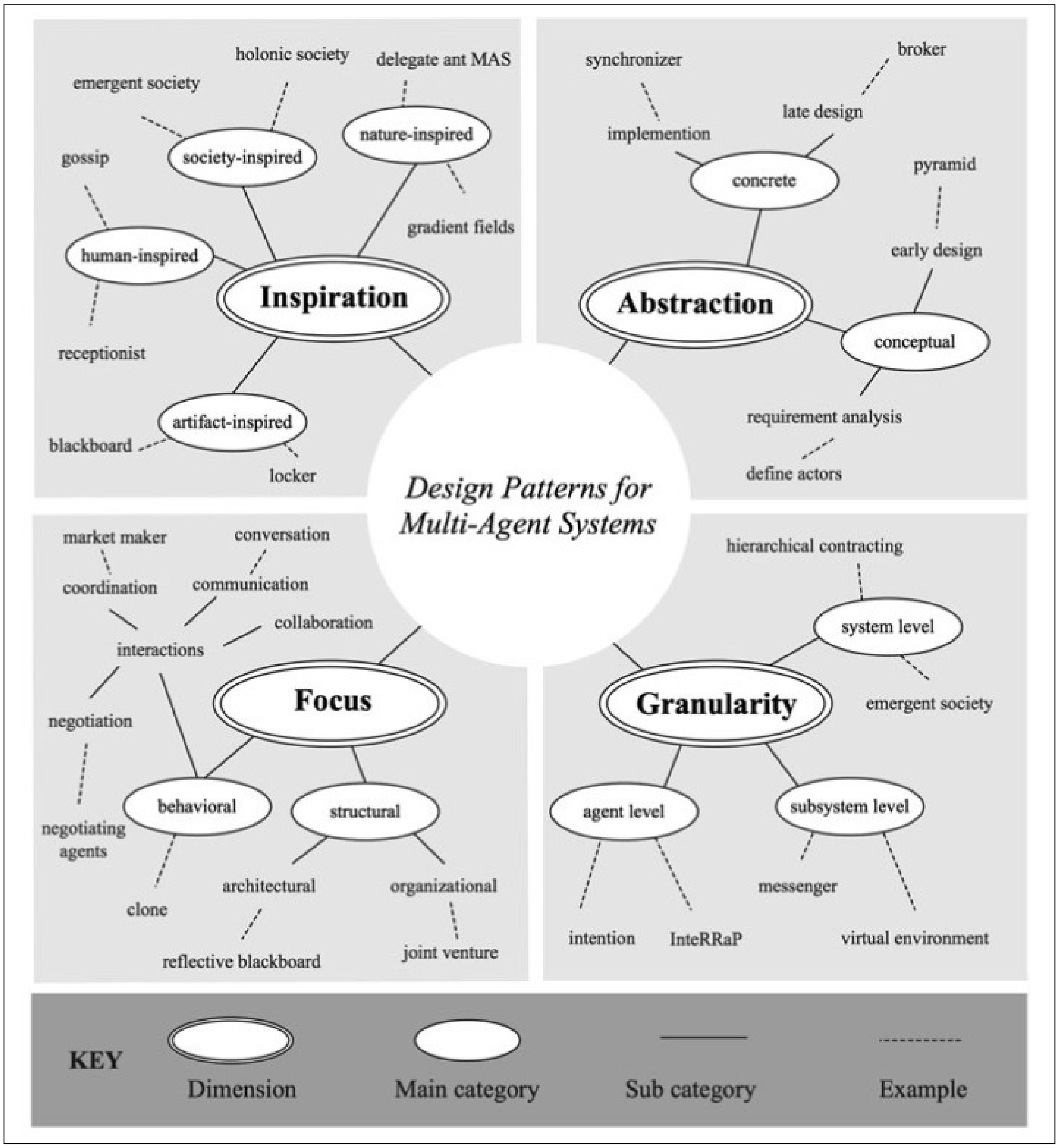


Fig. 2.1. Design Patterns for Multi-Agent Systems

2.1.2 Agent Oriented Metodologies

In this approach, also called ACMAS, the components of the agent and their faces could trigger a reorganization request with complete disregard about how changes can affect the global structure of the system. Hence, global change will be generated from the bottom level agents in a bottom-up communicative way. Agent-based systems focus on dynamically interacting components rather than in the components themselves. For Luck, McBurney, and Preist in [18] agent oriented-methodologies focus on reactive and deliberative behavior which rely in excess of custom-made solutions not giving space to a more general concept that could be replicated.

AAII model

PRS is the procedure reasoning system, where agents dynamically select plans from a plan library to accomplish their goals, taking into consideration their current status. The Australian AI Institute (AAII) model was developed for a range of agent-based systems using PRS-based belief-desire-intention and the distributed multi-agent reasoning system throughout the 1990s. AAII is an iterative process which will have a well-known outcome such as a model very similar to the PRS agent architecture. It basically provides both internal and external models. It identifies the relevant roles in the application domain to develop an agent class hierarchy. Then, AAII model also identifies the responsibilities associated with each role, services required and provided by the role, and the goals associated with each service.

Concurrent Metate M

The concurrent Metate M model is an executable temporal logic for reactive systems. Individual objects execute temporal specifications and communicate with their environment at certain times by broadcasting information as described by [19]. Consequently, in this model, rather than seeing computation as objects sending mail messages to each other, and thus invoking some activity, it can instead be visualized as independent entities listening to messages broadcast from other objects. As a result, the present” formula involves matching the antecedent of these rules against the history of incoming messages and then executing the "present and future" time consequences.

Agent UML

Bauer, Muller, and Odell [20] discovered the need for standards to boost agent-oriented development in the industry. At the time of the discovery, the unified modeling language (UML) was in a stage of maturity, so they took the decision of adding two new diagrams: Protocol diagrams, are an expansion (e.g. roles and others) and an incorporation to the UML state and sequence diagrams. Agent class diagrams extend the general class diagrams to agent-oriented flavor and properties.

Gaia model

Wooldridge [21] defines the Gaia model as a tool that lets an analyst take all the steps of the software cycle in an implementation of a MAS system. ***Abstract*** and ***concrete*** are the two most important categories in the Gaia model. Abstract concepts are used to conceptualize the system during the analysis stage and they are: roles, permissions, responsibilities, protocols, activities, liveness properties, and safety properties. Concrete concepts are used in the design process and are: agent types, services, and acquaintances. An organization is viewed as a collection of roles that stand in certain relationships to one another and that take part in systematic, institutionalized patterns of interactions with other roles.

Agent Oriented Software Engineering

One methodology to study MAS phenomena is the agent-oriented software engineering (AOSE). AOSE was created as the implementation of software systems based on software engineering and artificial intelligence principles, as mentioned in [22]. In terms of application fields, the same authors extend, AOSE can be focused in:

Agent-based grid computing, agents and services, agents for self-adaptive systems, integration of agent-oriented software into existing business processes and implications on business process re-engineering, integration of agents with legacy systems, and multi-agent-based simulation.

According to [23] an AOSE agent should work under the four following premises: autonomy, reactivity, pro-activeness, and social ability. The authors continue making an analogy between object-oriented programming and agent-oriented software engineering. The three differences between both approaches they mention are: autonomy, capability of flexible behavior, and multi-threaded control.

2.1.3 Organization Oriented Methodologies

There are different approaches for an organization oriented methodologies, also called OCMAS. However, for the different approaches to work, the theory of the different organizational paradigms, should be based, which will be explained in the next section. Later, this paper will mention different methodologies.

Organizational Paradigms

As stated by Horling and Lesser in [24] ten different organizational paradigms have been developed through previous research about agent-oriented. Figure 2.2 shows a comparison of those ten paradigms. Nouwens and Bouwman [25] shortened the list to three paradigms, hierarchy, market, and network, based in the different types of coordination in an organization. A more recent study [22], defines MAS organization as a hierarchical organization, a flat organization (or democracy), a subsumtion organization, and a modular organization. For the purpose of a deep research, in this document it was covered the paradigms presented by [24] given that the other two approaches are only simplifications.

Hierarchies are the ones where agents can be seen organized under a tree-like structure [26]. There, agents located in higher levels have more decision power than

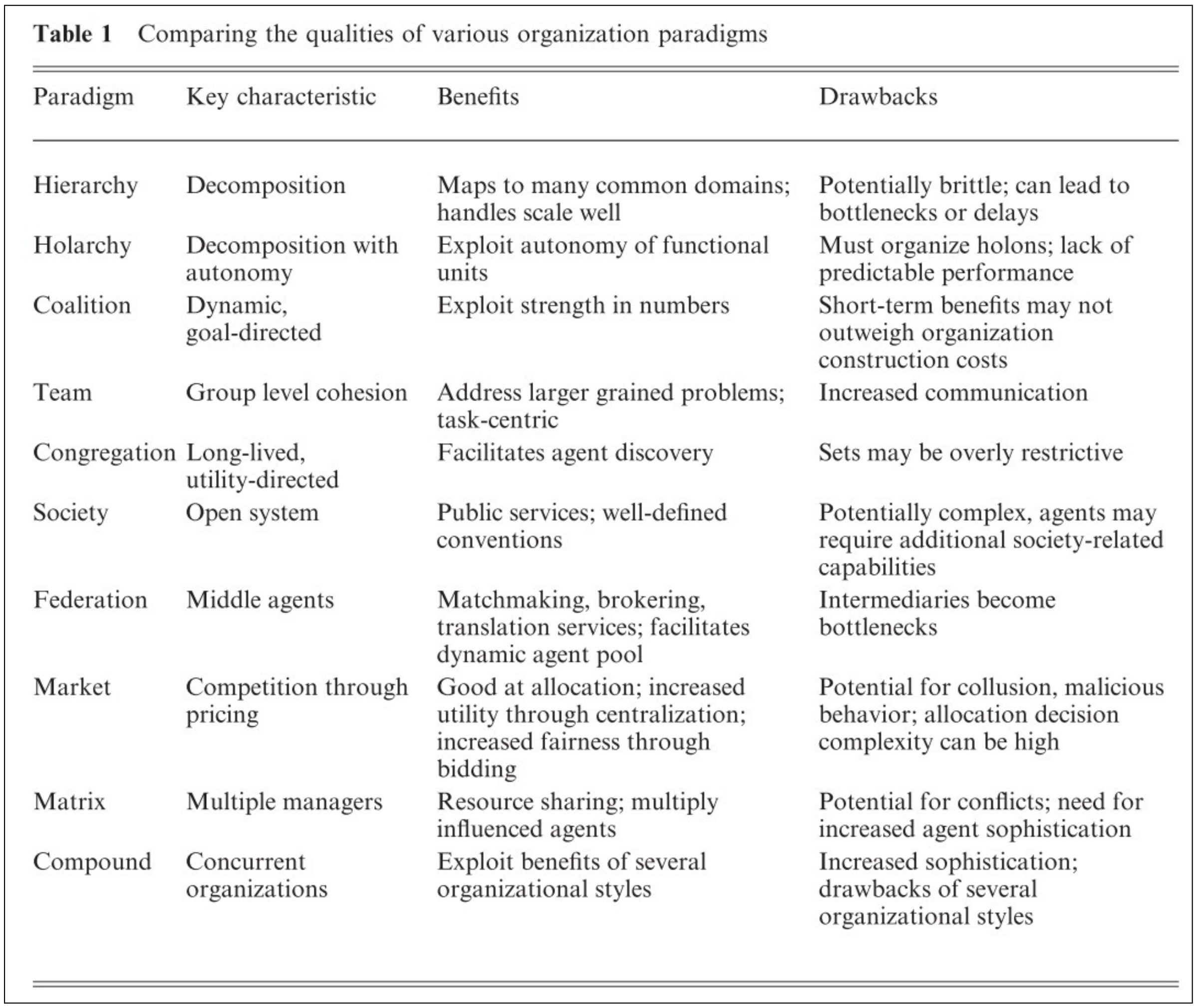


Table 1 Comparing the qualities of various organization paradigms

Paradigm	Key characteristic	Benefits	Drawbacks
Hierarchy	Decomposition	Maps to many common domains; handles scale well	Potentially brittle; can lead to bottlenecks or delays
Holarchy	Decomposition with autonomy	Exploit autonomy of functional units	Must organize holons; lack of predictable performance
Coalition	Dynamic, goal-directed	Exploit strength in numbers	Short-term benefits may not outweigh organization construction costs
Team	Group level cohesion	Address larger grained problems; task-centric	Increased communication
Congregation	Long-lived, utility-directed	Facilitates agent discovery	Sets may be overly restrictive
Society	Open system	Public services; well-defined conventions	Potentially complex, agents may require additional society-related capabilities
Federation	Middle agents	Matchmaking, brokering, translation services; facilitates dynamic agent pool	Intermediaries become bottlenecks
Market	Competition through pricing	Good at allocation; increased utility through centralization; increased fairness through bidding	Potential for collusion, malicious behavior; allocation decision complexity can be high
Matrix	Multiple managers	Resource sharing; multiply influenced agents	Potential for conflicts; need for increased agent sophistication
Compound	Concurrent organizations	Exploit benefits of several organizational styles	Increased sophistication; drawbacks of several organizational styles

Fig. 2.2. Organization Paradigms Comparison

the ones below. The less complicated example of a hierarchy is a single root having one or several branches. A ***holon*** is a self-conformed group of agents with very similar yet not identical characteristics as it was firstly conceived by [27]. The structure of each of these groups is a basic unit of organization that can be seen throughout the system as a whole. ***Coalitions*** in general are goal directed and short lived; they are formed with a purpose in mind and dissolve when that need no longer exists, the coalition ceases to suit its designed purpose, or critical mass is lost as agents depart. Klusch, Gerber, and Intelligence [28] explain that normally, there may not be any hierarchy within coalitions, however, there may exist one agent that represents and acts as the complete group. A number of agents that previously agreed to work together to solve a common goal is called a ***team*** by [29]. Agents give preference on boosting team work productivity which reduces their objectives to satisfy their own demands. Teamed discussions and knowledge are implemented in teams providing resilience and robustness to this organization model. The teams cohesion is derived primarily from the joint intentions created as part of executing the team plans. ***Congregations***, for [30], are integrated by individual agents that have a combination of their own beliefs, convictions, capabilities, and desires which are shared within a specific group. Dignum, Meyer, and Weigand [31] make an analogy between ***societies*** being the virtual counterpart of real-life societies. Agents in societies have free will to form part of this organization paradigm as they follow different patterns of knowledge, beliefs, interests and such. Social agents must follow some rules that are defined by an agent or group of agents that enforce social laws, norms or conventions. ***Federation*** agents should be able to harmonize contradictory and uniform pursuits which regularly can lead to collective disjunctive actions as viewed from a political stand point [32]. In federations, group members interact only with one agent that represents the whole group, often called the facilitator, mediator or broker. Kiani et al. in [33], argue the efficiency of a central broker where, depending on the organization performing a combination of interfaces and decisions between the federation, it represents other intermediaries. A set of buyers and sellers or third parties called auctioneers are the agents that interact in a ***market-oriented*** organization. The market-place competition among agents it is typically propitiated where only the most aggressive in terms of the transaction are the ones that get better results. Kang [34] mentioned that much research has focused up to now on optimizing customer demands satisfaction at enterprise-level design. Drawbacks to market-based organizations are complexity and security. ***Matrix*** organizations encourage the multiple management to one or many agents. Matrix organizations are very similar to human existence, where a person may receive instructions or suggestions by many other people or entities. Levinthal and

Workiewicz in [35] suggest that during the early twentieth century, multiple companies realized that centralized management reduced the ability of the units to answer to specific environment needs. Horling and Lesser in [24] defined the organization paradigms classification presenting the ***compound organizations***. A combination of organization paradigms risen up from the reality that organizations may not fit entirely in a specific paradigm but in a combination of some of them. If an agent takes different roles which require working in a different organization paradigm, it might complicate the problem. A detriment of accuracy to fulfill two roles can be noticed for a broker of a federation which also plays as a role in a different organization. However if the agent covers more information of both organizations, it may lead to more effective solutions.

AGR and AGRE models

An extension of a previous model called AGR (agent, group, and role) is presented by [36]. The new model aggregates to the new component called environment to AGR. The ***Agent*** is the concept that corresponds to any active entity playing roles within groups. Agents instantiation can range between a reactive or a clever that also can hold multiple roles in different groups. A set of agents that interact around a shared objective is called a ***group***. Agents in the same group are the only ones able to communicate with each other. A role is a set of characteristics and capabilities that agents can be divided into. Although, for an agent to play a role there must exist a request for it before instantiation. ***Environment*** is an abstraction of the different domains where an agent can interact. Physical spaces can be called areas and operations can be executed by agents only through modes.

MOISE model

Hannoun et al. in [37] presented an organizational model to control agents that is able to work with the two different methodologies of ACMAS and OCMAS. The

MOISE model is based in three different levels: responsibilities (individual level), aggregation (agency level), and global structuring and interconnection (society level). The MOISE organizational model was viewed as a normative set of rules that constrains the agent's behaviors.

OMACS model

Deloach, Oyenan, and Matson [38] developed OMACS as a framework for constructing complex, distributed systems than can autonomously adapt to their environment. In other words, a system that will be able to auto-organize itself during run-time. OMACS defines the requisite knowledge of a system's organizational structure and capabilities that will allow the system to reorganize during run-time. Typical OCMAS work accordingly to other approaches where goals (G), roles (R), and agents (A) are used. In OMACS four more entities are added: capabilities (C), assignments (Φ), policies (P), and a domain model (Σ). Capabilities are central to the process of determining which agents can play which roles and how well they can play them. Policies constrain the assignment of agents to roles thus controlling the allowable states of the organization. The domain model is a critical component that defines the ontology used to define behavioral policies and to allow agents to communicate effectively.

System of Systems Engineering

DeLaurentis, Crossley, and Mane in [39] refer to a system of systems as the one that is used when a specific goal is accomplished by several systemss; such systems need to operate autonomously and at the same time effectively interact with other systems. Furthermore, a system of systems methodology may fit in both agent or organizational paradigms. An analysis of a ROPE table or resources, operations, economics and policy contrasted with the different levels of organization is an important step in this methodology. Gomez et al. in [40] discuss an implementation of a Collaborative Air

Autonomous (CAA) SoS design approach. The final goal of the authors was to find an effective network configuration to increase robustness with limited number of existing agents.

Other models

Matson and Min in [41] defined an infrastructure called HARMS to integrate humans, agents, and robots into collectives to accomplish large-scale goals. They defined five different layers where each of these layers provide services to the top layers. The layers correspond to network, communication, interaction, organization, and collective intelligence. The novelty of this model is the indistinguishably goal to let humans and machines interact in a smooth way. Up to now, the organization layer hasn't been implemented yet.

2.2 Autonomous Behavior

Authors such as Arkin in [42], believe that behavioral or intelligent robots can be studied through two different system spectrums. On one hand, reactive or reflexive systems require low-level intelligence and simple computation. Hence, it leads to an instant responsive action. On the other hand, the other spectrum, deliberative or purely symbolic, is based on a representation which implies a high level of intelligence (cognitive). Such intelligence, leads to slow and very studied answers. Meticulous response and studied adaptation take latency and time dimension into consideration to make decisions. This work is focused on the second spectrum where the agents or the system as a whole execute actions to avoid disruption of service.

According to Geffner in [43], the basic problem of autonomous behavior is determining "what actions to take next". The same author continues stating that there are three approaches in AI for next action studies.

- First, programming-based approach where the programmer assigns directly by coding the next action

- Second, learning-based approach that is induced by the experience, let this be by itself, by other "instructors" agents or reinforced

- Third, the model-based approach is derived from a model of the actions, sensors, and goals. The latter approach is well related to the study of this research effort in such a way that models represent the future actions of the system.

Autonomous behavior, mentioned by Psaier and Dustdar in [44], should be able to operate and administer the whole system based on taking the adequate actions. The same authors continue stating that both trends of autonomic computing and self-adaptive agree on a combination of awareness of internal and external states allow proper adaptation. The self-* concept was first coined in executive Tek Reports as stated in [45], [44]. The Paradigm of self-* includes four self properties tied to self-managing systems: Self-configuring, self-healing, self-optimization, and self-protection.

2.2.1 Self-adaptive / Self-protection / Self-healing

Ghosh et al. in [46] define a self-healing system to the ones that have capabilities to detect a non expected situation. Detection is not enough up to now, also detecting should be done in running-time. This classification of systems is also considered as recovery oriented computing. Continuous availability is the driver for enhancing some systems with self-healing mechanism. Depicted in Figure 2.3 can be seen the relations and properties that were defined for self-healing [44].

Huebscher and McCann in [45] defined IBM's MAPE-K autonomic loop. MAPE-K contains the functions of monitor, analyze, plan, execute and knowledge. Detecting, diagnosing and recovery are the stages identified. Policies can be action, goal, or utility functions.

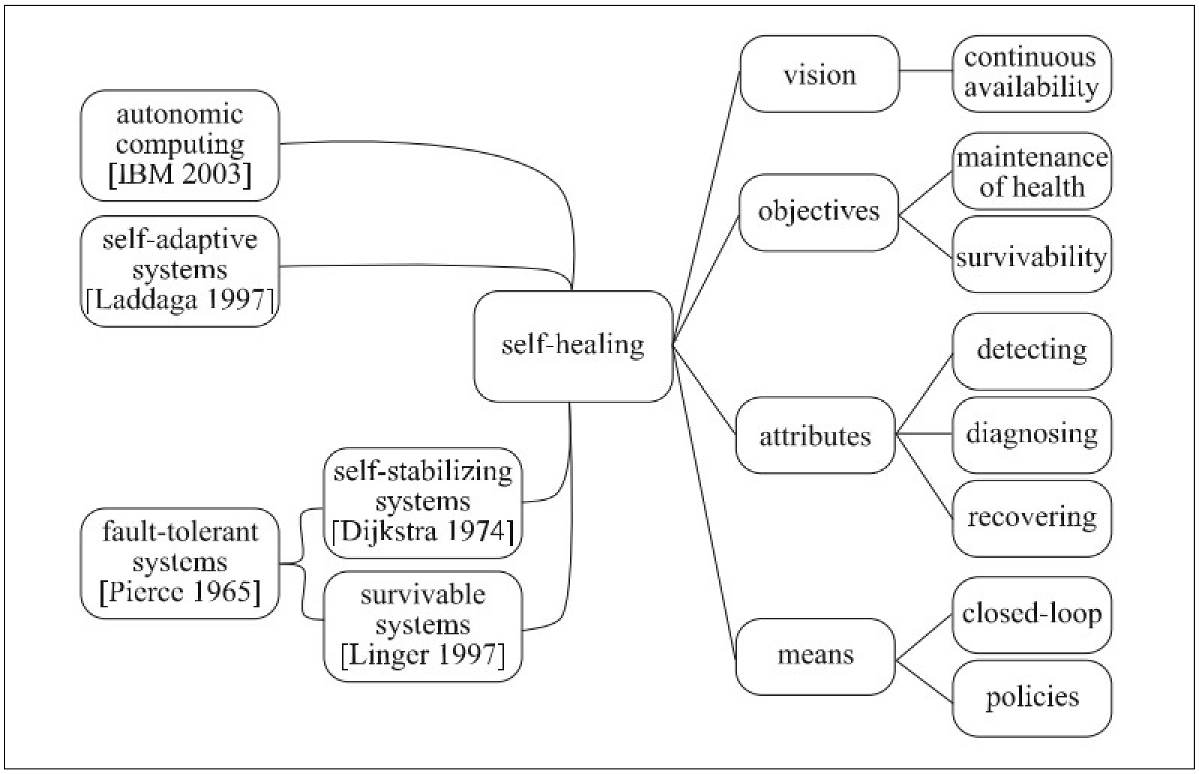

autonomic computing [IBM 2003]
self-adaptive systems [Laddaga 1997]
self-healing
fault-tolerant systems [Pierce 1965]
self-stabilizing systems [Dijkstra 1974]
survivable systems [Linger 1997]
vision
continuous availability
objectives
maintenance of health
survivability
attributes
detecting
diagnosing
recovering
means
closed-loop
policies

Fig. 2.3. Relations and Properties on Self-healing Research

2.2.2 Verification

One of the most reliable methods to perform detection on the stages mentioned in self-healing approaches, is formal verification. Verification techniques arose when the need of exhaustive software and hardware testing became impractical. It is considered applied mathematics for modeling and analyzing Information and Communication Technology (ICT) systems.

It is well known that catching software or system errors in early stages is more beneficial for the accomplishment of the goals. Camurati, Prinetto, and Torino [47] determined that in formal verification at design stage the designer specifies what the system under development should do and how it should do it. What the system should do is called its *specification*, while any one of the possible devices that realizes the specification is called an *implementation*. The same authors continue stating that verification is mostly done using two different classes: safety and liveness properties. Safety properties are related to assure that wrong situations should be always avoided. Liveness properties concerned with the possibility of correct status will occur in the future.

Verification techniques can be grouped into three: theorem proving, model checking, and testing [48]. Theorem proving, which allows showing correctness of programs similarly as a proof in mathematics shows correctness of assumptions. Hence, it is manual and requires too much time. Model checking is an automatic approach usually applied to finite state machines or automata. Testing is more precise and diverse, but developed for specific purposes, which some other authors call monitoring tools.

2.2.3 Model Checking

For Baier and Katoen in [49] model checking is a formal verification technique which allows for desired behavioral properties of a given system to be verified on the basis of a suitable model of the system through systematic inspection of all states of the model. It bases its results in a model of the system and a specification that will be verified if either it is satisfied or not.

The same authors continue indicating that the process for model-checking consists of three phases: The *modeling phase* which requires a model of the system and a formal characterization of the property to be checked. Models of systems describe the behavior of systems and are mostly expressed using finite-state automata, consisting of a finite set of states and a set of transitions. In model checking temporal logic is used as a property specification language of system properties. Functional correctness, reach-ability, safety, and real-time are properties used in temporal logic. The *running phase* starts setting up the correct initializing values for the options and directives, followed by an exhaustive revision of all the states of the system model. The *analysis phase* which can have three possible outcomes: success, failure of the model or the computer ran out of resources to execute the complete exercise.

Model checking is based on temporal logic that was devised by philosophers for making statements about changes in time [50]. A wide variety of applications have appeared as model checkers such as: SPIN [51] , NuSMV [2] , PRISM [52], [53] , DiVinE [54] , and many others.

Recently, the study of model checkers grew very fast as it is stated in the benchmark done by [55]. Out of 80 papers that were found in the two decades period of 1994–2006, only 3 papers were categorized as exhaustive study.

In terms of model checking multi-agent systems, the majority of efforts are focused on design stages verification in order to reduce costly failures. For instance, Mireslami in [56] present the use of Agent UML (AUML) methodology for MAS design as an extention of Unified Modeling Language (UML). The author uses Multi-agent Software Engineering (MaSE) methodology based on UML diagrams to generate message sequence charts, message sequence graphs and high-level message message sequence charts. those charts and graphs can lead to convert them into a models able to be verified using model checking techniques. This document is focused more in the survivability feature that takes place in the moment when the system is running and have no human activity at all. In other words, the specific moments where the autonomous multi-agent systems need to apply survivability features to overcome internal errors or external variables to accomplish the initial final goals.

2.2.4 Online Detection and Testing

Efforts for research on online detection or verification have not been around very long. Reactive adaptation results in an inefficient solution given that it may not be the best action that can be taken. They tend to be the kind of instant and myopic reflection where a lack of real adaptation happens due to not foreseeing what the next adaptations will be. In this section it will be mentioned some of the efforts in a non reactive adaptation.

Leucker and Schallhart [48] define that one of the main distinguishing features of run-time verification is due to its nature of being performed at run-time. It opens up the possibility to act whenever incorrect behavior of a software system is detected. The same authors make a comparison between run-time verification versus testing. They stated that run-time verification does not consider each possible execution of a system, but just a single or finite subset, it shares similarities with test, which terms a variety of usually incomplete verification techniques. Testing considers a finite set of finite input-output sequences forming a test suite. Test-case execution is then checking whether the output of a system agrees with the predicted one, when giving the input sequence to the system under test. In that study three different approaches were mentioned to react at run-time: FDIR, run-time reflection, monitor-based programming. Although recent studies talk about the model-based approach that will be added at the end of this section.

Fault Detection, Identification and Recovery

FDIR was presented in [57]. FDIR sometimes also known by Fault Diagnosis, Isolation, and Recovery. The general idea of FDIR is that a failure within a system shows up as a fault. A fault, however, does typically not identify the failure. Authors used Reiter's theory of diagnosis from first principles. Specifically, the detection of errors is performed by diagnosis techniques leading to reconfiguration that switch the server to work with the old version of the protocol.

Run-time reflection

The concept of short run-time reflection (RR) developed in [58], is an architecture pattern for the development of reliable systems. The main idea is that a monitoring layer is enriched with a diagnosis layer and a subsequent mitigation layer. The simple architecture of the run-time reflection framework comprises four layers: *Logging* which will be in charge of observing the system events and to provide them in a suitabe format for the monitoring layer. The *monitoring* layer will include a number of monitors to observe the information received from logging layer. *Diagnosis* that will make a differentiation between detection of faults and identification of failures. *Mitigation* happens to reconfigure the system to mitigate the failure, if possible.

Monitor-oriented programming

As proposed by Chen and Rou in [59] monitor-oriented programming is a software development methodology, in which the developer specifies desired properties using a variety of (freely definable) specification formalisms, along with code to execute when properties are violated or validated.

A taxonomy to analyze and differentiate recent developments in run-time software fault-monitoring approaches is presented by [60]. The taxonomy categorizes the various run-time monitoring research by classifying the elements that are considered essential for building a monitoring system. The same authors continue stating that the most important parts of the taxonomy are: specification language, monitor, event handler, and operational issues. Figure 2.4 shows the high-level view of a run-time monitor.

Specification language comprehends: language type, abstraction level, and property type. The language type to specify the properties, based in algebra, automata, or logic. The abstraction level can be domain-based, design-based, or implementation-based. There are two property types: safety property to express something bad never occurs and other temporal which refers to progress and bounded liveness including timing properties.

A *monitor* evaluates and inspects the state of the system. A comparison with the desired status to the one observed takes place in order to detect a possible failure. According to the taxonomy shown in [60], the summarized characteristics of the monitor are monitoring points, placement, platform, and implementation.

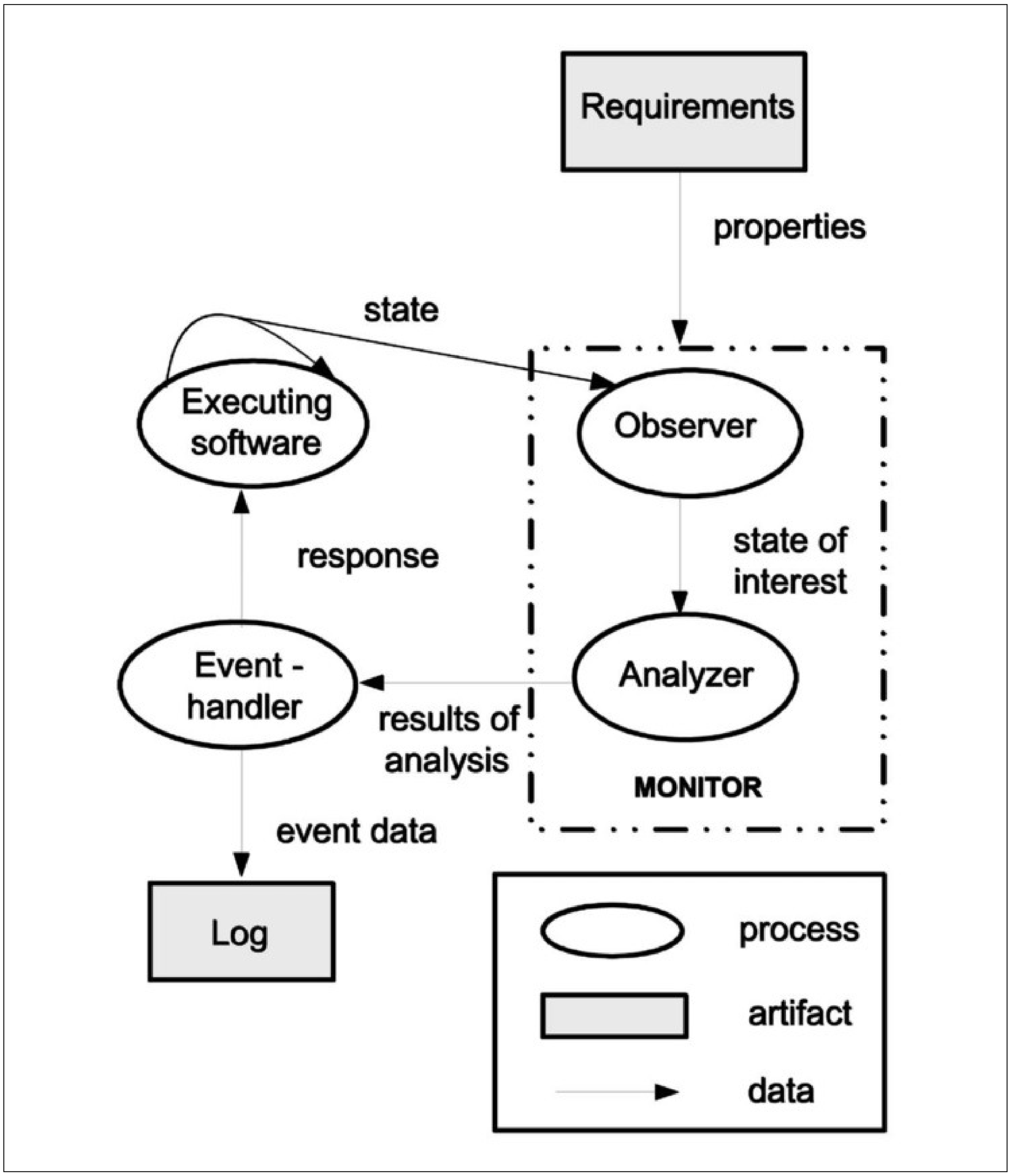


Fig. 2.4. High-level view of a run-time monitor

The *placement* refers to where the monitoring code is executed. The inline check is when the monitoring takes place embedded in the target code. Offline is when the monitor executes as a separate thread or process, even in different machine. Depending on if the application has to wait for the analyzer to execute or not is offline (asynchronous) or offline (synchronous).

The *platform* refers to if it is software or hardware. *Implementation* can be in three different ways: single process when it is executed along or inside the target program, multiprogramming for when each of the programs (monitor and target) run as different processes or threads, multiprocessor just for different processors.

Event-handler is the one in charge of taking action right after the monitor detects something. Response actions can alter the application state space, report application behavior, or start up another process. As the control level can be specified by the user or the monitors that react universally. The response effect reflects the extent to which the monitor's response to a violation can affect program behaviors.

Operational issues include program type, dependencies, and level of maturity.

Model-based testing

The goal of testing is failure detection. Accomplishment of the final requirements are led by the instant comparison between the status and performance of the implementation and the intended behavior [61]. Model-based testing, continued the same authors, is an alternative for testing based on explicit behavior models. Those behavior models abstract the desired paths of the system and its environment. In Figure 2.5 is depicted a general process of model-based testing in [61].

