Trusting robots : Contributions to dependable autonomous collaborative robotic systems

Jérémie Guiochet

Habilitation à diriger des recherches

présentée à

l'Université de Toulouse - Université Paul Sabatier

Spécialité : Informatique et Systèmes Embarqués

par

Jérémie GUIOCHET

Laboratoire d'Analyse et d'Architecture des Systèmes, CNRS
École Doctorale Systèmes

Titre de l'habilitation :

Trusting robots
Contributions to dependable autonomous collaborative robotic systems

Soutenue le 8 décembre 2015, devant le jury :

Présidente:	Marie-Pierre GLEIZES	Professeur Univ. Paul Sabatier, IRIT
Rapporteurs :	Jacques MALENFANT	Professeur Univ. Pierre et Marie Curie, LIP6
	Walter SCHON	Professeur Univ. Techno. de Compiègne, Heudiasyc
	Jean-Marc THIRIET	Professeur Univ. Joseph Fourier, GIPSA-Lab
Examinateurs :	Charles PECHEUR	Professeur Univ. Cath. de Louvain, LVL
	David POWELL	Directeur de Recherche CNRS, LAAS
Parrain :	Mohamed KAÂNICHE	Directeur de Recherche CNRS, LAAS

Foreword

[in French]

Comme il est illusoire d'imaginer un travail individuel de onze années, je profite de cette page pour exprimer ma gratitude.

En premier lieu, mes remerciements vont aux membres de mon jury, Marie-Pierre Gleizes pour l'avoir présidé, Charles Pecheur en tant qu'examinateur, Jacques Malenfant, Walter Schön et Jean-Marc Thiriet pour avoir accepté d'en être les rapporteurs. Dans ce jury, il y avait également Mohamed Kaâniche que je remercie pour son parrainage, son soutien indéfectible, et aussi pour son engagement en tant que chef d'équipe de TSF au LAAS dans un esprit de recherche collaborative. Il serait long de faire la liste des remerciements que j'adresse à David Powell, tant le travail présenté dans ce manuscrit, mais aussi les méthodes que j'ai acquises aujourd'hui, sont empreintes de son enseignement.

Je salue au sein du LAAS, Jean Arlat et Karama Kanoun, qui ont précédé Mohamed dans la charge de chef d'équipe de TSF et m'ont accueilli et accompagné dans ma mission de chercheur. Leur héritage est immense, et les opportunités qu'ils m'ont offertes sont inestimables. Sans tous les membres de l'équipe de recherche TSF et de l'environnement dont j'ai pu bénéficier au LAAS, je n'aurais pas tant de plaisir à remplir ma mission d'enseignant-chercheur ! merci à tous.

J'ai une pensée particulière pour deux chercheurs du LAAS, aujourd'hui disparus, qui représentent les deux directions vers lesquelles ma recherche se tourne : Georges Giralt qui m'a poussé à creuser ces histoires de robots sûrs de fonctionnement et Jean-Claude Laprie pour l'écoute et les réponses qu'il m'a données quand je suis arrivé dans TSF.

Cette habilitation n'aurait également pas vu le jour sans l'excellent contexte de travail offert par mes collègues à l'université de Toulouse III, au sein de l'IUTA, au département Génie Électrique et Informatique Industrielle. Je les remercie chaudement.

Enfin, et malgré les conventions, ceux qu'il faudrait mettre en premier, ils sont ma vie et m'ont fait avancer pendant ces années : mes amis, ma famille et par dessus tout, Audrey, je t'aime, Lou et César mes petits démons.

Vers des robots collaboratifs autonomes sûrs de fonctionnement

Résumé : Ce manuscript d'Habilitation à Diriger des Recherches (HDR) présente les travaux menés par Jérémie Guiochet au LAAS-CNRS au sein de l'équipe Tolérance aux Fautes et Sûreté de fonctionnement informatique (TSF). Ces travaux se sont principalement articulés autour de la problématique de la sûreté de fonctionnement des systèmes robotique collaboratifs autonomes. Les spécificités de ces systèmes, notamment les interactions physiques humain-robot et la présence d'incertitudes liées aux mécanismes de perception ou de décision, font que les méthodes de sûreté de fonctionnement ou d'analyse du risque utilisées pour les systèmes critiques doivent être reconsidérées. Les principales contributions se concentrent sur deux axes : les méthodes d'analyse de la sécurité-innocuité pour des systèmes robotique collaboratifs (Identification des dangers avec HAZOP-UML et évaluation quantitative de la confiance dans un argumentaire de sécurité de type *safety case*), et les mécanismes de tolérance aux fautes pour des systèmes robotique autonomes (planification redondante et synthèse de règles de sécurité vérifiables en ligne). Ces travaux ouvrent également des perspectives concernant le test des systèmes autonomes dans des mondes virtuels, la gestion des incertitudes pour la certification des robots autonomes en milieu humain, et la surveillance en ligne des différents niveaux d'une architecture logicielle de robot autonome.

Mots clés : Sûreté de fonctionnement, sécurité des robots, systèmes autonomes critiques, analyse du risque, tolérance aux fautes, argumentaire de sécurité, évaluation de la sécurité.

Trusting Robots – Contributions to dependable autonomous collaborative robotic systems

Abstract: This manuscript of HDR (*Habilitation à Diriger des Recherches*, french accreditation to supervise research) presents research work of Jérémie Guiochet carried out at LAAS-CNRS in the Dependable computing and Fault Tolerance (TSF) team. His research work is mainly related to the dependability of collaborative autonomous robotic systems. Specific challenges raised by these systems, including human-system physical interactions and the presence of uncertainties in the perception and decision mechanisms, induce the need to revisit dependability and risk analysis methods. The main contributions address the following topics: safety assessment of collaborative robotic systems (hazard Identification with UML-HAZOP and quantitative assessment of confidence in safety cases), and fault tolerance mechanisms for autonomous robotic systems (redundant planning and synthesis of on-line verifiable safety rules). This manuscript also opens perspectives in the fields of testing of autonomous robots in virtual worlds, uncertainty management for the certification of autonomous robots in human environments, and safety monitoring at different levels in an autonomous software architecture.

Keywords : Dependability, robot safety, safety critical autonomous systems, risk analysis, fault tolerance, safety case, confidence assessment

Contents

Introduction

Even if fictional fantasies are still far from real robots, technological improvements make them approaching reality. Besides ethical discussions, how to build such systems is a crucial issue. But if we plan that some of these fantasies come to reality in next decades, another issue can be raised, which is *how to better build them*? It is already a main challenge in critical applications, from transportation to aeronautics, and it will be obviously a core challenge for robots deployment. "Build better systems" may be interpreted in terms of quality of the functionalities, but also in terms of reliability or safety which are encompassed in the concept of dependability.

A first step is to apply dependability techniques used in other safety critical embedded systems to robotics. Nevertheless, these techniques need to be adapted and extended in order to manage specificities of new robotics. Indeed if such systems actually belong to more general classes of systems such as embedded or cyber-physical systems, the *collaborative* and *autonomous* properties induce important issues in dependability.

Collaborative robots, or cobots, enable closer collaboration between human and robots. This collaboration implements robots operating without fences, in contact with humans or other operators in the same workspace. It thereby allows collaborative activity, combining physical and cognitive abilities of man with machine. This decompartmentalization of space and activity between man and the machine might facilitate the deployment of robots in private or professional environments. The main curb to their deployment is then the confidence one can place during human-robot interaction. Uncertainties in robot perception or in robot and user reactions, lead to major challenges in safety analysis and argumentation that need to be addressed. A direct consequence would be to convince regulators of the ability of the system to deliver trustworthy and dependable services.

Autonomous robots are able to build plans to perform tasks without human interventions. This is of course a feature with unlimited potential applications. It is also a source of distrust (see for instance the recent open letter about autonomous weapons[1]), considering that zero failure is not possible in such a system. As for collaborative robots, uncertainties in the perception, combined with complex and non deterministic decisional mechanisms, limit the efficiency of classical dependability techniques.

[1] http://futureoflife.org/AI/open_letter_autonomous_weapons, accessed august 2015

Hence, considering these limitations in the application of risk analysis and dependability techniques, how can users and regulatory bodies trust a collaborative autonomous robot? More than an application domain for the dependability community, such systems raise new challenges further discussed in the introductory Chapter 1. This review chapter shows that the issue of trusting cannot be easely answered, and even if more and more researchers are contributing, it is still an important and open issue.

My contributions to this challenge which are presented in this manuscript, fall in two topics:

- analysis methods for safety assessment
- architectural mechanisms for dependability

Safety assessment methods have been widely used in safety critical embedded systems for years. An important recent improvement of those methods leads to risk analysis techniques based on system models (architectural or behavior). Nevertheless, very few are focusing on scenario description and analysis of their deviation, and no specific method to analyse safety of human-robot interactions is actually proposed in the standards or studied in the robotic community. In Chapter 2, we propose a method for the identification of hazardous scenarios, HAZOP-UML, mixing a well-known risk analysis technique (HAZOP, Hazard Operability) and a system description notation (UML, Unified Modeling Language). HAZOP-UML main advantages are that it is applicable at the very beginning of the development process, it includes the humans as a source of hazard (human error), it provides guidance for analysts with list of guide words, and it focuses on operational hazards, i.e., hazards linked with the robot tasks and interactions. This approach was applied to several real robots working in collaboration with workers. A second main issue in safety assessment, is the construction of safety argumentation or safety case (usually used for certification), initially done in safety critical systems through textual description. This activity has been recently studied through the use of graphical notations, aiming at structuring the arguments making more explicit the expert judgment. Such an approach might be an interesting direction in the field of autonomous and collaborative robots, where no standard actually fully covers such applications. Nevertheless, while constructing such an argument, analysts are faced to many uncertainties such as impacts of robot perception uncertainties, human or robot behaviors, or occurrence of unknown situations. Hence, if we consider that such systems will always have some uncertainties, the issue of how to assess confidence in the arguments of the safety case is raised. A recent challenge is to assess quantitatively the confidence associated to a safety case modeled in GSN (Goal Structuring Notation). We developed and present in Chapter 3 an approach based on belief theory and Bayesian Networks to define and assess the confidence in safety cases. This approach has been applied to an assistive robot for standing, walking and sitting, that has been validated during clinical trial tests.

My second research topic is motivated by the fact that even with the use of methods to reduce the number and consequences of hazards in the systems, there will still exist

some residual hazards. To cope with these residual hazards, fault tolerance, is widely used in safety critical applications (mainly in aeronautics). It is defined in the dependability community as the mean to avoid service failures in the presence of faults. More generally, it is carried out with error or hazardous situations detection, and system recovery mechanisms. A first contribution to this topic is the development of a fault tolerant task planner (FTPlan) presented in Chapter 4. As the main decisional software component of the autonomous architecture, many errors may occur coming both from the inference mechanism or the knowledge representation. For this, we show through a fault injection campaign, that using a redundant planner for a mobile autonomous robot increases dependability without reducing performances. Another contribution in the field of fault tolerance, presented in Chapter 5, is the development of a safety monitoring framework (SMOF). Safety monitors are a popular form of fault tolerance mechanism, aimed at assuring that the system will stay in safe states despite faults and adverse situations occurrence. They are designed to observe the system and its environment, and to react using a safety margin to keep the system in a safe state. Specification and design of such mechanisms is usually done in an ad hoc manner, with simple safety rules implemented (e.g., in case of contact with an obstacle, an bumper directly disconnect the power of the actuator of a mobile robot). We argue that in the future, versatile autonomous systems, will have to deal with complex safety rules, that might be activated or deactivated according to the tasks, and with the possibility of reacting with many different ways that might be non consistent. This chapter presents a framework to specify such rules, starting from a hazard analysis, and using formal verification techniques to synthesized them. This framework also integrates the issue of assuring safety while preserving functionality of the considered system. This approach has been applied to a real case study of a robotic co-worker.

The concluding chapter 6, outlines my main contributions in this field, but also present some other research activities that are not detailed in this manuscript. Finally, I present my main research perspectives in the fields of safety monitoring for autonomous architecture, robot testing in virtual worlds and certification of collaborative and autonomous robots

After my PhD on robot safety, I joined the dependable computing and fault tolerance research team at LAAS-CNRS in 2004, and had the opportunity to continue to contribute to the field of dependable robots. The work presented in this manuscript is the result of close collaborations with my colleagues at LAAS-CNRS coming from dependability research group but also from the robotics research group. After 11 years, my activities on this multidisciplinary topic resulted in several PhDs, postdocs, co-organization of workshops, and several French and European projects. Figure 1 presents a temporal overview of the four main contributions (and one perspective), and also my PhD students status.

.

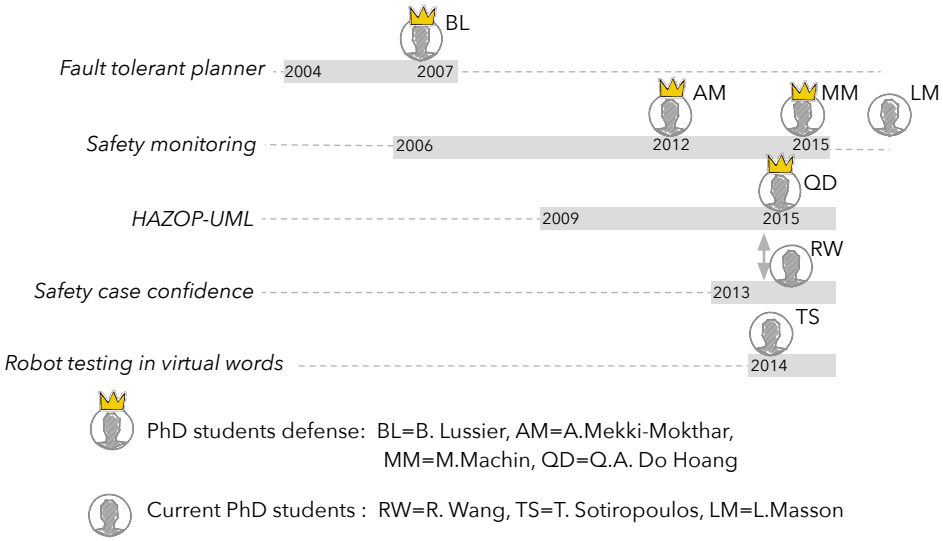


Figure 1: An overview of my main research activities, and associated PhD

1

Can we trust collaborative autonomous robots?

During my PhD on robot safety (defended in 2003), most publications on robot dependability were focusing on industrial automatic robots. Thirteen years later, many projects, Phds, dedicated research groups, and new standards, are focusing on dependable advanced robots. This chapter presents an overview of these activities and also provides some background for next chapters.

trust | trəst |
noun

1 firm belief in the reliability, truth, ability, or strength of someone or something: *relations have to be built on trust | they have been able to win the trust of the others.*

1* firm belief in the reliability, truth, ability, or strength of a robot: *interactions have to be built on trust | robots have been able to win the trust of the humans.*

Adapted from *New Oxford American Dictionary - online version - 2015*

As defined by international regulatory bodies, the accepted failure rate of a critical function in aeronautics is today 10^{-9} failure per hour of flight for one plane. Based on this calculation, manufacturers and end-users build a relation based on trust: (most of) people accept to travel in an aircraft, even if a failure during a flight is perceived as catastrophic. On the contrary, a failure of a robot which is actually potentially less catastrophic (compared to an aircraft, only few people are exposed), seems to be less

accepted. One reason may be that robots originally replace humans, and users expect that their failures and induced consequences are less frequent and severe than human ones[1]. Trust is thus partly based on the perception of that likelihood and consequences of failures (a more complete definition of trust in robotics is studied by Schaefer (2013)). Many robots developed in laboratories cannot be deployed in real life due to this lack of trust.

Nevertheless, several recent research European projects consider safety as the main challenge of human-robot cooperation like PHRIENDS (2006-2009); SAPHARI (2011-2015); SAFROS (2009-2013); ROBOT-PARTNER (2013-2016) or as a key objective in ROSETTA (2009-2013); BRICS (2009-2013); CHRIS (2008-2012); CARLOS (2012-2014). National projects such as ROBOSAFE (2013) in UK, SIMERO (2003) in DE, or NREC (2015) in the US, or dedicated research teams (e.g., Verifiable Robotics Research Group (2015)) also focus on robot dependability. Moreover, many research laboratories address robotics or embedded system dependability (or both like LAAS-CNRS). In this context, the number of PhD and published papers increased significantly in the last ten years. However, considering the vast and transversal domain, it is difficult to classify and review all works in this field. Hence, we focus in this section on activities which have a direct and explicit impact on dependability, and mainly on safety. Many works on robotics functions development may impact safety (for instance gripping issues), but as they do not explicitly focus on faults, they will not be reviewed in this chapter.

In order to present what is done to increase this trust, I start in Section 1.1 from a description of fundamentals elements of new robotics, i.e. autonomy and interaction, to an analysis of hazards and risks. It is followed by Section 1.2 dealing with European standards for robot safety. I also present major works done for dependability in robotics in Section 1.3. The last Section 1.4 presents main challenges in the field of dependable robots, and then focus on my research contributions, that presented in more detail in the following chapters.

1.1 From industrial robots to collaborative autonomous robots - Hazards and risks

Among the large diversity of robotics applications and their associated ethical issues (Royakkers and van Est, 2015), safety is not a new concept. It has been studied for years in industry, and particularly for industrial robots use. But the development of new robots leads to consider new paradigms which impact traditionally approaches by robotic community for addressing safety. This section starts from a definition of autonomy and its

[1]To illustrate this statement, several newspaper articles (e.g., http://www.ft.com/cms/s/0/0c8034a6-200f-11e5-aa5a-398b2169cf79.html#axzz3gWie0xAT) state that it is about eight times more dangerous to work in a bar in the US than in manufacturing (including manufacturing with robots, so it should be less for manufacturing with robots)

impact on software architecture, followed by a discussion on hazards and risks such systems induce.

1.1.1 Autonomous robots

Most of the robots used in industrial processes are automatic machines, performing repeated tasks, in a well defined workspace, segregated from operators with fences. These robots have been adapted to perform other useful tasks for humans at home, in hospitals, etc. Now, a new generation of robots, different from the industrial ones, have emerged completely dedicated to these new tasks, without any human intervention, in uncertain environments. Besides this development, autonomous mobile robots and drones are also developed to avoid human presence in the control loop.

Autonomy has several definitions in the literature; in its most generic sense, it is defined as "the ability to self-manage, to act or to govern without being controlled by others". A less generic and more suitable definition of autonomy of robotic systems can be found in Huang et al. (2005) as:

> *Autonomy is the ability of integrated sensing, perceiving, analyzing, communicating, planning, decision-making, and acting/executing, to achieve its goals as assigned. The autonomy level is determined by the complexity of the missions that the system is able to perform, the degrees of difficulty of the environments within which the system can perform the missions, and the levels of operator interaction that are required to perform the missions.*

This definition focuses on two important aspects of autonomous robots: the first is the uncertain environment in which the system operates, the second is its ability to make decisions in this environment. Nowadays, such robots are even considered to replace classic industrial robots in factories, in order to perform collaborative tasks with human operators (known as co-workers, see e.g., Haddadin et al. (2011)). Hence, current classifications of robots, segregating industrial robots from others (see for instance definition of service robots[2]) are about to be outdated.