New trends on the run-time verification was developed by [62]. The verification system was first intended especially for self-optimizing component based real-time systems where self-optimization is performed by dynamically exchanging components. It was also first offered as service of a real-time operating system (RTOS) as a novel on-line model checking approach. The same authors were not able to measure what the performance of the run-time verification will look like. However, they based their

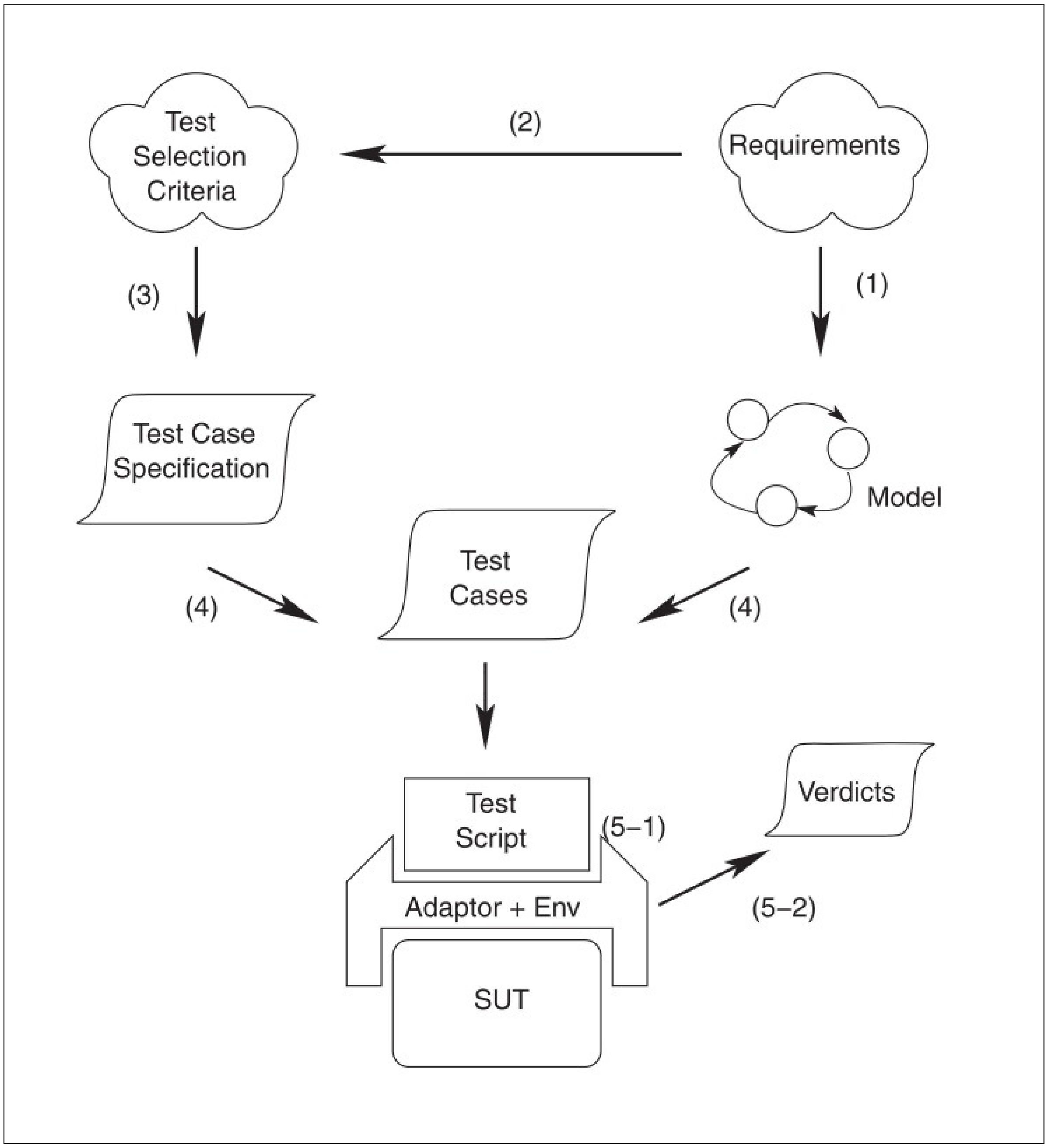


Fig. 2.5. Process of Model-Based Testing

assumptions on the thought of experience demonstrating that the properties to be checked in practice are usually not very complex.

Later, Zhao and Rammig in [63] extended their previous work stating that model-based run-time verification is an extension to the state-of-the-art run-time verification based in model driven engineering presented in [64]. One goal of the model to check consistency comparing during execution the system implementation against the system model. The second goal is safety checking where the system model is compared to the system specification. Because the model runs while the system is running as well it allows to go further than the current space of the real system. Monitoring states in compliance to the model or deleting branches of the model when the real system gets to specific states that may be already verified. One disadvantage can be the cost of flexibility, which is the computational complexity of the model which is less than the offline model checking but greater than the state-of-the-art run-time verification.

Calinescu et al. in [65] presented a new approach that includes model checking in complete harmony with quantitative verification. Quantitative verification is defined there as a technique that is based in mathematics to evaluate correctness, performance, an reliability of systems exhibiting stochastic behavior. There, the combination of model checking and quantitative verification is recommended to be used during execution to foresee and spot critical system errors and being able to plan in advance the steps to prevent or recover from those errors. A finite mathematical model is delineated and evaluated on how well the system requirements are met. The difference with a normal model checking fashion is that on top of the requirements expressed in formally temporal logistics, they are also complemented with probabilities and costs/rewards. Requirements also include external factors that may affect the system like probabilities that faults occur while the system is running including expected response time. This approach stresses the use of discrete-time Markov chains, or DTMCs, to express specification S and domain assumptions D, and probabilistic computation tree logic, or PCTL, to formalize the requirements R.

Filieri and Tamburrelli in [66] define an approach very similar to the one mentioned in the previous paragraph. They address the problem of online verification focusing on probabilistic run-time model checking where requirements expressed in logical expressions are used to verify reliable models in terms of Discrete Time Markov Chains. This is achieved using probabilistic model checking at run-time exploring and comparing all possible paths. The possible paths are grouped in two different algorithms: algebra-based and state elimination.

Moreno et al. in In [67] presented a proactive latency-aware system that takes emergent behavior using probabilistic model checking for determining the following decisions. Non-determinism is a key concept as the model is based on a underspecified decision. Hence, the model checker will find the solution to the non-deterministic choices so that the accumulated utility over the horizon is maximized.

Kim et al. in [68] and Gomez, Kim, and Matson in [69] make a study of a humanoid robot in a soccer environment. In [68] model checking is applied to a humanoid soccer player to verify that no inappropriate states are reached. A finite State Machine is used as model for the system. Hence, it can be validated using NuSMV. While [69] as a complement for the previous study presents the way that a humanoid learns in an iterative way how to intercept a ball on the same environment.

2.2.5 Self-adaptation for MAS

Verification for multi-agent systems is more complicated compared to the stand-alone solutions presented in this paper up to the previous section. The number of variables to verify increases exponentially when the view of the agent becomes social. In this section it is going to be presented a brief description of recent advances in terms of disconnected and online verification for MAS.

Common approaches for self-adaptation and model checking are model based such as the ones presented in [70], [71], [72], [73], [74] .

Lomuscio and Raimondi in [70] presented MCMAS or model checker for MAS. It allows the verification of specifications of common temporal; further more using epistemic, correctness, and cooperation modalities. Interpreted systems programming language (ISPL) is used to represent agents means. This approach is based on ordered binary decision diagrams (OBDDs).

[71] presented a multi-layer architecture approach to build self-aware and self-adaptive robotic multi-agent systems. It includes domain-specific meta-level component types and higher-order meta-level layering as improvements to meta-level components.

Persuasion between agents is what happens when an agent is influenced by surrounding agents to perform an action or decision. Exactly this behavior is the one that is verified by [72]. They used model checking fondness to introduce Perseus. Perseus is a model checker designed to verify satisfaction of AG_n language which describe properties of persuasion in a given model.

Calinescu et al. in [65] discuss the need of quantitative verification at run-time. Authors there base their self-adaptation in the continuous monitoring and projection of vital variables in order to adapt to the needs while the program is running. 2.6 shows the solution proposed in that work, where four basic steps take place: monitor, analyse, plan, and execute. Their ultimate goal is to help identifying, and, sometimes, predicting requirement violations resulting in a software supporting automated changes to meet requirements even if situations evolve.

In Figure 2.7, the authors presented an architectural approach that integrates MAS and self-adaptation(SA). There, the *agent behavior* contains all functionality concerning inside the agent. The *coordination module* is in charge of the coordination between agents. The *self-adaptation module* to address changing conditions in the system of its environment [73].

(Elakehal et al. in [74] present a software methodology called Self-managing Multi Agent Systems (MSMAS). It uses norms to capture the system specifications in a formal representation that can be verified either in advance or at run-time. Norms mentioned before can be system goals, system roles, the business activities, and communications.

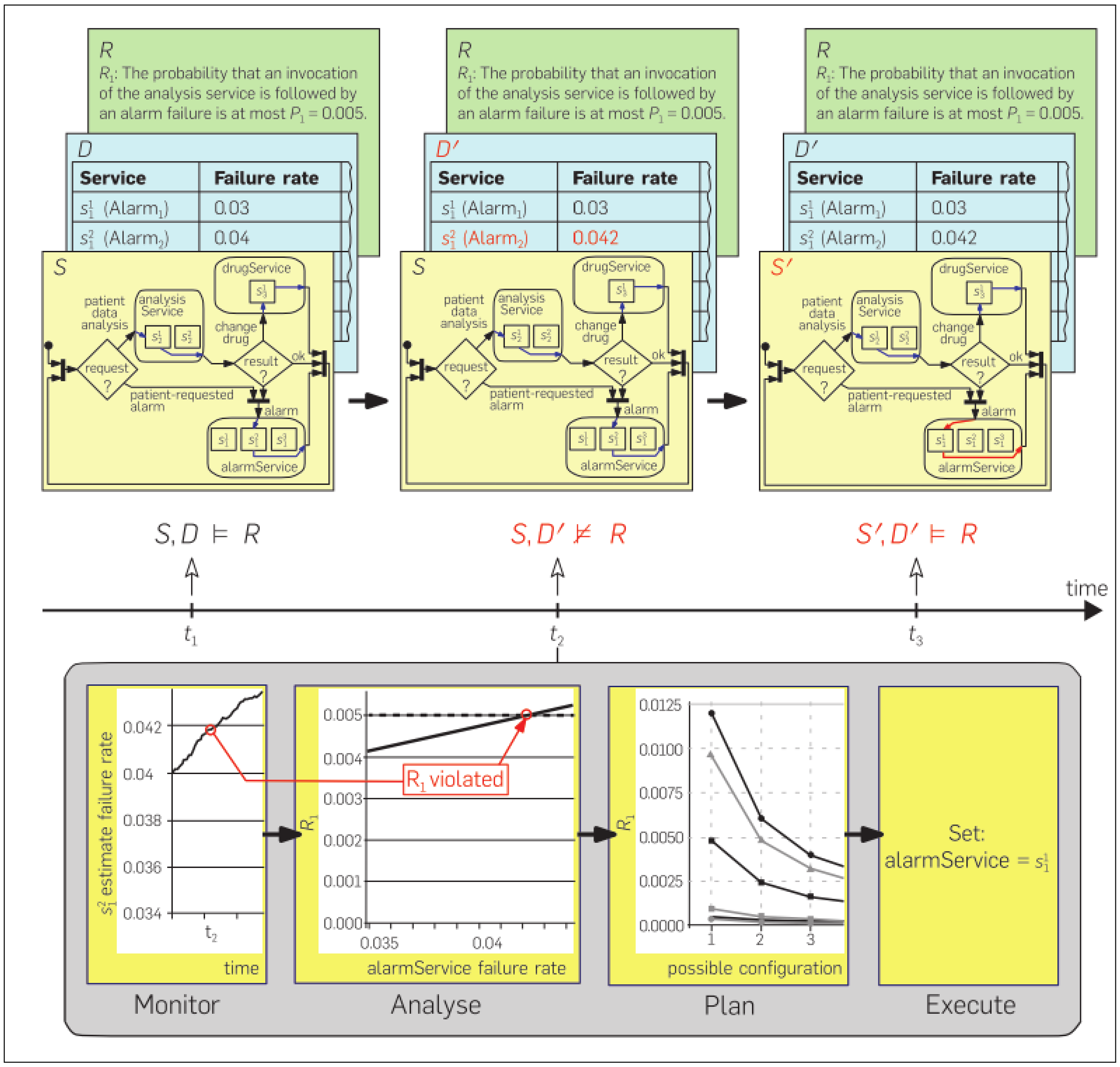


Fig. 2.6. Self-adaptive software solution

2.2.6 State Explosion Problem

Model checking, briefly explained in the previous section, relies on an exhaustive formal verification technique. Within this approach, it is important to verify all possible cases in the generated mathematical models.

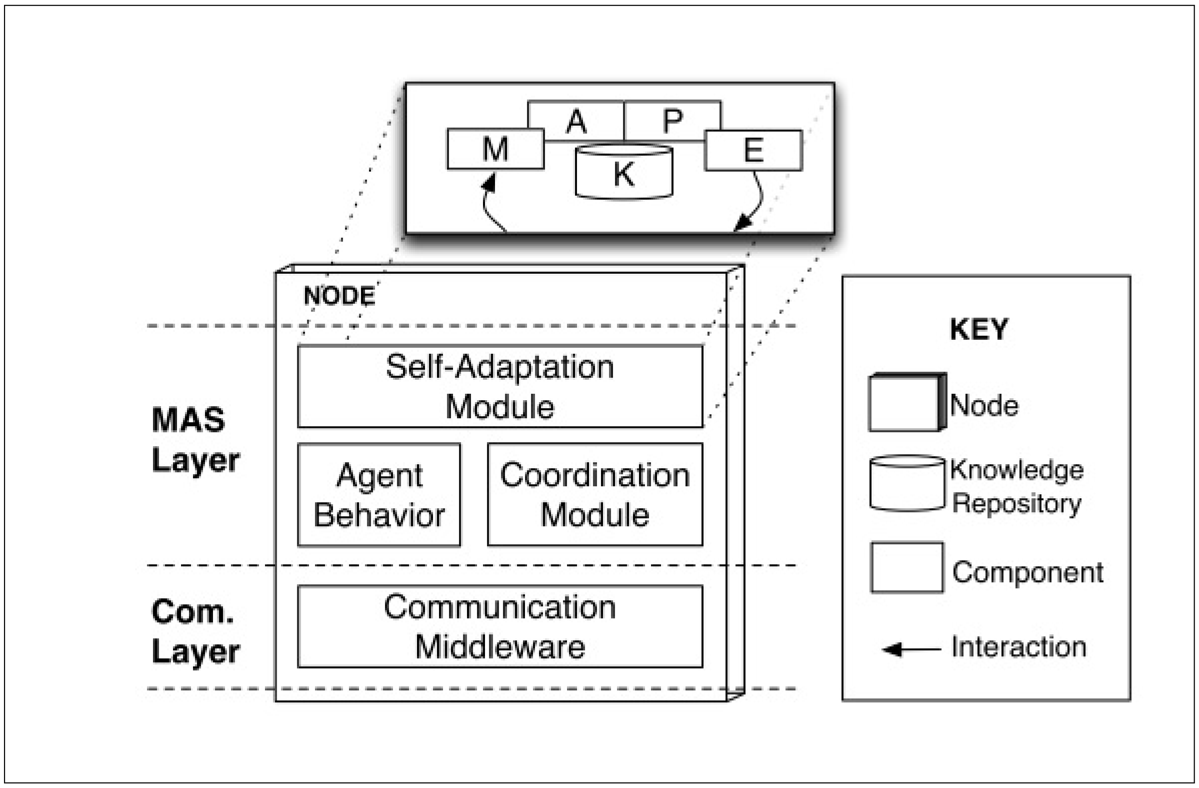


Fig. 2.7. Reference Model for SA-MAS

Prior research such as [75], [76], [77], [78], [79], [80], have argued that one of the problems of model checking is that resources and time needed to run the verification of a model increase in the same proportion to an exponential curve based on the number of states in a model.

2.3 State Explosion Solutions

Different authors have defined several advances in order to reduce the state explosion problem:

2.3.1 Symbolic Model Checking

This approach uses Binary Decision Diagrams (OBDDs) to represent the states instead of listing individually each state. It uses a fixed point algorithm which normally reduces the size in an exponential way. Some examples of solutions like this are proposed in [81], [82], and [83]. The latter example refers to a procedure called MCMT which symbolically computes pre-images of the set of unsafe states in terms of safety by a specific group of states generating partial Satisfiabilty Modulo Theories (SMT).

2.3.2 Partial Order Reduction

This approach is based on the asynchronous systems composition of processes exploiting the independence of actions. A recent study [84] supports the solution on online dynamic tracking interactions between concurrent processes/threads. The tracking information is exploited to generate a new partial-order reduction algorithm. The algorithm then pinpoints backtracking paths where alternative tracks in the state space need to be explored.

2.3.3 Counterexample-guided Abstraction Refinement

An appropriate level of refinements is the goal of this approach, where the property of interest is still possible to be validated but the details that only are abstracted add up to the delay of evaluation time. Clarke et al. in [85] describe an iterative and automatic refinement methodology. An initial abstract model is generated following an automatic analysis of the controls structures in the program to be verified. However, those abstract models may admit erroneous counterexamples that are evaluated to refine the final abstract model.

2.3.4 Bounded Model Checking

BMC is the most used model checking methodology in the industrial environment as of today. The technique is used to find errors in a finite state system using LTL. The method verifies if a propositional formula is considered true only if it can be disproved by a counterexample of length k. Depending on the result, if is true, it is stored as a boolean satisfiability (SAT) solver. If no counterexample of length k is found then the value k is increased and the process is repeated. [86] and [87] are examples of BMC where they follow the basic idea of restricting the model checking problem to a bounded problem. The verification changes then in terms of question. The new question now is if there is a counterexample with a specific length, instead of if the system violates a property.

2.3.5 Cloud Computing

Several approaches have appeared that have taken advantage of cloud computing and big data towards a favorable result in terms of model checking problems. These solutions are considered in more detail in the next section.

2.4 Cloud Computing Strategies

The basic idea of cloud computing is an economic principle summarized by the well-known statement "*pay as you go*". As an example, when a company is in need to execute a batch-oriented task there is a positive difference of cost between using 1,000 servers for an hour than using one server for 1,000 hours. The benefit in terms of cost is plausible, however, what is important here is the reduction of time that can be gained in a distributed or parallel processing manner.

The National Institute of Standards and Technology (NIST) has defined the different service models in [88] as: software as a service(SaS), platform as a service(PaS) and infrastracture as a service(IaS). As described by Armbrust et al. in [89], there is a wide difference of opinion in terms of the service models offered. Moreover, as mentioned by [90], the user interface layer of cloud computing facilitates the services to concealed XaaS or Anything as a service layers.

2.4.1 MapReduce

Functional languages like LISP were the inspiration for developing MapReduce. In the approach presented by Dean and Ghemawat in [91] a library is used to cover all the mess related to data distribution, load balancing parallelization, and fault tolerance. Fault tolerance is accomplished through the parallelization and re-execution mechanism of the operations of the map and reduce functions in a large group of computers.

The principal benefits of the Map and Reduce solution presented in [91] are a combination of an implementation of an interface which achieves high performance using a considerable easy to access PCs (or processes in the same computer) and a plainly potent interface which facilitates large-scale computations and parallelization.

Roughly speaking, this approach works using two functions, programmed by the user, where both of them receive and produce sets of key/values as shown in Figure 2.8. The *map* function receives a set of input pair and works to generate intermediate pairs. There is an iterator that is in charge of transfering the set of intermediate pairs. The same intermediate set of pairs that the *reduce* function receives to convert them into the final merged set of pairs.

Hadoop is an implementation of a distributed file system called Hadoop file system (HDFS) and MapReduce a large-scale data processing mechanism [92].

CTL Model Checking Algorithm Using MapReduce

Guo et al. in [78] propose a new solution of model checking running on a MapReduce platform. The proposed parallel algorithm is designed to compute the set of states of the model that satisfy a given CTL formula.

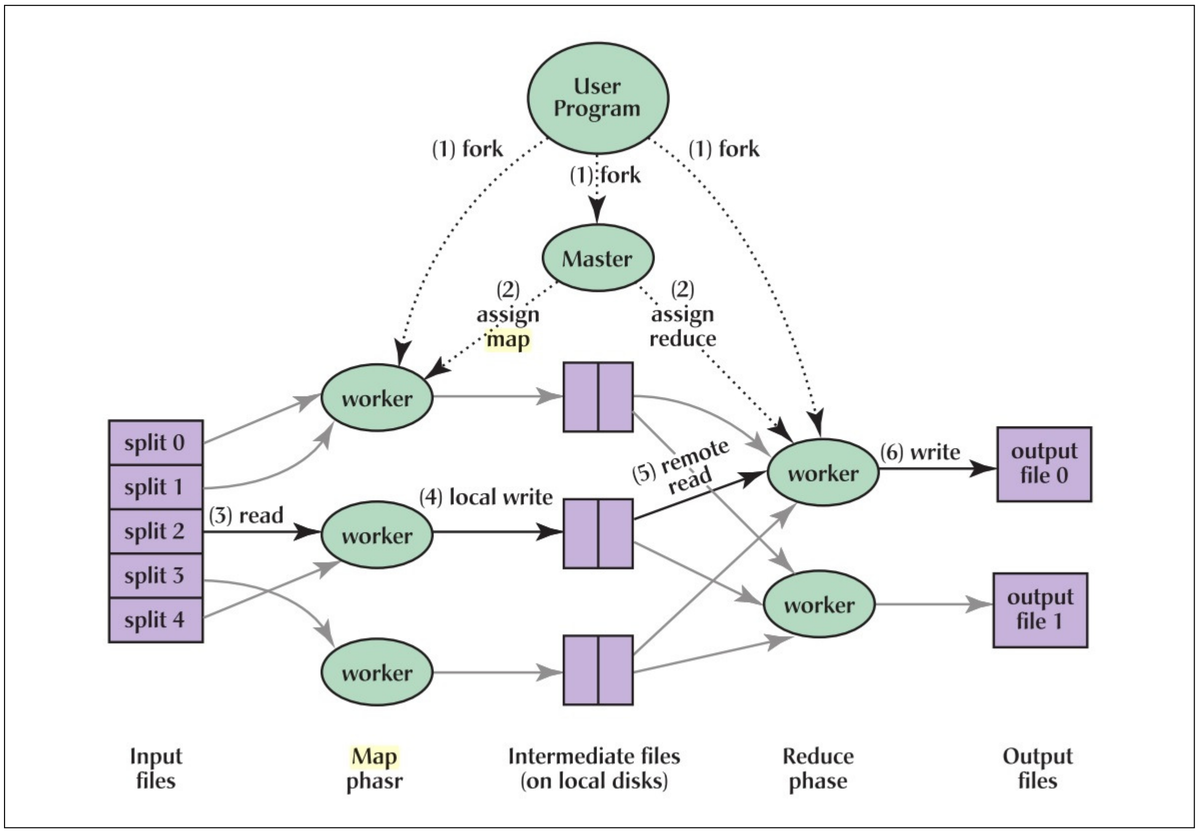
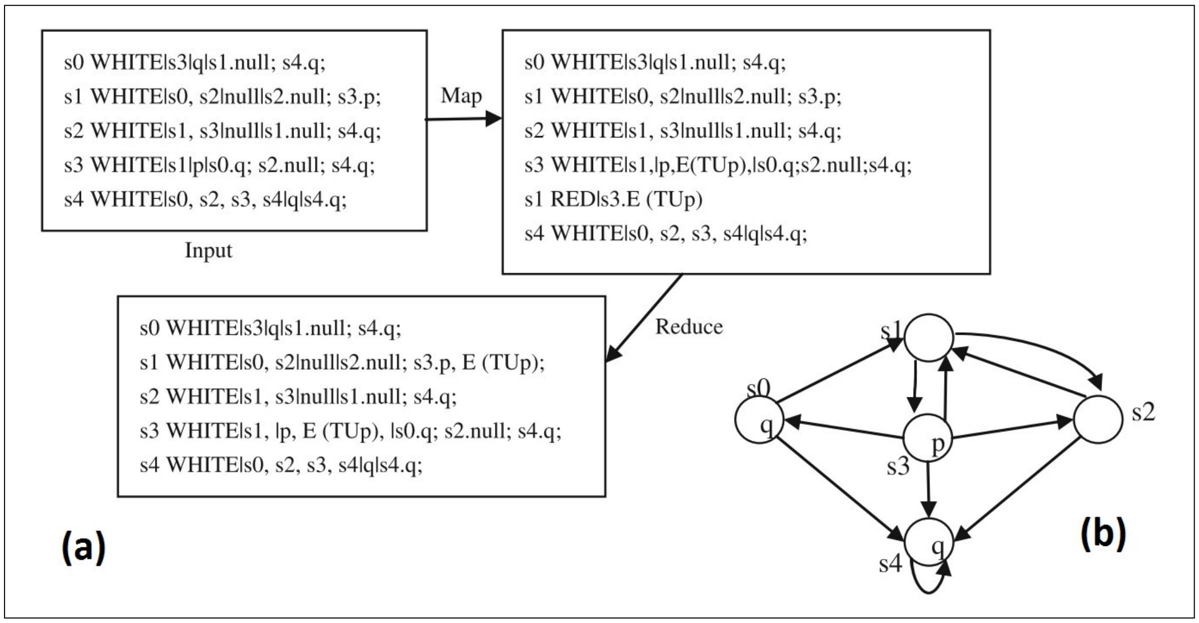


Fig. 2.8. MapReduce Execution Overview

An example is presented where a Kripke structure is defined as $M = (S, I, R, L)$. Consisting of a finite set of states S, a set of initial states $I \subseteq S$, a transition relation $R \subseteq S \times S$ and a labeling function $L : S \rightarrow 2^{AP}$. In MapReduce, the data structure is described as follows: consider the key-value pair where the key represents the state ID, and the value represents the state's information, such as its status flag, pre-successors, labels and successors' information. The formula to satisfy is $E(T \bigcup p)$ in the system shown in Figure 2.9(b). The first iteration of reduction of state explosion is shown in Figure 2.9(a). This study continues showing until the fourth iteration reduced state explosion. They analyzed the experiment and obtained that $s0, s1, s2$, and $s3$ but not $s4$ satisfy the formula. They conclude that the CTL model-checking



```
s0 WHITE|s3|q|s1.null; s4.q;
s1 WHITE|s0, s2|null|s2.null; s3.p;
s2 WHITE|s1, s3|null|s1.null; s4.q;
s3 WHITE|s1|p|s0.q; s2.null; s4.q;
s4 WHITE|s0, s2, s3, s4|q|s4.q;
```

```
s0 WHITE|s3|q|s1.null; s4.q;
s1 WHITE|s0, s2|null|s2.null; s3.p;
s2 WHITE|s1, s3|null|s1.null; s4.q;
s3 WHITE|s1,|p,E(TUp),|s0.q;s2.null;s4.q;
s1 RED|s3.E (TUp)
s4 WHITE|s0, s2, s3, s4|q|s4.q;
```

```
s0 WHITE|s3|q|s1.null; s4.q;
s1 WHITE|s0, s2|null|s2.null; s3.p, E (TUp);
s2 WHITE|s1, s3|null|s1.null; s4.q;
s3 WHITE|s1, |p, E (TUp), |s0.q; s2.null; s4.q;
s4 WHITE|s0, s2, s3, s4|q|s4.q;
```

Fig. 2.9. CTL Model Checking Algorithm Using MapReduce (a) first iterative procedure, (b) system

algorithm based on MapReduce is feasible given that the results are consistent with the previous analysis.

MaRDiGraS

Generating abstractions of the original state transition system is an approach to minimize the state explosion problem. Bellettini et al. in [93] provide MaRDiGraS as a generic library which is built on top of Hadoop MapReduce. MaRDiGraS is focused on breaking down the state explosion problem through different kind of formalisms. The benefit comes from simplifying the task that deals with very big state spaces by taking advantage of large clusters of machines. The name MaRDiGraS comes from a species of acronym of MapReduce-based DIstributed building of GraphS.

The model of Hybrid Iterative MapReduce depicted in Figure 2.10 is the one that MaRDiGraS follows. There is an initial state from where the computation starts constructing the sequential state-space until the set N of states not yet explored becomes large enough. It takes into consideration that the sequential approach is more efficient than the distributed one when the number of states is bellow a configurable threshold. After the threshold is passed, the algorithm starts running over a cluster of machines already set up in a MapReduce platform. The same authors determined that it is better setting up the threshold during run-time, depending on which number of new nodes are generated in each iteration. The *map* step is in charge of the computation of new states, while the *reduce* step identifies equivalence or inclusion relationships. The partitioner transfers the intermediate keys, meticulously checking that they belong to the same partition between the map and the reduce functions. The sequential algorithm is merged with the values of the reducers when the value of $|N|$ goes back bellow the threshold. At the end, when the set of unexplored states becomes empty, the entire state-space is stored either in a single or distributed file fashion.

The code of MaRDiGraS consists of two packages. The *data package* that comprises the state-space and the model, considered the data entities. The implementation of the algorithms of the framework of the hybrid iterative MapReduce model shown and explained above are fully contained in the *core package*. Due to lack of space, it is not possible to explain in detail each of the parts of the two packages, but for more information, please refer to [93].

The same research executed different experiments. One of those experiments was using the Amazon Elastic MapReduce on the Amazon Web Service cloud infrastructure for a benchmark real-time system model specified with Time Based Petri Nets called The Gas Burner. The MaRDiGraS based tool, executed on the input model, generates a graph with 14563 nodes (23635 states are generated during computation) reducing the time to only 39 minutes, over 8 *m2.2xlarge* machines. The reduction of time represents 80% less time than 175 minutes when executed in the same environment running in a complete sequential approach.

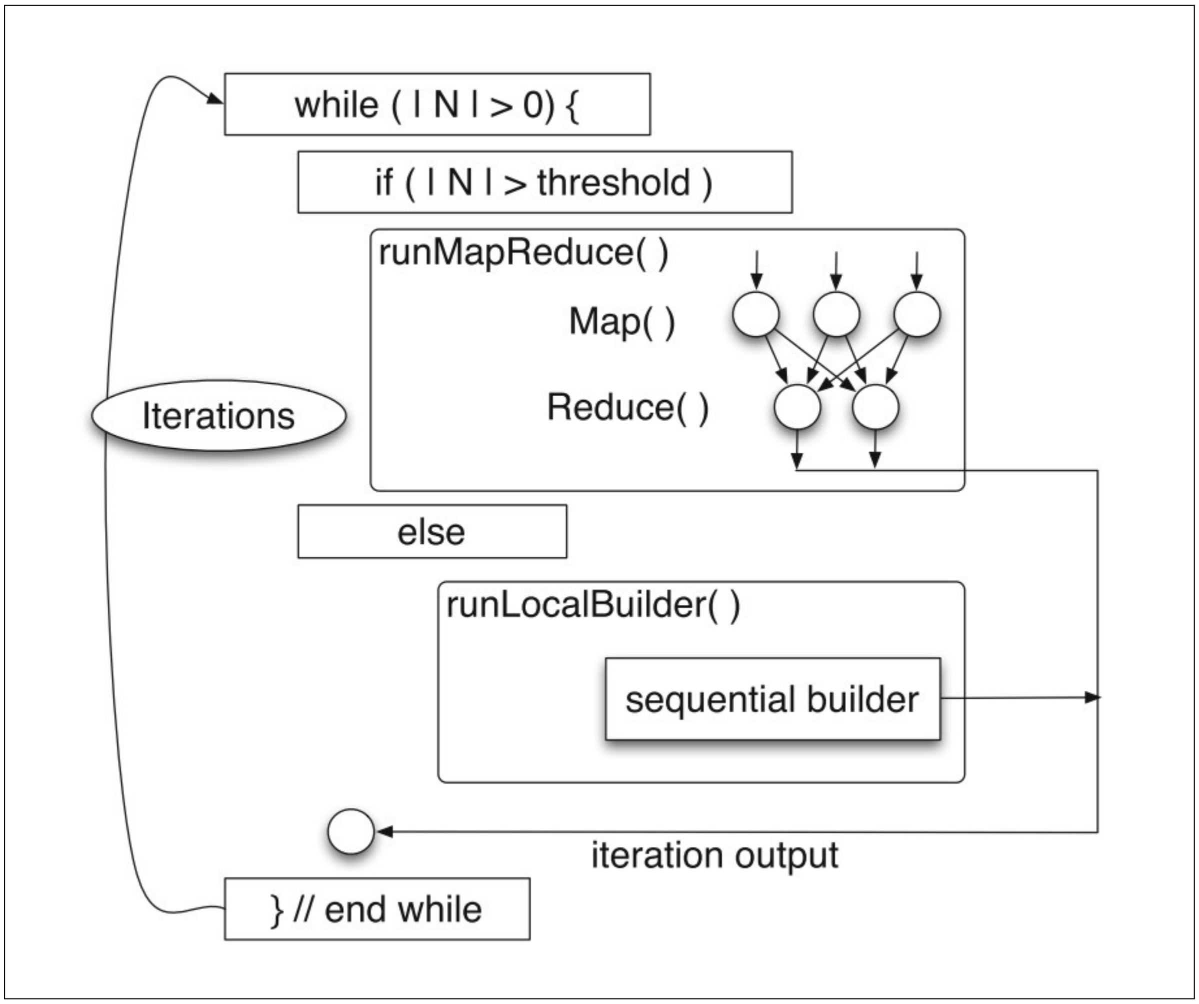


Fig. 2.10. MaRDiGraS: Hybrid Iterative MapReduce Model

2.4.2 Verification as a Service

As stated before, there are different models offered in the cloud than just the three mentioned in [88]. For instance, Mancini et al. in [94] presents SyLVaaS, a Web-based tool enabling VaaS which implements an assume-guarantee approach to perform a System Level Formal Verification (SLFV). In this case, VaaS refers to Verification as a Service, a new paradigm proposed to allow verification engineers to compute the simulation campaigns needed for their SLFV activities keeping both the system under verification (SUV) model and the property to be verified secret, thus achieving

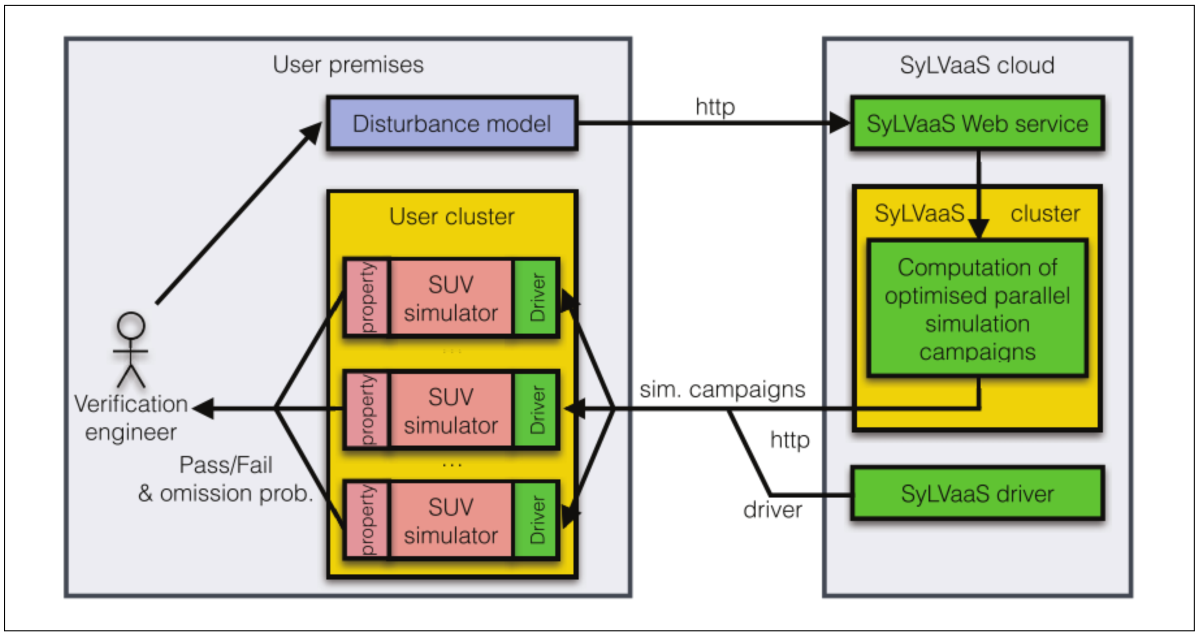


Fig. 2.11. SyLVaaS VaaS architecture

full Intellectual Property (IP) protection. According to the same authors, up to now model checkers for hybrid systems or cyber-physical systems (CPSs) cannot handle SLFV. Currently, Simulink and VisSim are the most used tools model based design which support Hardware in the Loop Simulation (HILS). According to Mancini et al. in [95] SLFV shows system correctness to meet the given specifications considering all possible scenarios. SyLVaaS is an acronym of System Level Verification as a Service and to their knowledge is the first Web-based software-as-a-service tool for HILS-based SLFV; its architecture is shown in Figure 2.11.