The issue of categorizing autonomous systems (and decide if a system belongs or not to autonomous systems) is actually complex. Some works present classifications (see for instance Parasuraman et al. (2000); Huang (2008)) from complete control by human (e.g., DaVinci medical robot[3]), to full autonomy (e.g., Mars Rover[4]), which permit to avoid a binary classification autonomous / not autonomous, which can be impossible for many systems at the boundaries of this classification.

[2] According to ISO13482 (2014) a service robot is a *Robot that performs useful tasks for humans or equipment excluding industrial automation applications*

[3] http://www.davincisurgery.com/

[4] http://mars.nasa.gov/mer/technology/is_autonomous_mobility.html

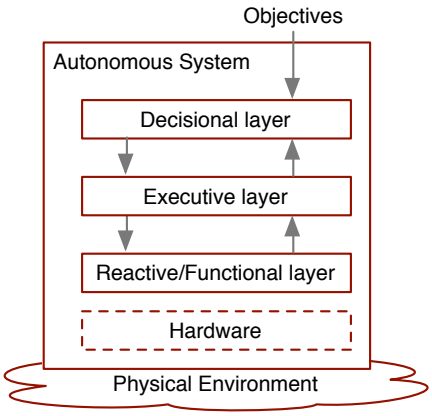


Figure 1.1: A three layer architecture for autonomy

In order to consider hazards induced by these applications, we will then consider two main characteristics of new robotics: the presence of a decisional layer which is non deterministic in the software architecture, and a highly interactive relation between a robot and humans, including physical human-robot interactions.

1.1.2 Autonomous software architecture

Considering the previous definition for an autonomous system, several architectures can be found in the literature. As stated in the survey on deliberative systems by Ingrand and Ghallab (2014), hierarchical architectures are probably the most used in autonomous robotics. It is usually composed of several layers, from hardware level to decisional (or deliberative) level. While some architectures might be classified as two-layer architectures, one for elementary tasks (like sensing), and another one for decisional aspects (e.g. see 4D/RCS Albus et al. (2002), ORCCAD Borrelly et al. (1998), CLARATY Volpe et al. (2001); Nesnas et al. (2003)) others propose to use a three layer architecture (Gat, 1998; Alami et al., 1998; Ingrand et al., 2007). In this latter case, the proposed software layers presented in Figure 1.1 are:

Decisional/deliberative layer is the highest abstracted level of the architecture. Objectives are received (from another system, or an operator) and it generates some plans according to an abstract representation of the system and its environment. Functions for deliberation (e.g., planning, learning or goal reasoning (Ingrand and Ghallab, 2014)) are usually composed of knowledge specific to the system's domain of application (such as heuristics or a model of the environment) and an inference mechanism used to solve problems by manipulating this knowledge. No guarantee

can be delivered for execution time, and output is also not deterministic. The use of heuristics is not guaranteed to be optimal or perfect, but sufficient to find solutions.

Executive/sequencing layer converts plans sent by the decisional layer, into primitive functions for the functional level. Real time requirements are included.

Reactive/Fonctional level is in charge of feedback control loops coupling sensors to actuators, of perception facilities and trajectory computation. It is composed of elementary functions sharing information without any global representation. Execution time is adapted to the sensors and actuators performances.

Each level sends to the highest level the results of task execution (including errors that cannot be managed at the lowest level). Hybrid versions with combined layers or direct communications links between functional and decisional layers also exist, but this simple three layer description is useful for aggregating most of the current hierarchical architectures.

1.1.3 Human-robot interaction

Among the topics of the research domain of human-robot interaction (see Yanco and Drury (2004) for a classification), two main branches can be cited: cognitive and social Human-Robot Interaction (cHRI), and physical Human-Robot Interaction (pHRI). Cognitive HRI focus on the social and psychological aspects of interactions mixing research domains as psychology, cognitive science, human-computer interfaces, human factors, and artificial intelligence. Physical HRI deals with issues due to physical interaction, especially from the view of robot design and control. As any classification, it has some limits, and both may impact or cover each other. For instance, in Mainprice et al. (2010), movement trajectories are computed to comply with users expectations and habits (human-aware movements, part of cHRI research), in the context of robot in human vicinity with potential contacts (part of pHRI research).

Apart from those human-robot interactions studies, we can first just consider the communication means between a human and a robot. A simple classification may be:

Remote communication : use of hardware components (teach pendant, buttons, graphical user interfaces, etc...)

Physical communication : physical contact between the human and a mobile part of the robot.

Cognitive communication : communication through cognitive signals like posture, movement, noise, speech, etc.

All these possible communication channels are bidirectional. For instance, a robot can engage an interaction through a desired physical contact with a user. Challenges in pHRI

require to cover many domains (Alami et al., 2006), and an important work was done in projects like SAPHARI (2011-2015) or PHRIENDS (2006-2009)). For pHRI, the way a human and a robot may interact depends on the level of "closeness" both can achieve. Physical interaction may be direct or indirect: if the robot is carrying an object the human is touching, then the interaction is termed indirect. A direct human-robot interaction involves the robot itself or an attached tool touching the human (or vice-versa). Direct pHRI may also be termed hands-on pHRI when the operator or user is supposed to work in physical contact with the robot, thereby exchanging forces and torques between the user and the robot. An extreme form of direct pHRI is (minimally invasive) surgery where robotic devices are used to work on or inside a human body. Another term to denote indirect pHRI is hands-on-payload, which is common with intelligent assist devices (IAD). To assess the risks associated with human-robot interaction it is important to not only classify these types of physical contact, but to also specify the level of closeness as outlined in the following taxonomy we developed during the PHRIENDS project:

1. Far (no pHRI possible): Human and robot are not sharing the same workspace; a direct physical contact is not possible. The interaction with a far robot is usually carried out via remote communication. This may be through a teach pendant or jogging device (when there is a line of sight with the robot) or over some network.

2. Close (accidental pHRI possible, "hands-off" pHRI): In this case the human and robot are sharing the same workspace. Since the human is within the robot's reach there is a risk of unwanted, potentially harmful physical contact. An example is the programming of the robot system while the programmer is within the robot's work cell. Another example could be the exchange of objects between a human and a robot through a dedicated exchange position within the workspace (e.g., a table or shelf).

3. Touching without simultaneous movement (direct or indirect pHRI): The robot shares its workspace with the human. Both are simultaneously moving through the workspace, but physical contact with the moving robot is avoided. In this category interaction only takes place when the robot stops. An example for this type of interaction is a robot delivering an object. The system approaches the human (or vice versa), the robot (safely) stops temporarily when the human reaches the object and only starts moving again after the interaction is completed.

4. Touching with simultaneous movement (direct or indirect pHRI): The robot shares its workspace with the human. Both are moving simultaneously and physical interaction is possible and intended. An example of direct pHRI may be a robot which is programmed by being manually guided through the workspace. Interaction could also take place during task execution. The robot may assist the human with its greater force and/or precision. An example of indirect pHRI may be a robot assisting

elderly people by feeding them. This may involve simultaneous direct and indirect pHRI with one or more manipulators.

5. Supporting (direct pHRI): Here the physical interaction occurs continuously over extended periods of time, usually in the form of exoskeletons which are worn by the user, or when the robot is carrying a human (for example in healthcare applications or rescue operations).

6. Invasive (direct pHRI): Surgical robots or their tools are designed to invade the human body, which is considered as the robot workspace.

We believe that these 6 levels should be part of a safety analysis. It is indeed obvious that the consequence of failure of the system will be strongly related to the type of closeness. A direct adaptation of such a classification could be the definition of the workspaces and associated safety rules in the case of a co-worker (Haddadin et al., 2011).

1.1.4 Harms, Risks and Hazards

The first and obvious concern when dealing with robot safety, is the possible harm due to an unwanted collision between a human and a robot. Most of work done on harm induced by robots is on the biomechanical analysis of human robot contact inducing impact, crushing, cutting, etc. and associated control loop or actuators for reducing harm severity (e.g. see Ulrich et al. (1995); Zinn et al. (2004); Povse et al. (2010); Haddadin (2014, 2015)). Some results of these researches are part of the technical specification of ISO/TS 15066 'Collaborative robots', which has been analyzed by the HSE (2012). Authors note that it is still difficult in this document to validate the forces calculation, as the situations in terms of probability of exposure and complexity of interaction (human moving or not, which direction, etc.) are difficult to describe. It is hence nearly impossible to determine an acceptable speed or force, without knowing the robotic application (diversity goes against generality).

Besides these researches specific to robotics, a more generic approach to study safety is based on the concept of *risk*, widely used in safety critical systems. It has been defined in ISO/IEC-Guide51 (1999) as the *combination of the probability of occurrence of harm and the severity of that harm*, where a *harm* is defined as *a physical injury or damage to the health of people, or damage to property or the environment*. We also include in harm definition the damage to the robot itself. A risk management process is usually composed of three steps (a more complete view is given in ISO31000 (2009)): hazard (any potential source of harm) identification and analysis, induced risk estimation, and risk evaluation to decide if the risk is acceptable. If not, actions to reduce risk are performed.

A major difficulty when using the risk management approach, is the occurrence probability estimation. It is hence, in such systems, very hard to estimate probabilities of failures of the software, human errors, adverse situations occurrence, etc. One approach is

then to set probability to one, and to deal with severity, considering that everything can occur! (It is even mentioned explicitly in risk management standards like ISO/FDIS14971 (2006), p. 37). This simplification has obviously some drawbacks, the most important being that this approach can outcome many unacceptable risks, and reducing all risks has a cost (performance, physical space, development time) that may cannot be covered for the development of robots (where power and physical space are limited for instance). Another difficulty is due to the definition of *hazard* (any potential sources of harm) which can be used to define its origin or the nature of the expected harm (e.g., electric shock hazard, crushing hazard, cutting hazard, collision hazard, fire hazard). This thus could designate a source or a consequence. For instance, in a hazard analysis of a manipulator, a hazard might be a controller reboot due to high temperature, and an unwanted movement is also a hazard. There is a possible cause/consequence chain between those two hazards. This could lead to ambiguities and difficulties into the application of hazard analysis techniques.

Main works aiming at identifying main hazards present in robotics applications are reviewed hereafter. Even if the study of Malm et al. (2010) states that crushing and clamping might be the major hazards in robot cells or in future applications, an important challenge is to identify all possible hazards that may change according to the task and the context. Most of classical hazards if robots are considered as *Machines* (according to the European Directive 2006/42/EC (2006)) induce same hazards as other industrial machines (electric shocks, cut, etc.), and more specific ones that can be found in ISO13482 (2014), Annex A. However, as illustrated in Table 1.1, many evolutions of robot properties need to be taken into account to identify the new hazards. A study of Carlson and Murphy (2003), was on failures of indoor and outdoor robots, not all autonomous. Thirteen robots were observed for nearly two years, displaying a MTBF (*Mean Time Between Failure*) of about 8 hours, and a reliability of less than 50%. Outdoor robots were seen to fail more often than indoor ones (maybe because of the more demanding outdoor environment), and while hardware faults were the most common cause of failures (42%), the control systems (including the operating systems) were also significant sources of failures (29%). On the contrary, when autonomy increases, so do the failures of the software part. For instance Tomatis et al. (2003a) presents the implementation of the autonomous museum tour guide RoboX9 and a study of its failures during five months of operation. 96% of failures were caused by the software components (80% due to the non-critical human interaction process, and 16% due to the critical navigation and localization process). Same conclusions were drawn by Steinbauer (2013), who presents a review on detected faults on 17 robots of the Robocup. Failures of the mission goal are considered. Software faults in these systems are more frequent than hardware faults, and belong to operational system, middleware or robot controller (including localization, or planners). Actuators and sensors fail with a similar rate, but the sensors ones were more critical for the mission success.

Among hazards induced by software, we identified in Lussier et al. (2005) that the software faults in decisional layer could be faults in the inference mechanism or in its knowledge representation. As presented in Figure 1.2, faults in the inference mechanism

	Industrial robotics	**Advanced robotics**	**New hazards examples**
Motion	No robot motion when human presence	Simultaneous motion (human and robot)	Bad synchronization between human and robot / Non human legible movements
Human-robot closeness	Human is far	Human is close / Physical contact	Collisions, contact forces too high
Human-robot interaction	Teach pendant	Advanced interaction (cognitive)	Mode confusion / communication errors
Robot control	Automatic	Autonomous	Hazardous decisions
Mechanical architecture	Heavy / Stiff / Powerful	Light / Compliant / limited power("intrinsically safe" Ulrich et al. (1995))	Precision hazards /energy storage due to compliance
Task complexity	Mono-function	Multi-functions	Safety rules not adapted (diverse and evolving rules)
Workspace	Structured	Non structured (uncertainties)	Adverse situations / uncertainties in perception

Table 1.1: Core properties of industrial and advanced robotics, and examples of induced hazards

may be introduced during development of the system, either as a design fault (for example if the decisional mechanism is not adapted to the system function, or if its principle is flawed) or as a programming fault (such as a typing or algorithm error). Faults in the system knowledge may be introduced either during development (design or programming) or in operation. Design faults may be either an explicitly-specified adverse situation that has not been covered by the developers (such as a missing procedure or an action needed to treat the adverse situation, or missing example sets used in learning for neural networks), or an imperfection in the choice criterion that possibly causes wrong conclusions to be drawn by the inference mechanism (such as faults in heuristics, or facts used for decision that are wrong in particular situations). Knowledge programming faults include both missing and faulty information in the knowledge of the decisional mechanism. Operational faults are incorrect dynamic information in the knowledge of the decisional mechanism, such as the current state of the system or information learned from the environment; these faults may be caused for example by sensor failures or imperfections, or undetected memory corruption.

Most of faults presented in Figure 1.2, like neglected situations or incorrect dynamic knowledge, result from a lack of knowledge (also called epistemic uncertainties), are hardly treated using common techniques like testing or formal verification for instance.

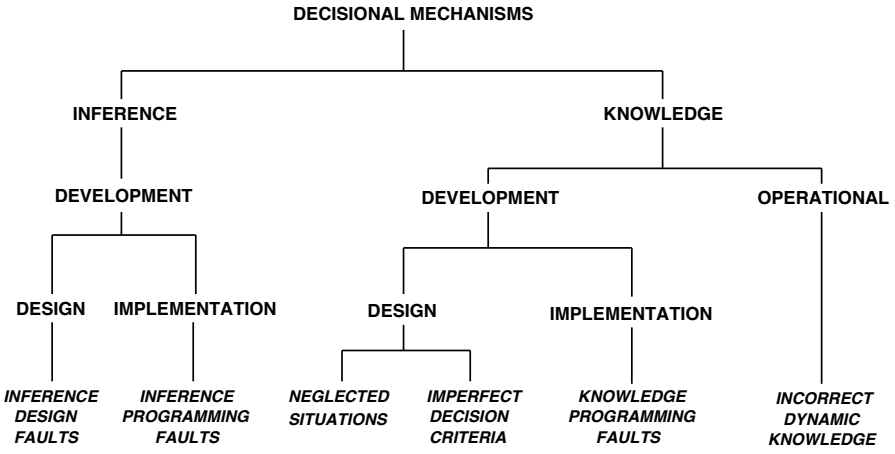


Figure 1.2: Internal faults in decisional mechanisms

1.2 Robot safety standards

In Europe, in order to commercialize a machine (including an industrial robot), the only requirement is to get a CE mark following the 2006/42/EC (2006). This is done through a process, from an auto-declaration of the manufacturer to a complete reviewing process by independent regulation bodies. ISO standards (e.g., ISO13849-1 (2006); ISO12100 (2010) for machines safety) are highly recommended as it gives confidence to the regulatory bodies to deliver certification. However, due to properties of new robots, such directives or standards are not entirely applicable, in particular due to physical contact between human and mobile parts of the robot. The generic standard IEC61508 (2010) dedicated to safety-related hardware and software, based on the concept of *Safety Integrity Level* (SIL), is also hard to apply because the entire robotic systems might be classified as a safety-related system. For cost reason, it is then quite impossible for a manufacturer to apply all the requirements of this standard. Moreover, in this standard it is recommended to avoid artificial intelligence technique for fault correction (IEC61508-3:2010, p48) for all the safety integrity levels except the lowest one !

More recently, the standard ISO10218-1 (2011); ISO10218-2 (2011)[5], for robot in industrial environment, has been followed by the standard ISO13482 (2014) for personal robots (a standard dedicated to collaborative robots is under development, current name is ISO/TS 15066 – Robots and robotic devices – Safety requirements for industrial robots – Collaborative operation). In this standard a list of typical safety related functions are given:

[5]In the US, the safety standard R15.06-2012 (2012) is an adoption of ISO 10218:2011 Parts 1 and 2, providing guidance on the proper use of the safety features embedded into robots, as well as how to safely integrate robots into factories and work areas.

emergency stop, protective stop, limits to workspace (including forbidden area avoidance), safety-related speed control, safety-related force control, hazardous collision avoidance. For each function a Performance Level (PL) equivalent to a Safety Integrity level (SIL), is assigned leading to a set of recommendations from ISO13849-1 (2006) (for software a more detailed list of recommendations are given in IEC61508-5 (2010)). This approach is appropriate when it is possible to clearly identify and separate the safety functions from the main robot controller. Nevertheless, if we consider for instance a mobile robot and the safety related function "hazardous collision avoidance", it is completely part of the main robot controller, with perception, decision and reaction. It then leads to assign the whole robot controller a high integrity level, which is too demanding for manufacturers. It is still hard to estimate adoption of such robotic standards by manufacturers, partly because their novelty, but considering the wide and growing variety of robots, we can expect that they may not be covered by such standards. Hence producing generic standards in this field is really challenging.

Until now, very few robots have been certified. For instance, in the technical documentation of the UR5 from Universal Robots (UR5-Robot, 2015), it is specified that 15 safety functions have been tested by the TÜV (Technischer Überwachungs-Verein) in accordance with the "EN ISO 13849:2008 PL d, and EN ISO 10218-1:2011, Clause 5.4.3". It is important to note that this certificate only validates the presence of a safety function (clause 5.4.3), with a Performance Level (PL) d. Such a level, is equivalent to an integrity level SIL2 in IEC61508 (2010) (maximum is SIL4). This does not give any guarantees when using the robot for a dedicated task, that safety will be respected. Same limitation will come with the ISO13482 (2014) for personal robots. Another limitation, is that few details are provided regarding the embedded software in such robots. It is important to note that this standard, explicitly does not include medical robots, which are "active medical devices" covered by the directive 93/42/EEC (1993), and by a set of standards from risk analysis (ISO/FDIS14971, 2006) to medical software development (IEC62304, 2006). Even if this domain, more standards seem to be applicable, only a part of the standard can be applied as presented in Guiochet et al. (2012).

A direct impact is a lack of widely accepted methods for certification of robots, and particularly for autonomous robots. Alexander et al. (2007, 2008, 2010) conclude that even if some formal methods can be efficiently applied to autonomous systems, it is not sufficient to build a safety argumentation to obtain certification (safety case). Even if important efforts have been done in recent standards towards new robotic systems, certification of collaborative robots with a decisional layer is still an open issue.

1.3 Dependability means

Dependability is defined by Avižienis et al. (2004) as the "ability to deliver service that can justifiably be trusted". Dependability encompasses many attributes, such as reliability,

safety or availability. In the dependability community, safety is defined as the absence of catastrophic consequences on the user and the environment, whereas in risk management standards it is defined as the absence of unacceptable risk. This latter definition is more adapted to the analysis of new robots, where an absolute definition of safety is actually difficult to applicable. Actually, another definition of dependability by Avižienis et al. (2004) takes into account this issue: "Dependability is the ability to avoid service failures that are more frequent and more severe than is acceptable". These authors define dependability's threats as failures, error and faults. A service *failure* happens when delivered service deviates from correct service. An *error* is a deviation in the system's state. Errors can propagate though the system and may ultimately lead to a failure. Finally, a *fault* is the adjudged or hypothesized cause of an error. A fault is active when it causes an error, otherwise it is dormant. To avoid service failures that are more frequent and more severe than is acceptable, dependability proposes four means:

Fault prevention : to prevent the occurrence or introduction of faults, including techniques coming from system engineering, good practices to design the system (section 1.3.1)

Fault removal : to reduce the number and severity of faults mainly using validation and verification techniques (section 1.3.2).

Fault forecasting : to estimate the present number, the future incidence, and the likely consequences of faults. It includes risk analysis methods. (section 1.3.3)

Fault tolerance : to avoid service failures in the presence of faults using redundancy, error detections, etc. (section 1.3.4)

The term "fault" which is used in these means may actually mean failure, error or fault. In robotics the term "fault" is sometimes misleading. For instance, uncertainties in environment perception, heuristics limits, or adverse situations, are not qualified as "faults" in its common sense, where are they are typical threats to dependability. To avoid confusion, I will use in this manuscript the term *fault* to encompass all threats to dependability.

Many techniques traditionally used for safety critical systems development may be used for autonomous robot development like hardware redundancy to increase reliability. But main techniques are actually not adapted to deal with new hazards of collaborative autonomous robots. For instance, redundancy with error masking which is used in all aircrafts, may be ineffective for a decisional component (two different plans may be correct), which lead to an impossible detection. I thus focus in the next sections on what has been done in dependability when it is specific to analysis and development of autonomous and collaborative robots, and different from what can be found in common dependability literature. As a robot is a multi-disciplinary artifact, techniques in order to improve its dependability come from computer science, robotics, automatics, mechanical engineering,

electronics, and cognitive sciences. Of course it is impossible to review all the works done in all those domains, hence I mainly focus on contributions coming from computer science.

1.3.1 Fault prevention

In a hierarchical architecture, developers have to deal with heterogeneous models and abstractions. As it is done in other domains, fault prevention in autonomous systems software is mainly implemented through the modularity of software components and the use of appropriate development tools for dealing with this heterogeneity. Component-based software or modularity first appears in generic architectures such as LAAS (Alami et al., 1998), RAX (Muscettola et al., 1998), CLARAty (Volpe et al., 2000) or IDEA (Muscettola et al., 2002). Such layered architectures can be supported by standardized middleware like ROS (*Robot Operating System*) (Quigley et al., 2009; ROS, 2015), OROCOS (Bruyninckx, 2001; Orocos, 2015), or Genom (Fleury et al., 1997; Mallet et al., 2010; Genom, 2015), which improves reuse, provide communication functions, and code generation.

Other environments, providing tools for formal specification and verification, has also been applied in the context of robotics (see ControlShell (Schneider et al., 1998), ORCCAD (Borrelly et al., 1998) or SIGNAL (Marchand et al., 1998)), but they were based on specific languages, which are not interfacing with current robotic development tools.

Associated to such tools, some researches also deal with model-driven engineering for autonomy (Brodskiy et al., 2014), in order the reduce the faults that may occur due to errors in specification or design. The Robotic Application development Process (RAP) (Kraetzschmar et al., 2010) proposed in the context of the project BRICS (2009-2013), is motivated by the absence of such methods in autonomous software development.

A way of preventing the occurrence of hazardous situations, is also to limit the robot performances or tend to reach an "intrinsically safe robot" (Ulrich et al., 1995). It concerns weight, forces, power, speed, acceleration, working area, mechanical mobility etc. Some of well known manipulators are the LWR (lightweight Robot III), developed by DLR[6] and commercialized by KUKA[7] or Universal Robots[8]. Another limitation concerns the number of degrees of freedom. It is indeed sometimes recommended like in the medical field (Glauser et al., 1993), to reduce to the lowest number of DOF necessary for the task in order to reduce complexity and thus potential faults. This approach may not be applicable when the robot must adapt itself to the environment. Additionally, all the work on compliance of robot actuation (passively safe actuators, control of active compliance/stiffness, see for instance Filippini et al. (2008); Albu-Schaffer et al. (2008); Flacco et al. (2012)) also contributes to prevent hazardous situation occurrence. Indeed, more compliant movements, will avoid harmful collisions and also induce more "natural" movements.

[6] http://www.dlr.de/rm/en/desktopdefault.aspx/tabid-3803/
[7] http://www.kuka-labs.com/en/medical_robotics/lightweight_robotics/
[8] http://www.universal-robots.com/GB/Products.aspx

Also in order to prevent hazardous situations occurrence, some work focus on human error avoidance through more human aware robot movements (see the review of Kruse et al. (2013). For instance human legible motion planning and reactive planning for collision avoidance is about calculating trajectories in order to produce movements which are more natural for humans (Mainprice et al., 2010). A robot holding an object to a human should move in front of him, and not behind; a robot arm movement should avoid to come close to human while performing a movement in its workspace (for instance move away the elbow joint from human body while interacting with him).

1.3.2 Fault removal

Fault removal aims to identify, diagnose and remove faults in the considered system. Only the first step is presented here, as the two others are quite generic. This activity includes verification activities which are mainly dynamic (run tests and detect faults through analysis of logs or with a run-time monitor) or static (static analysis, model checking, theorem proving). Nevertheless, as mentioned by Pecheur (2000); Tiwari and Sinha (2003); Menzies and Pecheur (2005), the classic issues faced by verification in control systems, are exacerbated for autonomous systems. Major issues are:

- *execution contexts* in autonomous systems are neither controllable nor completely known; even worse, consequences of the system actions are often uncertain.
- decisional mechanisms have to be *validated in a complete architecture*, since they aim to enhance functionalities of the lower levels through high-level abstractions and actions. Integrated tests are thus necessary very early in the development cycle, which is often impossible.
- the *oracle problem*[9] is particularly difficult since (a) equally correct decisions may be completely different (e.g., two different trajectories), (b) non-deterministic action outcomes and temporal uncertainties can cause otherwise correct plans to sometimes fail when executed, and (c) unforeseen adverse environmental situations may completely prevent any plan from achieving all its goals (for example, cliffs, or some other feature of the local terrain, may make a position goal unreachable).

Note that in dynamic and static verification techniques, models and mathematical tools are usually transversal and useful for other dependability means. For instance, model checking techniques might be used to specify testing oracle, or fault tolerance detection mechanisms as presented in Chapter 5.

[9] How to conclude on correctness of a program's outputs to selected test inputs?

Dynamic verification

Testing is the most intuitive way to reveal a fault: a test case is provided to system inputs, then its outputs are analyzed to determine if they are correct, which constitutes the oracle issue. When a complete behavioral model exists, it is used as an oracle: system and oracle outputs are compared. Otherwise, a partial oracle is used, which verifies properties of outputs. Mainly two types of test can be carried out in robotics. Conformity testing aims at revealing faults (and remove them), and robustness testing aims at assessing system resistance to stressful environmental conditions. In robotics, it is difficult to completely specify all situations the system is designed for. During navigation testing, it is quite impossible to define the boundary between conformity and robustness testing (e.g., when do environment conditions switch from "normal" to "stressful"? this is also called the "situation coverage" issue by Alexander et al. (2015)). This issue has an important impact on techniques chosen for testing autonomous systems. Robustness testing may also be deployed at the component level in an autonomous architecture, as it is done in Powell et al. (2012), where the functional layer is tested considering timing errors (typically delays between requests coming from the decisional layer).

One conclusion of Pecheur (2000) is that for autonomous controller, "scenario-based testing provides a very limited coverage". Indeed, in autonomous systems, their role is often limited to debugging rather than proving a thorough validation. Especially in the case of research platforms, developers check correct execution of the system for few situations. Intensive testing was however carried out on the RAX architecture for the DS1 project (Bernard et al., 2000): six test beds were implemented throughout the development process, incorporating 600 tests. The authors underline the relevance of intensive testing, but acknowledge particular difficulties regarding autonomous systems, notably the problem of defining suitable test oracles. However, new development of technologies (particularly in simulation and system modeling) lead to an increase of activities in this domain.

In Micskei et al. (2012), a framework has been developed to generate test cases for robustness testing of mobile autonomous systems. It is based on a model of system tasks (represented by UML sequence diagrams), and on a modeling of the environment (coming from ontology, specifying all possible objects of the environment and their properties). A complete oracle is impossible as it will be as complex as the system under test, and also because of the non-determinism of the decisional layer. Hence authors use a partial oracle described with sequence diagrams in order to compare messages occurrence (Horányi et al., 2013a). Also in the field of test case generation, Zou et al. (2014) present an approach based on genetic mutation. The goal is to generate cases to test collision avoidance between two drones. Considering that the oracle is based on estimation of a distance between drones equivalent to collision, the fitness function is easily implemented. Arnold and Alexander (2013) also generate test inputs including 2D worlds (map and obstacles), using procedural content generation as it is done in video games.

A major improvement for testing is the development of simulators, allowing to plug robot controller to a simulated mechanical and hardware architecture of the robot in a simulated environnement. Testing in robotics is indeed costly in terms of time, and can be harmful for the system or its environment (when testing safety for instance), and is usually performed with a limited set of environmental conditions. Simulators will cope with these issues. Actually a few simulators are sufficiently generic to integrate several software controller architecture, able to simulate gravity, frictions, and dynamic environment. We can cite Morse (2015); Echeverria et al. (2011) based on the 3D engine Blender Blender3D (2015), or Gazebo (Gazebo, 2015) (see a comparaison in Cook et al. (2014)). Most of work using those simulators for robots, aims at testing a function in relative simple conditions, and not to focus on fault identification or robustness estimation (Powell et al., 2012), but we can forecast that such testing campaigns in autonomous and collaborative robots using simulators will increase.

Another research direction in dynamic verification, is the use of runtime verification techniques reviewed by Leucker and Schallhart (2009); Delgado et al. (2004). This technique generates an oracle from properties (mainly temporal properties), which are specified by adding code into the controller software. Verification is then performed during operational life of the system. Such approach usually used in cyber-physical systems (e.g., Goodloe and Pike (2010), Kane et al. (2014)), has been used by Goldberg et al. (2005) for non regression testing of planning in robotics.

Static verification

The main difference with dynamic verification, is that static verification guarantees that all execution of a system are correct regarding the specifications. They are generally based on a system model which is an abstraction of the real system. It encompasses static analysis, model checking, and theorem proving. Some works address theorem proving (see Täubig et al. (2012) for obstacle avoidance algorithm proving) but most of the works applied to robotics are on model checking.

Model checking is based on the verification of properties of execution traces (or a reduced set) of a dynamic model (usually a state machine). Temporal logics, like CTL (Computation Tree Logic), are widely used to define these properties. In computer science the main drawbacks of these approaches are that the modeling step is not error prone (specific language) and also may not be representative of the real system. Tools also suffer from combinatory explosion. Nevertheless, increasing performances of calculators and algorithms should reduce such limitations, and increase model checking applicability in the future.

In robotics, Pathak et al. (2013) and O'Brien et al. (2014) propose to use model checking with an extension to estimate the probability that the properties are satisfied. In Scherer et al. (2005), the functional layer of a robot is verified using model checking. In this paper, verification is directly done on the Java code, which permits to avoid the modeling of the

software (but actuators have to be modeled). Simmons et al. (2000) present a way to verify the decomposition and synchronization of the controller tasks written in C++, using the model checker NuSMV.

Static verification of the planners is also an important issue in robotics. One way to validate a planning model is to define an oracle as a set of constraints that characterize a correct plan: plans satisfying the constraints are deemed correct. Such a technique was used for thorough testing of the RAX planner during the NASA *Deep Space One* project (Bernard et al., 2000), and is supported by the VAL validation tool (Howey et al., 2004). However, extensive collaboration of application and planner experts is often necessary to generate the correct set of constraints. Moreover, when the plans produced by the planner are checked against the same constraints as those included in the planning model, this approach only provides confidence about the planning engine and not the application-specific planning model. Some works (Khatib et al., 2001; Penix et al., 1998) have attempted to validate application-specific models by means of model-checking, which usually implies a manual conversion of the model into the syntax accepted by the model checker. This requires an intimate knowledge of the model checker and it is thus usually carried externally by a formal method expert, rather than by the system designer. However, some research has studied how this model transformation can be automated Cesta et al. (2010). More generally, Bensalem et al. (2014) show how planning and verification may contribute to each other.

A technique linked to model checking theoretical issues, is the supervisor synthesis originally defined by Ramadge and Wonham (1987) and Wonham (2005). Properties to check are combined with a dynamic model of the system in order to synthesize correct-by-design control software while providing formal guarantees of correctness and performance. Such approach has been used by Rutten (2001) and Bensalem et al. (2011) in order to guarantee properties like deadlock free or data freshness for instance. Johnson and Kress-Gazit (2015) studied the issue of synthesizing a robot controller taking into account uncertainties in sensing and actuating is studied (more generally robot controller synthesis is studied by Verifiable Robotics Research Group (2015)).

1.3.3 Fault forecasting

Fault forecasting aims at estimating the number, the future incidence, and the likely consequences of faults (Avižienis et al., 2004). It encompasses well-known risk analysis techniques usually classified into two categories :

- *Bottom-up*: a fault effect on the system is estimated in terms of cause-consequence, severity and probability, e.g. FMECA (Failure Modes Effects and Criticality Analysis), HAZOP (Hazard Operability). These methods are based on the use of tables listing deviations (or failure modes), their consequences and possible corrective actions.

- *Top-down*: induction of faults (and their combination) inducing an identified unwanted effect. FTA (Fault Tree Analysis) is used to deduce and represent with a logical tree the combinations of events (like faults) leading to an unwanted top event.

Such methods have been widely used for industrial robots development (Dhillon, 1991; Dhillon and Anude, 1993; Dhillon and Fashandi, 1997; Walker and Cavallero, 1996; Visinsky et al., 1994). However, several challenges appear when applying them to autonomous collaborative robots:

- cause-consequence analysis is limited due to the complexity and non determinism of the decisional layer
- probabilities of some unwanted events (e.g., software failures, human errors, adverse situations occurrence) are difficult to estimate
- hazardous situations may appear in the long term due to several decisions, and not by logical combinatory of events
- uncertainties in perception or heuristics, or human-robot interactions may induce hazardous behavior, which is difficult to analyse with the current risk analysis techniques usually focusing on fault propagation

A few studies in robotics are focusing on such issues. Suwoong and Yamada (2012) apply FMEA and FTA to a collaborative robot (not autonomous) for industry focusing on the safety related functions (emergency stop, etc.) using SIL from IEC61508 (2010). This paper actually applies those techniques to analyse safety functions which are not specific to collaborative robots. The conclusion is that new approaches are needed to analyse human-robot interactions risks. A similar approach is done for medical robots by Kazanzides (2009), where risk analysis is slightly adapted without taking account the previous issues. Böhm and Gruber (2010) chose to decompose the system into components and functions, and perform an analysis using HAZOP but they also do not adapt these techniques to take into account specificities like environment and interactions. Woodman et al. (2012) use a variant of HAZOP for software, SHARD (*Software Hazard Analysis and Resolution in Design*), associated with a predefined list of hazardous environmental conditions in the context of mobile robotics. A method called STPA (System Theoretic Process Analysis) developed by Leveson (2011), which provides guidance to users combining guide words (like in HAZOP) and fault models, applied to models, based on a process/controller/actuator/sensor representation. It has been applied to several safety critical systems, including robotics (Alemzadeh et al., 2015). All these studies actually do not address specifically the issue of deviations due to autonomy and human-robot interactions. For instance, none of them integrates a scenario-based analysis which is the basic requirement for interaction analysis as presented in Chapter 2.

Alexander et al. (2009) propose to associate several techniques. Hazard list templates and ETBA (*Energy Trace and Barrier Analysis*) are combined. This technique starts

from an unwanted release of energy, to infer the causes of this physical event. HAZOP and FFA *Functional Failure Analysis*) are used to analyse functions and data flow. Then, a FTA is performed using the results of the previous techniques. Combining all these techniques aimed at creating a reasonable approach for autonomous systems analysis, but as mentioned by the authors, it is still a combination of techniques, and further studies are required to improve applicability to autonomous systems. They also suggest (as it was done by Alexander et al. (2008)) to use the safety case approach based on the notation GSN (*Goal Structure Notation*)) for safety argumentation for autonomous system. This approach has the advantage to integrate in a single argument all evidences in favor of safety, which is an interesting approach particularly when no standards can be applied. We explore this direction in Guiochet et al. (2015) and Chapter 3 focusing on the issue of confidence in such argumentation, particularly when there are uncertainties. We propose a new model for quantitative confidence estimation based on Belief Theory for its definition, and on Bayesian Belief Networks for its propagation in safety case networks in the context of a collaborative robot development.

Taking into account the importance of the environment, Dogramadzi et al. (2014) develop a specific method called ESHA (Environmental Survey Hazard Analysis), for analyzing environmental hazardous situations that may occur (due to terrain, obstacles, etc.), without taking account the mission, or the robot tasks. The objective is to maximize the analysis using templates and checklists to stimulate the analyst study.

In conclusion, as mentioned in Dogramadzi et al. (2014), the method HAZOP-UML we developed at LAAS (Guiochet et al., 2010; Martin-Guillerez et al., 2010; Do Hoang et al., 2012; Guiochet et al., 2013), is the only approach focusing on human-robot interaction safety analysis. This method is presented in Chapter 2. It is based on an adaptation of a hazard identification technique, HAZOP (Hazard Operability), coupled with a system description notation, UML (Unified Modeling Language). This systematic approach has been applied successfully in research projects, and is now applied by robot manufacturers.