Faults, variation in system parameters, external inputs, and others are disturbances not easily visible events that may affect a SUV. An SUV is a deterministic system, while disturbances are used to model non deterministic behaviors. Sequences of inputs are bounded length (discussed previously), thus the problem is also approached through bounded SLFV. Counter examples are generated if errors are found during verification time. The same counter examples that would help to modify the SUV model and run again the SyLVaaS. Fast response time in SyLVaaS is achieved using a new parallel algorithm for the generation of operational scenarios from a disturbance model.

SyLVaaS requires two inputs: an integer $k > 0$ describing the number of computational cores available in each node machines and a disturbance model defining the operational environment. SyLVaaS generates k simulation campaigns, that will be executed in each of the node machines. The implemented workflow as shown in Figure 2.12(a) counts with an Orchestrator process and a number $S \in N^+$ of slaves. The Orchestrator takes care of the exploration of the state space of the disturbance generator, splitting and delegating the job in the slaves. Depth-First Search (DFS) is performed by the orchestrator up to bounded level (depth) $L < h$ and delegates the exploration of the subtrees rooted at each node at depth L to an idle slave as shown in Figure 2.12(b).

As experimental results they presented four different scenarios where due to limitations of space it will only cover the complete workflow scenario. Authors show in a table the time needed to compute the k simulation campaigns and the overall SyLVaaS response time, for two different disturbance models and each value for k. Those results were obtained using $S = 16$ slaves during trace generation and 16 cores to compute the k simulation campaigns. The results show a tendency to reduce the time of generating the simulation campaign for both disturbance models, comparing from 128 to 512 k slices. However, the overall time shows a contrast of reduction for the first disturbance model, while for the other disturbance model represents an increase of time.

For the sake of discussion, which was not presented by the authors, it was considered that is better not to increase the number of slices in all cases. Given that some of the times, to work in a parallel way will fall into the effects of what is very well-known as: "the whole is greater than the sum of the individual components". The latter expression means that there is a limit where expanding parallel processing makes a positive effect in the complete execution.

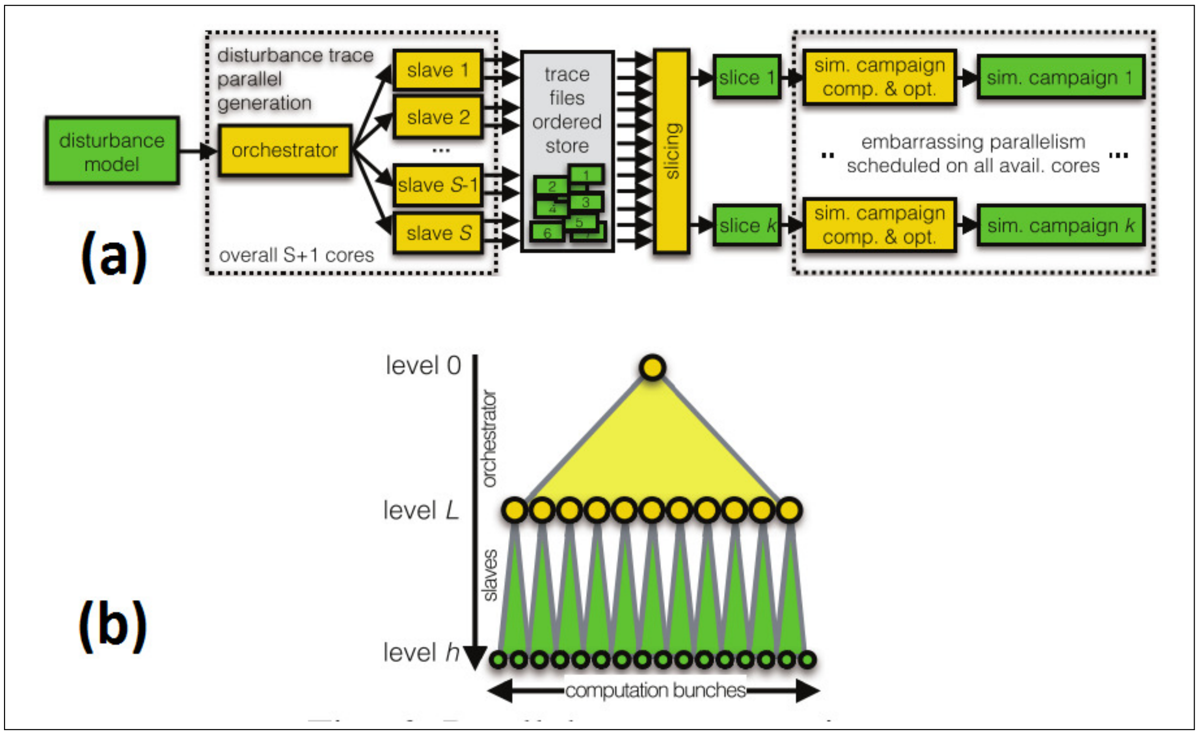


Fig. 2.12. SyLVaaS: (a) Workflow and deployment, (b) Parallel trace

2.5 Summary

In this section, it was presented the background information needed to put the reader in context with the solution that will be provided in this dissertation.

CHAPTER 3. HARMS - A MULTI-ACTOR SYSTEM MODEL

The HARMS model acronym comes from the wide range of type of actors that are expected to be able to connect through it [96]. Its name was inspired in the diversity of those actors, listed as: humans, software agents, robots, machines, and sensors. Implementations of the HARMS model have to insure the interaction will happen based in the capabilities that each actor possesses and the type of actors conforming the group in a network.

The HARMS model can be applied to different types of applications in the multi-actor spectrum. Additionally, the study of team formation from the point of view of leader or head selection is very important for such systems. Therefore, much research can be found related to that topic. Esmaeili et al. in [97] for example, proposed an algorithm for distributed leader selection. It is based on the capability and location of the actors within a network of multi-actor system. Based on the homogeneity of the actors, it is assumed that all of the can be selected as leader. The performance of the teams is also important and triggers the process. In this document instead, the process of leader selection is based in the capability model and a mesh communication between a number of agents that have the capability of leading configured, not all of them.

The HARMS model includes but does not exclude the following overall requirements:

- Self-organizing
- Adaptable
- Autonomous

- Indistinguishable actors
- Scalable
- Minimal human interaction
- Mobile

The HARMS model comprehends five different layers that are interconnected to provide the services required in order to accomplish smooth and effective interaction between actors. The five layers are listed as follows in a bottom up fashion:

- Network layer
- Communication layer
- Interaction layer
- Organization layer
- Collective Intelligence layer

What each of the layers comprehend will be extended in the next section.

3.1 Layers of HARMS

The model was divided into five different layers to provide the actors with the services needed to be able to integrate with each other. Those five layers are described in detail as follows.

3.1.1 Network Layer

The network layer represents the basic communication between the nodes of the system. Through it, each human, software agent, robot, machine or sensor, must count on basic capabilities to connect through a generic wire-bound or wireless network to connect to others. The network nodes have the ability to communicate via sending TCP/UDP messages using unicast, multicast or broadcast, depending on the message and which set of actors it is directed towards.

3.1.2 Communication Layer

Communication is the basic exchange capability between machines, agents and humans. Communication is defined by a specific syntax or a set of protocols between machines and semantics. All of these protocols are modeled in a generic sense and can be extended given a specific task domain. The HARMS model enables communication in a natural language format between all actors. The actors will exchange messages in terms of queries, imperatives and information share. Given that, the actors can send messages via unicast u, multicast m or broadcast b, they can send from $robot_a$ to any $robot_b$ . . . $robot_\infty$. Ryker et al. in [98] implemented the network and communication layers. Authors present examples proving the capability to handle increase of ubiquity in robotic systems within a controlled environment.

3.1.3 Interaction Layer

The interaction layer represents the means to let the wide diversity of actors being able to react depending on the message that is received. In other words, the actors will be able to interact with each other if they use a common "language". That language is based in the capabilities that each actor possesses. The interaction layer includes a set of techniques, algorithms and technologies that provide a layer for intelligent, rational decision making by a set of machines, agents and humans. In such kind of systems, there is an imperative of collectiveness, interest in cooperation in which there should exist negotiation and bargaining between actors in order to make decisions.

3.1.4 Organization Layer

Taking advantage of organization and multi-agent theories, HARMS model provides the form that the different actors can be assigned activities based in their indistinguishability and letting conform any of the different organization models.

3.1.5 Collective Intelligence Layer

The Collective Intelligence behavior in a collection of agents, robots and humans can lean in a number of different directions. In this case, the focus in this dissertation is on collective organizations with emergent and planned behavior. In a short study, Gomez and Matson in [99] started discussing how different level of intelligence of actors can enhance the effectiveness, and efficiency of a survivability feature in a MAS setting. In the case of that study, HARMS is the basis of all the assumptions where the collective intelligence could be achieved derived by the "level of intelligence" of each of the actors that conform the system.

3.2 HARMS model implementations

Given its intrinsic versatility of the HARMS model, it has been referenced and implemented in contexts such as: enabling features of multi-actor systems, human-robot interaction, ambient intelligence environments, and safety applications, to mention a few.

In this section it will be covered all the documented implementations of the HARMS model as an attempt to bring the reader to a broader level of knowledge of what can be done while implementing HARMS model.

3.2.1 Enabling features of multi-actor systems

The initial declaration of the model was presented by [41]. In that document, authors focused on defining what an effective infrastructure must posses to allow any combination of machines and humans to interact. The five layers of HARMS were defined in the same paper. Motivating scenarios were presented where machine to machine interaction takes place for intensive computation; while the man to machine interaction happens mostly for human in the loop cases.

One case of this is the paper where authors focused on implementing HARMS-based indistinguishability property in ubiquitous robot organizations [96]. Basically, the efforts in this paper relate directly to the implementation of HARMS where a specific instruction as a command was given to a group of actors in a speech format. They implemented a basic process to parse the string in such a way that they evaluated the accuracy of the robots understanding the instruction either with a blue-tooth headset or an internal microphone device. Their findings showed that using the blue-tooth headset reduced the ambient noise. Hence, a more effective identification of the command was more possible when using an external microphone rather than the one present within the internal parts of the robots.

When allowing digital interaction within heterogeneous actors, there exists the possibility where malicious actions could occur in any moment. DeWees in [100] defined the first steps on considering security in the communication layer of the HARMS model. Considering the confidentiality and authentication techniques lead to propose a secure communications protocol for the HARMS model. Experiments were conducted in three different scenarios considering unfriendly actions in the surroundings through the communication and network basic layers of HARMS.

Collaboration was the main idea behind the HARMS-based heterogeneous human-robot team for a gathering and collection function paper [101]. In that work, authors used HARMS model to enable coordination within a network of robots that could be arranged to harvest products such as apples. For example purposes, they developed the scenario where they used humanoid robots to perform the recollection of balls of different colors.

Adaptability in autonomous robots which work in a dynamic environment should be achieved when unwanted changes from the environment may happen [79]. Such adaptability consisted of a proposed model divided in two modules. In the one side it is the information exchange using HARMS model to allow interaction between cyber physical systems. On the other side it is the model validation that uses NuSMV to check whether the system can continue its mission toward the goal in the given environment.

The concern of what will happen when multi-actor systems encounter unplanned issues during run-time was also a motivation to start thinking on a survivability feature of the systems final goals [102], [103], and [104].

3.2.2 Human-robot interaction and AmI environments

Matson et al. in [105] presented a communication protocol between humans, software agents, robots, machines, and sensors using a natural language interface. Authors made the first step in developing a complete model of interaction which they denominated HARMS. Consequently, it can be seen that the basic motivation of the model is the indistinguishability that can be accomplished while using a human language to interact between any kind of actor.

Ontological Semantic Technology (OST) as the basis for implementing cooperation between different types of actors was the motivation in [106]. Authors there were concerned about the safety of firefighters and victims of fires which normally are exposed to many threats due to the dangerous environments of house fires. HARMS there was mentioned as an enabler to be applied in conjunction with OST techniques in firefighting environments.

In the case of human interaction, the majority of papers related to HARMS are the ones where authors discuss applications using HARMS and natural language and reasoning properties. A document related to HARMS mentions that the original goal of the model was to let heterogeneous robots execute actions following a speech command received by them [107].

The implementation of HARMS in Ambient Assisted Living (AAL) applications started using the Narrative Knowledge Representation Language (NKRL) [108]. NKRL includes an inference engine that provides the reason action on any spatiotemporal relation that exists within natural language narratives. It uses two different ontologies, the HClass and the HTemp. Later on, continuing this path of research authors proposed a semantic approach for enhancing assistive services in ubiquitous robotics [109]. There, a collective intelligence framework is proposed, based on narrative reasoning and natural language processing.

Wagoner and Matson in [110] performed tests for each of the three sentence types (imperative, interrogative, and declarative) obtaining an overall accuracy of 96.6 %. Authors in this work presented a robust human-robot communication system using natural language for HARMS. Motivations were that the user does not need any prior training to be able to communicate with machines.

The motivation of an actor to take the lead part of a multi-agent system when receiving a verbal instruction by a human took researchers to present a task manager for such activity in [111]. Requirements satisfied in their experiments were:

- The task manager retrieved the correct task from the dialogue manager
- The task was broken into the correct sub-tasks
- The task manager returned the actors that were capable of performing each sub-task

3.2.3 Safety applications

As mentioned before, the motivation for [106] was to use HARMS as an enabler to be applied in conjunction with OST techniques for firefighting robots.

The fist physical implementation of HARMS model was documented in [98]. In this work, authors present the detailed implementation of HARMS first two bottom layers, namely, network and communication. The specific scenario used to evaluate the feasibility was a fire suppression system on a top floor of a high-rise building. Several heterogeneous actors conformed the testbed, such as: arduino-based sensors, DARwIn-OP humanoid robots, iRobot Create robots, data servers, and human interfaces. Within the same big project, several teams were integrated to work together in different sub-problems of the whole solution. a) As mentioned in sections above, [98] was the first real implementation of HARMS two bottom layers was to provide the platform that allowed the complete interaction between all the actors of the scenario. b) [112] presented the implemented way using HARMS of the UGV to go finding the source of the fire. c) Park et al. in [113] worked in the part that the different actors, either humanoid or simple speakers and microphones around the room were using intuitive interaction allowing speech recognition. d) Wagoner et al. in Wagoner et al. in [114] introduced humanoid robots capable of moving towards and extinguishing a fire and locating and rescuing humans. e) Khaday et al. in [115] detailed the way that the wireless sensor network was implemented in such a way that used HARMS. Additionally, it presented the documentation for Big Data storage taking place for the scenario.

3.3 Analysis and Design of Third Layer of HARMS

Before the implementation of the scenarios and the experiments mentioned later in this document the only layers developed were the first two: Communication and Network layers. Consequently, in order to implement any kind applications it was necessary to design and implement the third layer. The third layer of HARMS, denominated Interaction layer pertains to any kind of collaboration that could exist between one or more actors towards accomplishing a common goal. In other words, to be able to use the HARMS model in a higher level of applications than the ones implemented in the past, it was necessary to build the different components of the interaction layer.

In this dissertation it is documented the steps that took place to bring to function the interaction layer of HARMS. We used the methodology of prototyping of a system life-cycle to take it from the ideas and requirements to a level of production. This section will talk in detail about the first two phases of the life-cycle, while the third phase will be explained in the next chapter.

3.3.1 Problem Analysis

The the third layer of HARMS should include the fulfillment of the following requirements:

- It has to be based in the capability model that is one of the pillars of the HARMS model
- Take advantage of the services provided by the lower layers: Network and communication
- Assure the indistinguishability of each of the actors taking part in the different scenarios
- Provide an easy way to manage any number actors, with their different specific capabilities. It is implicit to assume that an implementation of HARMS may scale up to a considerable number of actors. However, it should also be able to work in small scenarios where the number of actors could be reduced to even use self communication management. For instance, lets assume that the same actor.
- Intuitive way of instantiation for people of any background
- It should include the possibility to implement any kind of interaction between any kind of actors

- It should cover the cases when any specific command is sent to an actor has implemented all the different capabilities needed for accomplishing a complete scenario actor

We consider that the interaction layer corresponds directly to the capabilities that the actor possesses. Therefore, the individuality of each actor that can possess different capabilities must be implemented the interaction layer. We define an actor capability as the specific ability to react to certain circumstances. A capability can be attached to one or more tasks. Such tasks are the ones in which a big problem can be subdivided. The triggering circumstances can be started by a set of reasons enumerated as follows:

- A global variable that the specific capability should be monitoring, such as a determined time.
- The call from another process, within a bigger process of the same actor, such as sub-processes of a bigger process.
- The call to execute a command remotely from another actor.

It is also understood that the different activities, related to different tasks based on capabilities, should be ready to be executed or called to be executed at the initial moment when the scenario starts.

As a matter of explanation, it was considered that the individuality of each actor is given by the actions able to execute based on the corresponding capabilities that it has to react to specific stimuli. For instance, if an actor is able to perform a specific action, when he receives a message asking for the related capability, depending on his availability and ability, he will answer positively to that question. We also consider that the concept of actions and tasks based on the capability model can be extended to a more complex extent one where a task can be accomplished depending on many other factors such as time, space, and associated tools at hand. However, cases were studied where the interaction itself has its high level of complexity.

In earlier implementations of HARMS, the interaction layer was understood and taking place as a matter of natural language processing, [79], [110], and [111]. However, in the implementation shown in this research effort, it was considered that the basic interaction between humans is based more in terms of capabilities. Hence, it was implemented the interaction layer in terms of capabilities.

3.3.2 Interaction Layer Design

The design phase of the interaction layer was basically divided into each of the expected functionalities. However, before talking about the different functionalities that any interaction should cover, it was defined what are the components of the layer. Figure 3.1 shows the basic components that were defined for each of the implemented layers of HARMS. In that figure, it was presented the other layers in order to show the context in which the components are located. The implementation platform is shown in each of the components. As a matter of explanation, it was divided the solution in three group of components: executables, services, and data persistence, as shown in 3.1.

Data persistence

For a better understanding, it will start explaining the data persistance components of the framework. Data persistance corresponds to the term of the different databases shown in the right side of figure 3.1. The databases are supporting the autonomy of every actor. In other words, each actor has access only to its own database files. Consequently, actor$_a$ needs to ask to actor$_b$ when the value of a variable stored in the private database of actor$_b$ is required in a process of actor$_a$. Therefore, the interaction between actors is enforced due to data decentralization. The location of the files for anything related with data either execution or configuration should be located in the *DATA* folder which is located under the root folder of HARMS. In the specific implementation of both databases, they were implemented using JSON format due

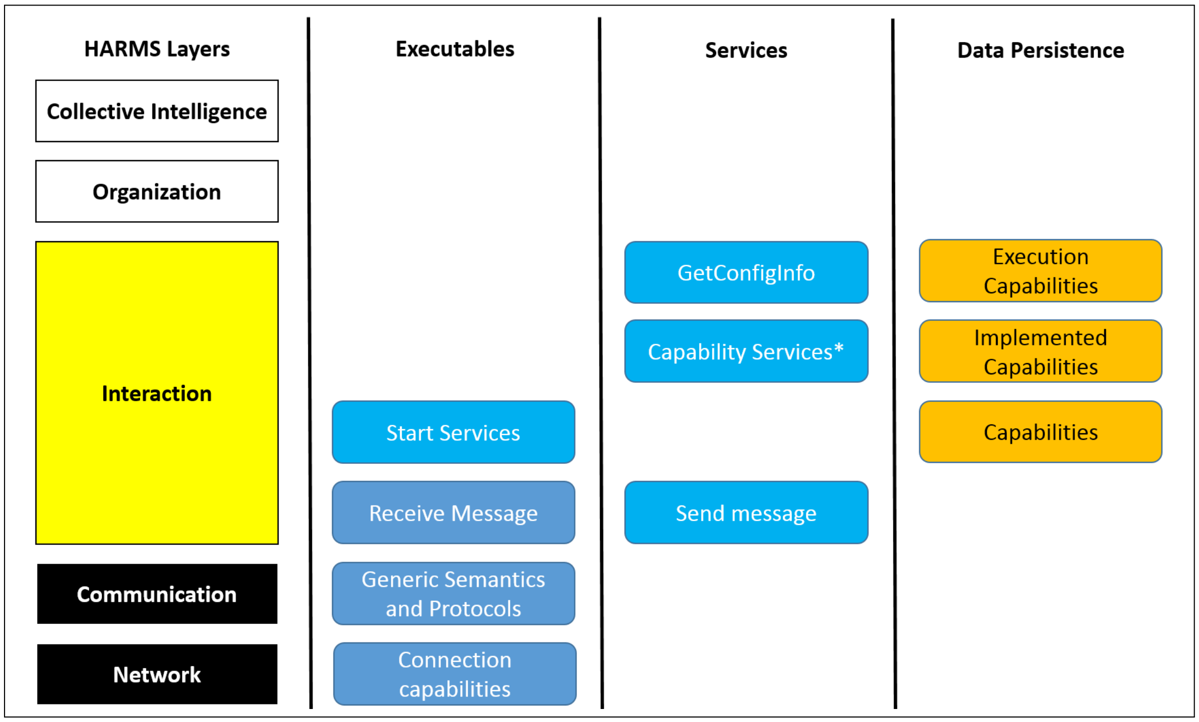


Fig. 3.1. HARMS interaction layer components

to the easiness of use. Another reason for using that format is the complete interface and library support in the programming languages used in this implementation, such as Java and Python.

Capabilities The JSON file of the capabilities is in charge of containing all the information concerning to the capabilities. Following the premise of the location of data, the location for the file containing the data of capabilities is *DATA\capabilities*. The file name of the database is *capabilities.json*. The basic structure of the capabilities database is listed as follows:

- **capability**: The capability ID. This is the identification that will be used to refer to a specific capability at any moment during run-time. We have used a nomenclature in which it is avoided a duplicate. An easy identification of the context and the capability is also enforced. As an example, *skit-lead-000* is used to identify the capability with the code 000 of the context *skit-lead* which corresponds to the leading part of the safe kitchen scenario. The provided nomenclature is a recommendation. However, architects or persons in charge of the integration of actors can name the capability as they consider more effective for them.

- **context**: This attribute is basically to group capabilities based in the context that the capability works.

- **description**: The space where the database administrator places a simple documentation of what is the capability in charge to do when it is executed.

- **type**: It concerns to the type of program that was implemented, it could be out of three cases: *logical* when it will not do anything, but it is used as a way to identify the actors that posses that capability such as the one used to identify which actor represents a patient. *executable*, Boolean value stored to identify if the file is an executable in the operating system. The more basic program of this kind is the program that contains the options of the lower and basic layers of HARMS. *webService*, when the program is going to be implemented and run as a webService. It is worth mentionig that the parameters that the program receives at the initial execution time are different to the ones received in each runtime execution through a webService call.

- **programmingEnvironment**: This attribute identifies the right programming environment at the moment of execution of the capability.
- **programName**: The name of the file that contains the program code to execute or the binary file if that is the case. It is worth mentioning that it is needed to avoid writing the extension of the file, since the program already adds it.
- **isExecutable**: Attribute used to determine if it is a binary file or a program that will be run from the specific programming environment.
- **needsToBeExecuted**: This is attribute serves to identify the files that will be executed at the beginning of the scenario execution.
- **programatStart**; If it also includes a different program program to execute at start time.
- **implemented**: A boolean attribute to determine if the capability is implemented or not in the specific actor where the code is running.
- **parametersExe**: This is a list of possible parameters that will be sent when the program is executed. The list of parameters can be 0 or more of them. The different attributes for each parameter is as follows:
  - *name*: The name or id of the parameter.
  - *description*: It is used as a documentation of the purpose of the parameter.
  - *type*: The type, it identifies if it is string or integer or any other type that can be used in the program.

  - *isFixedValue*: Attribute used to identify if the value is fixed from the beginning or can be changed depending on a condition during runtime.
  - *value*: The specific value for the specific instance.
  - *required*: If the value is required or not.

- **parametersIn**: This is a list of possible parameters that will be sent when the program is called to be executed. The specific case when this parameters are going to be used is when the capability is type *webService*. The list of parameters can be 0 or more of them. The different attributes for each parameter is as follows:

  - *name*: The name or id of the parameter.
  - *description*: It is used as a documentation of the purpose of the parameter.
  - *type*: The type, it identifies if it is string or integer or any other type that can be used in the program.
  - *isFixedValue*: Attribute used to identify if the value is fixed from the beginning or can be changed depending on a condition during runtime.
  - *value*: The specific value for the specific instance.
  - *required*: If the value is required or not.

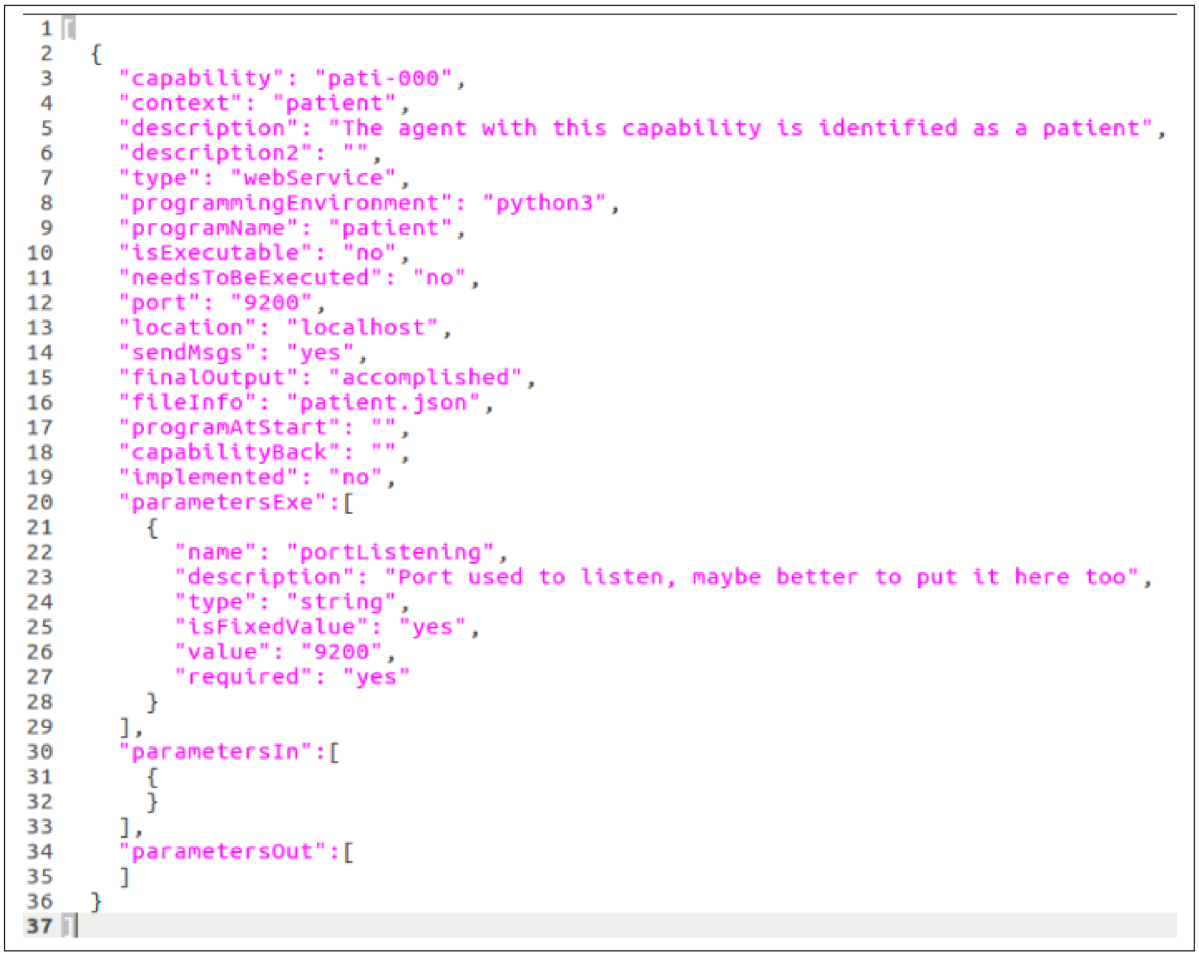
- **parametersOut**: This is a list of possible parameters that will be the output when the program is called to be executed. The specific case when this parameters are going to be used is when the capability is type *webService*. However, it was left open to send the different parameters as required by the different programs. Another reason to leave it open is that, at the end, the flow of the program is between different messages sent through the programs, not really as a reply received back as input from the actor that the message was originally sent to. The list of parameters can be 0 or more of them. The different attributes for each parameter is as follows:

  - *name*: The name or id of the parameter.
  - *description*: It is used as a documentation of the purpose of the parameter.
  - *type*: The type, it identifies if it is string or integer or any other type that can be used in the program.
  - *isFixedValue*: Attribute used to identify if the value is fixed from the beginning or can be changed depending on a condition during runtime.
  - *value*: The specific value for the specific instance.
  - *required*: If the value is required or not.

The figure 3.2 shows an example of a simple capability defined to identify an actor as a patient.

Implemented Capabilities This is the database containing the list of capabilities that are implemented in the specific actor where the HARMS stack is running. The name of the file is *capabilitiesImplemented.json*. The location for the file is in the folder with the name of *DATA\capabilities*. The structure of the database contains the following attributes:

- **actor**: Which is the ID of the actor. It is going to be the same value stored in the *config.txt*.
- **capabilities**: This attributes extends to be a list of the capabilities implemented in this actor. The sub-attributes are:
  - *capability*: The id of the capability. It has to be the same than the one in the file *capabilities.json* to be able to perform logical joins during runtime.
  - *implemented*: It is a boolean attribute determining *yes* if it is implemented and *no* otherwise.

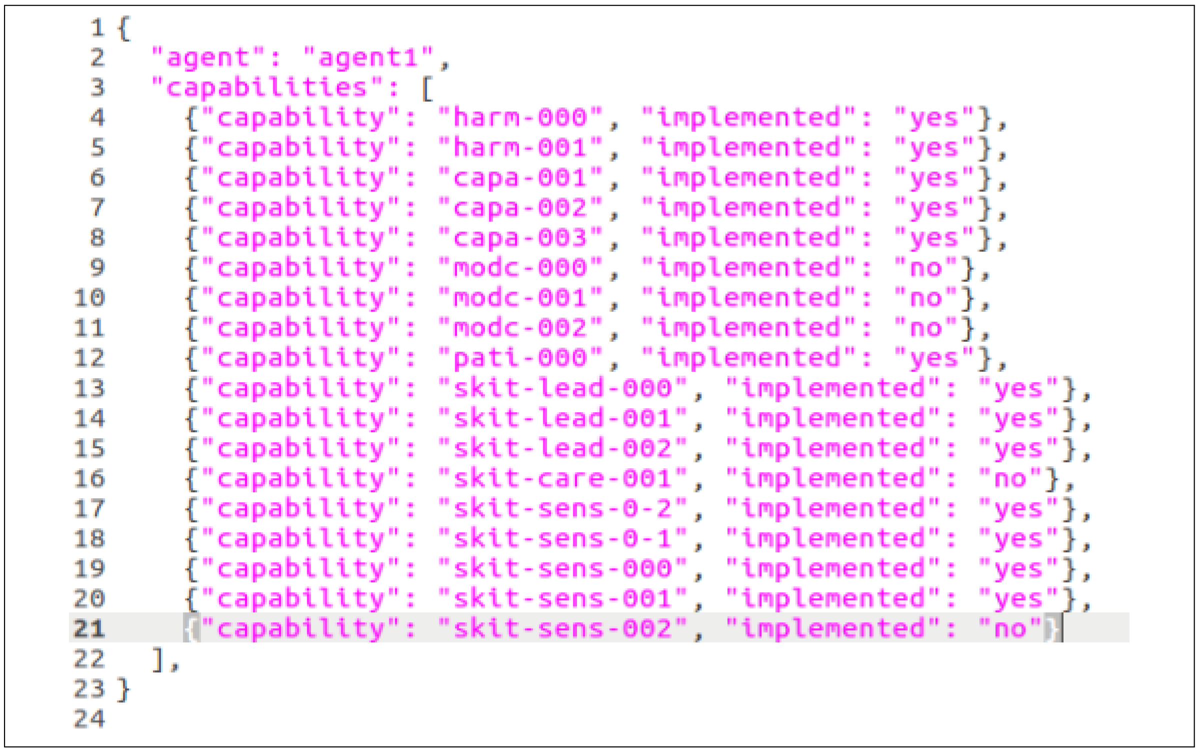


```
[
  {
    "capability": "pati-000",
    "context": "patient",
    "description": "The agent with this capability is identified as a patient",
    "description2": "",
    "type": "webService",
    "programmingEnvironment": "python3",
    "programName": "patient",
    "isExecutable": "no",
    "needsToBeExecuted": "no",
    "port": "9200",
    "location": "localhost",
    "sendMsgs": "yes",
    "finalOutput": "accomplished",
    "fileInfo": "patient.json",
    "programAtStart": "",
    "capabilityBack": "",
    "implemented": "no",
    "parametersExe":[
      {
        "name": "portListening",
        "description": "Port used to listen, maybe better to put it here too",
        "type": "string",
        "isFixedValue": "yes",
        "value": "9200",
        "required": "yes"
      }
    ],
    "parametersIn":[
      {
      }
    ],
    "parametersOut":[
    ]
  }
]
```

Fig. 3.2. Example of capabilities.json file

Execution Capabilitites In order to let the context of capability services work properly a database was developed. The name of that database is *executionCapabilities.json* which is located in the directory *DATA\capabilities*. An example of the file containing the execution capabilities database is shown in figure 3.4. The attributes included in the database are:

- **peers**: Which contains a list of attributes corresponding to each of the ones that has been receiving data. The attributes contained in this list are:
  - *hasCapability*: Which will correspond to the value whether the peer or actor has the capability implemented or not.



```
{
  "agent": "agent1",
  "capabilities": [
    {"capability": "harm-000", "implemented": "yes"},
    {"capability": "harm-001", "implemented": "yes"},
    {"capability": "capa-001", "implemented": "yes"},
    {"capability": "capa-002", "implemented": "yes"},
    {"capability": "capa-003", "implemented": "yes"},
    {"capability": "modc-000", "implemented": "no"},
    {"capability": "modc-001", "implemented": "no"},
    {"capability": "modc-002", "implemented": "no"},
    {"capability": "pati-000", "implemented": "yes"},
    {"capability": "skit-lead-000", "implemented": "yes"},
    {"capability": "skit-lead-001", "implemented": "yes"},
    {"capability": "skit-lead-002", "implemented": "yes"},
    {"capability": "skit-care-001", "implemented": "no"},
    {"capability": "skit-sens-0-2", "implemented": "yes"},
    {"capability": "skit-sens-0-1", "implemented": "yes"},
    {"capability": "skit-sens-000", "implemented": "yes"},
    {"capability": "skit-sens-001", "implemented": "yes"},
    {"capability": "skit-sens-002", "implemented": "no"}
  ],
}
```

Fig. 3.3. Example of capabilitiesImplemented.json file

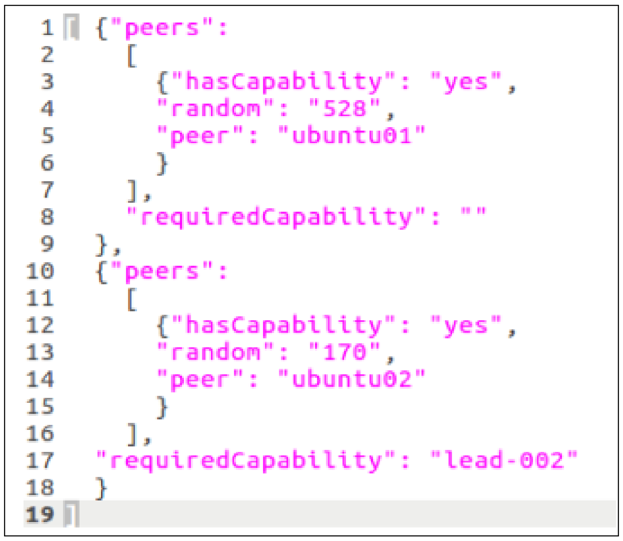
– *random*: Corresponds to the random number that the peer has assigned at the beginning of the execution of the specific capability.