1.3.4 Fault tolerance

In dependability community, fault tolerance defined by Avižienis et al. (2004) as the means to avoid service failures in the presence of faults, is carried out with error detection and system recovery mechanisms. Fault tolerance is rarely explicitly mentioned in literature on autonomous robotic systems, where the concept of *monitoring* is preferred when referring to planning (see Ingrand and Ghallab (2014) for a discussion on the subject). Although some techniques for error detection (such as temporal control by a watchdog, model-based diagnosis monitoring, redundancy and voting) or system recovery (error containment, positioning in a safe state, and hardware and software reconfiguration) are quite common, we believe that their use is far from systematic, partly because most autonomous systems are still research platforms focusing on autonomous function development (and not on their fault tolerance). Another important point is that fault tolerance increases significantly

the cost for the development in terms of physical space or power autonomy, which are all critical challenges for embedded systems, and a fortiori for autonomous robots. Several fault tolerance mechanisms are presented in the following sections, according to the layer they are implemented (using layers presented in Figure 1.1. There are of course several mechanisms which are observing data and able to perform recovery actions at different levels in the autonomous architecture. We thus classify the mechanisms according to the layer where most of their error observations and recovery actions are implemented.

Functional layer

At the functional level, fault tolerance in robotics has been experimented for actuators, sensors or perception software errors. For instance, Crestani et al. (2015) propose to develop dedicated monitors for each software component, which is also done by Zaman et al. (2013). In these papers, timing or reasonableness checks are performed for hardware and software modules as it is done in embedded systems, but with robotic specific recovery actions impacting the decisional level (for instance, in case of error detection, Crestani et al. (2015) propose to reduce the autonomy level of the robot).

In, (Bader et al., 2014), the authors use data fusion for tolerating faults of perception in an autonomous vehicle. Such an issue is an important challenge in current applications, and will certainly increase while mobile robots may mix indoor and outdoor tasks.

Works at this level of architecture may be comparable to the ones in safety critical embedded systems. Nevertheless, a specific issue is how recovery mechanisms at the functional layer may have impacts on the decisional level.

Executive layer

In Durand et al. (2010), even if they do not explicitly mention the three layer architecture, the environment and sensors faults are detected and recovered in layer responsible of actions sequencing and execution. In case of error detection, the corresponding function is executed in a fall-back mode, and choose other functions to deliver the same task. It is also proposed to reduce the level of autonomy switching to a tele operated mode. In this case, the decisional layer is disconnected.

As previously mentioned in Section 1.3.2, controller synthesis and model-checking techniques may be used to develop fault tolerant layers in an autonomous architecture. It is the case in Py and Ingrand (2004), where a layer has been developed (conceptually close to supervisor synthesis) to observe events coming from both decisional layer and functional layer, and to block requests from decisional layer or interrupt execution of functional modules. Inconsistent requests regarding the environment are some errors in functional modules are then covered. A comparable approach, with completely different technologies is used in Bensalem et al. (2009), where a robot controller is synthesized using the BIP

technology (Behavior, Interaction, Priority). This framework is composed of a language and a tool set, to support a rigorous design, including verification and code generation.

Major issues which are still open in these approaches are the coverage rate (completeness of covered faults) and performance decrease due to the additional components or behavior. It is thus an important direction to relate risk analysis techniques to such approaches, and to perform test campaigns to compare performances.

Decisional layer

Detecting plan execution errors is known in robotics as execution monitoring (Bouguerra et al., 2008; Pettersson, 2005; Mendoza et al., 2012)). These works actually do not focus on faults in the planner itself. Moreover, studied mechanisms focus on the capacity of the decisional level to deal with errors coming from other layers. For instance, in Ertle et al. (2010), the decision level integrates mechanisms to deal with environment hazards. The planner has a model of reachable states, and it checks if safety properties are respected. It computes a distance between intermediary states and hazardous states. Gspandl et al. (2012) point out that the decisional layer may also cover faults in the hardware layer. Observations and actuator states are compared to a supposed system state. A belief management system establish some hypothesis which are transmitted to the planner.

Very few works are focusing on faults of the planner itself. Chen et al. (1995) proposes a measure for planner software reliability and compares theoretical results to experimental ones, showing a necessary tradeoff between temporal failures (related to tractability of decisional mechanisms) and value failures (related to correctness of decisional mechanisms). Later work (Chen, 1997) addresses this tradeoff through a fault tolerance approach based on concurrent use of planners with diversified heuristics: a quick but dirty heuristic is used when a slower but more focused heuristic fails to deliver a plan in time. To our knowledge, no other fault tolerance mechanisms have been proposed in this domain for this layer. We thus proposed (Lussier et al., 2007c,b) an approach for temporal planners which are a major class of decisional software components in complex autonomous systems. It is presented in Chapter 4. The fault tolerance mechanisms focus on residual development faults in planning models and heuristics. Recovery from possible errors is achieved using redundant diversified planning models. A validation framework has been used to evaluate the cost and efficacy of the fault tolerance mechanisms of a real robot software with simulated robot hardware.

Independent safety monitoring layer

A popular form of fault tolerance dedicated to safety is *safety monitoring*, through which the functional system is forced to a safe state (recovery) should some hazardous behavior be detected (error detection) by an external and independent layer. Safety monitors appear in the literature under many different terms, such as *safety bag* (Klein, 1991), *diverse*

monitor (IEC61508-7, 2010) or *safety kernel* (Rushby, 1989). In robotics and decisional systems, it has also been called *safety manager* (Pace and Seward, 2000), *autonomous safety system* (Roderick et al., 2004), *checker* (Py and Ingrand, 2004), *guardian agent* (Fox and Das, 2000), or *emergency layer* (Haddadin et al., 2011). Nevertheless, most of these works deal with non completely independent layers, with dependency issues between the safety layer and the main controller. In Tomatis et al. (2003b), safety of a museum tour-guide robot is managed through several mechanisms like operating system exception handling, a redundant monitoring software, and a redundant monitoring hardware. As the redundant software runs on the same layer as the robot controller, an independent hardware monitor has been added as an ultimate barrier. Observations and reaction means are limited to velocity and bumper monitoring triggering a shutdown of the power. An *ad hoc* approach has been used to establish the safety rules implemented in this mechanism. On the contrary, Woodman et al. (2012) propose a method based on risk analysis to establish the safety rules, and an implementation in an independent layer. In case of uncertainties or when safety rules are not verified, commands to actuators are filtered, or the robot is stopped. However, the mechanism is not completely independent from the main controller for observation means, and thus its own system state representation can be erroneous due to failures of the main controller (whereas it should also cover failures of the main controller).

Even if those latter works can be extended to the development of independent safety layers, no process for safety rule production is studied, which is still an unexplored subject. We propose in Machin et al. (2014a, 2015) and in Chapter 5 a complete framework for the generation of these safety rules taking advantage of the concept of safety margin. It starts from a hazard analysis, and is based on formal verification techniques to automatically synthesize consistent safety rules. We also study and integrate in our framework the necessary tradeoff between safety and functionality level. This approach has been successfully applied to an industrial use case on a mobile manipulator robot for co-working. Safety rules have been implemented in a real safety monitor and a fault injection campaign has been carried out to validate the approach.

1.4 Challenges for dependability of autonomous systems

As presented in Robotics-VO (2013), the roadmap in the USA at 15 years for collaborative robots is to achieve the commercialization of systems that can recognize, work with, and adapt to human or other robot behaviors in an unstructured environment (e.g. construction zones or newly configured manufacturing cells). If we mix this roadmap with the one of autonomous vehicles, we get an objective of a robot that is also capable of moving in any environment in which humans can be. Robots will be able to learn on their own how to move in previously unseen scenarios (e.g., extreme weather, sensor degradation). It is of

course implicit that such services should be delivered with a justified level of trust, i.e., with an acceptable level of dependability.

To achieve such objectives, important efforts should be done in several directions. We propose the following ones, which are of course not exhaustive, and just focusing on dependability (and not on robotics features):

Learning and adaptation Adaptation to extreme conditions, or hazardous situations is of particular interest and an important issue. For instance, while the system accomplishes its missions, the safety rules checked online should also change and be adapted according to the context. The basic safety axiom "simple is safe" usually used in industrial safety mechanisms (usually to switch off power) cannot be used any longer.

Modeling, analysis, simulation and control This vast field is a key issue in safety analysis in robotics. Model-based safety analysis for collaborative and autonomous robotic systems will allow analysis at the first step of development. It might have then important impacts on the system design. The development of robot simulators integrating more accurately physical phenomena, and able to test the robot software, is also important in order to promote and increase testing methods.

Formal methods Verification of robot controllers is a real challenge, as many techniques used in embedded systems are hardly applicable due to the decision layer in autonomous architectures. For instance, verification of planners using formal methods is still an open issue. For instance, model-checking is more and more studied to improve dependability in robotics.

Control and planning Besides verification, we also point out the area of supervisor synthesis, which should lead to more confidence in the software of the controllers. Some works are on progress, but usually focusing on the functional layer, and not on the decisional one. Note that synthesis based on formal methods may also be used for safety monitors development.

Perception The important development of perception means (particularly in 3D) did not resolve the issue of hazardous situation perception which can be really complex for autonomous collaborative systems. The integrity, and its corollary, uncertainty, of perception mechanisms, is still an open issue, for robotics that may evolve indoor to outdoor, or in physical interaction with user. For instance, with manipulator robots, it is a challenge to detect if a collision between a human and a robot is wanted or not (in some case the human may want to stop the robot by touching it).

Mechanisms and actuator Even if it is out of the scope of this manuscript, we want to mention researches done in the field of mechanisms and actuators, like compliant (v.s. stiff) actuators or parallel architectures. They will actually have an important

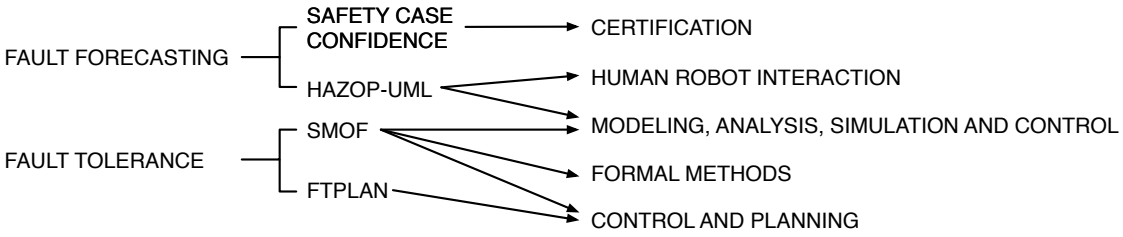


Figure 1.3: Relations between challenges and dependability means contributions

impact on the robot controllers behavior, and particularly safe reaction strategies in case of failures.

Human-robot interaction A major contribution, which is similar to the Modelling challenge, would be to have usable interaction models, in order to perform model-based risk analysis. Such models, should provide sufficient information to determine hazards, but also to induce sources of such hazards, regardless of whether they are human, mechanical, hardware or software, or a combination.

Certification The fact that such systems behavior and environmental conditions will never be deterministic, applying design standards (and adapt them) will not be sufficient. We believe that we need to provide tools in order to build safety argumentations for such systems. The issue of confidence in such arguments is particularly not explored in the robotics field.

1.5 Challenges addressed and contributions

Chapters in this manuscript address several challenges mentioned above, and cover two of the dependability means: fault forecasting (HAZOP-UML and Safety Case Confidence) and fault tolerance (SMOF and FTPlan) as presented Figure 1.4 (corresponding relations between our contributions and the field are given in Figure 1.3):

HAZOP-UML Model based hazard analysis for human-robot interaction: starting from a description of scenario of use in UML, we propose an adaptation of HAZOP to perform a systematic analysis of operational hazards occurring during robot tasks execution. This is presented in Chapter 2.

Safety Case Confidence Confidence quantification for safety argumentation for collaborative robots: based on the results of the HAZOP-UML (risk and recommendation list), we build a safety case (safety argument) aimed at making explicit all arguments.

Nevertheless, confidence in the arguments themselves need to be estimated, particularly when facing uncertainties. Chapter 3 presents a method based on quantitative techniques to estimate confidence propagation in the context of a collaborative robot.

Fault Tolerant Planner (FTPlan) Redundant temporal planners: a classic approach of dependability has been applied to a decisional component, a temporal planner. Two redundant planners are used to detect errors and perform recovery actions. Chapter 4 presents the main mechanisms and the validation of this approach through several fault injection campaigns.

Safety monitoring framework (SMOF) Producing independent safety monitoring rules has been achieved through the definition of a complete framework. It starts taking as input the results of HAZOP-UML (hazard list), and ends with the use of a tool we developed for automatic synthesis of the rules (this tool uses a model checker). Chapter 5 presents this framework. The synthesized rules are then implemented in a independent safety monitor.

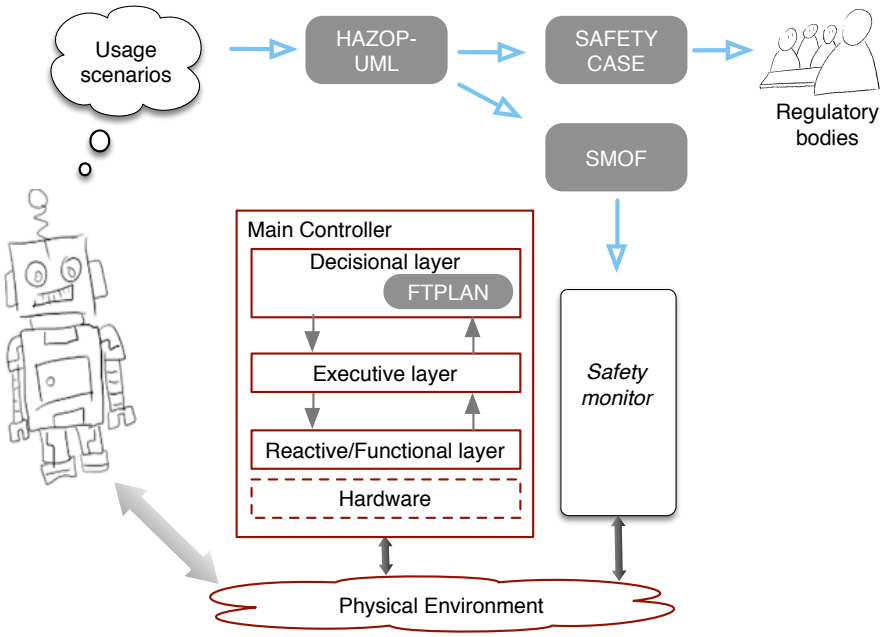


Figure 1.4: Contributions and chapters overview

2

Model based risk analysis for human robot interactions

This chapter presents a method we developed at LAAS, for the identification of hazards induced by human-robot interactions (HAZOP-UML website, 2015). This work has been done in collaboration with David Powell and Mohammed Kaâniche (LAAS-TSF), and has been published in international conferences (Guiochet et al., 2004, 2010; Martin-Guillerez et al., 2010; Do Hoang et al., 2012; Guiochet et al., 2013) and a journal (Pasqui et al., 2012), a journal has been submitted in 2015. I was co-advisor of one PhD (Quynh Anh Do Hoang, 2011-2015), a post-doc (Damien Martin-Guillerez, 2009-2010) and several internships. We validate this method on several case studies, in the context of a French Project (MIRAS, 2009-2013), and two European Projects (PHRIENDS, 2006-2009; SAPHARI, 2011-2015). We are starting to transfer this method to industry in the context of the European Project CPSELabs (2015-2018).

As already stated in Section 1.3.3, most safety analysis techniques coming from the dependability (Avižienis et al., 2004) or risk management (ISO31000, 2009) domains, could be used for the safety analysis of collaborative robots, but some specificities reduce their efficiency. For instance, the fact that robots move in unstructured and unknown environments make the verification and validation (mainly with test) non sufficient (it is impossible to guarantee that all main scenarios have been tested); the presence of users and complex non deterministic software (with decisional mechanisms) limit the use of quantitative risk analysis techniques; classical hazard analysis techniques are also not adapted to the com-

plexity of human-robot interactions. Safety standards describe overall requirements, but not specific risk analysis techniques.

Among requirements for the development of such methods, we want to stress the following ones :

1. applicable at the very beginning of the development process
2. includes the humans as a source of hazard (human error)
3. provides guidance for analysts (e.g., with guide words list)
4. focuses on operational hazards, i.e., hazards linked with the robot tasks and interactions

Among risk analysis techniques, the most widely used are Preliminary Hazard Analysis (PHA), Hazard Operability Analysis (HAZOP), Fault Tree Analysis (FTA), and Failure Mode, Effects, and Criticality Analysis (FMECA). The two first focus on hazard analysis at the very early steps of a development process, whereas FTA and FMECA are more dedicated to more advanced steps, and more focusing on reliability aspects. Thus, we chose to base our method on HAZOP, and to combine it with the system modeling language UML (Unified Modeling Language). This method developed at LAAS (HAZOP-UML website, 2015; Guiochet et al., 2010; Martin-Guillerez et al., 2010; Guiochet et al., 2013), has been successfully applied in several French and European projects (PHRIENDS, 2006-2009; SAPHARI, 2011-2015; MIRAS, 2009-2013) in collaboration with robot manufacturers (KUKA Robotics, AIRBUS Group and Robosoft). This chapter synthesizes our work on HAZOP-UML, and analyses the main lessons learned from the applications in these projects.

The remainder of this chapter is structured as follows. Section 2.1 provide background on UML and HAZOP. In Section 2.2, we present the HAZOP-UML method, and in Section 2.3, results of several experiments are analyzed and discussed. In Section 2.4, related work on model-based safety analysis is compared to our approach. We conclude in Section 2.5 by outlining the benefits and limits of HAZOP-UML, and listing some future directions.

2.1 Background

2.1.1 Unified Modeling Language

UML (Unified Modeling Language) is a graphical notation, widely used in software and system engineering domains to support early steps of the development process. Its specification is available on the Object Management Group UML page[1]. The current version

[1] www.uml.org : accessed 2015-05-15

Figure 2.1: MIRAS robot prototype during clinical investigation

(UML 2), has thirteen diagrams, that could be classified in static diagrams (e.g., class diagram) and dynamic diagrams (e.g., use case, sequence and state machine diagrams). UML is a language, and not a method, as it is not specified when each diagram must be used in which order. But, use cases and sequence diagrams are typically used at the beginning of project development.

As a running example, we will use some models of the case study MIRAS (2009-2013), an assistive robot presented Figure 2.1, for standing up, sitting down and walking, and also capable of health-state monitoring of the patients. It is designed to be used in elderly care centers by people suffering from gait and orientation problems where a classic wheeled walker (or “rollator"), is not sufficient for patient autonomy. The robotic rollator is composed of a mobile base and a moving handlebar.