– *peer*: The code or ID as the actor is identified when a message will be sent as it is stored in the file *peers.txt*.

Executables

The executables are the binary files that were developed in order to implement the different layers of HARMS.

In the case of the interaction layer, depending on the goal intended to accomplish by the executable it was decided which programming language was used. Java, and specifically the version JDE-1.8.0_171 was used to program, compile, and generate binary files of the two bottom layers and some functionalities of the interaction layer



```
{"peers":
   [
     {"hasCapability": "yes",
     "random": "528",
     "peer": "ubuntu01"
     }
   ],
   "requiredCapability": ""
},
{"peers":
   [
     {"hasCapability": "yes",
     "random": "170",
     "peer": "ubuntu02"
     }
   ],
"requiredCapability": "lead-002"
}
```

Fig. 3.4. Example of executionCapabilities.json file

of HARMS. Python version 3 was used to program the majority of the functionalities of the interaction layer of HARMS.

Receive Message As it can be seen in 3.1, the first component that the interaction layer contains and that is directly connected to the communication layer is the Receive Message component. This component implies a modification in the original program of HARMS executable Java code. The basic program of HARMS, which contains the network and and communication layers, originally developed in Java. Consequently, to align the implementation of the lower part of the interaction layer it was developed in the same executable. When mentioning it as a receive message functionality, it is understood that it is on top of the communication layer. It is triggered each time that the computer receives a message. That message can be, as stated by [98] either *command*, *notification*, or *query*. Therefore, the system knowing that will be able to react in the corresponding way each time that a message is received.

The workflow of the receive message process is shown in figure 3.5. It goes to consult *capabilities.json* database to see if that capability is implemented within itself. It is important to mention that the database *capabilities.json* will be updated with the

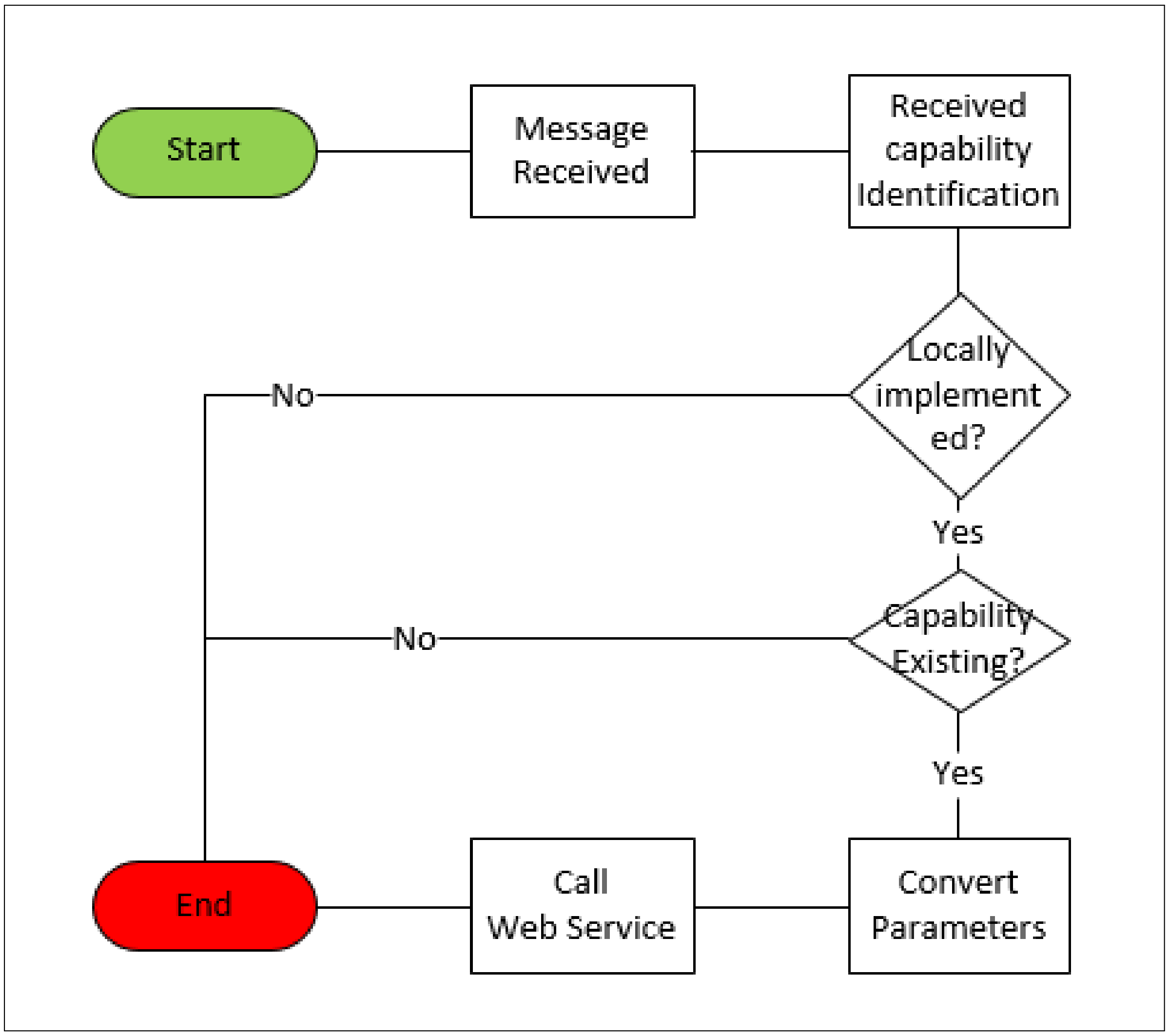


Fig. 3.5. Basic Workflow of the receive message process

attribute *implemented* at the start of the instance. If the attribute of that capability is *yes*, then it proceeds to make a call to the specific activity attached to it. As explained in the attributes of *capabilities.json* database, it can be either a program or a web service.

Start Services When implementing the third layer or HARMS, it was considered that, in order to be able to react to the messages or instructions received, the actor should be ready with his own capabilities enabled. For this reason, it was developed the *startServices.py* program which was implemented using python language version 3.4. The location of this file is *other_src\python* from the root of the implementation of HARMS. The goal of this program is to place at ready any of the activities corresponding to the capability ID that will be received afterwards.

Sevices

We decided to use web services as a mean to assure the ubiquity of the implementation of HARMS. In the basic implementation of HARMS it was noticed that in order to fulfill the indistinguishability of the model, it was needed to develop the capabilities functionalities.

As web services it is understood that they will be running all the time that the process is running. They also have specific requirements on basic parameters to receive. Those parameters are listed as follows:

- **peerTo**: Which corresponds to the peer that will be the one receiving a reply in the case that it would be needed. It also corresponds to the actor from which the message is receved from.
- **capability**: It will correspond to the capability that the system will use as a base to react to the message received..

Capability Services We decided to proceed to develop each capability as web services in order to assure that the execution will be easily accessed from within the same and different actors. Another reason to implement it in other language is to assure the independence of the code between layer 2 and 3, or communication and interaction. For that reason, it was also needed to implement the basic operations regarding capabilities. With that, all actors will have a base layer to work around capabilities that each of them has to provide towards a specific final goal. The language used to develop the webServices was Python and specifically the version was 3.4.

The capability services, as the name indicates, will provide the basic operations regarding capabilities which are:

- **Asked for implemented capability**: The id of this capability is **capa-001**. The actor with this capability receives a request to know if it has implemented a specific capability within itself. The steps that take place when this capability is requested are:
  - Gets a number stored in the file *executionCapabilities.json*. That number corresponds to a random number updated in each initial execution of a specific process.
  - Sends a message back to the peer from which the message was called. Within the message it also sends as parameter the random number generated.

  The message is sent as a unicast message to the peer from which the request was received. The corresponding program is stored with the name *askCapabilities.py* which is located in the folder *other_src\python\capabilities* under the root folder of HARMS code. It requires an additional parameter denominated *requiredCapability* which is the capability that will be corroborated if it is implemented within itself.

- **Receive capability replies**: The id of this capability is **capa-002**. The actor with this capability will be able to receive capability replies. That implies that will go to update the same database denominated *executionCapabilities.json*. Each time that a message with this capability associated is received it will go to either update or add a new tuple in the database in the case it does not exist.

3.3.3 Initial configuration of each execution

As it happens in any organization, when it is required to be formed, each of the actors already possess their own capabilities and are ready to be executed. The same way should be for a multi-actor system. Therefore, it was created the program that will be needed to run at the start point of the scenario. This needs to happen in order to have all the web services associated to each capability sound and running.

The file containing the instructions of the program is located in the foler *other_src\python* and the corresponding file is *start_services.py*. The program goes to the database *capabilities.json* to consult which capabilities are implemented within the actor. Depending on which capabilities are implemented, the program may run either the binary execution files or the raw programming files with the help of the corresponding instructions to run.

3.4 Negotiation and decision making using HARMS

The third layer of HARMS, denominated interaction layer, has the purpose of enabling the basic negotiations and decisions among a set of actors that would accept to cooperate to solve a problem. In this section an example will be documented.

3.4.1 Autonomous bootstrapping or Leader selection

As a matter of documentation, the general process of a negotiation between more than one actor will be documented here. Specifically speaking, the process implemented was denominated the autonomous bootstrap or leader selection.

Algorithm definition

Algorithms 1 and 2 present a negotiation that happens at the start of the execution of the scenario. There will be more than 2 agents involved in the initial negotiation to select the actor to perform the leading during all the execution. Such negotiation will be executed each time that the scenario is called to be executed. The initial instructions corresponding to bootstrapping are detailed in the algorithm 1. The subsequent instructions executed when a message is received is shown in the algorithm

2. Both algorithms combined and executed in each of the sides, the sender and the receiver, constitute the leader selection process.

Algorithm 1 determiningLeader(agent A, event e, time t)

```
Require: $\exists A$                                  ▷ agent where the algorithm is running
Require: $\exists e$                                  ▷ event that calls the execution
Require: $\exists t$                                  ▷ time for the delay
1: $A.ranNum \leftarrow$ generateRandomNumber()
2: $A$.query(broadcast, "leader capability?")
3: $A$.Delay(t)
4: $A$.query(broadcast, "execute leadership")
```

A broadcast message is sent from each agent that has the leader capability is the first instruction shown in algorithm 1. Given that at least two different actors will have the same capability configured within the network, the process is decided by a poll. A random number created while negotiating and each actor shares with all the peers with the same capability is included in the poll. A comparison of the values for each of the numbers created by each agent will be performed in each of those agents. That simultaneous comparison of all the values is possible due to the mesh communication between all actors. Such comparison in a mesh communication increases the possibility of equal distribution of leader assignments. Two sub-algorithms integrate the complete process. Basically, both are waiting for an event to happen. One of them starts when an external event provokes its initiation such as an specific time arrives. This algorithm 1 is the one in charge of generating the random number and store it for being able to share it alter on, when it is requested to be shared with other actors. After sending the number to all agents, that requested it, each actor waits until all numbers are received. One specific actor is selected by all the leader actors in an agreement based on which of them has the highest random number. The receiving messages part of the algorithm is detailed in the second algorithm. Upon reception of the message, the agent will first determine if the message is sent to itself. Even though the initial message is sent in a broadcast manner, the agent that will execute the leading process each time that it will be executed will be only one of the actors. The reason for that is to reduce the communication overhead caused by a distributed This last singularity of assignment, perfectly aligning with the indistinguishability expected of the implementation of the HARMS model, is accomplished by the instructions contained in the algorithm 2.

Complexity analysis

In this section a detailed complexity analysis is performed for both algorithms 1 and 2.

The formula 3.1 shows the evaluation equation for the algorithm 1 which is reduced to 2 instructions and 2 messages sent in a broadcast manner.

$$2n + 2 \tag{3.1}$$

Given that the instructions are executed in each of the actors, it leads to the $2n$, first part of the equation 3.1. The coefficient of 2 in the same equation corresponds to the instructions to designate the random number and the delay.

On the other hand, the algorithm 2 is detailing the instructions taking place when a message is received. Formula 3.2 shows how was evaluated that part of the program. A simple n is assigned due to a simple flow of of instructions starting with an if.

$$n \tag{3.2}$$

If both equations are multiplied as a combination of both algorithms, the result is as is shown in formula 3.3.

$$2n^2 + 2n \tag{3.3}$$

Ending in a simplification of the highest exponent as shown in equation 3.4.

$$\theta(n^2) \tag{3.4}$$

Algorithm 2 receiveMessageLeader(agent A, string msg, agent O, agent D)

```
Require: $\exists A$                    ▷ agent where the algorithm is running
Require: $\exists msg$                  ▷ message
Require: $\exists O$                    ▷ agent origin of the message
Require: $\exists D$                    ▷ agent destination of the message
1: if $A = D$ then
2:     if msg = "leader capability?" then
3:         if "leader" in $A$.capabilities then
4:             $A$.NOTIFICATION($O$, "yes " + $A$.ranNum)
5:         end if
6:     end if
7:     if msg = "yes " + $O$.ranNum then
8:         if $A$.ranNum < $O$.ranNum then
9:             $A$.selected ← $A$
10:        else
11:            if $A$.ranNum = $O$.ranNum then
12:                if $A$.ID < $O$.ID then
13:                    $A$.selected ← $A$
14:                else
15:                    $A$.selected ← $O$
16:                end if
17:            else
18:                $A$.selected ← $O$
19:            end if
20:        end if
21:        $A$.NOTIFICATION($O$, "leader " + $A$.selected)
22:    end if
23:    if msg = "leader " + $O$.selected then
24:        if $A$.selected <> $O$.selected then
25:            $A$.NOTIFICATION($O$, "leader correction " + $A$.selected)
26:        else
27:            $A$.NOTIFICATION($O$, "Ok " + $A$.selected)
28:        end if
29:    end if
30:    if msg = "leader correction " + $O$.selected then
31:        $A$.selected ← $O$.selected
32:    end if
33:    if msg = "execute leadership" then
34:        $A$.EXECUTEACTION(lead)
35:    end if
36:    if msg = "Ok " + $O$.selected then
37:        DONOTHING
38:    end if
39: end if
```

3.5 Summary

In this chapter of the dissertation it was presented the HARMS model to implement the means for enabling any kind of interaction between different types of actors. The HARMS model consists of five different layers that provide services in a bottom-up approach. In this chapter was also detailed the different implementations and related work that have been done at the moment. The documentation of the implementation of the third layer, interaction layer was also detailed in this chapter. Finally, as an explanatory case, it was meticulously analyzed an algorithm of interaction between actors using the HARMS model.

CHAPTER 4. INDISTINGUISHABILITY AND SURVIVABILITY IN MAS

Autonomous heterogeneous multi-agent systems are integrated by actors of many different types. They work together to accomplish a specific goal. Lately, many applications of MAS have been turned towards giving a solution to human societal problems. Ambient intelligence (AmI) environments comprehend digital environments that work towards supporting people in their daily lives [116]. Within that definition could be included several environments such as smart homes, health monitoring and assistance, hospitals, transportation, emergency services, education, and workplaces [117].

Out of an extension of the benefits of adopting the Internet of Things, newer concepts like the Internet of Robotic Things (IoRT) appear. Quotidian problems of society can be addressed by solutions that integrate different types of actors, including robots. On the one hand, Vermesan et al. in [118] show a detailed list as: sensing, cognition, perception, planning, actuating, and control as aspects to take into consideration for developing robotic systems. On the other hand, Chibani et al. in [119] present perception, actuation, and control as the fundamental functions that motivates the majority of robot systems. Advanced and elaborated Artificial intelligence (AI) techniques are common characteristics that devices and robots will be gifted to face close future problems. Chibani et al. in [119] presented IoRT systems as solutions to single actor system problems through the collaboration that can be achieved in a smart fashion within a network of heterogeneous actors. Expressed in a different way, individual abilities to solve big problems in a collaborative way; a problem that can not be solved otherwise. Such abilities include evaluation, actuation or both. Nonetheless, a good challenge is seen ahead to develop autonomous IoRT services that may populate greater defiance for the research community to accomplish a harmonic orchestration of all the actors.

A system that is able to accomplish its own mission, providing minimal acceptable values for aimed services and accomplishes the goals regardless of the hostility of the environment is a survivable system [120]. Hence, survivable systems ought to be able to autonomously recover at the moment when the problematic situation is over-passed without regard if the improvement was due to external or internal changes. Generally speaking, survivability of original goals for a system of an individual actor to manage a problem that is not programmed to overcome is essentially impossible. In recent times, different approaches of autonomous systems have been studied in more detail. Although, the network topics have been of more interest in terms of survivability proprieties for researchers up to now [121], [122], [123]. Nonetheless, one of the applications that researchers have recently focused on is Ambient Assisted Living systemd (AALs). Motivations for this gradual change to focus in AALs could be pushed by the gap population problem between elderly people and their caregivers. Problem that is more perceivable in developed countries. However, also the complexity of this kind of solutions such as the one that is generated for including several heterogeneous actors to the environment could be one of the reasons to have less related research at the moment. Moreover, when humans are considered not users, but actors within the solution, it generates that the scenario has to be able to evaluate any possible path that could happen responding to the unpredictability of such actors. Still, the IoRT systems should include reliable features that make them capable to succeed in what they were created for in a survivability fashion in spite of facing issues that are unknown before execution time. To the concern focused in this document, solutions that include survivability feature should have more related research, specially given that soon solutions will be provided as AALs and AmI, where humans will be depending in a higher level of systems suggestions.

Applying model checking in execution time and taking advantage of Cloud resources was presented in previous work [103]. Providing possible solutions to issues encountered during run-time as a survivability mechanism was the research focus of that work. In other words, the focus is set on providing possible solutions towards reaching the original final aims of the system in contrast to only being successful to identify errors while the system is running. This last mentioned is the target of model checking since its inception. Moreover, the cloud is used to reduce the response time used for the model checking activity while using several compute services a parallel execution. Such execution is used to increase resources destined to the state explosion problem. The state explosion problem is an implicit issue while running the model checking technique. In this research, an evaluation of the benefits of using the cloud resources to offload the model checking activity as a part of the survivability of the system final goals.

Governments of developed countries are turning their research priorities to the safety and security of elderly citizens at home, as the European Union [124]. Motivated by the gap difference increase between old people and their own caregivers [125]. Consequently, one of the strategies to tackle down this governments problem is with the AALs that are expected to be able to let old people live by their own in initial stages of specific deceases.

Multi-Agent Systems(MAS) paradigm could be used as a viable approach upon AALs solutions. Related to that, on the one hand, Sycara [7] defines MAS as the systems that has the ability to do the following.

1. Give solution to large problems which can not be managed by centralized actor.
2. Allow the interconnection and inter-operation throughout to any other system.
3. Solve problems related to organizational autonomous actors.
4. Optimize the use of sparse and distributed information.
5. Manage efficiently different levels and distribution of expertise.

On the other hand, Carley [15] gives the meaning of an organization as

1. collections of processes and intelligent adaptive actors
2. are task oriented

3. socially situated

4. technologically bound

5. and continuously changing.

Characteristics that are similar are easily identified in both definitions. Summarized as an integration of multiple heterogeneous actors that don't count with complete information and competences to give solution to a problem; Impossibility to count with global control and data fluctuation are enforced to work in a decentralized manner. In multi-actor systems each and every actor provides either computation, sensor, or actuation capabilities in a coordinated way. Many different forms have been defined at the moment to accomplish that coordination, such as centralized or the complete opposite betting in the autonomy of the actors. Many failures could be reached when a set of different independent and autonomous actors are set to collaborate. Detection of such failures may not be possible during analysis stage. Consequently, designs of such systems could lack of mechanisms to overcome them.

HARMS is based on the Machine-to-Machine (M2M) infrastructure to integrate humans, software agents, robots, machines and sensors into collectives [41]. The different layers defined for HARMS are: network, communication, interaction, organization, and collective intelligence. Each of those layers provide the means for the agents to interact. Indistinguishability is the most important feature for HARMS. Such characteristic lets the assurance of the activity to be done, no matter the actor performing it [96]. Services over the cloud are used to execute the model checking processing to accomplish the survivability feature over AALs using HARMS. Parallel running of model checking over the cloud is used to evaluate which of the modifications of the model could be applied as a solution for the issue encountered during run-time.

4.1 Guiding a person to perform a specific activity

There exist many applications where multi-actor systems help people to perform activities in an AmI setting. This section starts mentioning related work to the frame that will be presented next as means to provide indistinguishability and survivability features in the scope of AALs using HARMS.

4.1.1 Related work

The complete framework presented in this research effort includes and tacles down several different aspects of AmI and AALs spectrum. Survivability, for example, which is mostly investigated in different fields than in ambient assisted living systems. Ayara and Najjar in [126] present health care as an applicable scenario of a formal specification model for survivability in pervasive systems. There, authors evaluate the degree of survivability as a means to ensure the acceptable execution of the services offered. Based on that evaluation, in that research, they present an adaptation to the current situation. Model checking is not used in that research, consequently, that is the main difference to what is studied in this document. Also, even though it was applied to health care systems it was not applied to AALs.

Research community is focused mostly at the moment in topics such as adaptability [127] based in fault tolerance approaches to propose a re-configurable model framework in home automation. The research effort in this document presents an approach to accomplish the feature of survivability of original final goals of the system. In the present, exhaustively monitoring the targeted patients is one of the trends in AALs. Such trend is possible due to the information generated by pervasive devices located more and more in humans daily life environments. Forkan, Khalil, and Tari in [128] propose a cloud-based, real-time, context-aware platform to analyze the enormous amount of data generated by the different connected devices. Tailored services are used to provide global and individual needs are the goals of that solution. The latter kind of solutions are grouped in context-aware solutions which differ from the solution presented in this work. The cloud services are used only to execute the model check activity and it is triggered only if an error has appeared while the system is running. Consequently, the monitoring of possible errors is considered as out of the scope of this research effort. Using cloud resources with temporal logic and model checking during run-time has been used to monitor and recognize activities for smart environments in the kitchen room [129]. That work uses model checking to recognize activity patterns instead of trying to find a solution to the problems encountered during run-time. No deep studies were found for verification during run-time applied in AALs. A study that converts patterns of behavioral UML models of AALs architecture to specifications able to be formally executed and PRISM is used to verify the corresponding model can be found in [130]. That study was an effort on analysis of dependability. Changes of actors executing an activity is possible overtime due to the indistinguishability that is possible with the use of the HARMS model. Such difference to the case of using UML diagrams to code that can be used in a formal verification tool as explained in [130].

The novelty of the HARMS model could be one of the reasons why there is no deep related research. Nonetheless, a narrative knowledge representation language that uses defined reasoning rules as an example of implementation of the HARMS model as enabler of interaction between heterogeneous actors [109]. Model checking is not used to propose a solution as in this research effort differs to the one presented by [109] which applies intelligence to the mentioned perspective.

4.1.2 AAL systems

The goal of survivable systems is to induce corrective actions in order to maintain the ability to accomplish the original final goals even though unknown issues could be found during run-time. The relevance of the framework presented in this work is presented in this section. After that, two different use cases of AALs applying indistinguishability and survivability based in the implementation of the HARMS model will be presented.

An old person with Alzheimer is the motivation for the first scenario presented in this chapter. Persons with Alzheimer lose the sense of time. Because of that, when they sit to watch TV they may spend not hours but days. Consequently, guiding a person with those characteristics to go to bed covers the first motivating scenario. As shown in Fig. 4.1, the scenario involves several actors that work autonomously and agree to work together to fulfill the goal of guiding the end user to their bed.

A list of the problems that could be encountered by the previously mentioned system are:

- The mobile robot could stop with no explainable reason
- The communication may be not possible anymore
- The patient or the robot that need to be guided could not be following the instructions in the correct way
- Delays out of boundaries determined in the original plan
- Speakers not working properly
- One of the actors cannot get the information properly such as the actor needed to be assisted is not able to recognize the message

The aforementioned case of AAL system encapsulates guiding a patient or another actor to execute a specific activity such as going to sleep. Such solution includes the use of a heterogeneous multi-actor system where each of the actors provides their own capabilities to accomplish the big goal.

Independently of how often the situation takes place, it is understood that every single execution of the system will vary in a very dynamic environment such as the one where these framework is applied to. The uncertainty implicit in the scenario could drive the execution path to get changes depending on the values that each variable take during run-time. Consequently, the focus in this research effort is extended to the part of finding solutions to issues not seen before execution time in an implementation of the HARMS model for a person needing guide to go to bed scenario. Moreover, given the complexity to manage all the involved actors by a traditional dependable MAS approach the focus in this document will be on scenarios where unexpected circumstances take place during the run-time of AALs. Consequently, the HARMS model is the base to perform ad-hoc organizations. The problem of this section of the document is divided in three different part as:

- The bootstrap of the scenario is based in redundancy or actors where the system can rely in that if an actor is not working another can step in to cover it.
- The creation of the team of actors with activities assigned is done in an autonomous way.
- The system counts with a simple verification of the current status of the execution compared against the model of the system itself. If a discordance is found, an exception starts the process of solution generation which will propose the best possible solution to implement in order to overcome the issue found during run-time in a survivable way.

An elderly person is guided to sleep from the living room at an specific moment of the day as shown in figure 4.1. The list below show the steps taking place in that scenario:

- *Step 0:* The beginning of the run, depending on a specific time of day an event will start the negotiation to determine which actor will be the leader of the scenario.
- *Step 1:* A device is requested to execute an activity in order to accomplish a goal.
- *Step 2:* The device executes the different steps needed to accomplish the goal.

- *Step 3:* A normal on-line verification to confirm no errors occurred.
- *Steps 4 and 5:* Where the device in charge finishes one or more sequential activities.
- *Step 6:* (not shown in the figure) Is a confirmation indicating that the goal was successfully achieved.

4.1.3 Proposed Framework

This research effort presents a framework which extends what was presented in [103]. In that work authors were reaching self-healing feature based on the information and the work-flow of four not directly connected processes. First, the algorithms *leader selection* and *autonomous team formation* detail the steps for the implementation of the HARMS model in the scenario of guiding a patient to sleep. Second, the algorithms called *survivability* and *self-healing* show the steps for the implementation of the survivability feature.

HARMS implementation

The HARMS model could be used to provide indistinguishability and survivability features in AAL scenarios such as the one mentioned in the previous section as the motivating scenario. HARMS is a model that enables a multi-actor system to coordinate in an indistinguishable method the assignment and execution of activities [96]. HARMS is a model based on the capability model, where different autonomous actors have a number of capabilities and work together to accomplish a collective goal [41]. Consequently, in this research effort the first three layers of the HARMS model are used, namely, the network, communication, and interaction layers. Within an AALs there are several heterogeneous autonomous actors that are detailed as follows:

- Assorted sensors will be located in different places to be able to monitor important values from other actors such as patient and other robot. Context infor-

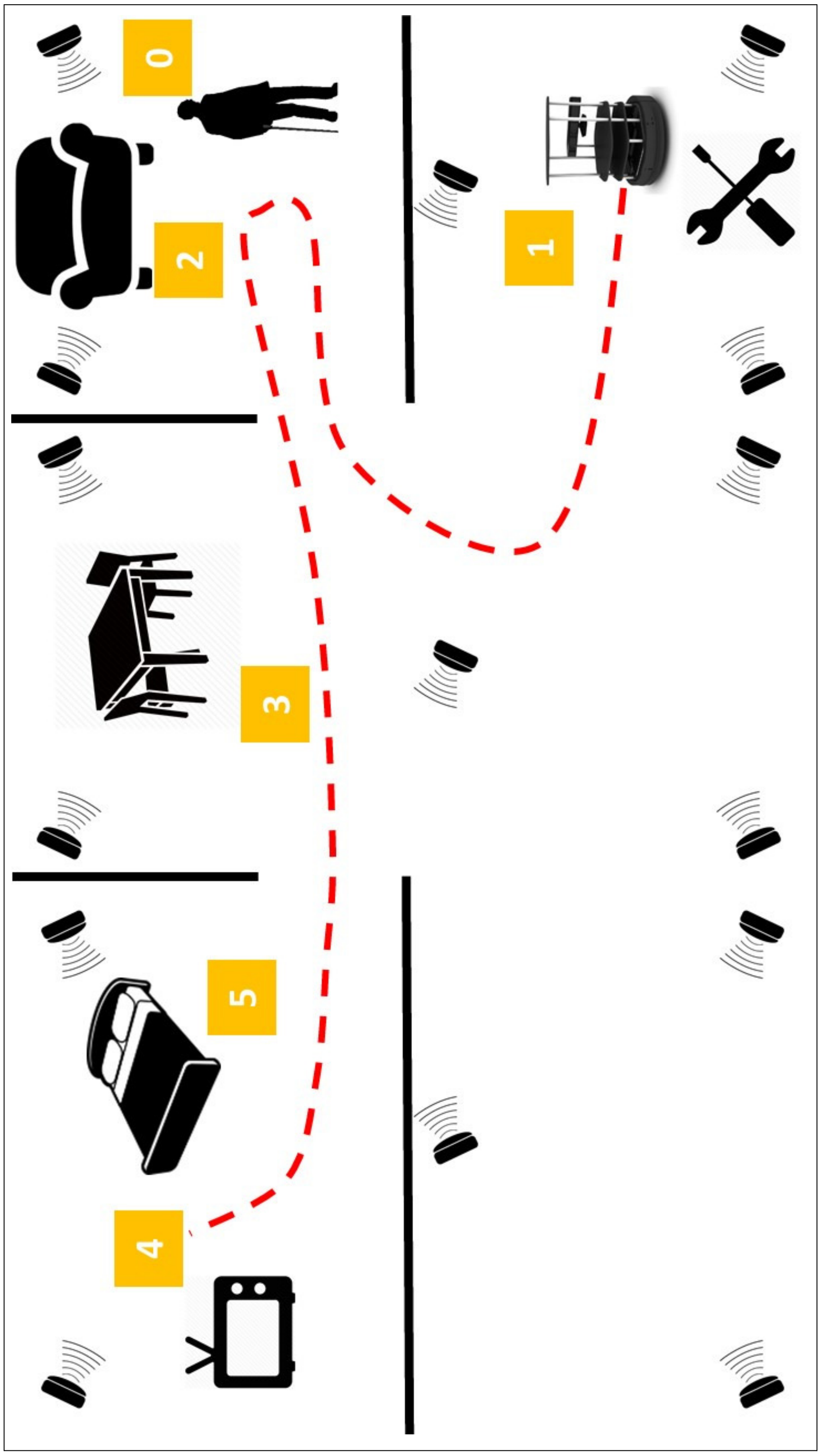


Fig. 4.1. Take me to sleep scenario drawing

mation related to other actors is the type of readings that those actors could get and provide to the system. Such context information could be the exact location of other actors such as the patient. The location of the patient could be accomplished as a triangulation of readings of several contextual information actors.

- It is assumed that the patient or dependable actor will always have in a reachable distance the way that will be used to detect their location. Such device would be like a phone which allows to the actors mentioned in the previous bullet to determine the exact location.
- There will exist more than one actor that will be able to start the scenario at a given time.
- It is assumed that within all the actors all there will exist capabilities redundancy. In other words, within all actors, at least 2 of them will have the ability of each of the capabilities.
- This framework is based in the redundancy of the actors, where all different capabilities are going to be able to

As a formal definition, for any AAL system using HARMS there should exist the following elements: $A_0...A_n$ actors with $C_0...C_m$ capabilities that can be held by any number of the actors to solve $T_0...T_l$ tasks or specific problems to be solved.

The term actors above refers to the different humans, software agents, robots, machines, and sensors eager to cooperate to accomplish a big goal. An actor capability is the way that the actor has to react to specific circumstances. As an example, if an actor receives a message with the parameters to execute an action, it will be able to accomplish it only if it has configured that capability. As mentioned tasks in the formal definition is understood all the different activities that the actors will execute or are able to execute to accomplish the final goal. As it can be seen in the above definition, the coordination and control of this kind of system is complex.

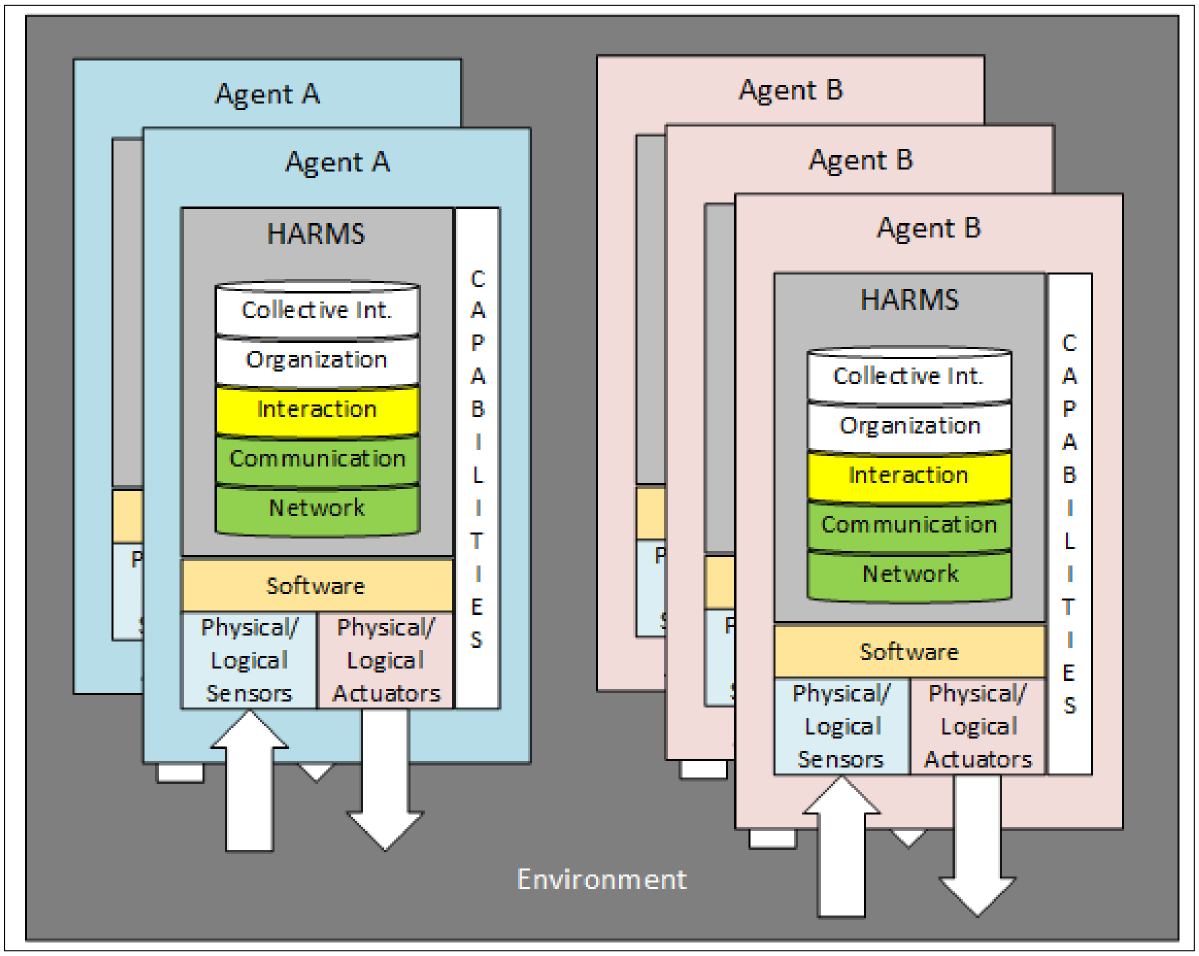


Fig. 4.2. HARMS actors

A series of steps are needed to be executed at the moment when a new actor joins a specific HARMS network. Next it will be detailed the assumptions of the steps that a new actor should go one by when when it firstly integrate the HARMS network.

1. The HARMS model software should be installed in the actor.
2. The complete information regarding the self-identification, self basic configuration. The list of peers should also be set in the system.
3. The actor count with the a list of capabilities that the actor can perform with the different physical and software configuration it has.
4. All the configuration of the capabilities that the actor is able to carry on have to be assigned and

Communication is enabled to let different actors to interact using the HARMS model protocol, as it is shown in figure 4.2. The other layers of the HARMS model let the different actors to interact, organize, even generate collective intelligence. Physical and logical setting of the different actors determines the first level of what they will be able to contribute to solve the big problem. In this scenario, the first three layers of the HARMS model are the only ones used. The first layer, allows basic network protocols. The second layer, provides communication syntax and semantics. The third layer, lets the actors define in an autonomous manner the way that they will collaborate.

Network layer In terms of the network layer, the basic physical communication functions are provided where the protocols to be used will depend on the actors. Basically the most common will be TCP/IP, or sockets. HARMS model is open to the possibility that in some cases it would be necessary to use other protocol such as XMPP. Also, the type of messages able to be sent are unicast, multicast, or broadcast. Each of the three cases were implemented in this scenario, depending on the context of information needed to be sent. As an example, a broadcast message could be the one sent by to leader to all the other actors to ask for the ones that count with a specific capability. An example of a unicast message in contrast could be the one sent as a reply to the previous question, from the one replying and sent only to the one that was asking for that capability. Lastly, an example of a multicast message could be the one that will be sent to the actors that count the capability to see the patient, specifically, when the leader will ask for the actors that see the actor at that moment.

Communication layer The basic information exchange is expected to be accomplished when implementing this second layer of the HARMS model. In this scenario all actors can use any type of message: query, imperative, or notification as options to send. Hereof, the kind of message sent between actors can be of any of the options available, depending on the specific communication objective between two or more actors. When the location of a specific actor is needed, the leader actor in the system will send a query to all the actors that possess the capability of location. The query type message requires answer from the actors that receive it. Consequently, the answer will be addressed sending the reading of the location for each actor that can see the actor. Lastly, the type of message imperative or command is when there is the need of a specific actor to perform an action. For example when the instruction to execute the guiding activity to the actor guide.

Interaction layer The interaction layer provides the basic means to let different actors to perform different group decisions among them. An auction could be accomplished as the decision of which actor will perform a specific activity directly related to a capability. As an example it is the one that takes place when the leader is selected.

The layers of organization and collective intelligence are not included nor required for the implementation of this scenario. Consequently, there is nothing specified for them in this document.

Model definition

This section is devoted to declare in a detailed manner each of the actors that are used in the scenario.

Leader actors (A) The actors that possess the leading capability are those ones that are able to coordinate the other different actors to work together to accomplish a big goal.

Contextual information actors (C) Contextual information actors are the ones able to read information about the environment related to other actors. For example, the RFID readers, used to get the location of the patient and the guide actor, working in a networked triangulation for improving accuracy.

Assistive actors(As) The assistive actors possess the capability to guide, monitor, give instructions, and physically help to other specific actors such as patient and robots.

Dependable actor (De) A dependable actor is the one that will be guided to do a specific activity. In this specific scenario, the dependable actor is the patient that needs to be guided to bed. Nevertheless, the way that it is proposed the framework includes any kind of actors in a generic algorithm presented. With that, the dependable actor could even be one of the actors that stop working properly during execution time, such as the guide actor.

Origin actor (O) The origin actor is the one that sends a message when a protocol of communication takes place.

Destination actor (D) Destination actor is the one that receives a message previously sent by another actor. Self communication is possible as well through the HARMS model.

Algorithm definition

In this section the algorithms related to the cloud survivability framework applied in AALs using HARMS are presented and detailed.

The algorithms are four and are presented next.

Autonomous bootstrap or leader selection Since the implementation of the HARMS model implies a decentralization of control and information, it also drives every scenario to start each activity without any actor assigned. Consequently, in the start of running all scenarios that use the HARMS model, there will exist an initial negotiation between the actors that possess the leading capability. This process was completely documented at the end of the previous chapter.

Autonomous team formation Continuing with the implementation of the HARMS model in this scenario when an old person needs to be guided to go to bed, the team needs to be created during run-time each time it is executed. A communication between the different actors exchanging information regarding their capabilities and availability will be important for the implementation of this algorithm. Expressed it in a different way, one actor asks with a broadcast message for the actors that have a specific capability configured. Following the reception of the request, the actors that count with that capability, they reply with other message to the requester with an affirmative answer.

A cohesive group is created in an autonomous fashion as shown in algorithm 3. In that algorithm the focus is to work in generic scenarios of AALs. What happens when in the destination actor upon reception of a message is shown in algorithm 4. Note that algorithms related to the reception of messages could be many. However, here it was presented only one as an example.

As mentioned before, algorithms 3 and 4 include the different instructions to integrate all the actors needed for each execution of the scenario, depending on the available actors. To easily understand the process lets divide the sequence in two sections:

1. Determining the most fit-able HARMS model between different AAL scenarios.
2. Since each activity, including the ones included in AAL scenarios have specific steps to be followed to accomplishing them, they need to be defined as well.

Negotiation as a basis of what happens between not previously grouped actors is presented in the algorithm 3. This algorithm starts from the assumption that the leader actor has already been selected, following the process detailed at the end of the previous chapter, specifically in algorithms 1 and 2. Consequently, the leader actor can be able to determine which actors posses the capability of ”see” a selected group of actors. The capability of spotting or provide location of other actors, status, and other environment variables are called contextual information actors. As an example could be mentioned the RFID readers that could calculate the location of actors that have the RFID tags that are visible for the readers. In the specific scenario presented in this section, humans needed to be guided will always have a phone with RFID tag in their own pockets. Basic communication runs back and forth between the general leader actor and the actors that have the specific contextual information capability. The leader actor receives a positive answer from the actors that have such capability configured and are also able to read it at that moment. As an example, in this case will be asked not only for the actors that have the capability, rather than asking which actors can really apply it now, such as ”seeing” at that specific location of the patient. Following that, the leader actor asks for the specific read of the actors that replied positively to the question of whether they were able to get that specific values at that moment. The calculation of the position of an actor is out of the scope of this work. Consequently, it is only called as a simple function that returns the location within the scenario. After getting the location of the patient or dependable actor, the leader actor starts the process to assign the guide actor. Such process is a handshake related to ask for capability of guiding, where only the ones with that capability will reply acknowledging that they have it. The assignment of the actor $_a$ in charge of guiding actor $_p$ is materialized based in the positive replies. A group of steps are then shown in the algorithm which correspond to the action of guiding the patient to bed. All those instructions are sequential, where non of them can be skipped. The process is considered finished or accomplished when the assistive actor sends a message to the leader actor stating that the patient is already in bed.

Algorithm 3 leadTeamFormation(actor A, actor D, time t, actor[] C*, actor[] As*)

```
Require: $\exists A$                          ▷ leader actor (selected previously)
Require: $\exists De$                         ▷ dependable actor
Require: $\exists t$                          ▷ time for the delay
Require: $\exists C*$                         ▷ group of actors w/context capability
Require: $\exists As*$                        ▷ group of actors w/assistive capability
Require: $*$                                  ▷ updated during runtime possibly by other procedure
 1: $A$.QUERY(broadcast, "[Context] capability?")
 2: $A$.DELAY(t)
 3: $A$.QUERY(multicast($C$), "[Context] info De?")
 4: $A$.DELAY(t)
 5: $D$.location $\leftarrow$ $A$.DETERMINELOCATION(C)
 6: $L$.QUERY(broadcast, "[Assistive] capability?")
 7: $A$.DELAY(t)
 8: $A$.QUERY(multicast($As$), "[status] info?")
 9: $A$.DELAY(t)
10: $As_{best} \leftarrow As_0$
11: for each a in $As$ do
12:     if $a$.[status] $>$ $As_{best}$.[status] then
13:         $As_{best} \leftarrow a$
14:     end if
15: end for
16: $A$.DIRECTIVE($As_{best}$, "Inst " + MoveToActorLocation + De.location)
17: $A$.WAITFORMSG($As_{best}$, "Done")
18: $A$.DIRECTIVE($As_{Best}$, "Inst " + AskActorPerformAction + De)
19: $A$.WAITFORMSG($As_{best}$, "Done")
20: $A$.DIRECTIVE($A_{Best}$, "Inst " + [AssistiveAction] + De)
21: $A$.WAITFORMSG($As_{best}$, "Done")
22: $A$.NOTIFICATION($O$, "Done")
```

The algorithms 1 to 4 corresponding to the scenario of guiding a patient to bed can be seen in the figure 4.3

Algorithm 4 receiveMessageTeamFormation(actor A, string msg, actor O, actor D)

```
Require: $\exists A$                  ▷ actor where the algorithm is running
Require: $\exists msg$                ▷ message
Require: $\exists O$                  ▷ actor origin of the message
Require: $\exists D$                  ▷ actor destination of the message
Require: $\exists De$                 ▷ dependable actor received in message
if $A = D$ then
    if msg = "[Context] capability?" then
        if [Context] capability in $A$.capabilities[] then
            $A$.NOTIFICATION($O$, "Yes")
        end if
    end if
    if msg = "[Context] info D?" then
        contextInfo ← GETCONTEXTINFO(De)
        $A$.NOTIFICATION($O$, "[Context] info = " + contextInfo)
    end if
    if msg = "[Assistive] capability?" then
        if [Assistive] in $A$.capabilities[] then
            $A$.NOTIFICATION($O$, "Yes")
        end if
    end if
    if msg = "[status] info?" then
        status ← GETSTATUSINFO
        $A$.NOTIFICATION($O$, "[status] info = " + status)
    end if
    if msg = "Inst " + MoveToActorLocation + De.location then
        MOVETOACTORLOCATION(De.location)
        $A$.NOTIFICATION($O$, "Done")
    end if
    if msg = "Inst " + AskActorPerformAction + De then
        while response != "Yes" do
            response ← ASKACTORPERFORMACTION(De)
        end while
        $A$.NOTIFICATION($O$, "Done")
    end if
    if msg = "Inst " + [AssistiveAction] + D then
        [ASSISTIVEACTION](D)
        $A$.NOTIFICATION($O$, "Done")
    end if
end if
```

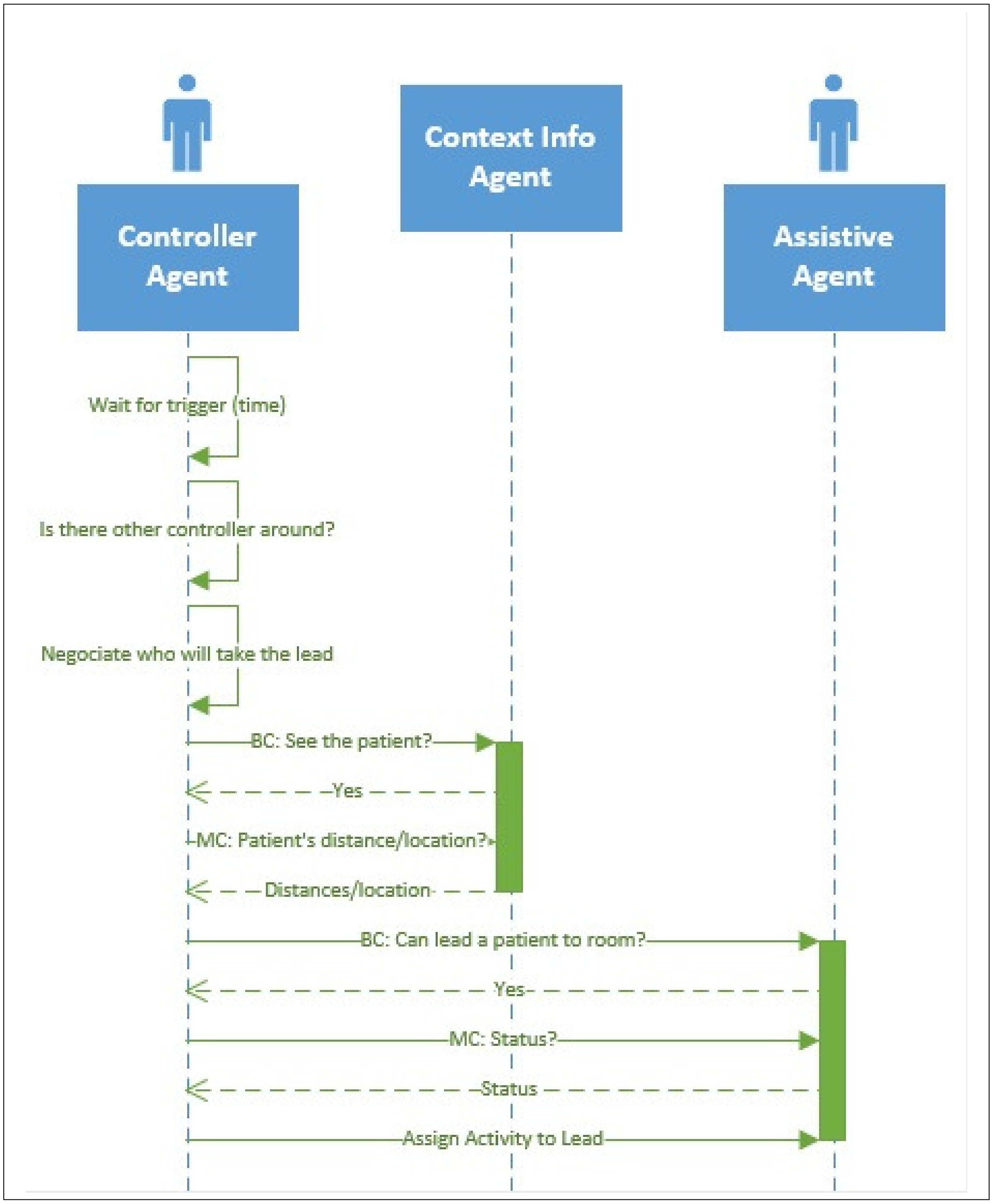


Fig. 4.3. Actor Interaction Workflow of HARMS Implementation

4.1.4 Complexity Analysis

It was considered important to evaluate the complexity of the algorithms presented in the previous section. Consequently, in order to better evaluate such algorithms, it was considered pertinent to divide them in two groups as:

1. Bootstrap or leader selection

2. Autonomous team formation

Bootstrap or Leader Selection

The complete complexity analysis of this algorithm was evaluated at the end of the previous chapter and the result is n^2.

Autonomous Team Formation

As an initial abstraction of the algorithm 3 it was determined that all steps contained after the line 14 will be evaluated with complexity of 1. Nonetheless, an n is assigned to the steps of broadcast and multicast given that it implies that they will be sent to all the actors or a group of them, not just one of them.

Formula 4.1 shows the evaluation in complexity terms of the algorithms mentioned in the previous paragraph and shown in algorithm 3

$$5n + 13 \tag{4.1}$$

While for algorithm 4 a simple evaluation can be seen in 4.2.

$$n \tag{4.2}$$

Consequently, formula 4.3 presents the consolidation of the two algorithms 3 and 4.

$$4n^2 + n + 13 \tag{4.3}$$

The conversion presented as $4^2 + n + 13$ is the result of the 4 n associated to the messages sent to more than one actor. The final is a loop. Therefore, it is only multiplied 4n to the n number of messages received.

In the end, the result of the evaluation of complexity of algorithms 3 and 4 is similar to what was the result obtained in the end of chapter 3 as it is shown in the equation 4.4.

$$\theta(n^2) \tag{4.4}$$

4.1.5 Experiments

Seven different virtual machines in the BDCF platform were used as the test bed for the experiments of the scenario where an elderly person who is performing other activity than sleeping needs to be guided to go to bed. Although it is a virtual environment, the experiments were conducted to evaluate the behavior of the system. The specific configuration for the seven virtual actors is depicted in the table 4.1. Data and control is not allowed to be centralized. Consequently, the only actor that knows which capability they have is only themselves given that the implementation is using the third layer of the HARMS model, as it was explained in the previous chapter. To be more specific in explaining that, the only way to implement indistinguishability is that actor$_1$ is the only one that knows what capabilities it possesses. The same for actor$_2$, actor$_3$, and all of them. Actors from 1 to 3 are configured with the capability of leading and model checking capability as well. The actor 6 acts as the patient and all the other actors have their own capabilities assigned. The capability of patient is a logical capability, given that it does not represent any specific activity to perform, but just to identify which actors have it configured. The last capability is still configured with indistinguishabiilty, even though the patient will always be the same, but it was

Table 4.1.
Capabilities by actor for HARMS implementation

No	Name	Description	Actors						
			001	002	003	004	005	006	007
1	harm-000	HARMS Java app	yes	yes	yes	yes	yes	yes	yes
2	harm-001	Ability to send messages	yes	yes	yes	yes	yes	yes	yes
3	capa-001	Ask for capabilities	yes	yes	yes	yes	yes	yes	yes
4	capa-002	Reply for asked capabilities	yes	yes	yes	yes	yes	yes	yes
5	lead-001	Determining master leader	yes	yes	yes	no	no	no	no
6	lead-002	Gets master lead of the execution	yes	yes	yes	no	no	no	no
7	lead-003	Assign guide, receiving location patient	yes	yes	yes	no	no	no	no
8	lead-004	Send message of result to second best	yes	yes	yes	no	no	no	no
9	pati-000	Identification as patient	no	no	no	no	no	yes	no
10	loca-001	Determine location of X actor (RFID)	no	no	no	yes	no	no	no
11	guid-001	Guide patient to specific activity	no	no	no	yes	no	no	yes

decided like that to follow a full indistinguishability scenario. Following that premise, the capability of patient is also requested to identify the actor that represents the patient. That actor that has the capability of patient assigned is used to determine the location of the patient that that actor represents.

4.2 Monitoring hazardous situations within a smart home environment

Several applications of Ambient Intelligence (AmI) environments have been developed in the context of monitoring hazardous situations within smart homes. For instance Skubic et al. in [131] implemented and monitored the activity for a space of time of over 2 years to residents in 17 apartments of an elderly care facility. Their purpose was to monitor and to find patterns to early detect possible alerts such as falls and extracting patterns of the usage of time for the people living within those apartments. While it is true that aging population is a concern for various countries in different continents, solutions for smart homes are targeted not only for people with those special needs. One of the reasons to develop this kind of solutions in a broader sense is to provide tools that are also created for people that are not suffering of any kind of disability.

Consequently, in this section, a framework is defined where while implementing the HARMS model that was able to set the environment to monitor a kitchen place. An special case was taken as the specific scenario where it could be used a gas stove, which given its own characteristics, it presents special hazards even for people with no special needs. Cases of house or building fires are countless where the ignition point has been a stove such as the one mentioned earlier in this paragraph.

4.2.1 Problem definition

Figure 4.4 shows the sequence and interaction diagram that all the actors need to perform in order to monitor a safe kitchen environment. However, the different activities could represent more than only one interaction or message between actors.

For instance, the actor that will perform the data analysis, in order to retrieve the information regarding the most recent value of a sensor needs to receive the raw data from the sensor that will be assigned to sense that variable.

The basic constraints for a problem as such are:

- The scenario will include a set of actors that may agree to cooperate towards solving the specific problem of monitoring a kitchen environment.
- There should be at least two or more actors able to accomplish each specific capability, with the exception of the actor that will count as the care giver, which normally it is a human.
- The surrounding environment will be specifically a set of a kitchen in a normal apartment.
- The stove has to be one that works with gas in any denomination, such as butane.
- The specific point to trigger the scenario is the turn on, turn off of the burners of a gas stove.
- There could be cases where the stove may not ignite, however, the gas may be flowing causing possible problems leading to a fire hazard.
- The end user could be leaded to turn off the burner by any means. For example by playing the sound of a string trough a TTS program.
- The different variables to measure should be a combination of at least one related to possible fire hazards, such as smoke, gas, and temperature.
- The solution should include the implementation of a multi-actor stack model that assures indistinguishability of the actors such as HARMS.

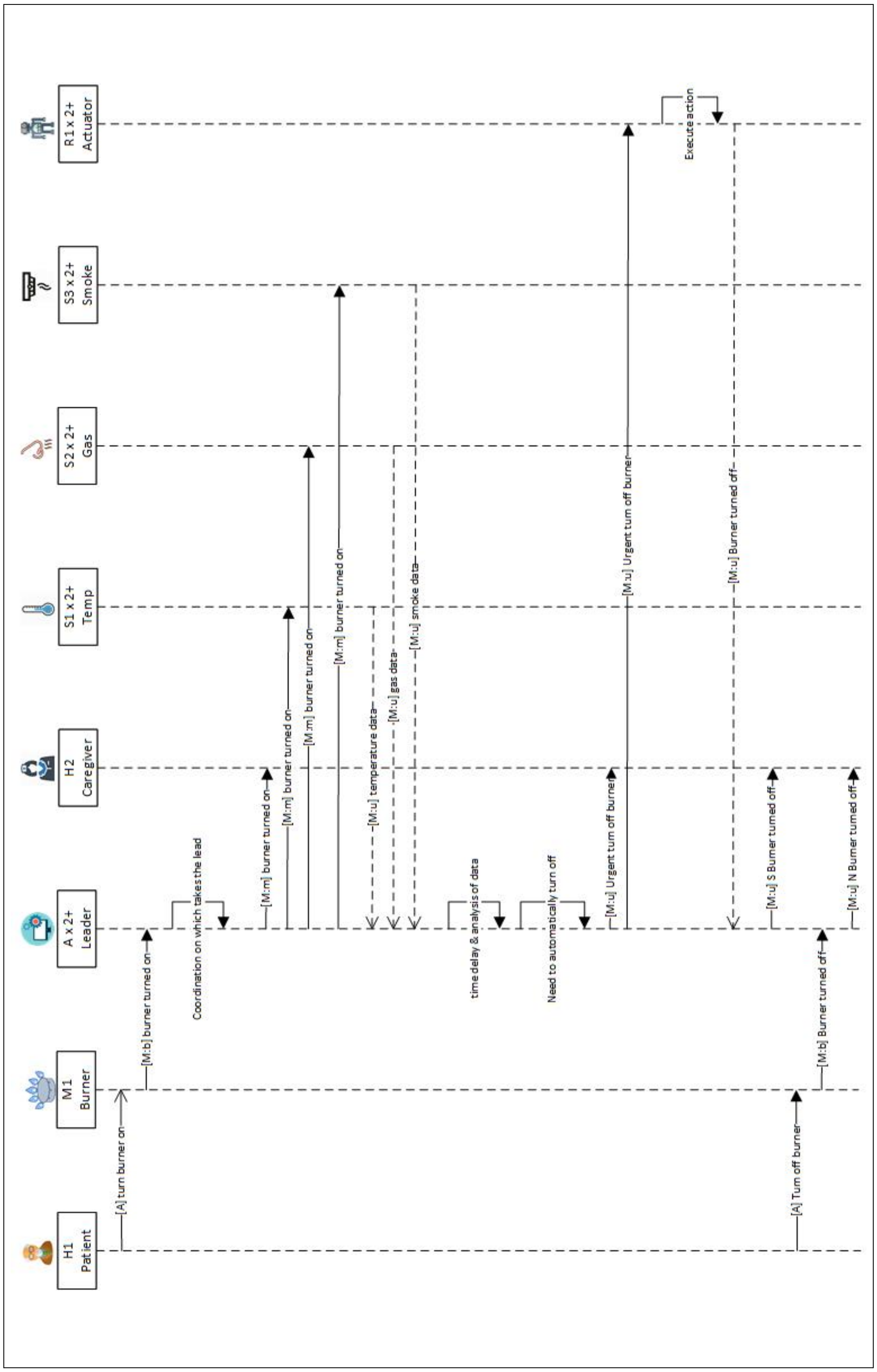


Fig. 4.4. Safe cooking interaction diagram

4.2.2 Model definition

Figure 4.5 shows a macro view of the processes taking place for the safe kitchen scenario. The four different processes that take place are:

- In the case of the determining leader was implemented exactly the same way that it was implemented for the example of guiding a human to perform an action.

- The process denominated lead start is analogous to the process defined in the previous example as team formation, however, the process takes different configuration. Hence, it requires a different definition given that different actors take action due to the specific capabilities that each actor possesses and are required by the solution. Within this process, the assignation of activities to different actors will take place.

- The next big process is called sensor data analysis, which represents the important part where there is at least one actor that performs the analysis of the data. Such data will be received from other actors concerning the values that the sensors capture in run-time. The data gathered while the system is running will be compared to some values previously stored for allowed gaps (e.g. minimum and maximum values) for each variable sensed.

- The last (not external) process that will take place is the scenario in question is the one going by the name of perform action. Specifically, this process corresponds to the activity that an actuator will perform. For instance, a speaker could play the sound of a string using TTS technology. Other example could be to have a humanoid robot that goes to turn off itself the burner. However, due to the lack of time, it was decided to use the first example.

It is understood that both, the start, and end process flags are performed possibly by external actors to the complete solution. In other words, the start presented in red color in the figure 4.5 takes places when a contact sensor perceives that the burner

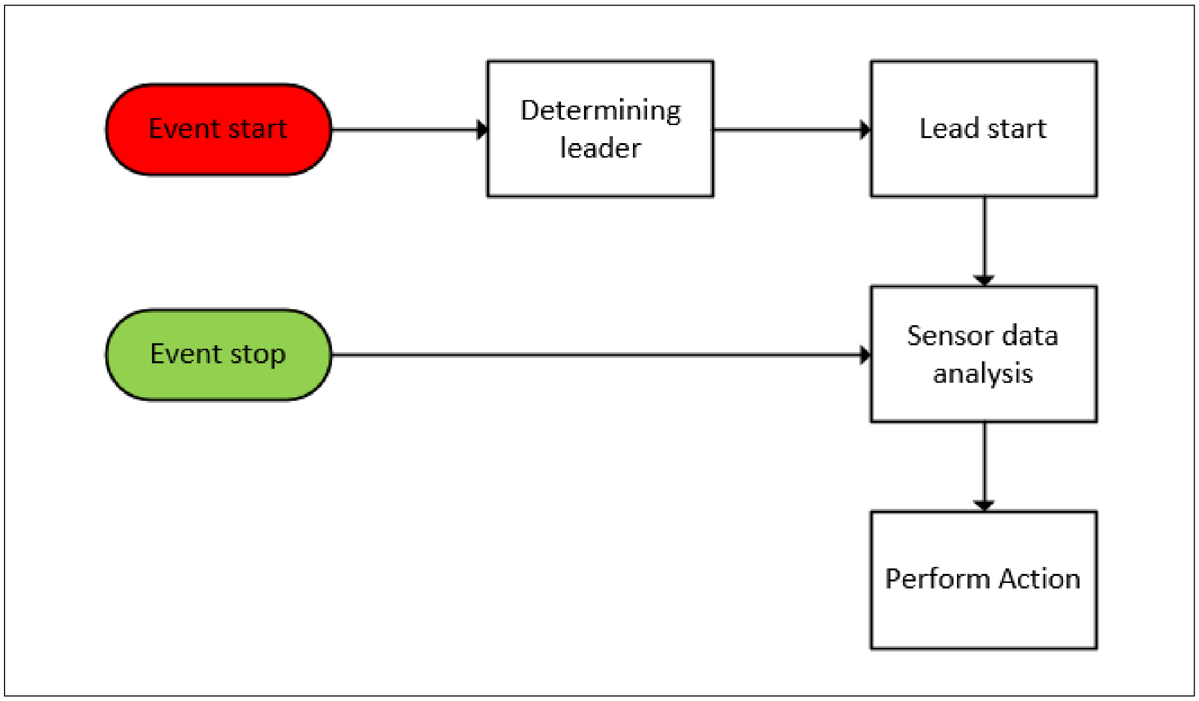


Fig. 4.5. Safe Kitchen Processes

gauge has been turned on. The same case will happen for the one presented in green color, corresponding to the event when the burner has been turned back off, either by the end user, or any other actor of the scenario.

4.2.3 Algorithms definition

It was considered necessary to define the algorithms for two of the big processes present in the complete solution. The first to define is the *Lead start*, which starts with values needed from the initial execution of the process. That means the time when the last setup happened. Values required before the algorithm starts are the ones that let the process know which sensors are already implemented and working in the complete solution. Next, in line 1, the actor care giver is found out by calling the function that determines it within the ones that have the capability 'skit-care-001' configured in their own capabilities. In line 2, it is verified if the actor exists or not among all the actors in the HARMS configured network. If it does not exist, the process finishes right after displaying a notification to the user that it was not able to continue given that situation. In line 6, the actor that will perform the data analysis is determined. That action leads to the same verification that took place for the care-giver. Then, the same definition of each specific actor of the implemented variables will be specified. Due to the lack of space in this document, it was only presented in the algorithm the temperature actor. However, in the program it has the complete code for each actor. The only variant for the validation if the actor exists is that it also verifies if the specific sensor was implemented. The value corresponds to the one that is required to exist since the last configuration of the actors took place. After determining all the actors, the process will start sending the variables of the run to the analysis actor. Such variables are, which are the actors for specific action and the values of which sensors are active in the scenario. Then, the instruction to get the specific information of the sensor and be sent directly to the actor that will perform the data analysis. This set of two instructions is written in the code for all the sensors. In spite that in the algorithm presented it is omitted for the other two sensors. Lastly, the command sent to the actor in charge of the data analysis to start performing it is sent. Consequently, the lead is then transferred to that actor which will be coordinating all the next efforts. Specifically the actor with the analysis sensor data capability assigned.

Algorithm 5 leadStart()

```
Require: $\exists tempImplemented$                         ▷ Is temperature implemented?
Require: $\exists gasImplemented$                                 ▷ Is gas implemented?
Require: $\exists smokeImplemented$                             ▷ Is smoke implemented?
 1: $careGiverActor \leftarrow$ findOutActor($'skit - care - 001'$)
 2: if careGiverActor == '' then
 3:     ShowMessage('No actor care giver, process aborted')
 4:     Return
 5: end if
 6: $analysisActor \leftarrow$ findOutActor($'skit - sens - 0 - 1'$)
 7: if analysisActor == '' then
 8:     ShowMessage('No analysis actor, process aborted')
 9:     Return
10: end if
11: $tempActor \leftarrow$ findOutActor($'skit - sens - 003'$) ▷ for all sensors (gas, smoke)
12: if tempActor == '' and tempImplemented then
13:     ShowMessage('No temperature actor, process aborted')
14:     Return
15: end if
16: sendVariable(analysisActor, 'actorAnalysis', analysisActor)        ▷ Parameters:
    actorTo, variable, value
17: sendVariable(analysisActor, 'actorTemp', tempActor)      ▷ for all sensors (gas,
    smoke)
18: sendVariable(analysisActor, 'tempImplemented', tempImplemented)  ▷ for all
    sensors (gas, smoke)
19: if tempImmplemented then                        ▷ for all sensors (gas, smoke)
20:     getInformation('temperature', tempActor, analysisActor)    ▷ Parameters:
    sensor, actorSource, actorDestination
21: end if
22: SendCommand(actorAnalysis, 'start')
```

The way that the sensor values are verified is shown in figure 4.6. In the case of the analysis of the sensor data, the algorithm 6 starts calling the functions to get the information corresponding to the context values. Then, it directly gets into a while loop which will keep running until the value running is set to 'no'. It is important to mention that the value of that specific variable changes only by external commands. Then, inside the while loop the values of each sensor will be extracted and verified, if they are implemented. The comparison will be between the current value, which will be constantly updated with the values received from the actor that is in charge of that activity in the run. If the current value goes beyond the allowed gap, then it starts the call to the process of performing action of 'Turn off the stove'. Such action, as it was mentioned before, it can be as complex as needed. However, for this experiment, the selected was for the simplest, which is to play the sound of a string using a TTS tool. It is important to note in this algorithm that all the values that the actor will get are the ones that are stored in its own memory. However, those values are updated when receiving the values read by the different sensors by normal message communication through HARMS. This allows the indistinguishability and decentralization of the parts.

Algorithm 6 sensorDataAnalysis()

```
1: getContextValues                          ▷ running, max, min, actors, etc
2: while running do
3:     if tempImplemented then               ▷ for all sensors (gas, smoke)
4:         currentTemp ← getValue('temperature')
5:         if currentTemp > maxTemp then
6:             performAction('turn off the stove')
7:         end if
8:     end if
9:     getContextValues                      ▷ specifically, running variable
10: end while
```

4.2.4 Experimental setup

For the experiments, in this section it was determined to use 8 different actors. Each node was set up with its own different capabilities. except the two actors with leading capability which also share other capabilities. It was decided like that given

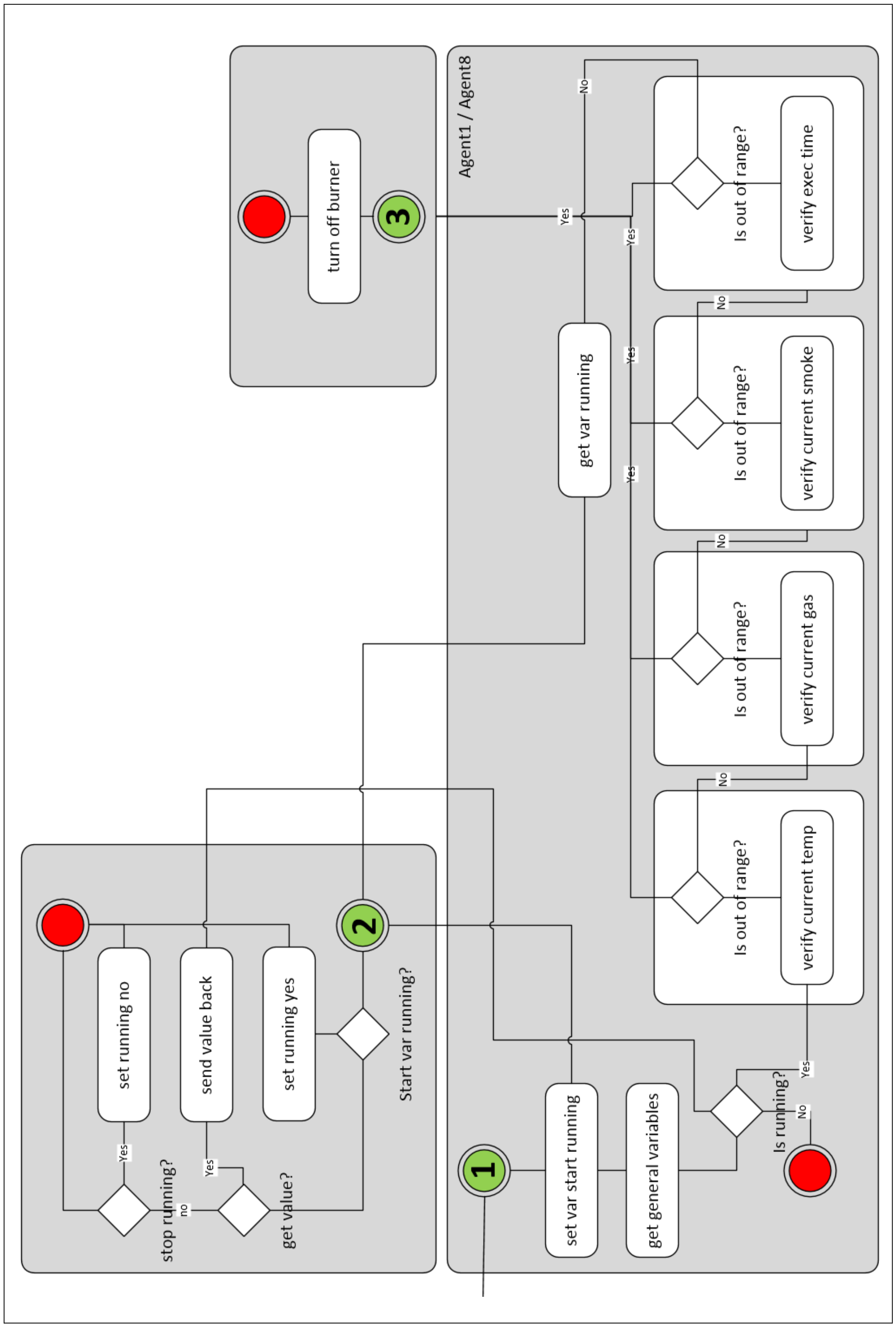


Fig. 4.6. Data Analysis Component Workflow

that there were available only 8 actors to work with. The reason to count with at least two actors with each capability was in a matter of redundancy which will be taken advantage of, while assuring the indistinguishability feature. The capability that is only assigned to one actor is the care giver due to the reason that a care giver is a person responsible to oversee the activities and status of the user and the environment being monitored.