Use case diagrams This diagram is the basic requirement UML model, presenting the system to analyse, the actors communicating with it, and the objectives for the use of the system: the use cases. The example of Figure 2.2 only presents a subset of the complete use case diagram (15 use cases), and the two involved actors. In this diagram, the proposed services are to help the patient to stand up (UC02), deambulate (UC01), and sit down (UC03). The system is also able to detect physiological issues and trigger an alarm (patient heartbeat and fatigue, in UC08). We also represent that the system offers the profile learning facility (UC10). In some projects using UML the mechanical part of a robot is represented as a UML actor, and the system boundary (the box around use cases) defines the robot controller (including software and hardware). We do not recommend using such an approach to perform the hazard identification, indeed, the complete system has to be studied as a whole.

This diagram provides an expressive and simple mean to communicate between developers, analysts and users. This graphical representation is always completed with a textual descriptionImportant information such pre and post conditions, and non-functional requirements are included. Use case diagram only represents functional requirements. Textual description of the normal, alternative and exception flows may also be presented with sequence diagrams as presented hereafter.

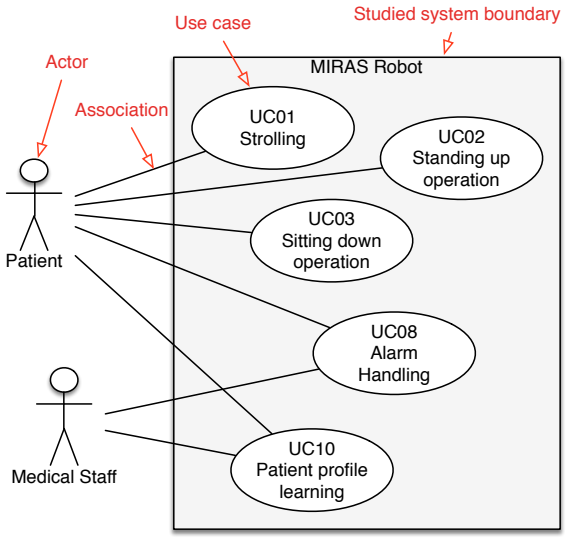


Figure 2.2: Extract of MIRAS use case diagram from Guiochet et al. (2013)

In order to prepare the HAZOP-UML study, an extract from the use case textual description should be done, with only the pre and post conditions, and also the invariants coming from safety properties in the “Non functional requirements" category. An example of such a table is given Figure 2.4 for the UC02 of the MIRAS running example.

Sequence diagrams As presented Figure 2.5, a sequence diagram describes a possible scenario of use, which is actually an instance of an UML use case. This diagram shows a nominal scenario for the UC02. Other scenarios are possible for the UC02, like alternative flow of events (e.g., the patient releases the handles while she is standing up). This second scenario will be represented with another sequence diagram (not presented here). The expressiveness of such diagram is well adapted to represent human-robot interactions, and have proven to be useful while discussing with other stakeholders who are not experts of this language (doctors, mechanical engineers, etc.). All messages exchanged between actors and the system are represented along their lifelines. In our case three types of messages are used:

- *indirect interaction* through robot teach pendant (hardware or software interfaces)
- *cognitive interaction*, e.g., gesture or voice/audio signals are exchanged
- *physical interaction*, direct contact between physical structure of the robot and the user

Use Case Name		[Name of the use case]
Actors		[An actor is a person or other entity external to the system being specified who interacts with the system and performs use cases to accomplish tasks]
Preconditions		[Activities that must take place, or any conditions that must be true, before the use case can be started]
Normal Flow	Description	[User actions and system responses that will take place during execution of the use case under normal, expected conditions.]
	Postconditions	[State of the system at the conclusion of the use case execution with a normal flow (nominal)]
Alternative flows and exceptions		[Major alternative flows or exceptions that may occur in the flow of event]
Non functional requirements		[All non-functional requirement: e.g., dependability (safety, reliability, etc.), performance, ergonomic]

Figure 2.3: Use case textual description template

Use case name	**UC02. Standing up operation**
Abstract	The patient stands up with the help of the robot
Precondition	The patient is sitting down The robot is waiting for the standing up operation Battery charge is sufficient to do this task and to help the patient to sit down The robot is in front of the patient
Postcondition	The patient is standing up The robot is in admittance mode
Invariant	The patient holds both handles of the robot The robot is in standing up mode Physiological parameters are acceptable

Figure 2.4: UC02 use case textual description with pre,post conditions and invariant

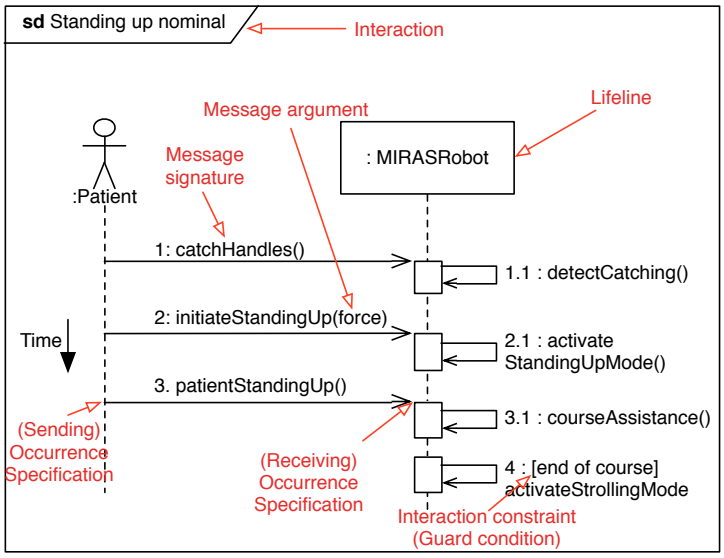


Figure 2.5: Sequence diagram for the nominal scenario of *UC01: Standing up operation*

In the example of Figure 2.5, the messages are all physical contacts, so we did not add this information which can be done using a UML annotation. In UML, a sequence diagram is a representation of an *Interaction*, where actors and the system (*Lifeline*), send some *Message* that might have *Arguments* and *Constraints*. Here the message *2:initiateStandingUp* is sent to the robot with a *force* exerced on the handles. As the time increases from top to bottom, each message has a *sending* and *receiving occurrence event*. It is also possible to represent on a message a *guard condition* for its execution (e.g., *[end of course]* of message 4).

2.1.2 HAZOP

HAZOP (HAZard OPerability) is a collaborative hazard identification technique, developed in the 70's, and is widely used in the process industries. It is now standardized by the standard IEC61882 (2001). Its success mainly lies in it simplicity and the possibility to apply it at the very beginning of the development process. It is also adaptable to the formalism used to describe a system as presented in the standard DefStan00-58 (2000). HAZOP does not consider failure modes as FMECA, but potential deviations of the main parameters of the process. For each part of the system, the identification of the deviation is systematically done with the conjunction of:

- system parameters, e.g., in the case of an industrial process : `temperature`, `pressure`, `flow`, etc.
- guide words like: `No`, `More`, `Less` or `Reverse`

Guideword	Interpretation
No/None	Complete negation of the design intention / No part of the intention is achieved and nothing else happens
More	Quantitative increase
Less	Quantitative decrease
As Well As	All the design intention is achieved together with additions
Part of	Only some of the design intention is achieved
Reverse	The logical opposite of the design intention is achieved
Other than	Complete substitution, where no part of the original intention is achieved but something quite different happens
Early	Something happens earlier than expected relative to clock time
Late	Something happens later than expected relative to clock time
Before	Something happens before it is expected, relating to order or sequence
After	After Something happens after it is expected, relating to order or sequence

Figure 2.6: Guide words list adapted from IEC61882 (2001)

The role of the guide word is to stimulate imaginative ideas and initiate discussions. A proposed list of guide words is given Figure 2.6. For instance, we can have the following conjunctions (e.g., for a chemical process):

- `Temperature` $\otimes$ `More` $\rightarrow$ `Temperature too high`
- `Flow` $\otimes$ `Reverse` $\rightarrow$ `Product flow reversal`

For each deviation, the procedure is then to investigate causes, consequences and protection, and produce document usually in a table form (similar to FMECA), with columns like: Guide word, Element, Deviation, Possible causes, Consequences, Safeguards, Comments, Actions required, etc.

Even though the HAZOP method has proved to be efficient, the results may be questionable when the boundary of the study is too vast or not well defined, or when the guide words are either too numerous or too limited for the analysis to be relevant. Another limitation is that there is no systematic method to adapt the guide words to the considered domain, so adaptation depends on the expertise of the initiators of the method. Additionally, the HAZOP method needs the allocation of human resources and suffers from combinatorial explosion when too many deviations are considered or when the analysts go into too much details. Hence, the success of a HAZOP study depends greatly on the ability of the analyst and the interactions between team members. The choice of the considered “system parameters", is of high importance, because all the study relies on it. The HAZOP-UML method proposed in this chapter is aimed at providing more guidance to analysts to identify which parameters they have to consider.

2.2 HAZOP-UML

One main issue when applying HAZOP is to identify the system parameters. We propose to use UML to partition and describe the system. The considered parameters will be then some elements of the UML diagrams. In this section we will give guidelines to identify those parameters, and the associated guide words to identify possible deviations. This work is the result of several applications and refinement, and may also be completed or modified by the analysts. Even if our objective is to propose a systematic approach, it is important to note that HAZOP-UML does not identify all hazards; First because no single hazard identification technique is actually capable of finding all the hazards (Cantrell and Clemens, 2009), and also because we will focus on the identification of the operational hazards, i.e. hazards linked to the human-robot interactions, through dynamic models of the system.

As already presented, we propose to focus on the three main dynamic UML diagrams: use case, sequence and state diagrams. For those diagrams, some generic deviations are presented in Section 2.2.1. The whole process is then introduced in Section 2.2.2, and Section 2.2.3 presents a prototype of a tool for HAZOP-UML.

2.2.1 Guide words

Instead of using the term “parameter" usually used in HAZOP studies, entities and attributes of UML elements are introduced in this section. Then for each element, a generic interpretation for a deviation is proposed. This analysis is based on the UML metamodel (OMG-UML2, 2007). The selected UML entities are : use case, message. A similar work has been done for state machines, and is presented in Do Hoang (2015).

Guide words for use cases

Figure 2.7 presents an extract from the UML metamodel, focusing on a use case. The UML class diagram notation is used to represent this metamodel. This diagram specifies that a use case may ne composed of 0 to several (noted as “*") *Behaviors*. Indeed, a use case is usually composed of a nominal behavior (or nominal scenario), and several exceptions. Each Behavior may have 0 to several *Constraints*, which are pre and post conditions. As introduced in section 2.1.1, we add to this metamodel one constraint to the *Behavior* of a *UseCase*: the invariant. Indeed, when an analyst will study all possible deviations, we argue that the non-functional requirements, which may be safety invariants (e.g., robot velocity should not exceed 20cm/s) must be taken into account. We should then consider that the attributes of a use case are: preconditions, postconditions, and invariants, which are all UML *Constraints*. For this reason, we apply the classical HAZOP guide words to the concept of constraint in a generic way and formulate an interpretation to guide the analyst. The result of this work is given in Figure 2.1. Only six guide words

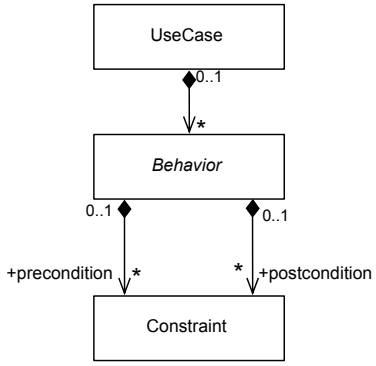


Figure 2.7: Reduced concepts for specification of use cases

Entity = Use Case		
Attribute	**Guideword**	**Interpretation**
Preconditions/ Postconditions/ Invariants	No/none	The condition is not evaluated and can have any value
	Other than	The condition is evaluated true whereas it is false, or vice versa
	As well as	The condition is correctly evaluated but other unexpected conditions are true
	Part of	The condition is partially evaluated Some conditions are missing
	Early	The condition is evaluated earlier than required for correct synchronization with the environment
	Late	The condition is evaluated later than required for correct synchronization with the environment

Table 2.1: Guide words list and generic interpretation for use cases

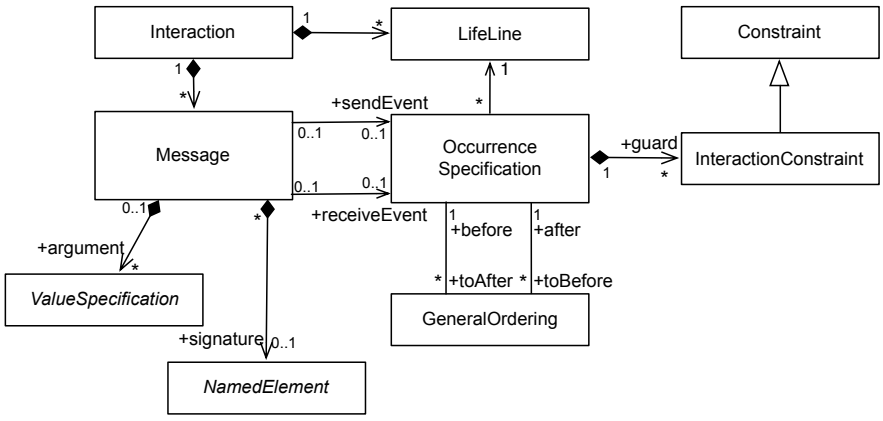


Figure 2.8: Reduced metamodel for interactions in UML (sequence diagrams) extracted from OMG-UML2 (2007)

were interpreted, we also remove many redundancies in the interpretation. Taking the example of use case “UC02 : standing up operation" described in Figure 2.4, the resulting combination of the precondition “The robot is in front of the patient" with the guide word “No", leads to the following scenario: the patient tries to standup while the robot is not properly positioned, this might induce excessive effort for the patient and a fall which is catastrophic in our application. If we consider this use case, with 9 conditions and 6 guide words, this leads to 54 possible deviations. Moreover, the interpretation of a guide word may change from an analyst to another. Nevertheless, the objective is to finally identify all hazards, and the original guide word used for the identification is of no real importance.

Guide words for sequence diagrams

Sequence diagrams are one of the graphical representation of the *Interaction* UML concept. It is composed of *Lifelines* exchanging *Messages*. This is represented in the simplified metamodel Figure 2.8. This metamodel extracted from OMG-UML2 (2007) has very little differences with the version (OMG-UML2, 2011), so we kept this representation which is simpler, and enough expressive for its use in HAZOP-UML. Based on this metamodel, we define five attributes for the Message:

1. General Ordering: the general order of the messages within the interaction
2. Send/receive event timing: event related to the clock time
3. Lifelines: send and receiving lifelines of a message
4. Interaction Constraint: guard condition on a message

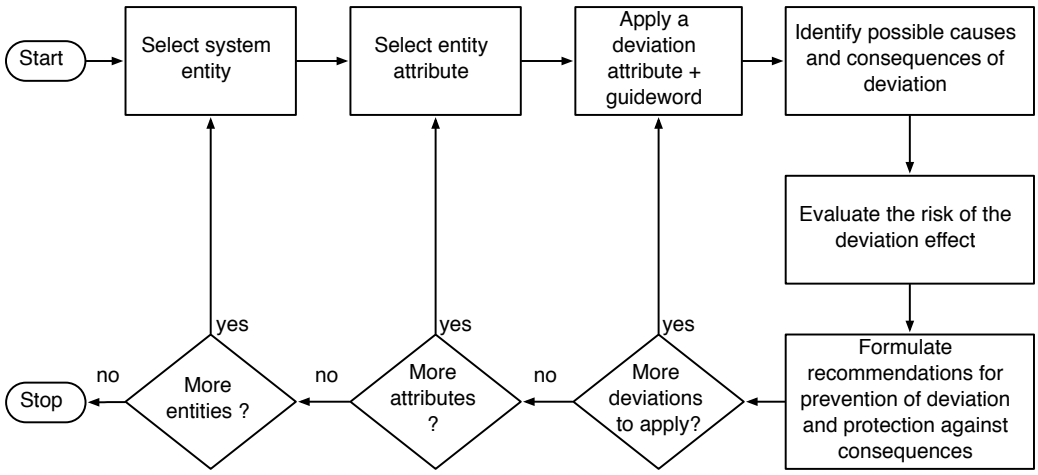


Figure 2.9: HAZOP-UML process

5. Message argument: parameters of a message

Other elements of the metamodel have not been considered, as we did not find any possible deviation or we intentionally avoid to consider them because they would have produced redundant possible deviations (interested reader may find more about UML *interaction fragments in OMG-UML2 (2011)*. The resulting table for the generic deviations and their interpretation is given Table 2.2. In tOMG-UML2 (2011) the following explanation is given: "A *GeneralOrdering* represents a binary relation between two *OccurrenceSpecifications*, to describe that one *OccurrenceSpecification* must occur before the other in a valid trace. This mechanism provides the ability to define partial orders of *OccurrenceSpecifications* that may otherwise not have a specified order." This could be interpreted as the fact that in some diagrams a *GeneralOrdering* relation can be added as a constraint. But in a sequence diagram, the physical position of the message already specifies an order for a valid trace. Hence, in our approach, we will interpret a sequence diagram as a valid trace, i.e. with a valid specified ordering of the message. This trace is descriptive (and not prescriptive like the state machine), but changing the ordering may lead to hazardous interactions.