Hardware and software configuration

Each of the eight different actor, used for the experiments, was configured according to its own function. There were two actors with gas sensor, two other actors with temperature sensor, two more actors configured with smoke sensor capabilities, and two more actors that were configured with more than one capability. The way that each of those configurations were made are documented next.

Common actors configuration All the actors share a basic specific configuration in terms of hardware which is listed as follows:

- Raspberry Pi 3 (RPi3) Model B Quad-Core 1.2 GHz 1 GB RAM with On-board WiFi and Bluetooth Connectivity.
- MicroSD card of 32 Gb used to store the different specific software and configuration of each of the actors.

Likewise, in terms of software and configuration, all actors were configured with the following list of programs:

- Raspbian as the oficial supported operating system for Raspberry Pi. The version downloaded and installed was 4.14, which comes with the installation package denominated NOOBS V2.7.0 The basic installation contains software and programs that were used as:

  - Java 1.8.0_65 that is used for the execution of the general program of HARMS
  - Python version 3.5 that was used to execute the HARMS programs

- Network configuration with an IP address used by the HARMS model to allow the communication between actors
- SSH connectivity and configuration to let a general actor to coordinate the initial execution of all the actors to put them ready to start each time that the scenario was planned to be executed
- Java binary files of HARMS that contains the implementation of the first two layers (Network and communication). It also contains the interface for the third layer.
- Different python programs that correspond to the basic implementation of the HARMS model. Those programs were developed either as web services or as normal executable programs needed for the interaction layer to work properly.

Actor with leader, data analysis, and actuator capabilities Apart of the basic configuration of all the actors mentioned above, the actor with leader, data analysis, and actuator capabilities was configured with a speaker using Bluetooth connection. In terms of software it was covered in the Python program corresponding to the capability of actuation. It specifically uses a text to speech (TTS) library to play the sound of an instruction that will be given to the end user. An example of the instruction that the speaker will play could be sent through the communication layer to the actuator actor such as: *"Please, turn off the stove burner because the temperature sensor is perceiving a value of 150 degrees, which is higher than the threshold of 100 degrees".*

Actor with leader, data analysis, actuator, and care giver display capabilities This type of actor, has the complete same set of capabilities and configuration

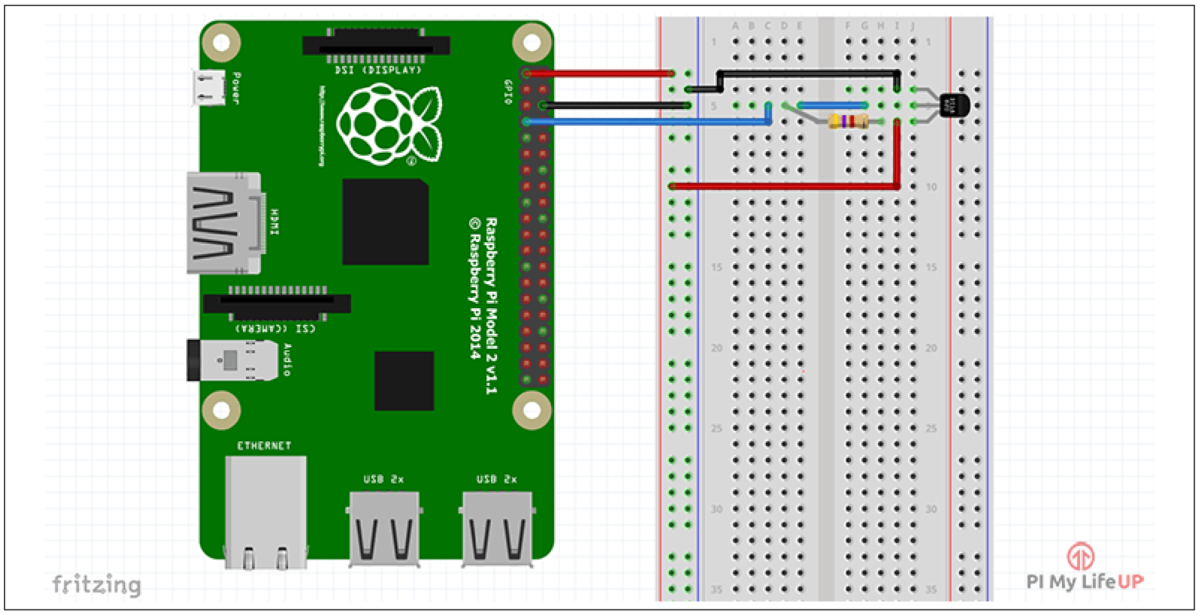


Fig. 4.7. Temperature sensor configuration

of the one just mentioned in the previous paragraph. However, it only has one more capability assigned which is the display of the status of the user and the environment being monitored. The display is used as a medium to provide online feedback of the activities and variables related to the system that the care giver is in charge of overseeing. As an example, when the end user turns on and off a stove burner, this actor will receive a notification containing that information.

Actors with temperature sensor capability The actors with temperature capability were implemented following the online instructions found in [132]. The specific circuit configuration for the sensor recommended in the same source is shown in figure 4.7. The actors with this configuration also include the common configuration of all the actors mentioned above as the common actor configuration.

In terms of hardware, the list of parts used for each of the actors with temperature capability is detailed as follows:

- DS 18B20 Temperature Sensor

- 4.7k Resistor
- GPIO breakout kit
- Breadboard
- Breadboard Wire

In the same way, the actors with temperature capability were accommodated with the program in Python that receives the values in a specific general-purpose input/output (GPIO) port. In the implementation that was performed for the set of experiments it was used the port number 4.

In terms of calibration of the DS 18B20 Temperature Sensor. It was used an average of the values presented in a range of 30 minutes by both of the devices. The specific value obtained in a controlled environment like the apartment where the experiments took place with the stove turned off was 76.6616 Fahrenheit degrees. Consequently, in the system it was set up a 10% extra or 84.32776 Fahrenheit degrees as the maximum threshold allowed for the experiments performed. Even if these values do not reflect a real scenario of a maximum value representing a hazard of a fire. They were considered as a change noticeable enough for the systems for the experiments to perform.

Actors with gas sensor capability The way that the actors with gas sensor capability were implemented is documented in [133]. The circuit configuration documented in the link mentioned above in this paragraph is also as shown in figure 4.8. The actors with this configuration also include the common configuration of all the actors mentioned above as the common actor configuration.

In terms of hardware, the list of parts used for each of the actors with gas capability is detailed as follows:

- MQ-2 (Methane, Butane, LPG, smoke) sensor
- Analog-Digital Converter (8 Ports)

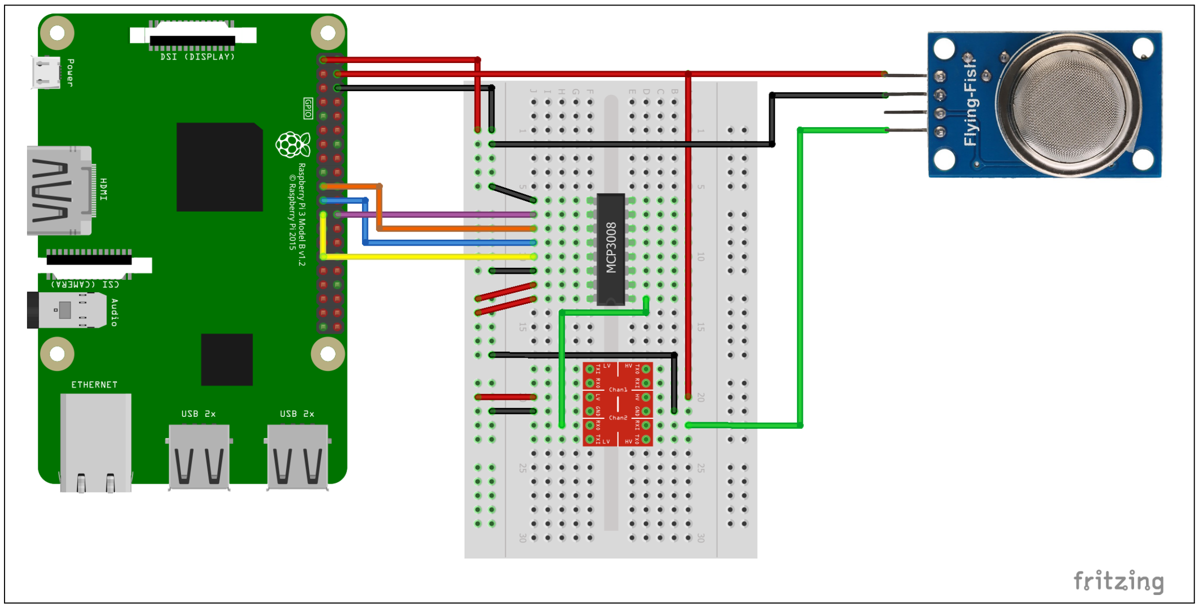


Fig. 4.8. Gas sensor configuration

- 5V to 3.3V Logic Level Converter
- Breadboard
- Breadboard Wire

In the same way, the actors with gas capability were accommodated with the program in Python that receives the values in a specific serial peripheral interface (SPI) port. In the implementation that was performed for the set of experiments it was used the analog port number 0.

For calibration of the MQ-2 gas sensor, before the real tests took place, and given the possible hazards related to manage a flammable element within an apartment setting, it was considered to take an initial reading of 30 minutes without the gas gauge open. The average value thrown by both readers or sensor was of 0.0077195. Consequently, in the same way, a 10% above that value, namely 0.00849145 was used as the upper limit allowed for the conducted tests.

Actors with smoke sensor capability The smoke sensor configuration follows the same configuration than the gas sensor, since the MQ-2 sensor reads gas and smoke at the same time. What was done is that the capabilities of sensing smoke were configured to only two of the actors as in opposition to the four actors that count with the physical capability.

For calibration purposes of the MQ-2 smoke sensor, it was run the reading of smoke value for a space of time of 30 minutes without having the stove burner functioning. The value thrown was 0.0148069, for both sensors. It was calculated a 10% value above the average or 0.01628759 to consider it as the maximum allowed value of smoke to have inside a closed environment such as an apartment complex, where the experiments took place.

Actors configuration for experiments

Table 4.2 shows the basic configuration that was set up for each actor. Accordingly, figure 4.9 also shows the following distribution of capabilities that will be explained more in detail later in the chapter:

- Actor1 is configured with the capabilities of leader, data analysis, and actuator
- Actor2 is configured with the capabilities of gas sensor
- Actor3 is configured with the capabilities of smoke sensor
- Actor4 is configured with the capabilities of gas sensor
- Actor5 is configured with the capabilities of temperature sensor
- Actor6 is configured with the capabilities of temperature sensor
- Actor7 is configured with the capabilities of smoke sensor
- Actor8 is configured with the capabilities of leader, data analysis, actuator, and care giver display

In table 4.3 it is shown in detail all the different capabilities that were required to be implemented in order to make the scenario of safe kitchen work. Each of those capabilities are related with each of the actors. Also figure 4.9 shows the configuration diagram, where icons are used to represent the capabilities assigned to the different actors. In the same figure, it is shown that all the actors process any interaction using the third layer of the HARMS model. That process provides standardization within HARMS implementations to assure the effectiveness of the interactions between actors. In other words, when a message is received through HARMS stack, the layer 3 is the one in charge of letting the actor know what should be done with the received information, in the case that the capability is configured for that actor. The flow of the process when a message is received is documented at

Table 4.2.
List of actors in Kitchen Monitoring

Peer	***IP Address***	***role***
Actor1	192.168.1.18	leader, data analysis, actuator
Actor2	192.168.1.19	gas sensor
Actor3	192.168.1.20	smoke sensor
Actor4	192.168.1.21	gas sensor
Actor5	192.168.1.24	temperature sensor
Actor6	192.168.1.25	temperature sensor
Actor7	192.168.1.22	smoke sensor
Actor8	192.168.1.26	leader, data analysis, actuator, and care

the end of the previous chapter. In the same figure, it can be seen that each of the actors is composed by at least the Raspberry pi configuration plus each capability. As an example, Actor1 counts with the capabilities of leading, which is represented by the machine with the different cogs; the other capability it possesses is the actuating part, represented by a small robot. Another example would be Actor2 that possesses the capability of gas sensor which is represented by a nose that can detect smells and odors. A third example is the Actor3 which has a thermometer to represent the temperature sensor capability. The only actor that posses the capability of care giver is Actor8, given that it is considered that the care giver has to be a person. Such capability is represented by a nurse drawing.

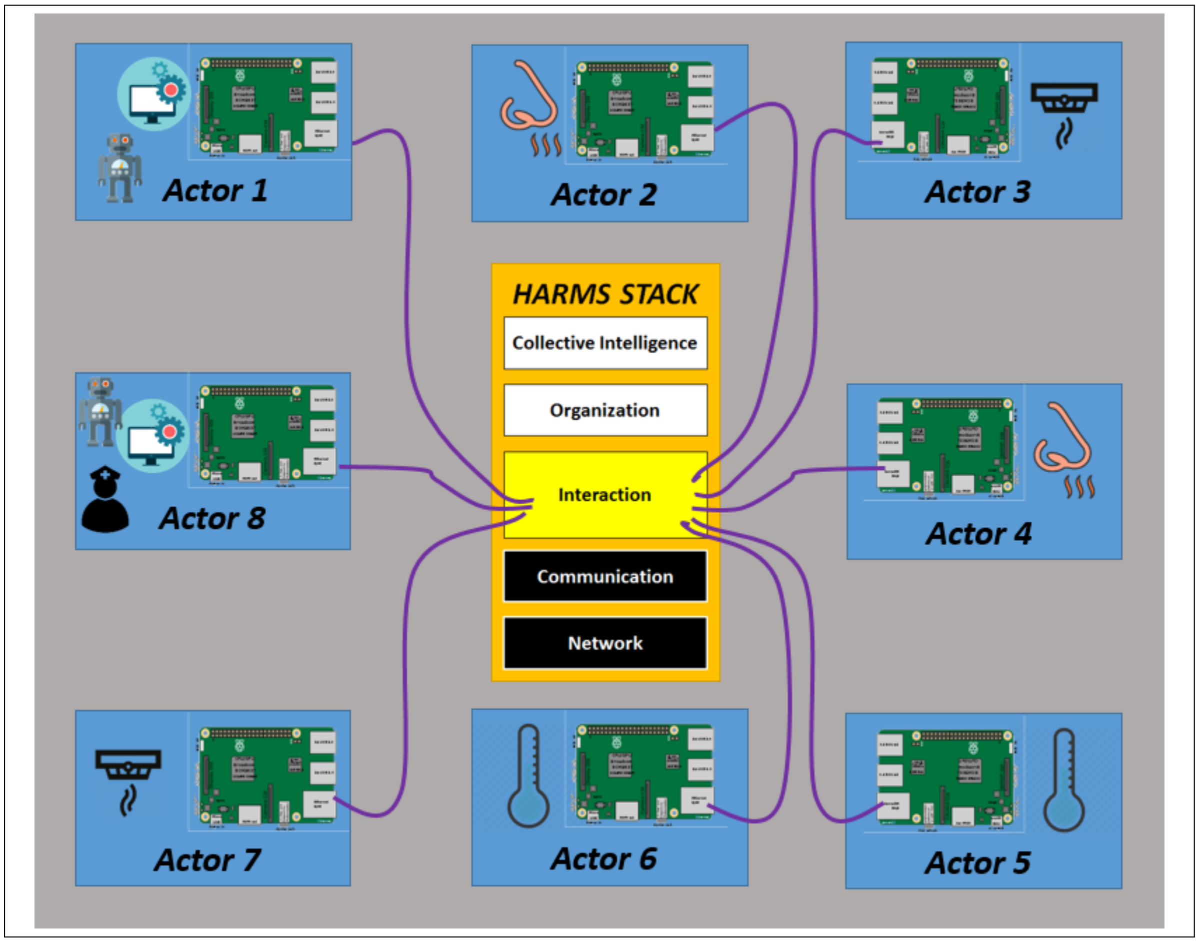


Fig. 4.9. Safe kitchen actors diagram

Table 4.3.
List of capabilities assigned by actor in Safe Kitchen Scenario

Capability	***Description***	***Actor***								***Name***	***location***	***run at start***
		1	***2***	***3***	***4***	***5***	***6***	***7***	***8***			
harm-000	HARMS java application	X	X	X	X	X	X	X	X	HARMS.jar	\	yes
harm-001	Ability to send messages	X	X	X	X	X	X	X	X	sendMessageSocket	\other_src\harms	no
capa-001	Ask for capabilities	X	X	X	X	X	X	X	X	askCapabilities	\other_src\capabilities	yes
capa-002	Reply for asked capabilities	X	X	X	X	X	X	X	X	replyCapabilities	\other_src\capabilities	yes
skit-lead-000	Timer execution that will trigger the scenario to run	X							X	combinedLead	\other_src\skit-lead	yes
skit-lead-001	Leader capability of safe kitchen scenario (negotiation)	X							X	determineLeader	\other_src\skit-lead	yes
skit-lead-002	Execute leading capability of safe kitchen scenario	X							X	Lead	\other_src\skit-lead	yes
skit-sens-001	Stove burners capability (on/off sensor)?		X	X						OnOffBurner	\other_src\skit-sens	yes
skit-care-001	Caregiver capabilty (receive messages)								X	receiveMessageCareGiver	\other_src\skit-sens	yes
skit-sens-000	Sensor Variables (Get / upd, etc)	X	X	X	X	X	X	X	X	sensorVariables	\other_src\skit-sens	yes
skit-sens-001	Receive sensor info	X	X	X	X	X	X	X	X	ReceiveSensorData	\other_src\skit-sens	yes
skit-sens-002	Read sensor info (sensor or general info)	X	X	X	X	X	X	X	X	readSensorData	\other_src\skit-sens	yes
skit-sens-003	Temperature sensor capability (read & store value)					X	X			readTemperature	\other_src\skit-sens	yes
skit-sens-004	Gas sensor capability (read & sending if value on)		X		X					readGas	\other_src\skit-sens	yes
skit-sens-005	Smoke sensor capability (read & sending if value on)			X				X		readSmoke	\other_src\skit-sens	yes
skit-sens-0-1	Analysis of sensor data	X							X	dataAnalysis	\other_src\skit-sens	yes
skit-actu-001	Actuator (robot / speaker / other) capability	X							X	TurnOffBurner	\other_src\skit-actu	yes

4.2.5 Experiment cases

To test the success in accomplishing the indistinguishability of the different actors involved in the setup, the following different experiment cases were conducted:

- **Case 1: No alarms issued** This case is the one happening when the end user, such as the person with special needs, follows a normal set of actions to cook with no special issue encountered by the system.
- **Case 2: Alarm(s) issued** This case is characterized when the end user, such as the person with special needs, unconsciously forgets or misses to identify a possible hazard. That issue may be noticed by the system, action by which, it starts the corresponding set of instructions in order to avoid an emergency.

In order to conduct the tests in an environment as close as possible of a real test bed, the tests took place in an apartment complex in an ambience of a normal living-kitchen room located in a building of apartments around the city of West Lafayette, Indiana, between the first days of July of 2018.

To evaluate the indistinguishability, a set of 10 exercises were performed, to verify which actor was assigned for each task. According to Harnard [134], the term indistinguishability has its origins in the thoughts about the Turing tests. Where the goal of those test was motivated in such a way that the user would not be able to identify if the entity in the other side of the computer was, either another computer or a human. In these experiments it was expected to assure that the task related to a capability is accomplished independently of which actor performs the activities related to it. It is considered that a good measure of indistinguishability could be not having a 100% of task assignations to the same actor in all the times that the experiment was ran.

For that purpose, within the implementation of the system a file is stored in the actor that performs the data analysis. That file denominated "logConfigFile.json" contains the log of all variables of each execution. By terms of log of all variables it includes but it does not exclude the following list of information:

- **actorLeader** is the actor that was determined by the process explained in detail in the previous chapter.
- **actorAnalysis** identifies the actor that verifies that the run-time variables are within the acceptable parameters to consider that a possible hazard alarm has not been issued.
- The different actors in charge of gathering the information of the different variables during execution. This list includes **temperatureActor**, **gasActor**, and **smokeActor**
- The different variables of what sensors are implemented for each of the runs. This list includes **temperatureImplemented**, **gasImplemented**, **smokeImplemented**
- **lastStart** and **lastStop** correspond to the start and stop time stamps of the run. Those values correspond to the time when the burner was turned on, and the time when the burner was turned off.

Since the *analysisActor* itself can be indistinguishable, the log may be scattered all around the actors. To counterpart that, it was proceeded to gather the log of each actor and merge it to one single file to have all the log data.

4.2.6 Results

Table 4.4 shows the results that were obtained while executing the two different real life scenarios mentioned as experiment cases. As mentioned before, if the percent of the load distribution is more likely to be assigned the majority of the times to a specific actor, then that would imply that the indistinguishability measure would also be not effective as expected.

In the case of the leading capability, given that it was implemented with the algorithm of leading selection, it was considered that the indistinguishability value

Table 4.4.
Measuring indistinguishability

Actor	***First 10***	***Second 10***	***Overall***
1	50%	70%	60%
8	50%	30%	40%

obtained corresponds to an acceptable value. However, for the assignation of other activities, that were assigned by the leader, the percentage of assignation was 100% for the same actor all the execution times. Nevertheless, the execution of the activity is performed given that the tasks needed for accomplishing the capability can be performed by more than one of the actors in the configured network of actors.

Such behavior corresponds to the algorithm of broadcast which is a sequential process that goes to send message by message in the order that the actors are in the file peers. It was noticed that the specific case of the experiment case does not affect for the indistinguishability of assignation of tasks to the actors.

4.3 Survivability as feature of MAS

Within recent times the development of systems that will be in close contact with living creatures, specially humans, has been increased. In spite if the person is or not lacking of determined abilities, solutions have been oriented to permit human-system or human-robot collaborations. Due to their distinct functionalities and the problems they are commended to work on, those systems can be categorized as complex systems. However, it comes to the focus concern in this dissertation what could happen right after the moment when the system encounters either external events or internal errors that could affect the plan of action of the system, hence the results. Consequences could be disastrous in any of the two possible outcomes of a system without a survivability feature. On the one side, the system could stop working, and imagining that the end users that it are directly interacting and depending on the system could not be aware of the situation, it could lead to a stressful moment for them. On the other side, if the system overpass the situation and guides the scenario not to the desired original final goal rather than other point. Drivers for the need of survivable solutions could be based in the premise that there is no place for pure reliable methods, given that systems such as AmI are immerse in a very dynamic environment. Destructive events are not predictable enough either by probabilistic methods or calendar calculations.

4.3.1 System survivability

Efforts to measure survivability as the one mentioned in previous sections can be as the natural thought of the effectiveness of a system after a given set of impacts [135]. Such definition is rather vague and the focus was set in the situations that may appear during run-time of the systems. In this research effort, survivability is defined as the number of times that the system can accomplish the goals set before run-time in spite of finding unknown problems. The survivability feature is based in the assumption that the system is well documented from its inception through a valid model that works in all situations to what it was designed to work. Also, based on the possibility that future robotic systems may encounter a very dynamic environment which could bring many variables to play during run-time and may not be know before execution time. In those cases, the only effort to find a solution to an unknown issue responds to the natural instinct of survivability.

4.3.2 Algorithms and complexity analysis

Survivability

To accomplish the survivability feature, it was implemented using the first three layers of the HARMS model. Survivability is divided into two parts.

First, a simple mechanism is used to be verifying the current status of the system and validate it with what should be the state of the system at that moment generates awareness of possible problems during run-time. Such elementary mechanism provides the characteristic expected of the multi-actor system to identify errors not easy to find during the analysis and design stages of the system. For the error or uncertain situation discovery, it is proposed to have a time interval that is considered acceptable for each specific capability to fulfill the related activities. Consequently, the time interval to have either finished the activity or to have received answer from other actor is part of the information stored in the capabilities file and will be active for each or certain capabilities during run-time. Messaging is the basis of the assumption for the different actors to communicate and detect possible errors encountered. For example, a maximum wait time will be assigned when an actor sends a message containing an instruction to execute activities related to a capability. There could also exist capabilities that have no maximum wait time to receive the desired answer. However, for the ones that have the timer, if that threshold is reached without receiving any of the expected answers, it will start the process of self-healing as a part of the survivability feature. Such maximum time will be set as an average of the time for that activity to happen in previous executions. Identifying the right time when the execution does not match what the model of the system says it should be doing is important from the stand point of issue detection. That is important due to be able to know in what state of the model the changes could start bein doing. In the solution proposed in this research effort the focus is on finding a possible solution. That is the reason why the simple on-line verification is used for error or issue detection, avoiding to make exhaustive studies of what is in the environment. Also, another reason is that a detailed analysis of the environment variables is out of the scope of this research effort.

Second, at the time that an issue has been identified, the self-healing process as a survivable feature will start. What is expected to receive back from the execution of the self-healing process is a possible solution, including a path course of steps to follow to still achieve the original final goals. Such solution or set of instructions as solution will be confirmed if it is possible to be executed with the capabilities existing in the actors that are in the network at that time. In other words, solution corroboration process consists of another round of handshaking to reassure requirements of capabilities availability as shown in lines 4 to 7 of the algorithm 7. If the solution provided first is not possible to be executed if the actors present do not count with one of the capabilities needed, the process will corroborate more than one possible solution. Such corroboration also includes the selection of the actors that better suit for accomplishing the new set of actions. The process finishes with the assignment of the new activities to the selected actors.

As stated before, the survivability process will be triggered by the detection of a process that exceeds the maximum time to accomplish the activity set for an activity.

Algorithm 7 survivability(actor A, event error, model OriginalModel)

```
1: if Error != 0 then
2:     $A$.SolutionFinder(OriginalModel)
3:     $As_{new} \leftarrow$ BetterActor($As$, [$Assistive$] + "$capability$")
4:     Directive($A$, $As_{new}$, assistiveAction)
5: end if
```

The four processes presented in [103] and shown in figure 4.10 are:

- **Base model generation** is the assumption that the model that is running is always updated and ready to be sent as parameter in case of needing it.
- **Current activity verification** implies the process of validation of the current status of the system against the status that the model says it should be happening during all the time that the system is running.
- **Solutions generation and evaluation** auto-generates model variations and coordinates models validation through a model checker, such as NuSMV.
- **Solutions corroboration** insures the viability of any of the actors that could receive the assignment to accomplish the new set of steps.

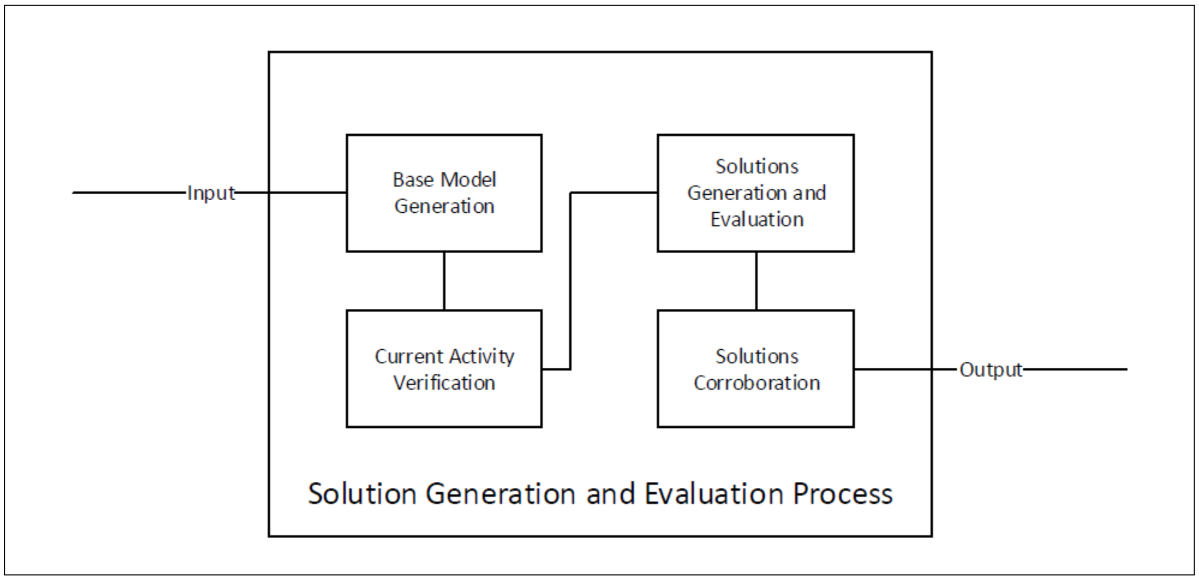

Input
Base Model Generation
Current Activity Verification
Solutions Generation and Evaluation
Solutions Corroboration
Output
Solution Generation and Evaluation Process

Fig. 4.10. Workflow of survivability process

An actor adaptive task-based model was presented in [79], in that work, authors defined the model as a tuple $robot = \{S, T, I, \rightarrow, AP\}$, where, S is a set of states, T is a set of transitions, $I = S_0 \in S$ is an initial state, $\rightarrow \subseteq STS$ is a transition, and AP is a set of atomic propositions. The point of view of that work differs to the one presented in this research effort in such a way that here it is based the survivability feature in the communication taking place between all actors of a multi-actor system using the HARMS model.

Other strategies to formally verify open systems are contrasted in [136]. Authors in that work present a comparison between module checking CTL and model checking alternating-time temporal logic (ATL). The analysis discusses that there are properties that can be expressed in module checking that can not be represented in ATL.

Self-healing

The self-healing process, as shown in 8 is compound of four different sub-processes. Creation of several different models based on the original take place at first. Such changes to the model should take place in the specific state where the issue was encountered during run-time. An exhaustive evaluation of all the possible paths of each of the different models created is the following step. For that evaluation is used the NuSMV tool [2]. This part will be explained in more detail in the following section. After the results of the NuSMV tool are returned, a comparison in terms of time response is executed. In this research effort, the value to compare is the time response, however, in future work it could be extended to evaluate other variables. The model execution that got a better time response is the solution that will be proposed to the agent that requested the execution of the self-healing process. After the result is returned to the general leader, it is followed by a corroboration or a new handshake to determine if there is another actor that can execute the solution provided by the self-healing part. It is assumed that there would be other actors that can receive the assignment of the missing activities to accomplish the original final goal as a survivability feature. However, it is not assumed that there would be actors waiting to receive a new instruction. Expressing it in a different way, since the cost of the actors could be elevated, it is assumed that actors will be working in other activities such as cooking, surveillance, and actor support, nonetheless they will be able to respond based in priorities. Such priorities will let them stop doing what they may be doing and go to support a system that needs to solve an issue at that moment. Figure 4.12 presents the flow diagram of the self-healing process.

In figure 4.11, the actors can be found in in the interaction diagram that illustrates how it happens for the algorithm 7, survivability and algorithm 8, self-healing.

Algorithm 8 SelfHealing

```
Require: Parameter of original model
1: ModelVariationGeneration(OriginalModel)
2: ParallelExecution(ModelVariations)
3: $BestSolution \leftarrow$ ResultsComparison()
4: Return(BestSolution)
```

Complexity analysis for the algorithms 7 and 8 is less complex than the ones studied before as is shown in the equation 4.5.

$$n + 2 \tag{4.5}$$

n in that algorithm is related to the number of iterations of the loop happening to obtain the right actor in terms of availability and possesing the capabilities required. The coefficient presented as 2 corresponds to the two following instructions.

On the other hand, for the algorithm 8 the complexity is calculated as it can be stated in the equation 4.6.

$$3n + 1 \tag{4.6}$$

The consolidation of the algorithms 7 and 8 would represent an evaluation result as shown in equation 4.7.