2.2.2 HAZOP-UML process and outputs

According to the previous tables, the process to perform HAZOP-UML is the following procedure: for each entity, for each attribute, for each guide words, identify one or several possible deviations and analyse it (them). A graphical view is given Figure 2.9. The analysis of the deviation may include the identification of possible causes and consequences. Depending on the project, it is also possible to evaluate the risk (consequence of the deviation effect, and likelihood of the considered deviation). Nevertheless, this information is

Entity = Message		
Attribute	**Guideword**	**Interpretation**
General Ordering	No	Message is not sent
	Other than	Unexpected message is sent
	As well as	Message is sent as well as another message
	More than	Message sent more often than intended
	Less than	Message sent less often than intended
	Before	Message sent before intended
	After	Message sent after intended
	Part of	Only a part of a set of messages is sent
	Reverse	Reverse order of expected messages
Send/receive event timing	As well as	Message sent at correct time and also at incorrect time
	Early	Message sent earlier than intended time
	Later	Message sent later than intended time
Lifelines (receiving and sending objects)	No	Message sent to but never received by intended object
	Other than	Message sent to wrong object
	As well as	Message sent to correct object and also an incorrect object
	Reverse	Source and destination objects are reversed
	More	Message sent to more objects than intended
	Less	Message sent to fewer objects than intended
Interaction Constraint (Message guard condition)	No/none	The condition is not evaluated and can have any value
	Other than	The condition is evaluated true whereas it is false, or vice versa
	As well as	The condition is well evaluated but other unexpected conditions are true
	Part of	Only a part of condition is correctly evaluated
	Late	The condition is evaluated later than correct synchronization with the environment
Message arguments (Parameters)	No/None	Expected parameters are never set / returned
	More	Parameters values are higher than intended
	Less	Parameters values are lower than intended
	As Well As	Parameters are also transmitted with unexpected ones
	Part of	Only some parameters are transmitted Some parameters are missing
	Other than	Parameter type / number are different from those expected by the receiver

Table 2.2: Guide words list and generic interpretation for sequence diagram messages

usually too complex or impossible to obtain. On the contrary, such analysis always includes identification of recommendations to treat the deviation or its causes or it consequences (prevention and protection means). To establish such a study, the columns of a table as in Figure 2.10 are given hereafter:

1. Entity: the UML element on which the deviation is applied (here UC02 is the same for all the table so it is in the head of the table)
2. Line number: for traceability (UCx.line_number)
3. Attribute: the considered attribute (e.g., a use case precondition)
4. Guide word: the applied guide word
5. Deviation: the deviation resulting from the combination of the entity attribute and the guide word based on Tables 2.1 and 2.2
6. Use Case Effect: effect at the use case level.
7. Real World Effect: possible effect in the real world.
8. Severity: rating of effect of the worst case scenario in the real world.
9. Possible Causes: possible causes of the deviation (software, hardware, human, etc.).
10. Safety Recommendations for prevention or protection
11. Remarks: explanation of analysis, additional recommendations, etc.
12. Hazard Numbers: real world effects are identified as hazards and assigned a number, helping the users to navigate between results of the study and the HAZOP-UML tables.

In the given example Figure 2.10, a precondition of UC02 (previously presented Figure 2.4) is analyzed using the guide words No and Other than. It leads to identify the hazard HN6 (Fall of the patient due to imbalance caused by the robot).

The resulting documents are the tables as the raw artefacts, but also:

- a concatenated list of identified hazards
- a list of hypotheses made to perform the analysis, which need to be confirmed by domain experts to validate the study
- a list of safety recommendations

All those documents reference each others using numbered labels for lines, hazards (HN), recommendations (Rec), and hypothesis. Examples of a hazard table and recommendation list are given Figure 2.11 and Figure 2.12. As an example, recommendation Rec2 from Figure 2.12, covers hazards HN6 (fall of the patient), and has been formulated in the HAZOP table UC02 line 15 (UC02.15).

Project: MIRAS **Date:** 04/08/2009
HAZOP table number: UC02 **Prepared by:** DMG
Entity: UC02.Standing up operation **Revised by:** JG

Line Number	Attribute	Guide word	Deviation	Use Case Effect	Real World Effect	Severity	Possible Causes	Safety Recommandation	Remarks	Hazard Num.
UC02.15	Battery charge is sufficient to do this task and to help the patient to sit down (precond)	No/ none	Battery charge is too low but the robot starts the standing up operation	The robot interrupts its movement (standing up or walking)	Loss of balance or fall of the patient	Serious	HW/SW Failure Specification error	Worst-case electrical consumption must be evaluated beforehand. Take the lower bound of the battery charge estimation	If the robot stops during standing operation, the most probable scenario is that the patient will fall back on the seat.	HN6
UC02.16		Other than	Battery charge is high enough but the robot thinks otherwise	Robot refuses to start stand up operation	Patient is confused	None	HW/SW Failure Specification error	None		

Figure 2.10: HAZOP-UML Table extract

Num.	Hazard	Severity	References
HN4	Fall of the patient without alarm or with a late alarm	Severe	UC13.SD01.29
HN5	Physiological problem of the patient without alarm or with a late alarm	Severe	UC03.SD02.57
HN6	Fall of the patient due to imbalance caused by the robot	Severe	UC12.SD01.19,30
HN7	Failure to switch to safe mode when a problem is detected. The robot keeps moving	Severe	UC12.SD01.62,89
HN1	Incorrect position of the patient during robot use	Serious	UC13.SD01.1,2,3

Figure 2.11: Hazard list extract

Num.	Safety recommandation	Hazard Num.	References
Rec1	The standing-up profile should be validated by a human operator	HN8, HN12	UC03.SD02.91,96
Rec2	Worst-case electrical consumption must be evaluated beforehand (and display of the mean battery time left by the robot)	HN6	UC02.15
Rec22	Send regularly a network heartbeat from the robot to the medical staff control panel. Launch alarm on time-out.	HN6	UC01.SD1.15,24
Rec31	Safety margins should determined for maximum and minimum height of the robot (monitoring is required)	HN8	UC03.SD02.91

Figure 2.12: Recommendation list extract

2.2.3 A tool for HAZOP-UML

To ease the analysis of complex systems, we developed a prototype of a tool to support the method. It helps to manage the combinatorial aspects of the HAZOP method by maintaining consistency between UML models and HAZOP tables and by providing document generation and management features. The tool is built as an Eclipse plugin (www.eclipse.org) using the Graphical Modelling Framework (GMF). In this tool presented Figure 2.13, the analyst can draw UML use case and sequence diagrams. Using guide word templates, HAZOP tables are automatically generated, ready to be filled out by the analyst using choices lists.

The list of guide words, the list of columns and the list of severities are editable using the main project view. Using the template, the analyst can add a line in the table by selecting a message, and then select applicable deviations and fill the corresponding columns. When filling the table, the recommendation list and corresponding hazards are automatically generated in the project view. The toolbox of the HAZOP guide words enables deviations to be added (for example, several deviations for the same keyword). Finally a report in HTML can be generated consisting of HAZOP tables, UML diagrams, and hazards, recommendations and hypotheses lists.

2.3 Experiments and results

This section provides results of the experimentation of HAZOP-UML on three robotic applications developed within the following projects:

- ANR-MIRAS (Multimodal Interactive Robot of Assistance in Strolling) (MIRAS, 2009-2013) an assistive robot for standing up, sitting down and strolling already presented in Section 2.1.1.
- FP6-PHRIENDS (Physical Human-Robot Interaction: depENDability and Safety) (PHRIENDS, 2006-2009). The system is a mobile robot with a manipulator arm. The considered environments are workshops and factories with human workers. Collaborative work between a human and a robot is possible (e.g., the robot can give an object to the human). The arm is the KUKA Light Weight Robot (LWR), a seven degrees of freedom arm which contains torque and motor position sensors. The mobile base is the KUKA omnirob product.
- FP7-SAPHARI (Safe and Autonomous Physical Human-Aware Robot Interaction) (SAPHARI, 2011-2015). As in PHRIENDS, an Industrial co-worker operates in a manufacturing setting accessible to human workers. The mobile manipulator may encounter humans while moving between the different workstations because the operation area is freely accessible to human workers. It takes and places part boxes on

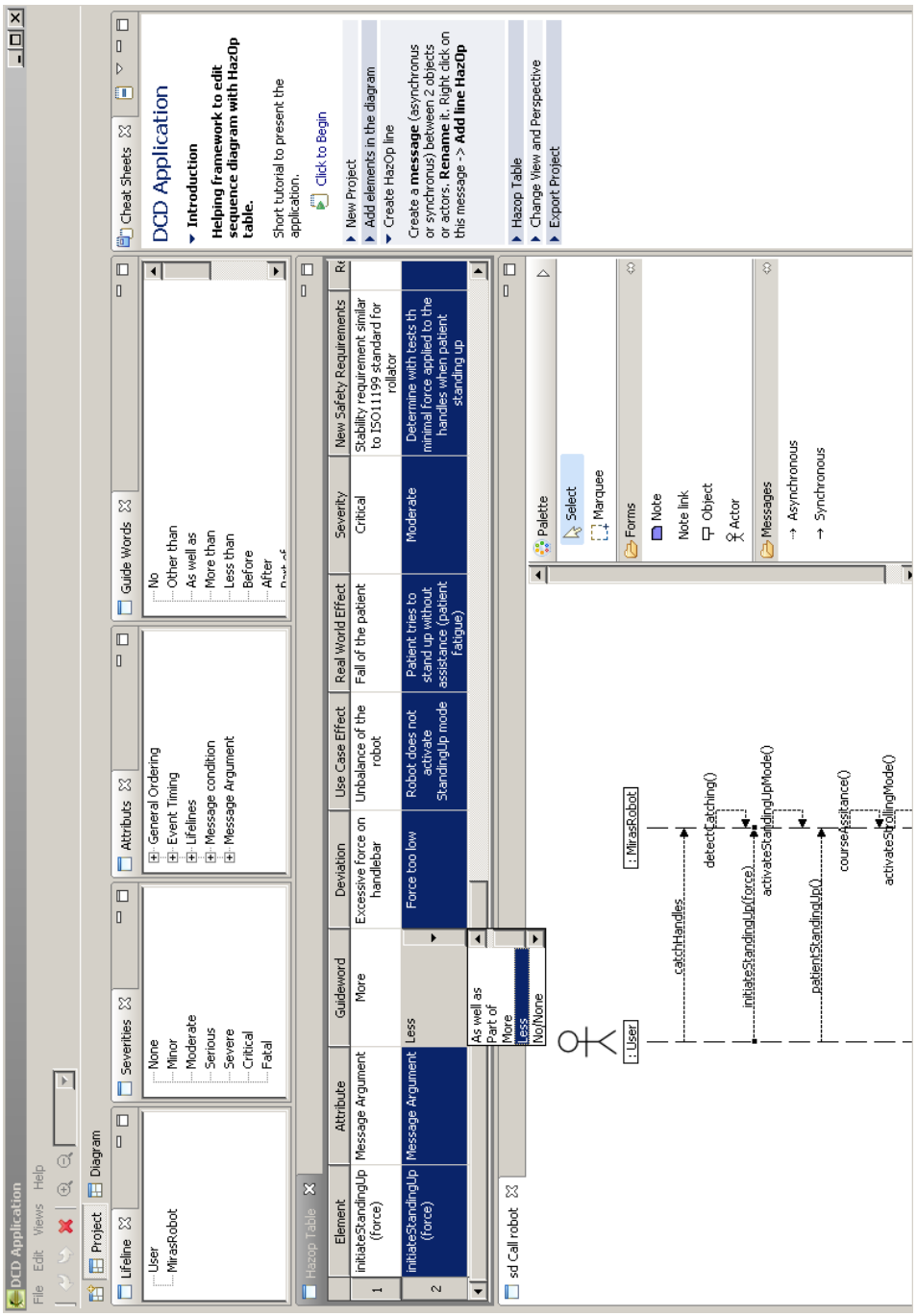


Figure 2.13: Main view of the tool to support the HAZOP-UML method

	PHRIENDS	MIRAS	SAPHARI
Use cases	9	11	15
Conditions	39	45	54
Analyzed deviations	297	317	324
Interpreted deviations	179	134	65
Interpreted deviations with recommendation	120	72	50
Sequence diagrams	9	12	16
Messages	91	52	122
Analyzed deviations	1397	676	2196
Interpreted deviations	589	163	87
Interpreted deviations with recommendation	274	85	36
Number of hazards	21	16	28

Table 2.3: Statistics for the application of HAZOP-UML for the three projects

shelves, work stations, or on the robot base in order to convey them. The robot navigates autonomously in its operation area. When the robot encounters unexpected or difficult situations the worker might intervene and help by giving the robot direct haptic instructions.

2.3.1 HAZOP-UML applicability

Classic HAZOP is usually applied in collaborative workshops, involving many partners to maximize the chances of study completeness. On the contrary, HAZOP-UML can be applied by a single analyst and then validated by experts. This comes from the fact that the study is always based on a UML model, which has been done in collaboration with stakeholders (e.g., robotic engineers or medical staff). The fact that their knowledge has been captured by UML models, makes the safety analyst task more independent from domain experts. Of course, during analysis many questions raise, and hypotheses need to be done to carry out the analysis. They need then to be validated by the experts (this is why we propose to produce a hypotheses list).

Considering that a single analyst can perform most of the work, we also analyse the effort to perform the complete analysis. Numbers are given in Table 2.3 for the three robotic projects.

For the three projects, the complexity was nearly the same (between 39 and 54 use case conditions, and 91 and 122 messages in sequence diagrams). For each project one analyst has been recruited. Those three analysts, were a post-doctoral, an engineer, and

a Dr-engineer. “Analyzed deviations" stands for the number of deviations the analyst has considered, but only a part of them leads to an ‘Interpreted deviations".

The resulting numbers show that no combinatory explosion happens, and less than 0.5 man-month was necessary for each study. Few iterations for table updates were needed (between 2 and 3). The presented tool in Section 2.2.3 was under development during those three projects, so we used a classic spreadsheet software with templates and macros. The cross checking between HAZOP tables and UML diagrams was then done by hand, which is clearly a limit that we want to reduce with our tool. However, those three projects were successful regarding the applicability of our method.

2.3.2 HAZOP-UML guide words relevance

For all projects, statistics of guide word usage have been done (see Guiochet et al. (2008b,c, 2009) for details for PHRIENDS project). Results shows that all guide words have been used by the analysts, and no guide word were redundant (i.e. no couple of guide words lead to exactly same deviations). It is possible to find the same deviation using two different guide words used by two different analysts, but this is actually not an issue, because our main objective is to find a list of hazard, what ever guide word used to identify it. To determine if the guide words list is not limiting, we only rely on the results of the application on the three projects. A formal demonstration is actually impossible, and as already discussed, no single hazard identification technique is actually capable of finding all the hazards. We thus consider that in order to propose a systematic approach, the selected guide words are sufficient to identify all the major hazards.

2.3.3 HAZOP-UML validity

Table 2.6 presents two results for validity. First, this study shows that all hazards found during the PHA (Preliminary Hazard Analysis), done by collaborative workshop between a safety analyst and robotic experts, were also identified during HAZOP-UML (performed by the analyst), and that new hazards were also found. The fact that all scenarios of use were modeled in UML significantly improves the analysis. For instance, the hazard HN11 (Disturbance of medical staff during an intervention), was only identified during use case analysis, and never mentioned during the PHA, whereas it is highly relevant in case of emergency intervention.

The second analysis presented in this Table shows that use cases (UC) and messages (Seq) analysis are complementary, whereas state machine analysis has a redundant contribution for hazard identification. For instance, HN4 identified 11 and 13 times during use case and sequence diagrams analyses, has been identified 32 more times during state machine analysis. Nevertheless, we believe that state machine analysis is also interesting to identify more sources of deviations that could be used in other risk analysis methods,

Message attributes	Guidewords	Deviations	Interpretations
1. General Ordering	No	91	75
	Other than	97	25
	As well as	91	13
	More than	91	7
	Less than	0	0
	Before	92	32
	After	91	15
	Part of	0	0
	Reverse	91	43
2. Message timing	Early	91	28
	Later	91	28
3. Lifelines	Not applicable in our case study, which considers only a single robot (and a single human)		
4. Interaction Constraint	No constraint were specified in the UML models		
5. Message arguments	No/None	91	59
	More	91	52
	Less	91	62
	As Well As	71	2
	Part of	95	31
	Other than	112	98

Table 2.4: Sequence diagram guide words utility in PHRIENDS

Use case attributes	Guidewords	Deviations	Interpretation
Conditions (39) (pre/post/inv)	No/none	42	39
	Other than	95	95
	As well as	41	23
	Part of	40	10
	Early	40	9
	Late	39	3

Table 2.5: Use case guide words utility in PHRIENDS

Num	Description	PHA	HAZOP-UML		
			UC	Seq.	State Machine
HN1	Incorrect posture of the patient during robot use	2	4	3	4
HN2	Fall of patient due to imbalance not caused by the robot		29	27	30
HN3	Robot shutdown during its use	1	2		5
HN4	Patient falls without alarm or with a late alarm		11	13	32
HN5	Physiological problem of the patient without alarm or with a late alarm		15	10	
HN6	Fall of the patient due to imbalance caused by the robot	10	51	37	10
HN7	Failure to switch to safe mode when a problem is detected. The robot keeps on moving		8		
HN8	Robot parts catching patient or clothes	3	5	4	
HN9	Collision between the robot (or robot part) and the patient	2	14	14	
HN10	Collision between the robot and a person other than the patient		5	14	2
HN11	Disturbance of medical staff during an intervention		1		
HN12	Patient loses his/her balance due to the robot (without falling)	11	1	70	1
HN13	Robot manipulation causes patient fatigue	12	1	53	21
HN14	Injuries of the patient due to robot sudden movements while carrying the patient on its seat			3	
HN15	Fall of the patient from the robot seat	2	10	12	
HN16	Frequent false positive alarms (false alarm)			3	

Table 2.6: Hazard list and occurrences in PHA and HAZOP-UML in MIRAS

and also provide safety recommendations which are different from use cases and messages ones.

2.3.4 HAZOP-UML usability

A major advantage of HAZOP-UML lies in its simplicity. Indeed, UML models have been simplified to be easily understandable by non experts without reducing its expressiveness. HAZOP is also an intuitive method. Several engineers from different domains (electronics, computer science or risk management), have been trained to the method in few days.

HAZOP-UML is completely integrated and consistent with the development process. Indeed, same UML diagrams were used in the projects, to define the scenarios. This helped us for each iteration in the development process to easily update the HAZOP tables. This traceability is an important issue in safety analysis methods, which are usually applied once due to the cost to apply them.

Among HAZOP-UML limitations, we remind that HAZOP-UML is focusing on operational hazards (linked with the robot tasks). We thus do not consider “machine" hazards already defined in many standard, like electrocution, explosion, etc. As already mentioned, this method should be completed by other hazard analysis techniques. A second limitation is the fact that the UML models and HAZOP tables do not explicitly mention the environment conditions of execution. For instance, a similar scenario but with high or low level of light might change the deviations and their consequences. It is still an open issue and an integration in the UML models would be an interesting direction. Last but not least, the HAZOP-UML has the same drawback as other risk analysis methods, which is a difficult

determination and expression of the hazard because of the fuzziness of a hazard definition ("potential source of harm", from ISO/IEC-Guide51 (1999)) which may designates both a cause or a consequence. Three columns in the HAZOP table can represent a hazard: deviation, use case effect, real word effect. In many tables, we found that some real word effects were already mentioned as use case effects in other HAZOP table lines. We chose to reduce the number of hazards, taking into account only the "real word effect" as a hazard, but for some cases where it was obvious that the treatment would be completely different, we also took into account the deviation and use case effect. For instance, in Table 2.6, the hazard HN2 (Fall of patient due to imbalance not caused by the robot) and HN6 (Fall of the patient due to imbalance caused by the robot), lead both to the fall of the patient, but have been differentiated. Even if we provide a well guided method, extraction and formulation of hazards list requires a high level of expertise of the safety analyst, in order to choose the right level of description of a hazard.