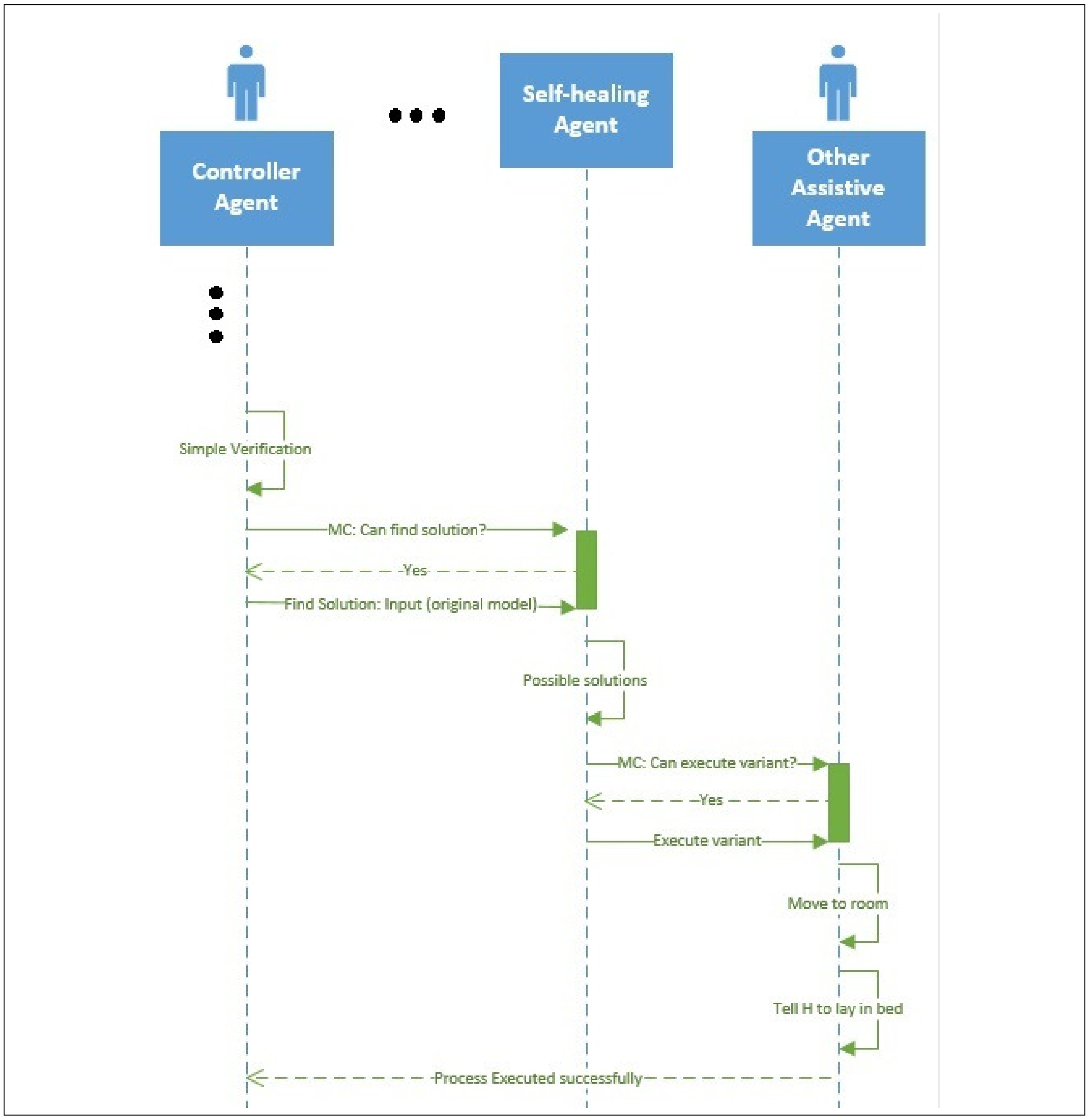


Fig. 4.11. Self-healing actors interaction diagram

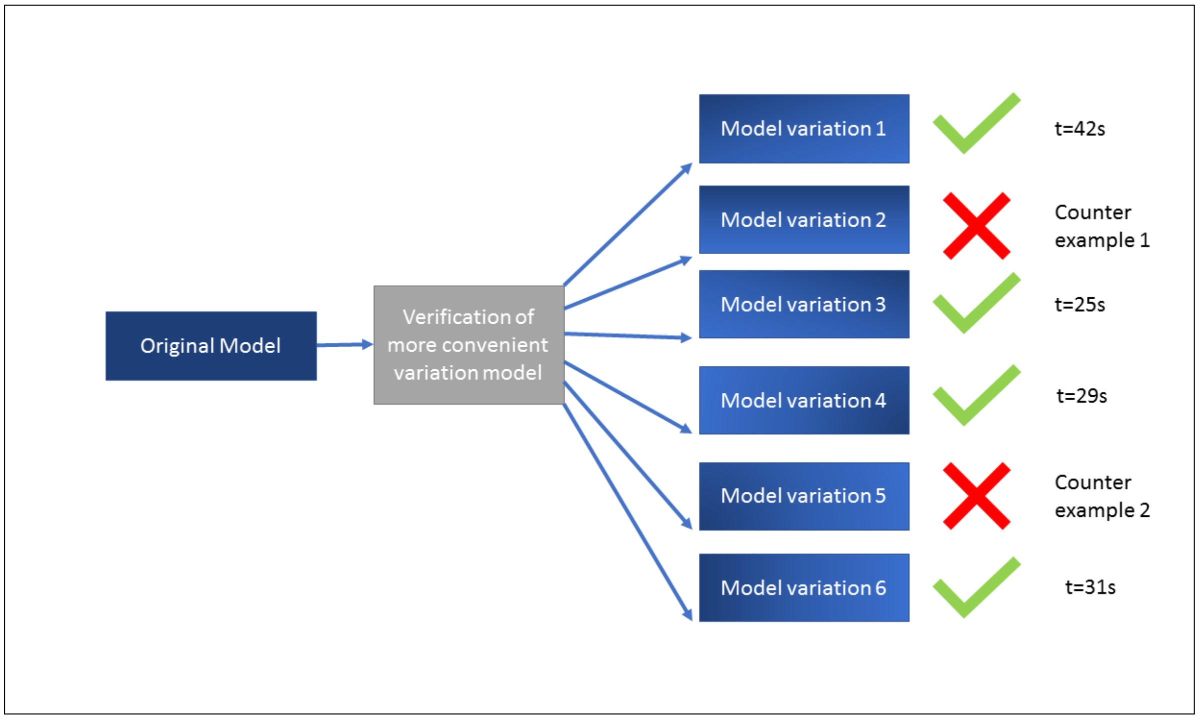


Fig. 4.12. Self-healing original diagram

Table 4.5.
Actors with self-healing capability

No	Name	Description	Actors						
			001	002	003	004	005	006	007
1	modc-001	General model checking (coordinator)	no	no	no	yes	no	no	no
2	modc-002	Model verification (model checking)	yes	yes	yes	yes	yes	yes	yes

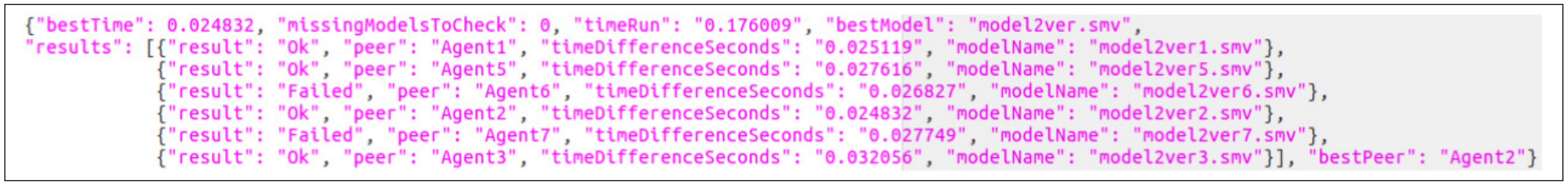
$$4n + 2 \tag{4.7}$$

where one of the 2 coefficient of equation 4.5 converts in the $(3n + 1)$ because it carries out the complexity of equation 4.6.

The final complexity evaluation results in a simple n as shown in equation 4.8.

$$\theta(n) \tag{4.8}$$

Self-healing mechanism The implementation of the self-healing mechanism was possible due to the use of seven different virtual machines obtained from the BDCF platform. Each of the machines worked in an autonomous way. That means that each machine corresponded to an actor with its own autonomy and isolated information including the capabilities configured. The table 4.5 shows that all the actors were configured with the capability of model checking. However, only one of the actors was configured with the capability to coordinate the model checking activity. That capability, as explained before, will perform the modifications of the original model and consequently distribute those models to the different actors that count with the capability of model checking. This process takes advantage of the use of pynusmv library which is executed each time that capability is requested.



```
{"bestTime": 0.024832, "missingModelsToCheck": 0, "timeRun": "0.176009", "bestModel": "model2ver.smv",
"results": [{"result": "Ok", "peer": "Agent1", "timeDifferenceSeconds": "0.025119", "modelName": "model2ver1.smv"},
            {"result": "Ok", "peer": "Agent5", "timeDifferenceSeconds": "0.027616", "modelName": "model2ver5.smv"},
            {"result": "Failed", "peer": "Agent6", "timeDifferenceSeconds": "0.026827", "modelName": "model2ver6.smv"},
            {"result": "Ok", "peer": "Agent2", "timeDifferenceSeconds": "0.024832", "modelName": "model2ver2.smv"},
            {"result": "Failed", "peer": "Agent7", "timeDifferenceSeconds": "0.027749", "modelName": "model2ver7.smv"},
            {"result": "Ok", "peer": "Agent3", "timeDifferenceSeconds": "0.032056", "modelName": "model2ver3.smv"}], "bestPeer": "Agent2"}
```

Fig. 4.13. Results of model check activity of general scenario

4.3.3 Experimental setup

This section is devoted to explain in detail what was the configuration of all the different aspects needed to execute the experiments. Such experiments were conducted to determine the feasibility and benefits obtained while uploading the model checking processing to the cloud in scenarios like the ones mentioned before.

Cloud Tools

BDCF platform was the cloud environment used for the experiments. A number of seven virtual compute resources were configured using the BDCF platform. In the seven virtual machines the different configurations were assigned for the seven different actors running on top of each of the different virtual machines. The machines were configured in terms of platform and software as shown in figure 4.14. Each machine, as shown in that figure, has a public network configuration. Such connection gives the possibility to be managed from outside or from other machines as one of the benefits of the cloud. Web services were configured to allow the communication between the different actors of the experiments. Each call was made using the third layer of the HARMS model, which imply messages based in capabilities. Those capabilities have a specific web service to be executed upon receiving a message of that kind. Those machines were previously configured with two programming environments which were Java and Python.

Java Programming Environment

The HARMS model in the first two layers was implemented using JAVA language. Consequently, the Java programming environment was set up in each of the actors. The HARMS model has been programmed through Java code and requires at least to have the JRE component to run.

Python Environment

For the third layer of the HARMS model it was mostly implemented in Python to allow the web service to be implemented and run smoothly. Each capability has a web service associated to run each time that a message with that code is received. All the code for those web services was made using Python developing environment.

HARMS Model

The problem statement was specifying that the use a multi-actor system was required to accomplish the specific scenarios. Consequently, for the communication an interaction part of this other sub-scenario also used the HARMS model. The use of decentralized databases is one of the key components of the third layer of the HARMS model. Consequently, in the framework proposed in this paper, the first three layers of the HARMS were implemented: network, communication, and interaction layers.

JSON Format Database

The complete information pertinent to each of the capabilities that an actor has configured is located in a JSON format file. With the files configured in a decentralized manner the indistinguishability of HARMS is insured. In other words, each actor has different files that have the configuration of the capabilities that it has configured.

4.3.4 Models evaluation

For the different models' evaluation, model checking will be used, specifically a tool called NuSMV [137]. NuSMV is a temporal logic model checker, which takes as input an automata model and a temporal logic formula. An automata is a finite state machine (FSM) which is used as a model on which it will be verified a specification in a temporal logic formula fashion. The models to be verified will be the different variations created based on the basic original model that represents the system in the

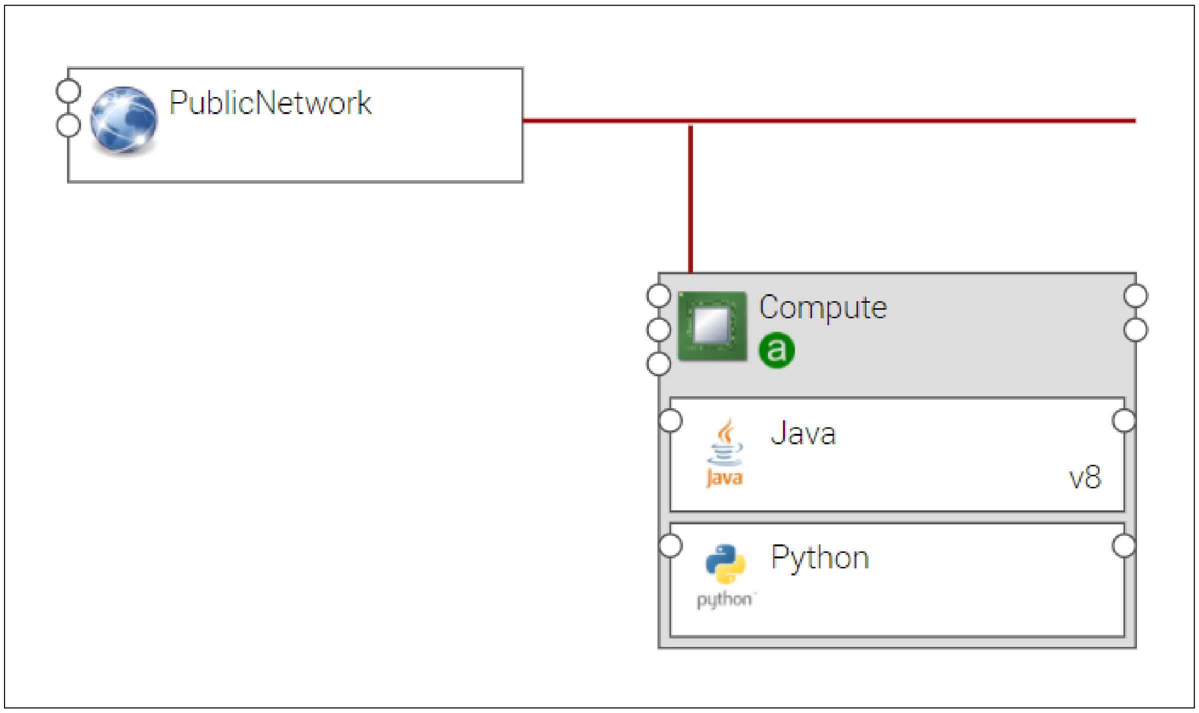


Fig. 4.14. General configuration of virtual machines in BDCF

way that it is running. To mimic the natural behavior of living creatures, the change of the model is set up in a dynamic way where the changes will take places only after the error was encountered. Specifically those changes will be made at the same point represented in the model for the status that the system was at the moment of finding the issue. In other words, the change in the models will be reflected in transitions or states in the model just after the state where the problem emerged.

FSM model

As stated before, to verify the results of time response with the possibility of successfully accomplishing the original final goals a model checker such as NuSMV will be used. NuSMV uses SMV extension files which contain the description of finite state machines and to express a set of requirements in computational tree logic (CTL) and linear temporal logic (LTL) [2]. In this approach, it was used the original model which is the implementation of HARMS discussed in previous sections of this work. The way that was used to represent the model is following the rules of writing an automaton model in the SMV format which is the one required by NuSMV.

The SMV program is subdivided into four different files:

- *Actor localization* RFID readers are actors that correspond to this category and also called as contextual information actors in the previous sections of this document.
- *Actor Guide* is the one in charge of guiding the patient to go to sleep. The term assistive actor was used previously for this kind of actors.
- *Actor Controller* or the leader actor as mentioned in previous sections is the one in charge of leading the complete scenario.
- *Code main* is the code that contains the macro part of the system.

In the code listing 4.1 it is presented the code for the FSM of the localization actor, addressed before as contextual information actor. Basically the actor's states stay the same until the state of the general leader is set to asking for the location.

Listing 4.1 Code Actor Localization (RFID)

```
MODULE agt_loc(ag_ct_state)
VAR
   state        :       {
      idle,
      cal_loc,
      snd_loc};

ASSIGN
   init(state) := idle;

   next(state) := case
      state=idle
    & ag_ct_state=snd_cmd_loc : cal_loc;
      state=cal_loc : snd_loc;
      state=snd_loc : idle;
          TRUE : idle;
        esac;
```

The code listing 4.2 presents the code in SMV representation fro the actor that guides the patient to bed. In the description given before this actor is called the assistive actor. Upon receiving a message stating the capability associated to the actor locator or contextual information, the code listing represents the model pertaining that part of the system. Later on, when the actor arrives to the desired location of the dependable actor it switches state to state that it has arrived and other set of actions will follow such as giving instructions to follow him towards the bedroom. A validation of a positive answer of the patient will be performed in order to continue with the following instruction. And when it is located in the bedroom, it will verify if the person is laid in bed. Finishing with with turning off everything in the bedroom and moving back the robot to the place where it was at the beginning of the execution.

The actor controller or leader actor will lead all the scenario. The code listing shown in 4.3 comprehends the states and transitions related to that actor.

Listing 4.2 Code Actor Guide

```
MODULE agt_guide(ag_ct_state)
VAR
   state :  {
            idle,
            mov_to_loc_pat,
            play_follow_inst,
            mov_to_bed,
            play_lay_inst,
            verify_patient_pos,
            error};

   location     :  {
            anywhere,
            on_track,
            on_target_loc};

ASSIGN
        init(state) := idle;
        init(location) := anywhere;

        next(state) := case
      state=idle & ag_ct_state=snd_cmd_guide
    & location=anywhere : mov_to_loc_pat;
      state=mov_to_loc_pat : play_follow_inst;
      state=play_follow_inst : mov_to_bed;
      state=mov_to_bed : error;
          TRUE : error;
        esac;

   next(location) := case
      location!=on_target_loc
    & state = mov_to_loc_pat : on_track;
          location=on_track
    & state = mov_to_loc_pat : on_target_loc;
          TRUE : anywhere;
   esac;
```

In the next code listing 4.4 it is presented the main code which corresponds to the declaration of the actors.

Hereof, it will be used the Kripke structure for the formal definitions that start as follows.

Listing 4.3 Code Actor Controller

```
MODULE agt_ct
VAR
   state    : {idle,
               rq_other_ct,
               ans_ct_cap,
               dt_ag_ct,
               snd_cmd_ct,
               rq_ag_loc,
               snd_cmd_loc,
               rq_ag_guide,
               dt_ag_guide,
               snd_cmd_guide,
               fwd_complete_msg,
               error};

ASSIGN
   init(state) := idle;

   next(state) := case
      state=idle : rq_other_ct;
      state=rq_other_ct : ans_ct_cap;
      state=ans_ct_cap : dt_ag_ct;
      state=dt_ag_ct : snd_cmd_ct;
      state=snd_cmd_ct : rq_ag_loc;
      state=rq_ag_loc : snd_cmd_loc;
      state=snd_cmd_loc : rq_ag_guide;
      state=rq_ag_guide : dt_ag_guide;
      state=dt_ag_guide : snd_cmd_guide;
      state=snd_cmd_guide : fwd_complete_msg;
      state=fwd_complete_msg : idle;
          TRUE : error;
   esac;
```

Definition 4.3.1 *Define an AAL generic problem actor* $= \{S, T, S_0, \rightarrow, AP, PT\}$ *where* S *represents a set of states that the actor can adopt in any moment of the execution time space. Whereas, a specific state* $s_i \in S$. T *substitutes all the turning points or transitions from one actor's state to another.* S_0 *denotes all the initial actor's states which should be part of all valid* S *states. In other words,* $S_0 \in S$. *A right arrow* $\rightarrow$ *is a transition relation which implies the step from one actor's state to other as* $\rightarrow \subseteq S \times S$. AP *relates a set of atomic propositions* PT *groups the*

Listing 4.4 Code Main

```
MODULE main
VAR
   ag_ct_state : {idle,
                  rq_other_ct,
                  ans_ct_cap,
                  dt_ag_ct,
                  snd_cmd_ct,
                  rq_ag_loc,
                  snd_cmd_loc,
                  rq_ag_guide,
                  dt_ag_guide,
                  snd_cmd_guide,
                  fwd_complete_msg,
                  error};
   ct_1: process agt_ct;
   ct_2: process agt_ct;
   loc_1: process agt_loc(ag_ct_state);
   loc_2: process agt_loc(ag_ct_state);
   loc_3: process agt_loc(ag_ct_state);
   guide_1: process agt_guide(ag_ct_state);
   guide_2: process agt_guide(ag_ct_state);

ASSIGN
   init(ag_ct_state) := idle;
```

validation of possible Atomic propositions depending on the specific actor's state as $PT : S_i \to 2^{AP}$.

In this structure it is insured that the only possible transitions are the ones detailed there. For example, to insure that the system changes of state from guiding while moving the patient to arrived to the bedroom. There could be an extra validation loop where if the distance between the guide actor and the patient does not exceed a maximum threshold. The other possible transition should be the one where the robot has already arrived to a specific location as a temporal goal for it. With that it is possible to validate that the system is behaving as expected, because if the system goes out of what it is expected, could be easily identified as an issue during run-time.

Definition 4.3.2 *Define an AAL generic problem system* $= \{sS, Agt, sT, sS_0, \to, sAP, sPT\}$ *where* sS *represents a set of states that the system can adopt in any* *moment of the execution time space. Whereas, a specific state $ss_i \in sS$. Agt groups all the actors that are in the system at the runtime sT substitutes all the turning points from one state to another. sS_0 denotes all the initial states which should be part of all valid sS states. In other words $sS_0 \in sS$. $\rightarrow$ is a transition relation which implies the step from one state to other as $\rightarrow \subseteq sS \times sS$. And sAP relates to the possible set of atomic propositions which depends on the specific state as $sS_i \rightarrow 2^{sAP}$.*

The general SMV or original code that is shown in listings 4.1, 4.2, 4.3, and 4.4 exemplify one of the generic problems discussed above. Even though the code listings present a generic problem, the specific problem focus in this section of the document is the guiding the patient to bed. It is considered necessary to mention that in the code that NuSMV will read does not have to have every single part of the code of the scenario. The only parts important for NuSMV are regarding the different states and transitions that force to change between states. The previous sentence is written in this document as a matter of explanation of why not all the instructions are coded in SMV format. The only ones that are important for the model checking part is when the status changes.

As it was explained previously, the system will receive the original code with one parameter, which is the model also indicating the state where the error was found. Modifications to the original model will be done in different files with different file names. The changes will be also done starting from the state where the error was found. Changes in the SMV code may vary from adding a new actor, adding different transitions, deleting others, adding other actions, subdividing goals, and others. A validation will be performed to see if the original final goals are achieved with those changes.

Logic formula

The model checking technique requires a model that represents the system and a logic formula which will be evaluated to verify if the model of the system holds all properties stated.

All models will be evaluated using the same specification that will be hold only if the original final goals are accomplished. In the experiments for this section, the formula that will be checked as shown in this work in formula 4.9. It was previously decided to use the final state of $fwd\_complete\_msg$ for the actor Controller 1 ct_1. Controller 1 is the actor that is assumed to take the lead role to the complete execution of the scenario.

$$EF(ct\_1.state = idle- > ct\_1.state = fwd\_complete\_msg) \tag{4.9}$$

Offloading the model checking part to the cloud

The motivation of reducing time of processing the model checking activity drives the decision to offload that activity to the cloud services. Another reason is to reduce the risk on the systems running out of resources while executing the model checking activity. A parallel execution is possible using the cloud services where they are configured as actors in an implementation of the HARMS model. The solution finder part involves the offloading of the model checking part.

The big data capabilities framework (BDCF) is an open source topology model that provides parallel execution by composing software components which are associated and extend the processing capabilities [138]. The BDCF framework was selected out of several options that were assessed. This framework targets to provide solution for three different layers, which are virtualization, orchestration, and provisioning. Within this context, virtualization where an infrastructure as a service(IaaS) allocates the resources needed for each execution within all the shared and available hardware assets. Orchestration takes care of the allocation of the different resources (compute, network, disk space). And software requirements, installation, and configuration are managed by the provisioning layer.

BDCF uses Alien4Cloud, a technology adopted by researchers as aplications of TOSCA components [139] BDCF includes as an intuitive graphical user interface (GUI) software which manages in a web fashion the catalog and application design. The applications that can be addressed using BDCF are based in different tools, technologies, and platforms, such as, Hadoop MapReduce and Hortonworks for distributed storage and process apps, elastic stack components for log analysis. Furthermore, other components that are offered in BDCF can be Kafka as a message broker, Consul as a consensus system, Rstudio for data scientists and researchers, and other development environments such as Java and Python. Alien4Cloud assures the top-notch standard for cloud environment through the utilization of topology and orchestration specification for cloud applications(TOSCA).

In the BDCF platform the solution presented in this research effort includes the components as shown in 4.15:

- One Compute, which will be the one of running each of the processes
- Webservice of an automatic run of PyNusMV [140] which will have as an output the result of the execution of that specific thread received as input
- Python code that will run the webservice for the different variations created
- Python code component which will contain the code that will modify, or in this case will simulate the part that will make the changes of the SMV code [137] for variations and save them in different files

4.3.5 Experiment cases

Two distinct cases were selected for experiments to be conducted for determining the benefits of offloading the model checking part to the cloud as a part of the survivability feature that can be accomplished using the HARMS model.

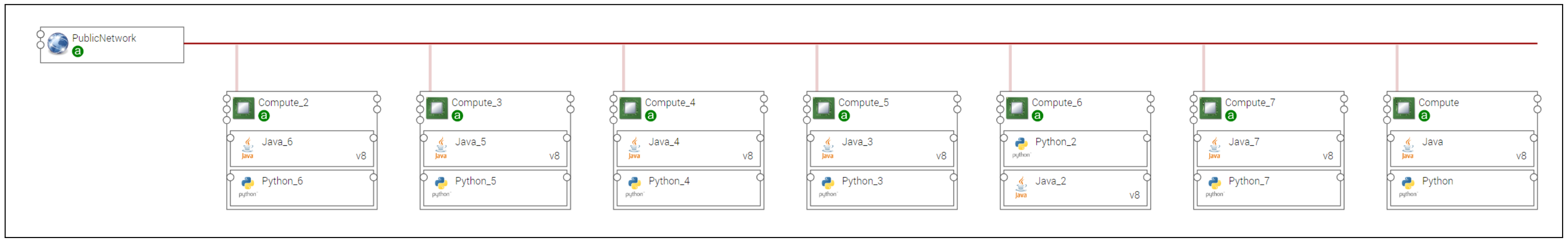


Fig. 4.15. BDCF platform diagram for self-healing

- In the one hand, experiments were performed to evaluate the way that offloading the model checking to the cloud could benefit the overall performance of the system.
- In the other hand, a set of experiments were programmed to measure the effectiveness of the survivability feature.

While performing an offload of processing of activities to the cloud many variables could be monitored to evaluate the performance of such implementation. Figure 4.13 shows the results obtained in one of the executions of the process. Such result is presented in the JSON format that is the basic format used to store important information for each of the actors. Actor 4 is the only that has different configuration within the seven different actors used for these experiments. The reason for the difference is that Actor 4 will be in charge of coordinating the model checking activity, while the others actors contribute with the model checking capability. Consequently, that actor cannot call himself to run that capability. After finishing the corresponding of model verification capability all actors send the result as a message reply to the actor that ordered them to execute that validation process. In that specific run, the actor that executed the verification capability with less time was Actor2. Therefore the model variation 2 is the selected by the coordinator actor and will be implemented with another handshake. The model variation that was verified in the Actor2 was model2ver2. Validation is the last stage included in the results within this work.

The way that the system reacts when the number of actors able to execute the model checking over the cloud increases and the number of models to verify stays fixed is shown in 4.16. The results show that the time to verify the 6 possible changes of the original model goes decreasing according to the number of actors where parallel execution will take place.

Also, the way that the system behaves when the number of actors that execute the model checking capability stays fixed, and the number of models to verify increases

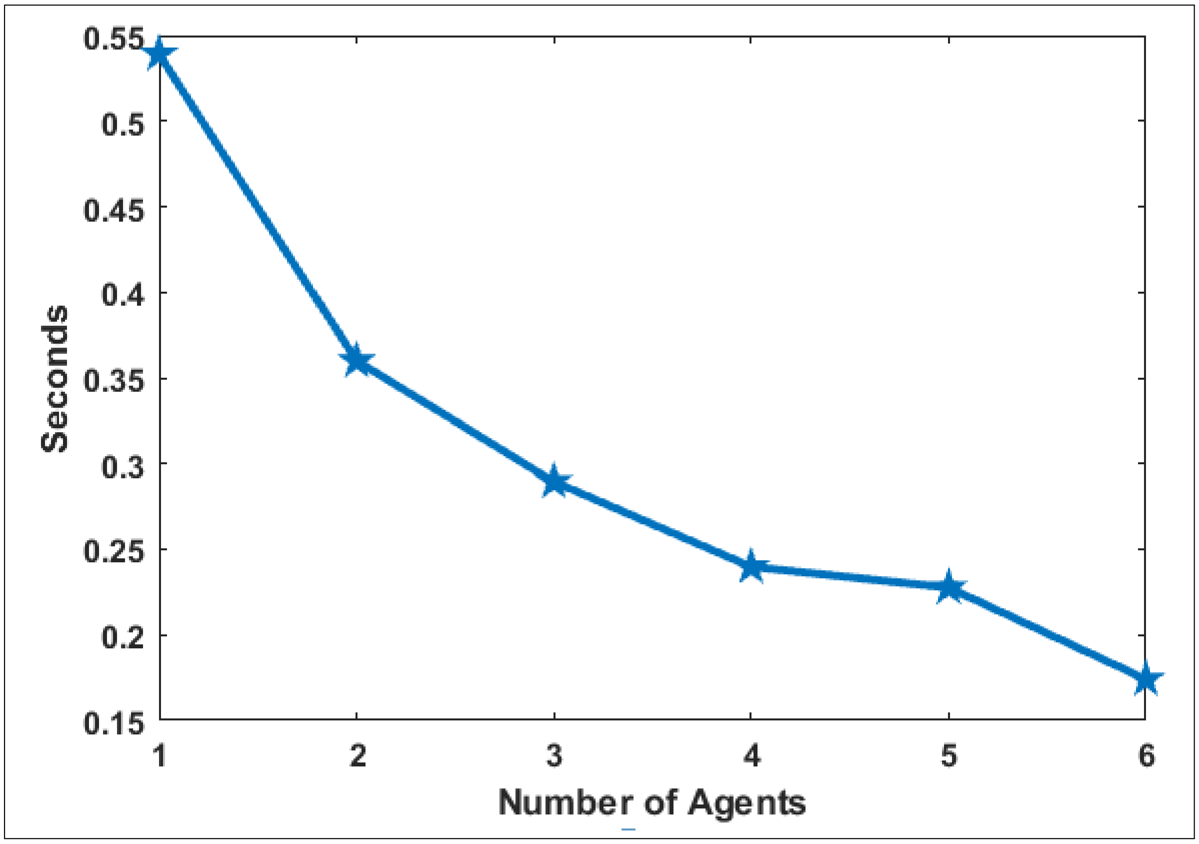


Fig. 4.16. Execution time depending on number of actors with model checking capability

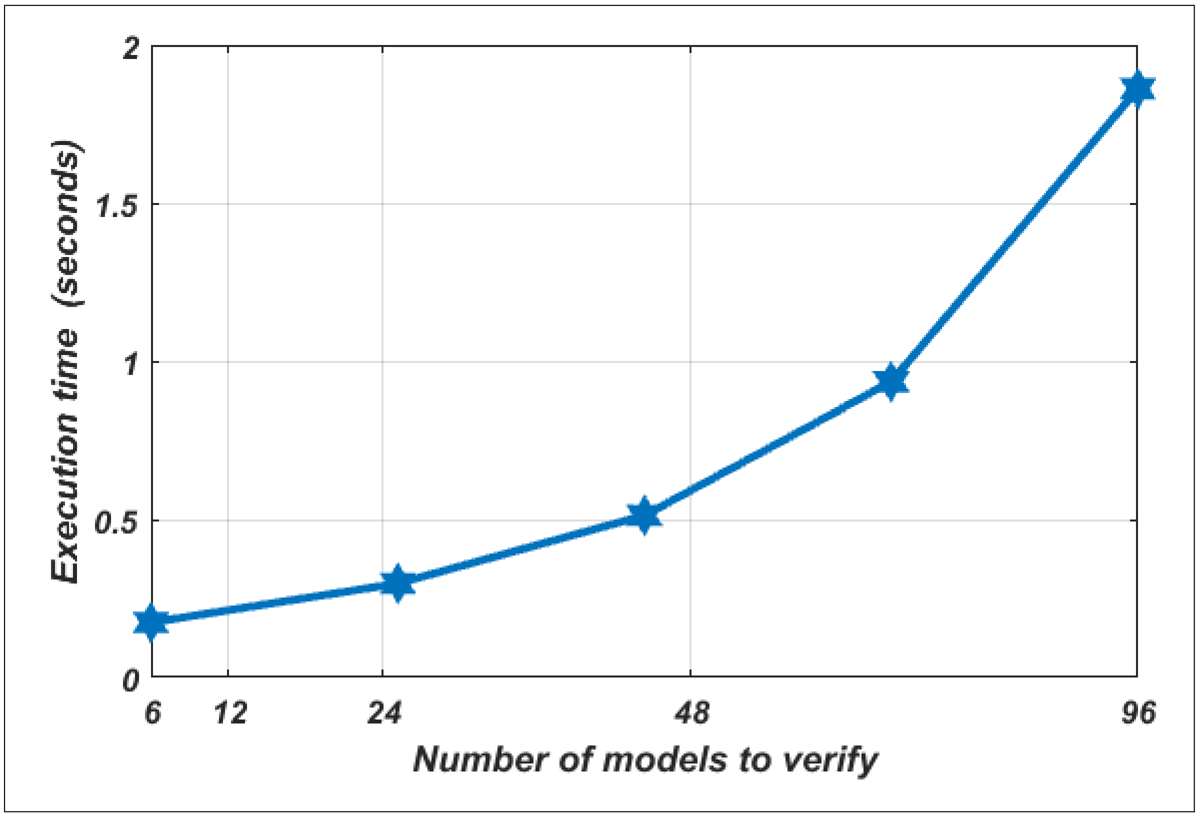


Fig. 4.17. Execution time depending on number of models to verify

is documented in 4.17. There, the trend shows that the value of time goes increasing strictly related to the number of models to verify.

4.4 Summary

In this chapter of the dissertation it was explained three different cases where it can be implemented both, the indistinguishability or the survivability of a multi-actor system with the use of the interaction layer of the HARMS model. The first scenario that was detailed in this chapter corresponds to the one when a MAS has the goal of guiding a human to perform a specific activity. The second scenario that was mentioned in deep in this chapter was the scenario where monitoring hazardous situations within a smart home environment can be solved using HARMS model.

The last scenario that was presented in this chapter was the one that permitted the evaluation of how offloading model checking processing to the cloud can help reducing that the survivability feature takes to provide possible solutions to a problem identified during execution time.

CHAPTER 5. CONCLUSIONS

In this section, the important contributions, findings, impact, conclusions, and future work related to this dissertation are summarized.

5.1 Contributions

There are four main contributions out of this research effort which are listed as follows.

- First, The implementation of the third layer based in the capability model for HARMS applications has augmented the decentralization of control and the information stored in the private databases of each of the actors.
- Second, the indistinguishability of the agents executing activities based on their own capabilities for a multi-agent system was completely implemented and verified. This contribution was enhanced with the use of the interaction layer of the HARMS model.
- Third, the benefit of distributing and offloading the model checking calculation to the cloud was evaluated. The aim of the performed experiments was to verify the minimization of the time to propose possible solutions during run-time.
- Fourth, the framework that contains the survivability feature that works towards solving problems identified only during run-time was presented with the use of model-checking during execution time of the system. Such attribute is of great importance for MAS and systems in general, specially when the end user will be relying in a high level of the system recommendations, such as in AALs.

5.2 Conclusions

This dissertation presented solutions to the problems that any system could face during run-time. The concern of this dissertation is motivated by how, from a suvivability point of view, a multi-actor system could reduce the risk of having human lives in danger if those systems find problems that are not programmed to overcome. In this work two techniques were introduced to let multi-agent systems take advantage of the different interactions that happen between the different actors to assure the full accomplishment of the original final goals in a survivable fashion.

At this moment, the feature of multi-agent systems based on the behaviour of living creatures survivability instinct does not exist. Similar efforts in which survivability has been mentioned are well studied in multi-agent systems as a different approach than the one addressed in this dissertation in a way where the problem has already occurred. In other words, the majority of the studies that exist today try to find the reason why a problem exist, making probabilistic studies before it is really happening, in a prevention manner. However, the dynamic environments that systems like the ones mentioned at the beginning of this section will represent a high complexity to be able to study for solutions given the variables at stake during run-time, specially for actors that count with small set of processing resources. Consequently, this study presents a solution where with a simple verification if the execution of the system goes transitioning as expected to detect a possible issue, which will trigger the execution of the survivability feature. Experiments provided show the efficacy in applying the survivability feature to let the system continue working towards accomplishing the original goal in spite of finding issues that they were not originally designed to surpass. Offloading the computation of the model checking process to the cloud was also evaluated in how it could help to reduce the time to provide a possible solution to the issue encountered in execution time.

Our first intention in this dissertation was to propose the implementation of a survivability feature to multi-agent systems, as explained in the previous paragraph.

However, while trying to implement such element for multi-agent systems the implementation and assurance of the indistinguishability was needed to let the survivability feature work in a proper way. The indistinguishability then was reached with the implementation of models based in capability such as HARMS. This finding is important given that, the majority of systems developed so far are using fixed and specific agents to perform activities which could generate bottlenecks. Problems like the one that was just mentioned can be materialized due to many reasons. Nevertheless, the focus in this dissertation is not, to find the reason why the problem was initiated. On the contrary, the importance is given to find the solution, rather than finding the cause of the problem. Survivability behaviour happens in nature, where trial and error is implemented. Further evaluation of the success of those trials follows to determine how effective the implementation was. Experiments were performed to corroborate that the indistinguishability feature can be achieved with the implementation of an algorithm of leader selection in multi-agent systems using HARMS model.

While implementing the third layer of the HARMS model it was possible to accomplish the direct interaction enforcing due to the decentralization of data. Consequently, if an actor needs to to know a value that it is only in the private database of another agent, the only way to get it, is through an exchange of messages.

5.3 Future Work

A variety of questions appeared when implementing the approaches mentioned above which lead to sketch possible further work. One possible future path of this work is to apply other artificial intelligence methods to the mechanism that will propose the different changes to the model that will be checked. It is considered that with solutions as such the time to provide the solution will be less perceivable by the dependable agent, such as the patient, and also the accuracy of the options to verify can be improved. This work also has other paths that can be studied in depth, such as autonomous fault detection for AALs, since it was one assumption for this work and could be improved applying techniques as mentioned that went out of the scope of this dissertation. Another direction that this study could take in forthcoming stages can be to evaluate to mix a context-aware solution to not only detect an error but also to propose the possible changes to the model. Moreover, another avenue for this study is to conduct experiments taking into consideration that the complexity of the different capabilities can be extended. For example, if the availability to execute a specific capability will depend on the presence of other factors. To be more specific, if a system has the need to paint a house, then, in order to paint a house, there may be specific tools and resources that are needed to perform the action in a proper way. That could also be another path for the future of this work, since only the implementation of a basic capability model was studied.