2.4 Related work on model-based hazard identification, tools and methods

This section presents related work, focusing on model-based safety analysis, and more particularly those using UML. The concept of "model-based" refers to the fact that a safety analysis technique (e.g., FTA) is based on an abstract representation of the studied system. This was already done at the very first hours of the risk analysis techniques using for instance block diagrams, or had-hoc representations. The quite recent model-based term, usually refers to the use of standardized models (like UML) and the possibility to have tools assisting analysts to produce automatic, or semi-automatic safety analysis based on a system model. Generally, model-based safety analyses focus on the following issues (Blanquart, 2010):

1. Fault propagation analysis
   (a) *bottom-up*: a fault effect on the system
   (b) *top-down*: induction of faults inducing an unwanted effect
2. Dependability (or safety) properties verification
3. Quantification of probability of unwanted events

Many high-level modeling languages for safety analyses have been defined to cover those points. Just to cite some of them, HIPS-HOPS (*Hierarchically Performed Hazard Origin and Propagation Studies*) and its associated tool developed at Hull university [2], automatically generates fault trees and FMECA tables starting from system models (e.g., Simulink

[2] http://hip-hops.eu (accessed 2015-05-15)

models). For each component, fault annotations are given, and the tool propagates those faults to build safety models (e.g., Fault trees). Altarica (Boiteau et al., 2006; Lipaczewski et al., 2015) also propose means for fault tree generation or properties verification from system and reliability models. Additionally, many European research projects addressed model-based safety analysis: ESACS (2001-2003)[3] in transportation domain, followed by ISAAC (2004-2007) [4] in avionics, then CESAR (2009-2012) [5] followed by CRYSTAL (2013-2017) [6] for embedded systems. Previous techniques and works, usually rely on a precise description of the system behavior, which is usually not available at the beginning of a human-robot project.

The proposed method in this chapter falls within the scope of fault propagation analysis, and can be described as a "middle-up approach", as we do not start from "faults" but from deviations. Our objective is then to identify hazards (and hazardous situations) during human-robot interaction. Very close work is proposed by Leveson (2011), with a method called STPA (System Theoretic Process Analysis), which provides guidance to users combining guide words (like in HAZOP) and fault models, applied to models, based on a process/controller/actuator/sensor representation. Many recent applications of STPA can be found, e.g., in robotics (Alemzadeh et al., 2015), space (Ishimatsu et al., 2010), railway (Thomas and Leveson, 2011) or automotive (Sulaman et al., 2014). One difference with our approach is that scenarios are actually not modeled in this approach. Users are represented as "controllers", which is not clear while describing human-robot interactions. STPA objective is also different in the way that it really focuses on the identification of cause-consequence chain, which is not the objective of HAZOP-UML (only find the hazards and hazardous situations). We also propose to use UML which is not the case in STPA. On the contrary, the work done in the CORAS project (CORAS, 2014; Bjørn Axel Gran and Thunem, 2004), is based on UML to analyse security. Even if we focus on safety, our objectives are the same as this project. A major difference is that we strongly interconnect UML models and the risk analysis technique HAZOP, which was not addressed in CORAS.

Our risk analysis approach is based on a re-interpretation of HAZOP guidewords in the context of some UML diagrams. A similar approach has been followed in some previous studies considering UML structural diagrams (Hansen et al., 2004; Gorski and Jarzebowicz, 2005; Jarzebowicz and Górski, 2006) and dynamic diagrams (Johannessen et al., 2001; Allenby and Kelly, 2001; Arlow et al., 2006; Frantz et al., 2007; Srivatanakul, 2005). In all those papers, the guide words were quite reduced (e.g., only omission and commission) or the link with UML language elements was not fully explored. We actually extend the results of those studies, focusing only on use case, sequence and state machine diagrams (not presented here), in order to explore deviations during operational life. We also paid

[3] www.transport-research.info/web/projects/project_details.cfm?ID=2658
[4] http://ec.europa.eu/research/transport/projects/items/isaac_en.htm
[5] www.cesarproject.eu
[6] www.crystal-artemis.eu

a particular attention to the human errors expression and analysis in this method, which was absent from the previous papers.

2.5 Conclusion

We proposed in this chapter a new method for the safety analysis of human-robot interaction called HAZOP-UML. To build this method we used the UML metamodel to identify the basic elements of three dynamic models. We then proposed three guide words tables for use cases, messages of sequence diagrams (and state machines as presented in Do Hoang (2015)). Those guide words tables help the safety analyst to imagine possible deviations for every elements of those dynamic models. Those deviations are then reported in HAZOP tables, where causes, consequences, and recommendations are formulated. This process produces lists of hazards, recommendations, and hypotheses.

This method has been applied successfully on several projects, and we presented in this chapter a general analysis of the benefits and the limits of the method. We particularly focus on the applicability and validity of the approach. Main advantages of HAZOP-UML are:

- simple (training and application)
- applicable at the first step of the development process
- limits the combinatory explosion
- consistent with system models, and inherits of system modeling benefits: traceability and modifiability
- easily supported by a computer assisting tool

Even if the models and HAZOP tables can be easily achieved, the main limit lies in the necessity of a high expertise to formulate hazards from HAZOP tables. It is up to the safety analyst to determine the right level of detail for the hazard identification.

Additionally to the three projects presented in this chapter, HAZOP-UML has also been used as a first step of the safety argumentation presented in Chapter 3. It is also an entry point for our method to build independent safety monitors in the context of autonomous robots presented in Chapter 5. We also plan to use it as an entry point for defining virtual words for testing mobile robots in simulation. A future direction is the complete transfer to industry, which is already started in the European project CPSELabs (2015-2018).

3

Safety argumentation confidence assessment

The work presented in this section is more recent than the previous one on HAZOP-UML. Nevertheless, it is a new important direction we want to develop at LAAS. I collaborated with David Powell and Mohamed Kaâniche (LAAS-TSF) while we were supervising PhD of Quynh Anh Do Hoang (2011-2015). Preliminary results have been published in workshops and international conferences (Do Hoang et al., 2012; Guiochet et al., 2013, 2015). Since 2014, I collaborate with Gilles Motet (LAAS-TSF) on this subject. We are now supervising a PhD (Rui Wang, started in 2014), to extend the results to the confidence assessment of safety critical development processes.

Safety cases are used in several critical industrial sectors to justify safety of installations and operations. As defined in the standard DefStan 00-56 (2004): "a Safety Case is a structured argument, supported by a body of evidence, that provides a compelling, comprehensible and valid case that a system is safe for a given application in a given environment". An important research work has also been initiated to deliver guidelines to document safety cases. An initial work developed at York University Kelly (1998), based on an adaptation of Toulmin argumentation model Toulmin (1958), led to the proposal of the Goal Structuring Notation (GSN). Other proposals such as CAE for Claims-Argument-Evidence Bishop et al. (2004) and KAOS (Knowledge Acquisition and autOmated Specification) Dardenne et al. (1993), but they did not reach the maturity of GSN GSN-Standard (2011). The Object Management Group (OMG) has also delivered a metamodel for the argumentation

approach OMG-ARM (2013). The goal of these approaches is to make more explicit the supporting arguments for a top-level claim.

Given a claim and a supporting argument, an important and growing issue is to understand how much confidence one could have in the claim and how the different arguments contribute to such confidence. For instance, let us consider the classical example of the claim "{System X} is safe", supported by the evidence that all specific hazards have been eliminated as presented in Figure 3.1. Main concepts of GSN are presented here: goals present claims forming part of the argument; Strategies describe the nature of inferences that exist between a goal and its supporting sub-goal(s); Solutions present a reference to an evidence item (results of a fault tree analysis for instance); Contexts present contextual artifacts (they could be a reference to contextual information, or statements). Other elements are used in GSN but not presented here as our proposal focuses on these main components of GSN. Each element of such an argument may be subject of uncertainties, such as "do all the hazards have been identified?" or "is the treatment of hazard n effective?". Moreover, considering that argument structures tend to grow excessively, it may become too complex for third parties to analyse the argument. Therefore, appropriate methods to assess confidence in the argument structures and supporting evidence are required. Three main challenges are of particular interest: how confidence could be formally defined, how confidence could be quantitatively estimated, and how confidence in argument leaves could be propagated to assess the impact on the main claim confidence.

In this chapter we mainly address the first and third issues by introducing a new method for defining and propagating a quantitative estimation of confidence of a safety case. After presenting related work in Section 3.1, we introduce our definition of confidence based on belief theory in Section 3.3. This definition is used in Section 3.4 where details about confidence propagation are given. Finally, in the conclusion we will discuss about first results and open issues in this area.

3.1 Related work

The issue of confidence in argument structures has already been addressed by several works, with slightly different objectives and scopes. Table 3.1 presents a common framework to analyze some relevant related work considering the following dimensions:

- Argument modelling: construction of the "case" which may be based on GSN or other notations
- Argument uncertainties identification: uncertainties in inferences and arguments elements are identified
- Confidence modelling: construction of a confidence case, with explicit representation of dependencies between the uncertainties

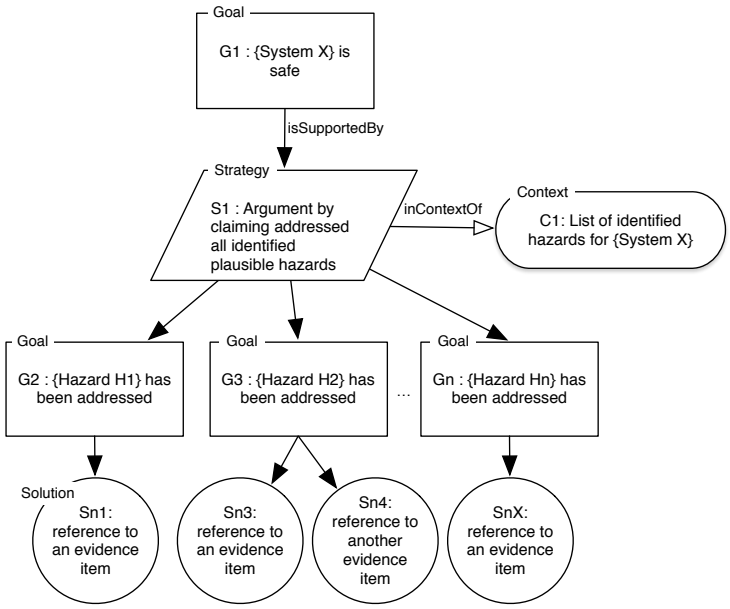


Figure 3.1: GSN example adapted from Hazard Avoidance Pattern Kelly and McDermid (1997)

- Confidence estimation: theoretical framework for quantitative estimation of the confidence
- Decision support: provide support based on the quantitative estimation in order to make a decision for the acceptability of the argument, or its improvement.

Qualitative approaches

In Hawkins et al. (2011), the inventors of GSN address the confidence issue, by proposing to split a traditional safety case in two pieces. The first is the safety argument, showing all evidences, and the second is a confidence argument that addresses confidence in evidences, contexts, and individual inferences. This confidence argument is also represented with GSN. It starts by adding to the safety case some possible uncertainty sources, which are called Assurance Claim Points (ACP), that are attached to inferences (the arrows connecting claims), contexts (explanatory information), or solutions. Then, for each ACP, an argumentation mainly focuses on demonstrating that the ACP is trustworthy and appropriate, which is built using GSN. Another proposal Anaheed et al. (2012), is based on the ACP but only focuses on Context and Solution elements. The authors propose to use a map (Common Characteristic Map) as a check list to identify sources of uncertainties,

	Argument modelling	**Argument uncertainties identification**	**Confidence modelling**	**Confidence estimation**	**Decision support**
Littlewood and Wright (2007)			Bayesian network	Probability law	
Zhao et al. (2012)	Argumentation Metamodel (ARM) based case	Based on Toulmin model	Bayesian network (with Hitchcock criteria)	Probability law (with basic logical gates)	
Denney et al. (2011)	GSN		Bayesian network	Probability law and tool support with AgenaRisk	
Cyra and Górski (2011)	Trust case based on Toulmin model			Dempster-Shafer Theory	Decision level associated to confidence level
Anaheed et al. (2013)	GSN			Dempster-Shafer Theory	Decision based on the confidence value
Anaheed et al. (2012)	GSN	Common Characteristic Map (CCM)	Confidence case based on GSN		
Goodenough et al. (2012)	GSN	Based on Assurance Claim Points (ACP)	Confidence case in GSN	Baconian probability	
Hawkins et al. (2011)	GSN	Based on Assurance Claim Points (ACP)	Assurance case in GSN		

Table 3.1: Different approaches for managing confidence in safety case

with recursive dependencies. For instance, if a safety case includes a solution which is a "Process result", they propose the generic uncertainties related to "the use of a language", "the use of a tool", "the use of a mechanism", "the involved artifacts", etc. All those characteristics are then refined, with possible recursive dependencies.

The proposed approach in Goodenough et al. (2013) is quite similar, adapting the defeater concept from Defeasible Reasoning theory introduced by Pollock (2008). These defeaters that could be compared to previous ACP, or weaknesses in the argumentation, are then analyzed to be reduced one by one.

Both previous proposals focus on the identification of the weaknesses in an argumentation, and present methods for a well structured approach. Nevertheless, such approaches may lead to complex confidence cases. Although controversial, we believe that quantitative estimation approaches may help to analyze the safety case confidence. For instance, it can support sensitivity analyses to identify the weak elements of an argumentation.

Quantitative approaches

This group of approaches tries to apply mathematical formalism to capture lack of confidence in argument elements. Apart from some proposals based on simple mathematical models as in Goodenough et al. (2012) where the number of uncertainties is estimated, two main ways of approaching the problem can be identified:

- Bayesian Belief Networks (BBNs): in this case the uncertainty is interpreted as a probability. BBNs are then applied to deduce the confidence in a goal from credibility of its backing arguments. Some authors directly use BBNs for modeling arguments and confidence. For instance, in Hobbs and Lloyd (2012), they only use BBNs and commercial tools to calculate "trustworthy", which is actually a conditional probability. With a similar approach, authors of Littlewood and Wright (2007) particularly focus on the diversification in argumentation, calculating how a "multilegged" argument (a claim is supported by two evidences) impacts the probability (interpreted as a confidence level) of achieving the main claim. However, they directly use BBNs, without any safety case. On the contrary, Zhao et al. (2012) propose to apply to each claim of a Toulmin model argument, a Bayesien network pattern showing relationships between uncertainties in the argumentation based on Hitchock criteria Hitchcock (2005). However, confidence propagation is not clearly analyzed and justified. In Denney et al. (2011), the authors present an interesting approach to build a BBN from the safety case, and use the work of Fenton and Neil (2012), to define a distribution of confidence for each argument element, but they do not propose transformation rules between safety case in GSN and the confidence BBN. The confidence propagation formulas are also not justified.

- Dempster–Shafer (D-S) theory of evidence. These approaches are based on the belief theory developed by P. Dempster in 1967, and extended by G. Shafer in 1976.

A common justification for its use, is that probability theory does not make difference between epistemic and aleatory uncertainties Aguirre et al. (2013). In the D-S approach, belief, disbelief and epistemic uncertainty are explicitly quantified. An important work by Cyra and Górski (2011) is based on this theory. The authors, propose to build "Trust cases" based on Toulmin concepts, and to directly associate levels for belief and uncertainties, linked with a decision to accept or not an argument element. In this case, they do not build a confidence case, but directly propose a method and a tool for decision support. As presented later, they do not explicitly take into account confidence in the inferences of the argument. Anaheed et al. (2013) directly reuse the previous work, with a limited version, only considering that for each argument element it exists a level for "sufficiency".

In summary, defining and measuring confidence in assurance claims is an important and open issue. A framework for determining confidence is needed, and this chapter presents some initial steps to fulfill this objective.

3.2 Proposed approach overview

Our objective is to propose a method to identify weaknesses in safety case, in order to improve it. Referring to Table 3.1, our contribution focuses on the following steps presented Figure 3.2:

- Argument modelling: the safety case is built using GSN
- Confidence modelling: we propose to annotate the GSN models and transform them into a confidence network
- Confidence estimation: confidence in the network leaves are estimated and propagation formulas are used
- Sensitivity analysis: impact of confidence variations is analyzed to identify weaknesses of the safety case.

3.3 Measuring confidence

Confidence may be used as a common concept for different theories, including probability, and D-S. As Cyra and Górski (2011); Anaheed et al. (2012), we define confidence using the D-S approach. In this theory, a belief function is defined from the powerset $\mathcal{P}(\Omega)$ of possible events into $[0; 1]$. For instance, let ω be the state of an indicator light that can have two values on and off, then $\Omega = \{on, off\}$ and $\mathcal{P}(\Omega) = \{\{on\}, \{off\}, \{on, off\}, \varnothing\}$. In this example the belief function Bel, is defined as the mass m of belief such as $Bel(\{on\})$

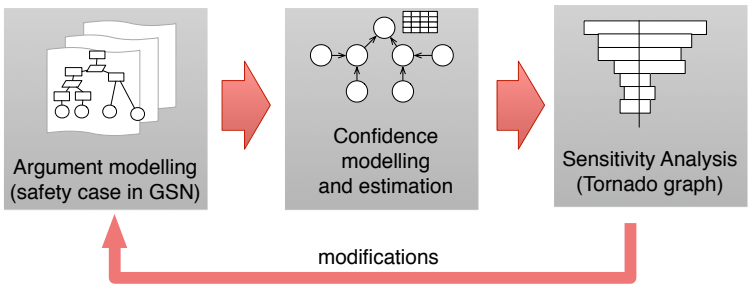


Figure 3.2: Overview of the proposed method

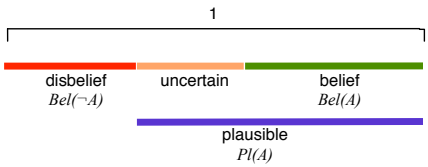


Figure 3.3: D-S theory concepts, with a Boolean set

represents the credibility of the light to be ON. As an example, a possible estimation would be $Bel(\{on\}) = m(\{on\}) = 0.2$, $Bel(\{off\}) = m(\{off\}) = 0.5$ et $m(\{on, off\}) = m(\Omega) = 0.3$. When events are Boolean, like in this example, we can sum-up the D-S concepts with the Figure 3.3 (Plausibility is another D-S concept which is not included in this chapter).

We will consider in a safety case that all elements leaves are observed, and that they cannot be false. Hence, for an element A, $Bel(\overline{A}) = 0$. This led us to define confidence and uncertainty as the belief functions:

$$\begin{cases} m(A) = Bel(A) = g(A) \in [0,1] & : \quad confidence \\ m(A, \overline{A}) = 1 - g(A) \in [0,1] & : \quad uncertainty \\ m(\overline{A}) = 0 \end{cases} \tag{3.1}$$

In the context of safety case, we consider two types of uncertainty sources, which are similar to those presented in Hawkins et al. (2011) named "appropriateness" and "trustworthiness". For instance, in the very simple safety case presented in Figure 3.4, two sources of uncertainties may be identified:

- uncertainty in the fact that B is appropriate for the inference "A is Supported by B"
- uncertainty in the solution B itself : are the tests trustworthy?