REFERENCES

REFERENCES

[1] W. L. Oberkampf and T. G. Trucano, “Verification and validation,” Sandia National Labs, Albuquerque, New Mexico, Tech. Rep., 2007.

[2] A. Cimatti, E. Clarke, F. Giunchiglia, and M. Roveri, “NUSMV: A new symbolic model checker,” *International Journal on Software Tools for Technology Transfer*, vol. 2, no. 4, pp. 410–425, 2000.

[3] B. Chaib-Draa, P. Millot, R. Mandiau, and B. Moulin, “Trends in distributed artificial intelligence,” *Artificial Intelligence Review*, vol. 6, pp. 35–66, 1992. [Online]. Available: http://link.springer.com/article/10.1007/BF00155579{\%}5Cn{\%}3CGotoISI{\%}3E://A1992JD63000002

[4] E. H. Durfee, V. R. Lesser, and D. D. Corkill, “Trends in Cooperative Distributed Problem Solving,” *IEEE Transactions on Knowledge and Data Engineering*, vol. 1, no. 1, pp. 63–83, 1989.

[5] M. Wooldridge, “Coherent social action,” in *Proceedings of the Eleventh European Conference on Artificial Intelligence (ECAI-94)*, 1994, pp. 279–283.

[6] ——, *An Introduction to MultiAgent Systems*, 2nd ed. Glasglow, Great Britain: Wiley, 2009.

[7] K. P. Sycara, “Multiagent Systems,” *AI Magazine*, vol. 19, no. 2, p. 79, 1998. [Online]. Available: http://www.aaai.org/ojs/index.php/aimagazine/article/view/1370

[8] K. M. Carley, “Computational and mathematical organization theory: Perspective and directions,” *Computational and Mathematical Organization Theory*, vol. 1, no. 1, pp. 39–56, 1995.

[9] S. N. P. Russel, *Artificial Intelligence: A modern approach.* New Jersey: Prentice Hall, 2009.

[10] M. V. Dignum, “A model for organizational interaction: based on agents, founded in logic,” Ph.D. dissertation, Utrecht University, 2004.

[11] E. T. Matson, “Transition in Multi-Agent Organizations,” Ph.D. dissertation, University of Cincinnati, 2008.

[12] E. Oliveira, K. Fischer, and O. Stepankova, “Multi-agent systems: Which research for which applications,” *Robotics and Autonomous Systems*, vol. 27, no. 1, pp. 91–106, 1999.

[13] V. Dignum, F. Dignum, and L. Sonenberg, “Towards dynamic reorganization of agent societies,” in *Proceedings of Workshop on Coordination in Emergent Agent Societies*, 2004, pp. 22–27.

[14] H. A. Abbas, S. I. Shaheen, and M. H. Amin, “Organization of Multi-Agent Systems: An Overview,” *International Journal of Intelligent Information Systems*, vol. 4, no. 3, pp. 46–57, 2015. [Online]. Available: http://www.sciencepublishinggroup.com/journal/paperinfo.aspx?journalid=135{\&}doi=10.11648/j.ijiis.20150403.11

[15] K. M. Carley, “Computational organization science: a new frontier,” in *Proceedings of the National Academy of Sciences of the United States of America*, vol. 99 Suppl 3, no. 1, 2002, pp. 7257–62. [Online]. Available: http://www.pubmedcentral.nih.gov/articlerender.fcgi?artid=128594{\&}tool=pmcentrez{\&}rendertype=abstract

[16] F. Zambonelli, F. Zambonelli, A. Omicini, and A. Omicini, “Challenges and Research Directions in Agent-Oriented Software Engineering,” *Autonomous Agents and Multi-Agent Systems*, vol. 9, pp. 253–283, 2004. [Online]. Available: http://www.springerlink.com/openurl.asp?id=doi:10.1023/B:AGNT.0000038028.66672.1e

[17] J. Juziuk, D. Weyns, and T. Holvoet, “Design Patterns for Multi-agent Systems: A Systematic Literature Review,” *Agent-Oriented Software Engineering: Reflections on Architectures, Methodologies, Languages, and Frameworks*, vol. 9783642544, pp. 79–99, 2014.

[18] M. Luck, P. McBurney, and C. Preist, “A manifesto for agent technology: Towards next generation computing,” *Autonomous Agents and Multi-Agent Systems*, vol. 9, no. 3, pp. 203–252, 2004.

[19] M. Fisher, “Concurrent {METATEM} - A Language for Modelling Reactive Systems,” *Parallel Architectures and Languages Europe*, pp. 185–196, 1993. [Online]. Available: citeseer.nj.nec.com/fisher93concurrent.html

[20] B. Bauer, J. P. Muller, and J. Odell, “Agent UML: a Formalism for Specifying Multiagent Software Systems,” *International Journal of Software Engineering and Knowledge Engineering*, vol. 11, no. 03, pp. 207–230, 2001. [Online]. Available: http://www.worldscientific.com/doi/abs/10.1142/S0218194001000517

[21] M. Wooldridge, N. R. Jennings, and D. Kinny, “The Gaia Methodology for Agent-Oriented Analysis and Design,” *Autonomous Agents and Multi-Agent Systems*, vol. 3, no. 3, pp. 285–312, 2000. [Online]. Available: http://eprints.soton.ac.uk/253748/{\%}5Cnhttp://link.springer.com/10.1023/A:1010071910869

[22] O. Shehory and A. Sturm, *Agent-oriented software engineering: Reflections on architectures, methodologies, languages, and frameworks.* Berlin: Springer, 2014.

[23] M. Wooldridge and P. Ciancarini, “Agent-Oriented Software Engineering: The State of the Art,” in *Agent-Oriented Software Engineering*, 2001, vol. 1957/2001, pp. 55–82.

[24] B. Horling and V. Lesser, “A survey of multi-agent organizational paradigms,” *The Knowledge Engineering Review*, vol. 19, no. 04, p. 281, 2005.

[25] J. Nouwens and H. Bouwman, "Living Apart Together In Electronic Commerce: The Use Of Information And Communication Technology To Create Network Organizations," *Journal of Computer-Mediated Communication*, vol. 1, no. 3, p. 0, 2006. [Online]. Available: http://onlinelibrary.wiley.com/doi/10.1111/j.1083-6101.1995.tb00169.x/full

[26] Y.-P. So and E. H. Durfee, "Designing tree-structured organizations for computational agents," *Computational and Mathematical Organization Theory*, vol. 2, no. 3, pp. 219–245, 1996.

[27] A. Kestler, *The Ghost in the Machine*, 1st ed. New York: Macmillan, 1967.

[28] M. Klusch, A. Gerber, and A. Intelligence, "Formation among Rational Agents," *IEEE Intelligent Systems*, vol. 17, no. 3, pp. 42–47, 2002.

[29] G. Beavers and H. Hexmoor, "Teams of agents," *2001 IEEE International Conference on Systems, Man and Cybernetics. e-Systems and e-Man for Cybernetics in Cyberspace (Cat.No.01CH37236)*, vol. 1, pp. 574–582, 2001. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=969875

[30] C. Brooks, E. Durfee, and A. Armstrong, "An introduction to congregating in multi-agent systems," in *Proceedings Fourth International Conference on MultiAgent Systems*, 2000, pp. 79–86. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=858434

[31] V. Dignum, J.-J. Meyer, and H. Weigand, "Towards an organizational model for agent societies using contracts," in *Proceedings of the first international joint conference on Autonomous agents and multiagent systems*, 2002, pp. 694–695. [Online]. Available: http://portal.acm.org/citation.cfm?doid=544862.544909

[32] D. J. Gordon, "An Uneasy Equilibrium: The Coordination of Climate Governance in Federated Systems," *Global Environmental Politics*, vol. 15, no. 2, pp. 121–141, 2015.

[33] S. L. Kiani, A. Anjum, M. Knappmeyer, N. Bessis, and N. Antonopoulos, "Federated broker system for pervasive context provisioning," *Journal of Systems and Software*, vol. 86, no. 4, pp. 1107–1123, 2013. [Online]. Available: http://dx.doi.org/10.1016/j.jss.2012.11.050

[34] N. Kang, "Multidomain Demand Modeling in Design for Market Systems," Doctoral Dissertation, University of Michigan, 2014.

[35] D. A. Levinthal and M. Workiewicz, "Are Two Heads Better than One: The Multi-authority Form and Organizational Adaptation," *Social Science Research Network*, pp. 1–36, 2015.

[36] J. Ferber and F. Michel, "AGRE: Integrating Environments with Organizations," *Agent-Oriented Software Engineering IV*, vol. 3374, no. Chapter 2, pp. 48–56, 2005. [Online]. Available: http://www.springerlink.com/index/10.1007/978-3-540-32259-7{_}2{\%}5Cnpapers2://publication/doi/10.1007/978-3-540-32259-7{_}2

[37] M. Hannoun, O. Boissier, J. S. Sichman, and C. Sayettat, "MOISE: An organizational model for multi-agent systems," *Advances in Artificial Intelligence*, vol. 1952, pp. 156–165, 2000.

[38] S. A. Deloach, W. H. Oyenan, and E. T. Matson, "A capabilities-based model for adaptive organizations," *Autonomous Agents and Multi-Agent Systems*, vol. 16, no. 1, pp. 13–56, 2008.

[39] D. A. DeLaurentis, W. A. Crossley, and M. Mane, "Taxonomy to Guide Systems-of-Systems Decision-Making in Air Transportation Problems," *Journal of Aircraft*, vol. 48, no. 3, pp. 760–770, 2011.

[40] M. Gomez, Y. Kim, E. Matson, M. Tolstykh, and M. Munizzi, "Multi-agent System of Systems to Monitor Wildfires," in *System of Systems Engineering Conference (SoSE)*, 2015, pp. 262–267.

[41] E. T. Matson and B. C. Min, "M2M infrastructure to integrate humans, agents and robots into collectives," in *IEEE Instrumentation and Measurement Technology*, Binjiang, 2011, pp. 408–413.

[42] R. C. Arkin, *Behavior Based Robotics*. Masachusetts: MIT Press, 1998.

[43] H. Geffner, "The Model-based Approach to Autonomous Behavior : A Personal View," in *AAAI Conference on Artificial Intelligence*. Atlanta, Georgia: Association for the Advancement of Artificial Intelligence, 2010, pp. 1–4.

[44] H. Psaier and S. Dustdar, "A survey on self-healing systems : approaches and systems," *Computing*, vol. 91, no. 1, pp. 43–73, 2011.

[45] M. C. Huebscher and J. a. McCann, "A survey of autonomic computingdegrees, models, and applications," *ACM Computing Surveys*, vol. 40, no. 3, pp. 1–28, 2008. [Online]. Available: http://portal.acm.org/citation.cfm?doid=1380584.1380585

[46] D. Ghosh, R. Sharman, H. R. Rao, and S. Upadhyaya, "Self-healing systems survey and synthesis," *Decision Support Systems*, vol. 42, pp. 2164–2185, 2007.

[47] P. Camurati, P. Prinetto, and P. Torino, "Formal Verification of Hardware Correctness : Introduction and Survey of Current Research," *Computer*, vol. 21, no. 7, pp. 8–19, 1988.

[48] M. Leucker and C. Schallhart, "A brief account of runtime verification," *Journal of Logic and Algebraic Programming*, vol. 78, no. 5, pp. 293–303, 2009. [Online]. Available: http://dx.doi.org/10.1016/j.jlap.2008.08.004

[49] C. Baier and J.-p. Katoen, *Principles of Model Checking*. Masachusetts: MIT Press, 2008.

[50] K. L. McMillan, *Symbolic Model Checking*. New York: Springer Science+Business Media, LLC, 1993, vol. 1.

[51] G. J. Holzmann, "The Model Checker SPIN," *Ieee Transactions on Software Engineering*, vol. 23, no. 5, pp. 279–295, 1997.

[52] M. Kwiatkowska, G. Norman, and D. Parker, "PRISM : Probabilistic Symbolic Model Checker," *In Computer Performance Evaluation Modelling Techniques and Tools*, vol. 2324, pp. 200–204, 2002.

[53] ——, "Verification of Probabilistic Real-Time Systems," *Lecture Notes in Computer Science*, vol. 6806, pp. 585–591, 2011.

[54] J. Barnat, L. Brim, V. Havel, J. Havlíček, J. Kriho, M. Lenčo, P. Ročkai, V. Štill, and J. Weiser, "DiVinE 3 . 0 An Explicit-State Model Checker for Multithreaded C & C ++ Programs," *Computer Aided Verification*, vol. 8044, no. LCNS, pp. 863–868, 2013.

[55] R. Pelánek, "BEEM: Benchmarks for Explicit Model Checkers," *Lecture Notes in Computer Science*, vol. 4595, no. 201, pp. 263–267, 2007. [Online]. Available: http://link.springer.com/chapter/10.1007/978-3-540-73370-6{_}17

[56] S. Mireslami, "Verification of Multi-Agent Systems Using AUML Methodology," Ph.D. dissertation, University of Calgary (Canada), 2013.

[57] J. Crow and J. Rushby, "Model-based reconfiguration: Diagnosis and recovery," SRI International Corp., Menlo Park, CA., Tech. Rep. NASA-CR-4596, NAS 1.26:4596, 1994.

[58] A. Bauer, M. Leucker, and C. Schallhart, "Model-based runtime analysis of distributed reactive systems," in *Proceedings of the Australian Software Engineering Conference, ASWEC*, vol. 2006, 2006, pp. 243–252.

[59] F. Chen and G. Rou, "MOP : An Efficient and Generic Runtime Verification Framework ," *ACM SIGPLAN Notices*, vol. 42, no. 10, pp. 569–588, 2007.

[60] N. Delgado, A. Q. Gates, and S. Roach, "A Taxonomy and Catalog of Runtime Software- Fault Monitoring Tools," *IEEE Transactions on Software*, vol. 30, no. 12, pp. 1–16, 2004.

[61] M. Utting, A. Pretschner, and B. Legeard, "A taxonomy of model-based testing approaches," in *Software Testing, Verification and Reliability*, vol. Volume 22, 2012, pp. 297–312. [Online]. Available: http://onlinelibrary.wiley.com/doi/10.1002/stvr.450/pdf

[62] Y. Zhao, S. Oberthür, M. Kardos, and F. Rammig, "Model-based Runtime Verification Framework for Self-optimizing Systems," *Electronic Notes in Theoretical Computer Science*, vol. 144, no. 1, pp. 125–145, 2006.

[63] Y. Zhao and F. Rammig, "Model-based Runtime Verification Framework," pp. 179–193, 2009.

[64] S. Kent, "Model Driven Engineering," *Integrated Formal Methods*, vol. 2335, pp. 286–298, 2006.

[65] R. Calinescu, C. Ghezzi, M. Kwiatkowska, and R. Mirandola, "Self-adaptive software needs quantitative verification at runtime," *Communications of the ACM*, vol. 55, no. 9, pp. 69–77, 2012.

[66] A. Filieri and G. Tamburrelli, "Probabilistic verification at runtime for self-adaptive systems," *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 7740 LNCS, pp. 30–59, 2013.

[67] G. a. Moreno, J. Cámara, D. Garlan, and B. Schmerl, "Proactive Self-Adaptation under Uncertainty : a Probabilistic Model Checking Approach Categories and Subject Descriptors," in *Proceeding of the 10th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering (ESEC/FSE)*, 2015, pp. 1–12.

[68] Y. Kim, M. Gomez, J. Goppert, and E. T. Matson, "Model Checking of a Training System Using NuSMV for Humanoid Robot Soccer," *Robot Intelligence Technology and Applications 3*, vol. 208, no. 96, pp. 531–540, 2015. [Online]. Available: http://link.springer.com/10.1007/978-3-642-37374-9

[69] M. Gomez, Y. Kim, and E. T. Matson, "Iterative learning system to intercept a ball for humanoid soccer player," *ICARA 2015 - Proceedings of the 2015 6th International Conference on Automation, Robotics and Applications*, pp. 507–512, 2015.

[70] A. Lomuscio and F. Raimondi, "MCMAS: A model checker for multi-agent systems," in *Proceedings of Tools and Algorithms for the Construction and Analysis of Systems*, vol. 3920, 2006, pp. 450–454.

[71] G. Edwards, J. Garcia, H. Tajalli, D. Popescu, N. Medvidovic, G. Sukhatme, and B. Petrus, "Architecture-driven self-adaptation and self-management in robotics systems," in *Proceedings of the ICSE Workshop on Software Engineering for Adaptive and Self-Managing Systems, SEAMS.* Vancouver, Canada: IEEE, 2009, pp. 142–151.

[72] K. Budzyńska and M. Kacprzak, "Model checking of persuasion in multi-agent systems," *Studies in Logic, Grammar and Rhetoric*, vol. 23, no. 36, pp. 99–122, 2011.

[73] D. Gil de la Iglesia and D. Weyns, "SA-MAS: Self-adaptation to Enhance Software Qualities in Multi-agent Systems," in *Proceedings of the 2013 International Conference on Autonomous Agents and Multi-agent Systems*, 2013, pp. 1159–1160. [Online]. Available: http://dl.acm.org/citation.cfm?id=2484920.2485120

[74] E. E. Elakehal, M. Montali, and J. Padget, "Run-time Verification of MSMAS Norms Using Event Calculus," in *International Conference on Self-Adaptive and Self-Organizing Systems*, 2014, pp. 110–115.

[75] E. M. Clarke, W. Klieber, M. Novacek, and P. Zuliani, "Model Checking and the State Explosion Problem," in *Tools for Practical Software Verification*, B. Meyer and M. Nordio, Eds. Springer, 2012, no. 2005, ch. 1, pp. 1–30.

[76] E. Clarke, "Counterexample-Guided Abstraction Refinement*," in *International Symposium on Temporal Representation and Reasoning.* IEEE, 2003, pp. 7–8.

[77] A. Valmari, "The state explosion problem," *Lectures on Petri Nets I: Basic Models, Lecture Notes in Computer Science Volume 1491*, vol. 1491, pp. 429–528, 1998. [Online]. Available: http://www.springerlink.com/index/10.1007/3-540-65306-6

[78] F. Guo, G. Wei, M. Deng, and W. Shi, "CTL Model Checking Algorithm Using MapReduce," in *Emerging Technologies for Information Systems, Computing, and Management*. Springer, 2013, vol. 236, ch. 39, pp. 341–348. [Online]. Available: https://books.google.com/books?id=zVpHAAAAQBAJ

[79] Y. Kim, J.-W. Jung, and E. T. Matson, "An Adaptive Task-Based Model for Autonomous Multi-Robot Using HARMS and NuSMV," in *Procedia Computer Science*, vol. 56, no. MobiSPC. Elsevier Masson SAS, 2015, pp. 127–132. [Online]. Available: http://linkinghub.elsevier.com/retrieve/pii/S1877050915016610

[80] C. Bellettini, M. Camilli, L. Capra, and M. Monga, "Distributed CTL model checking using MapReduce: theory and practice," *Concurrency Computation Practice and Experience*, pp. 685–701, 2015.

[81] M. Kacprzak, A. Lomuscio, and W. Penczek, "Verification of multiagent systems via unbounded model checking ," in *Autonomous Agents and Multiagent Systems*. New York: IEEE, 2004, pp. 638 – 645.

[82] R. E. Bryant, "Graph-Based Algorithms for Boolean Function Manipulation," *IEEE Transactions on Computers*, vol. C-35, no. 8, pp. 677–691, 1986.

[83] S. Ghilardi and S. Ranise, "MCMT: A model checker modulo theories," in *Automated Reasoning*, vol. 6173 LNAI. Trento, Italy: Springer-Verlag, 2010, pp. 22–29.

[84] C. Flanagan and P. Godefroid, "Dynamic partial-order reduction for model checking software," *ACM SIGPLAN Notices*, vol. 40, no. 1, pp. 110–121, 2005.

[85] E. Clarke, O. Grumberg, S. Jha, Y. Lu, and H. Veith, "Counterexample-guided Abstraction Refinement *," *Computer Aided Verification*, vol. 1855, no. LNCS, pp. 154–169, 2000.

[86] T. Latvala, A. Biere, K. Heljanko, and T. Junttila, "Simple bounded LTL model checking," *Formal Methods in Computer-Aided Design*, vol. 3312, no. LCNS, pp. 186–200, 2004. [Online]. Available: http://www.springerlink.com/index/A1JNFCB7Q9KNC1Q1.pdf

[87] A. Biere, K. Heljanko, T. Junttila, T. Latvala, and V. Schuppan, "Linear Ecodings of Bounded LTL Model Checking," *Logical Methods in Computer Science*, vol. 2, no. 5, pp. 1–64, 2006. [Online]. Available: http://www.lmcs-online.org/ojs/viewarticle.php?id=116

[88] P. Mell and T. Grance, "The NIST Definition of Cloud Computing Recommendations of the National Institute of Standards and Technology," *National Institute of Standards and Technology, Information Technology Laboratory*, vol. 800-145, pp. 1–3, 2011. [Online]. Available: http://csrc.nist.gov/publications/nistpubs/800-145/SP800-145.pdf

[89] M. Armbrust, I. Stoica, M. Zaharia, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, and A. Rabkin, "A view of cloud computing," *Communications of the ACM*, vol. 53, no. 4, pp. 50–58, 2010.

[90] X. Xu, "From cloud computing to cloud manufacturing," *Robotics and Computer-Integrated Manufacturing*, vol. 28, no. 1, pp. 75–86, 2012. [Online]. Available: http://dx.doi.org/10.1016/j.rcim.2011.07.002

[91] J. Dean and S. Ghemawat, "MapReduce: Simplified Data Processing on Large Clusters," *Communications of the ACM*, vol. 51, no. 1, p. 107, 2008. [Online]. Available: http://dl.acm.org/citation.cfm?id=1327452.1327492{\%}5Cnhttp://portal.acm.org/citation.cfm?doid=1327452.1327492

[92] T. White, *Hadoop : The Definitive Guide*, 4th ed. O'Reilly Media, Inc., 2015.

[93] C. Bellettini, M. Camilli, L. Capra, and M. Monga, "MaRDiGraS: Simplified building of reachability graphs on large clusters," *Reachability Problems*, vol. 8169 LNCS, pp. 83–95, 2013.

[94] T. Mancini, F. Mari, A. Massini, I. Melatti, and E. Tronci, "SyLVaaS: System Level Formal Verification as a Service," in *Euromicro International Conference on Parallel, Distributed, and Network-Based Processing.* IEEE, 2015, pp. 476–483. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=7092763

[95] T. Mancini, F. Mari, A. Massini, I. Melatti, F. Merli, and E. Tronci, "System Level Formal Verification via Model Checking Driven Simulation," in *International Conference on Computer Aided Verification (CAV)*, vol. 8044, no. LCNS, 2013, pp. 296–312.

[96] J. Lewis, E. T. Matson, S. Wei, and B. C. Min, "Implementing HARMS-based indistinguishability in ubiquitous robot organizations," *Robotics and Autonomous Systems*, vol. 61, no. 11, pp. 1186–1192, 2013. [Online]. Available: http://dx.doi.org/10.1016/j.robot.2013.04.001

[97] A. Esmaeili, N. Mozayani, M. R. Jahed Motlagh, and E. T. Matson, "A socially-based distributed self-organizing algorithm for holonic multi-agent systems: Case study in a task environment," *Cognitive Systems Research*, vol. 43, pp. 21–44, 2017. [Online]. Available: http://dx.doi.org/10.1016/j.cogsys.2016.12.001

[98] A. Ryker, E. Matson, S. Kim, S. Lee, and I. Jang, "Implementing a HARMS-based software system for use in collective robotics applications," *ICARA 2015 - Proceedings of the 2015 6th International Conference on Automation, Robotics and Applications*, pp. 416–420, 2015.

[99] M. A. Gomez and E. T. Matson, "Survivability of MAS through Collective Intelligence," in *2018 Second IEEE International Conference on Robotic Computing (IRC)*, IEEE, Ed., Los Angeles, CA, 2018.

[100] M. D. DeWees, "Securing Communication Within The HARMS model for use with Firefighting Robots," Ph.D. dissertation, Purdue University, 2015.

[101] M. Kim, I. Koh, H. Jeon, J. Choi, B. Cheol Min, Y. Im Cho, and E. T. Matson, "A HARMS-based Heterogeneous Human-Robot Team for a Gathering and Collection Function," in *16th International Symposium on Advanced Intelligent Systems (ISIS2015)*, Mokpo, South Korea, 2015, pp. 4–7.

[102] M. Gomez, Y. Kim, J. Goppert, and E. Matson, “Using Online Model Checking Technique for Survivability, Evaluating Different Scenarios on Runtime,” in *Procedia Computer Science*, vol. 94, 2016.

[103] M. Gomez, A. Chibani, Y. Amirat, and E. T. Matson, “Self-healing Mechanism over the Cloud on Interaction Layer for AALs Using HARMS,” in *De la Prieta F. et al. (eds) Trends in Cyber-Physical Multi-Agent Systems. The PAAMS Collection - 15th International Conference, PAAMS 2017. PAAMS 2017. Advances in Intelligent Systems and Computing*, 2018, vol. 619, pp. 264—-267. [Online]. Available: https://doi.org/10.1007/978-3-319-61578-3{_}33

[104] M. Gomez, A. Chibani, Y. Amirat, and E. Matson, “IoRT cloud survivability framework for robotic AALs using HARMS,” *Robotics and Autonomous Systems*, vol. 106, 2018.

[105] E. T. Matson, J. Taylor, V. Raskin, B. C. Min, and E. C. Wilson, “A natural language exchange model for enabling human, agent, robot and machine interaction,” *ICARA 2011 - Proceedings of the 5th International Conference on Automation, Robotics and Applications*, pp. 340–345, 2011.

[106] J. H. Hong, B.-C. Min, J. M. Taylor, V. Raskin, and E. T. Matson, “NL-Based Communication With Firefighting Robots,” in *IEEE International Conference on Systems, Man, and Cybernetics*. Seoul, Korea: IEEE, 2012, pp. 1461–1466.

[107] D. Erickson, M. Dewees, J. Lewis, and E. T. Matson, “Robot Intelligence Technology and Applications 2012,” in *Robot Intelligence Technology and Applications 2012*. Springer, 2013, pp. 873–882. [Online]. Available: http://link.springer.com/10.1007/978-3-642-37374-9

[108] E. T. Matson, “Towards the design of intelligent assistive services by integrating NKRL and the HARMS model,” in *International Workshop on ASROB at IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2013), Tokyo Big Sight*, Tokyo, Japan, 2013, pp. 1–5.

[109] N. Ayari, A. Chibani, Y. Amirat, and E. Matson, “A semantic approach for enhancing assistive services in ubiquitous robotics,” *Robotics and Autonomous Systems*, vol. 75, pp. 17–27, 2016. [Online]. Available: http://dx.doi.org/10.1016/j.robot.2014.10.022

[110] A. R. Wagoner and E. T. Matson, “A robust human-robot communication system using natural language for HARMS,” *Procedia Computer Science*, vol. 56, no. 1, pp. 119–126, 2015. [Online]. Available: http://dx.doi.org/10.1016/j.procs.2015.07.178

[111] ——, “A task manager using an ontological framework for a HARMS-based system,” *Journal of Ambient Intelligence and Humanized Computing*, vol. 7, no. 4, pp. 457–463, 2016.

[112] M. Gomez, E. Matson, J. Song, S. Baek, and J. Kim, “UGVs spotting fire location for Cooperative Fire Security System using HARMS,” in *ICARA 2015 - Proceedings of the 2015 6th International Conference on Automation, Robotics and Applications*, 2015.

[113] S. Park, Y. Kim, E. T. Matson, H. Jang, C. Lee, and W. Park, "An intuitive interaction system for fire safety using a speech recognition technology," *2015 6th International Conference on Automation, Robotics and Applications (ICARA)*, pp. 388–392, 2015.

[114] A. Wagoner, A. Jagadish, E. T. Matson, L. Eunseop, Y. Nah, K. K. Tae, D. H. Lee, and J. E. Joeng, "Humanoid robots rescuing humans and extinguishing fires for Cooperative Fire Security System using HARMS," *ICARA 2015 - Proceedings of the 2015 6th International Conference on Automation, Robotics and Applications*, no. 207689, pp. 411–415, 2015.

[115] B. Khaday, E. T. Matson, J. Springer, Y. K. Kwon, H. Kim, S. Kim, D. Kenzhebalin, C. Sukyeong, J. Yoon, and H. S. Woo, "Wireless Sensor Network and Big Data in Cooperative Fire Security system using HARMS," *ICARA 2015 - Proceedings of the 2015 6th International Conference on Automation, Robotics and Applications*, no. 207689, pp. 405–410, 2015.

[116] C. Ramos, J. C. Augusto, and D. Shapiro, "Ambient Intelligence the Next Step for Artificial Intelligence," *IEEE Intelligent Systems*, vol. 23, no. 2, pp. 15–18, 2008.

[117] D. J. Cook, J. C. Augusto, and V. R. Jakkula, "Ambient intelligence: Technologies, applications, and opportunities," *Pervasive and Mobile Computing*, vol. 5, no. 4, pp. 277–298, 2009. [Online]. Available: http://dx.doi.org/10.1016/j.pmcj.2009.04.001

[118] O. Vermesan, A. Broring, E. Tragos, M. Serrano, D. Bacciu, S. Chessa, C. Gallicchio, A. Micheli, M. Dragone, A. Saffiotti, P. Simoens, F. Cavallo, and R. Bahr, "Internet of Robotic Things Converging Sensing/Actuating, Hyperconnectivity, Artificial Intelligence and IoT Platforms," pp. 1–35, 2017. [Online]. Available: http://www.riverpublishers.com/pdf/ebook/chapter/RP{_}9788793609105C4.pdf

[119] A. Chibani, Y. Amirat, S. Mohammed, E. Matson, and N. Hagita, "Ubiquitous robotics : Recent challenges and future trends," *Robotics and Autonomous Systems*, vol. 61, no. 11, pp. 1162–1172, 2013. [Online]. Available: http://dx.doi.org/10.1016/j.robot.2013.04.003

[120] Z. Yanjun, "Survivable RFID Systems: Issues, Challenges, and Techniques," *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, vol. 40, no. 4, pp. 406–418, 2010. [Online]. Available: http://ieeexplore.ieee.org/xpl/freeabs{_}all.jsp?arnumber=5440949

[121] J. P. Sterbenz, E. K. Çetinkaya, M. A. Hameed, A. Jabbar, S. Qian, and J. P. Rohrer, "Evaluation of network resilience, survivability, and disruption tolerance: Analysis, topology generation, simulation, and experimentation: Invited paper," *Telecommunication Systems*, vol. 52, no. 2, pp. 705–736, 2013.

[122] E. K. Çetinkaya, D. Broyles, A. Dandekar, S. Srinivasan, and J. P. Sterbenz, "Modelling communication network challenges for Future Internet resilience, survivability, and disruption tolerance: A simulation-based approach," *Telecommunication Systems*, vol. 52, no. 2, pp. 751–766, 2013.

[123] S. Jha and J. M. Wing, "Survivability analysis of networked systems," *Proceedings - International Conference on Software Engineering*, pp. 307–317, 2001.

[124] A. Moschetti, L. Fiorini, M. Aquilano, F. Cavallo, and P. Dario, “Preliminary Findings of the AALIANCE2 Ambient Assisted Living Roadmap,” in *Ambient Assisted Living*, S. Longhi, P. Siciliano, M. Germani, and A. Monteriù, Eds. Cham: Springer International Publishing, 2014, pp. 335–342.

[125] H. Steg, H. Strese, C. Loroff, J. Hull, and S. Schmidt, “Europe is facing a demographic challenge Ambient Assisted Living offers solutions,” *IST project report on ambient assisted living*, 2006.

[126] A. Ayara and F. Najjar, “A formal specification model for survivability in pervasive systems,” *Proceedings of the 2008 International Symposium on Parallel and Distributed Processing with Applications, ISPA 2008*, pp. 444–451, 2008.

[127] M. Darwish, “Architecture and deployment of services of assistance to the person,” Ph.D. dissertation, Universite de Bretagne-Sud, 2017.

[128] A. Forkan, I. Khalil, and Z. Tari, “CoCaMAAL: A cloud-oriented context-aware middleware in ambient assisted living,” *Future Generation Computer Systems*, vol. 35, pp. 114–127, 2014. [Online]. Available: http://dx.doi.org/10.1016/j.future.2013.07.009

[129] T. Magherini, A. Fantechi, C. D. Nugent, and E. Vicario, “Using Temporal Logic and Model Checking in Automated Recognition of Human Activities for Ambient-Assisted Living,” *IEEE Transactions on Human-Machine Systems*, vol. 43, no. 6, pp. 509–521, 2013.

[130] G. N. Rodrigues, V. Alves, R. Silveira, and L. A. Laranjeira, “Dependability analysis in the Ambient Assisted Living Domain: An exploratory case study,” *Journal of Systems and Software*, vol. 85, no. 1, pp. 112–131, 2012.

[131] M. Skubic, G. Alexander, M. Popescu, M. Rantz, and J. Keller, “A smart home application to eldercare: Current status and lessons learned,” *Technology and Health Care*, vol. 17, no. 3, pp. 183–201, 2009.

[132] Pi My Life Up, “Raspberry Pi Temperature Sensor: Build a DS18B20 Circuit,” 2016. [Online]. Available: https://pimylifeup.com/raspberry-pi-temperature-sensor/

[133] Host-On.de, “Configure and read out the Raspberry Pi gas sensor (MQ-X),” 2017. [Online]. Available: https://tutorials-raspberrypi.com/configure-and-read-out-the-raspberry-pi-gas-sensor-mq-x/

[134] A. S. Harnad, “Minds , Machines and Turing : The Indistinguishability of Indistinguishables Source : Journal of Logic , Language , and Information , Vol . 9 , No . 4 , Special Issue on Alan Published by : Springer Stable URL : http://www.jstor.org/stable/40180236 Minds,” *Journal of Logic, Language, and Information*, vol. 9, no. 4, pp. 425–445, 2000.

[135] I. Ushakov, “System survivability,” in *Probabilistic Reliability Models*, 2012, ch. System Sur, pp. 162–168.

[136] W. Jamroga and A. Murano, “On module checking and strategies,” in *Proceedings of the 2014 international conference on Autonomous agents and multi-agent systems*, no. Aamas, Paris, France, 2014, pp. 701–708. [Online]. Available: http://dl.acm.org/citation.cfm?id=2615845{\%}5Cnhttp://delivery.

acm.org/10.1145/2620000/2615845/p701-jamroga.pdf?ip=66.254.241.109{\&}id=2615845{\&}acc=ACTIVESERVICE{\&}key=EA62C54EFA59E1BA.367AD4ADD93F5D2F.4D4702B0C3E38B35.4D4702B0C3E38B35{\&}CFID=762133550{\&}CFTOKEN

[137] A. Cimatti, E. Clarke, E. Giunchiglia, F. Giunchiglia, M. Pistore, M. Roveri, R. Sebastiani, and A. Tacchella, "NuSMV 2: An OpenSource Tool for Symbolic Model Checking," in *International Conference on Computer Aided Verification.* Springer, Berlin, Heidelberg, 2002, pp. 359–364.

[138] W. T. François Exertier , Mathis Gavillon , David Kuik , Mihai Mitrea , Anil Sinaci , Béchir Taleb Ali, "D4 . 1 Medolution Platform APIs and Specification V1 ITEA3 Project 14003," Medolution Consortium, Tech. Rep., 2016.

[139] J. Carrasco, F. Durán, and E. Pimentel, "Trans-cloud: CAMP/TOSCA-based bidimensional cross-cloud," *Computer Standards and Interfaces*, vol. 58, no. August 2017, pp. 167–179, 2018. [Online]. Available: https://doi.org/10.1016/j.csi.2018.01.005

[140] S. Busard and C. Pecheur, "PyNuSMV: NuSMV as a Python library," *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 7871 LNCS, pp. 453–458, 2013.