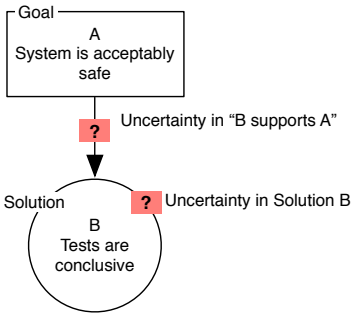


Figure 3.4: Uncertainty points in a simple inference

3.4 Propagating confidence

3.4.1 Argument types

The very basic inference is the simplest one, "A is Supported by B". Nevertheless, most of arguments are more complex than direct one-to-one inference. For instance, let us consider the example presented with the main claim "A: System is fit for use", supported by both "B: Tests are conclusive" and "C: Formal verification has been performed". In that case, we can expect that both evidences independently increase the level of confidence in A. This concept is presented as "alternative argument" in Cyra and Górski (2011): even if there is no confidence in B, the fact that C also independently supports A will preserve some level of confidence.

An another form of inference, is presented in the GSN «Hazard Avoidance Pattern» proposed in Kelly and McDermid (1997), presented Figure 3.1. In that case, the main Goal "System is Safe", depends on all the sub goals together (we do not consider "Strategy" as a node, because it is only a descriptive element). Each of the premises covers a part of the goal. Cyra and Górski (2011) propose to name such an argument a "complementary argument".

Figure 3.5 present those two types of arguments, with the inference "A supported by B and C". We also illustrate the fact that in both types of argument, the sub nodes may have a different weight in the overall confidence in the claim A. Other types of arguments may be included, as introduced in Cyra and Górski (2011); Anaheed et al. (2013), but they are not included in this chapter.

3.4.2 Simple argument

The basic inference, "A is supported by B" can apply to the cases (a) a goal is refined into a subgoal and (b) a goal is supported by an evidence, as presented in Figure 3.6. In this

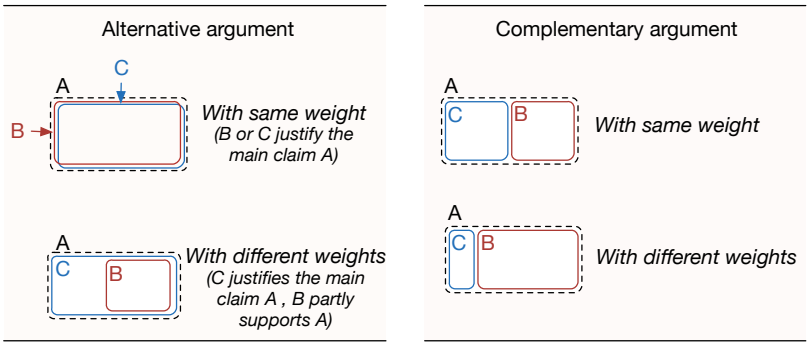
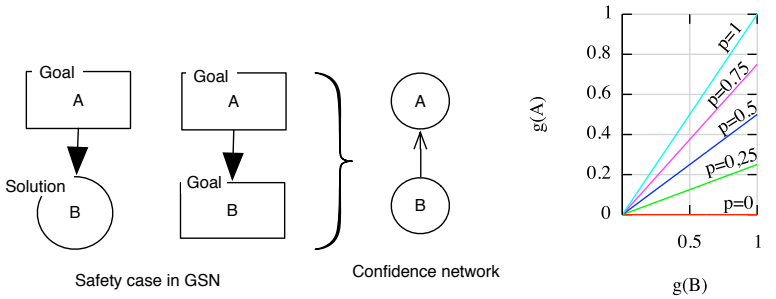


Figure 3.5: Alternative and complementary arguments



(a) (b)

Figure 3.6: (a) GSN Simple argument transformation into confidence network and (b) $g(A)$ in function of $g(B)$, for $p \in [0; 1]$

case, the confidence network is represented like a BBN, using two nodes and one edge. We propose to use the following table to describe the confidence propagation:

g(B)	0	1
g(A)	0	p

In this table, the confidence in A is estimated when there is no confidence in B (i.e. when $g(B) = 0$), then $g(A) = 0$, and when there is a maximum confidence in $g(B)$. In this case, the confidence in A depends on a factor p, which represents the confidence in the inference "A is supported by B". The final confidence is obtained using this table as a probability table: $g(A) = p * g(B)$. The result is a linear dependency $g(A)$, illustrated in Figure 3.6 considering different values for p and g(B).

3.4.3 Alternative arguments

As presented Figure 3.5, several arguments may support a claim with an independent influence. It is important to note that in this Figure, we do not represent the confidence, but the way each argument supports the main claim. In this case, the confidence in A, may be increased by the confidence in both B and C. Such approach could be applied to Solutions or sub-goals as presented Figure 3.7. The Strategy node is not part of the

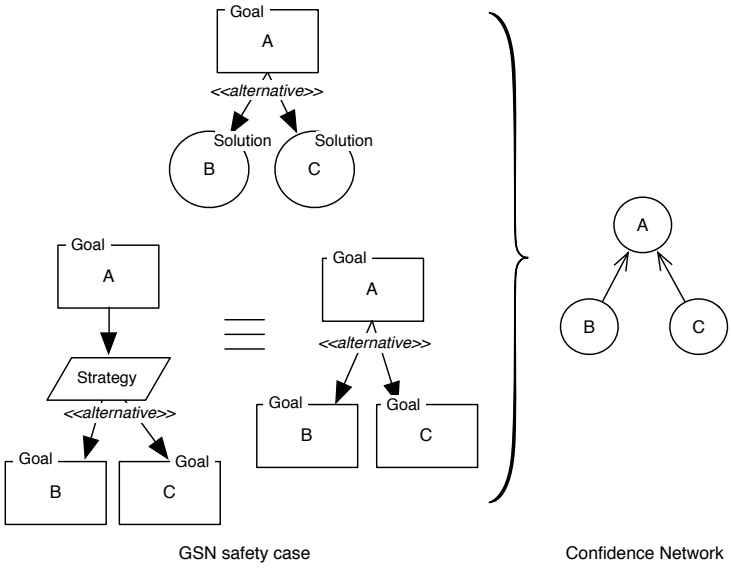


Figure 3.7: Alternative argumentation transformation into confidence network

confidence network, as it only gives explanations on the choices made for argumentation.

We chose for this argument type to use a *leaky noisy-or* as defined in probability theory Díez and Druzdzel (2007). It was originally introduced in Pearl (1988), and it is based on

a logical OR between parent nodes (Y_i) and a child node(X), but it includes the fact that the relationship between parents and the child node are not necessary deterministic. The *leaky* effect corresponds to the fact that even when both parents (B and C) have 0-value probability, there is still a "leaky" probability for the child node. For probabilities, the mathematical function is, with Y_v the set of Y_i in state $\{True\}$:

$$P(X = \{True\}|Y_i) = 1 - (1 - l) * \prod_{Y_i \in Y_v} (1 - p_i) \tag{3.2}$$

with $p_i = P(X|Y_i, \{\overline{Y_i}\}_{j \neq i})$. In its application to confidence, we do not consider the leaky effect, it is indeed obvious that if there is no confidence in B and C ($g(B) = g(C) = 0$), then the confidence in A is zero, i.e. $g(A) = 0$. Consequently, we obtain the following equation:

$$g(X|Y_i) = 1 - \prod_{Y_i \in Y_v} (1 - p_i) \tag{3.3}$$

According to 3.3, the resulting table for two parents is:

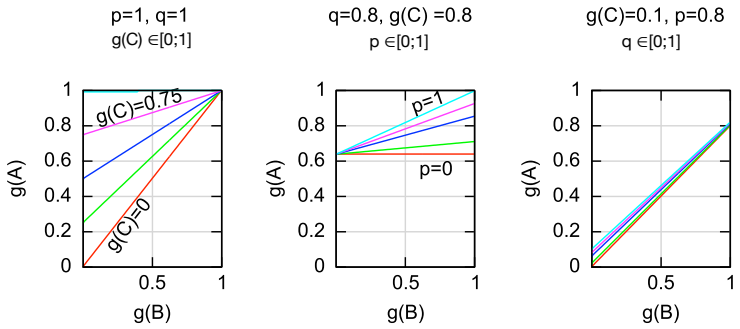
g(B)	0		1	
g(C)	0	1	0	1
g(A)	0	q	p	1-(1-p)(1-q)

This leads to the confidence formula $g(A) = p * g(B) + q * g(C) - g(B) * g(C) * p * q$. p and q respectively represent the confidence in A in case one only has confidence in B or C. Figure 3.8 illustrates the evolution of confidence g(A) for different situations:

- Figure (a) where $p = q = 1$ illustrates that increasing the confidence in $g(B)$ alone or $g(C)$ alone, automatically increases $g(A)$. For instance, for $g(C) = 0.75$ and $g(B) = 0.5$, the resulting confidence is 0.875. Confidence of 1 for A, occurs only if $g(B)$ or $g(C)$ reaches 1.
- Figure (b) shows influence of p on $g(A)$. For a low confidence p in the inference "A is supported by B", the confidence in A only depends on confidence in C ($g(A)$ is constant for $p = 0$).
- Figure (c) shows that for a low value of $g(C)$ (0.1), the variation of q, which is the confidence in the inference "A supported by C", has no effect on $g(A)$.

3.4.4 Complementary arguments

Complementary arguments are used when a set of solutions or subgoals are required simultaneously for supporting the main goal. However, a weight for each element is assigned to rate its relative importance. For instance, in the "Hazard Avoidance Pattern", some



(a) (b) (c)

Figure 3.8: Alternative argument: g(A), in function of g(B), g(C), p and q

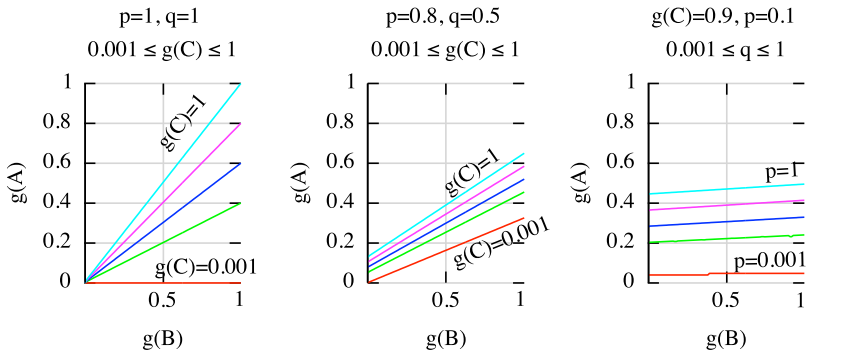
hazards may have a less impact on the overall safety, and the lack of confidence in their treatment, may induce less reduction in the main confidence, than for other more severe hazards. Several models are used in the literature for such arguments, such as simple And-gate Zhao et al. (2012), weighted mean Denney et al. (2011), or Noisy-And Hobbs and Lloyd (2012). In our case, after several simulations, we decided to define our own Noisy-And, to obtain the trends that are relevant for complementary argumentation. In this case, we based our calculation on the uncertainty as defined in equation 3.1 and using the leaky noisy-or defined in equation 3.2, but taking for the leak $v = 1 - l$. We then obtain the following confidence table:

$m(B, \overline{B})$	0		1	
$m(C, \overline{C})$	0	1	0	1
$m(A, \overline{A})$	$1 - v$	$1 - v.(1 - q)$	$1 - v.(1 - p)$	$1 - v.(1 - p).(1 - q)$

To calculate the confidence table, we apply the relation $g(X) = 1 - m(X, \overline{X})$, and we also decided to fix $g(A) = 0$ when $g(B) = g(A) = 0$ (which should be obtain for whatever weight of B and C). We thus obtain the following table:

$g(B)$	1		0	
$g(C)$	1	0	1	0
$g(A)$	v	$v.(1 - q)$	$v.(1 - p)$	0

One main difference with other research works, lies in the interpretation of the parameters. In our case, p and q represent the weight of B and C to decrease confidence (increase uncertainty). In the context of confidence calculation, we also propose to introduce a relation between leak value v, p,and q such as: $v = (p + q)/2$. Indeed, when p and q are lower than 1, it means that the confidence in the inference is less than one. The generalization



(a) (b) (c)

Figure 3.9: Complementary argument: g(A) in function of g(B), p and q

of this constraint to a complementary argument with n parents is:

$$v = \frac{1}{n}\sum_{i=1}^{n} p_i \tag{3.4}$$

The values in the confidence table are:

$$g(X|\overline{Y_1}, ..., \overline{Y_k}) = v.\prod_{i=1}^{k}(1 - pi)$$

where p_i represent the weight of Y_i in the argument. We consider in the following discussion that having a value of 0, for any confidence is not considered, has such an element (no confidence at all), will be removed from a safety argument. Figure 3.9 presents the result for 2 parents, B, and C, and one child, A. In (a) and (b) we illustrate that when q decreases (q=1, q=0.5) then the influence of $g(C)$ decreases. On the figure, the lines for different values of $g(C)$ are close depending only on $g(B)$ (with a value of 0.5, not presented here due to limitation space). We also illustrate in (b), that when p and q are less than 1, we obtain a residual confidence when $g(B) = 0$ and $g(C) > 0$. This is actually an expected result, because, when the weights are less than one, this means that the argument is not a perfect AND gate. In (c), p is low (0.1), which is interpreted as a low influence of $g(B)$, and characterized by the fact that all lines are nearly horizontals (i.e. with no influence of $g(B)$). A complete analysis of limits, which is not presented here, has demonstrated that the variations of $g(A)$ are compliant with a complementary argument Do Hoang (2015).

3.4.5 Mixed arguments

The previous arguments could be used also to integrate the confidence in the GSN "Context" element. Indeed, a context is actually a complementary element for the considered

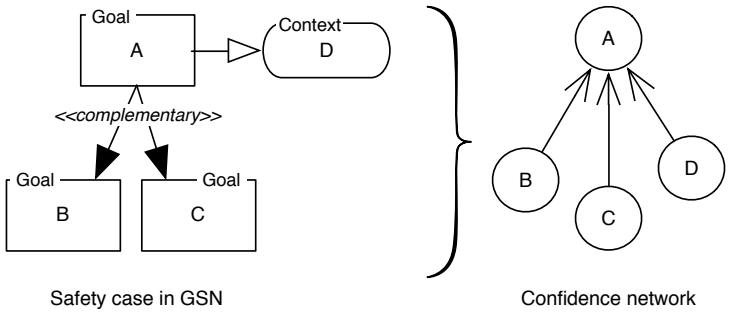


Figure 3.10: Mixed argumentation 1

argument. Figure 3.10 presents a complementary argument, where a context has also been defined. In this case, the resulting network, is a node A, with three parents (B, C, D), and a noisy-and table for node A. When an alternative argument is used between B and C, then, an intermediate node I_BC is included, with an alternative table for B and C. The confidence table in A is a noisy-and between D and I_BC.

3.4.6 Sensitivity analysis

We propose to perform a sensitivity analysis using a tornado graph. It is a simple statistical tool, which shows the positive or negative influence of basic elements on main function. Basically, considering a function $f(x_1, ...x_n)$, where values $X_1, ..., X_n$ of the variables x_i have been estimated, the tornado analysis consists in the estimation for each $x_i \in [X_{min}, X_{max}]$, of the values $f(X_1, ..., X_{i-1}, X_{min}, X_{i+1}, ...X_n)$ and $f(X_1, ..., X_{i-1}, X_{max}, X_{i+1}, ...X_n)$, where X_{min} and X_{max} the maximum and minimum admissible values of variables x_i. Hence for each x_i, we get an interval of possible variations of function f. The tornado graph is a visual presentation with ordered intervals. In our case, we estimate the intervals of $g()$ with $X_{min} = 0$ and $X_{max} = 1$.

If we take the example of alternative argument, with arbitrary values $q = 0.7$ and $p = 0.9$, we get the following table:

g(B)	0		1	
g(C)	0	1	0	1
g(A)	0	0.7	0.9	0.97

If we choose the values of $g(B) = 0.8$ and $g(C) = 0.7$, the confidence table leads to the value $g(A) = 0.8572$, also computed with the tool AgenaRisk[1], presented Figure 3.11. In this example, to determine the sensitivity to $g(B)$, we keep all the values for p, q, and

[1]http://www.agenarisk.com

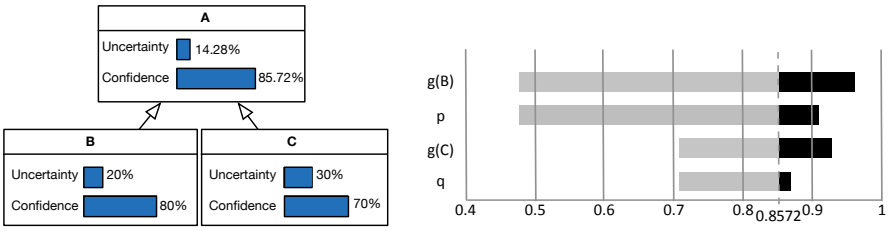


Figure 3.11: (a) Example of an alternative argument with the tool Agenarisk and (b) Corresponding Tornado graph

$g(C)$, and only calculate the values $g(A)$ for $g(B) = 0$ and $g(B) = 1$ (we obtain the values 0.49 and 0.949).

The same approach is used for other variables p, q, and $g(C)$. The result is presented Figure 3.11 (b). In this tornado graph, g(B) appears to be the most influent parameter to decrease or increase the confidence in A. The left part is between 0.49 and 0.872, which means that if g(B) is equal to its lower limit, then the confidence in A could be reduced to 0.49. On the opposite, with a maximum value of g(B), then confidence in A could reach 0.949.

Such an analysis leads to identify some sensitive points in a confidence network. This could be used to increase the confidence focusing first on the most positive sensitive points, or to focus on the elements where confidence should never be decreased (to consolidate the safety case confidence). Nevertheless, two main limits appear: it is not possible to identify combination of confidence variations, and such a diagram does not identify which variables are the easiest to increase. For instance, even if $g(C)$ appears to be less influent, it may be easier to increase its confidence than the one in $g(B)$. Our approach does not focus on those aspects, but they are important points to study.

3.5 Conclusion

This chapter presented a new approach for the definition and estimation of confidence in a safety case. We argue that it is important to have a separation between the safety case and the confidence case. Our aim is to analyze uncertainties that may be present in a safety case, using a sensitivity analysis. Our approach is based on the Dempster-Shafer theory for the definitions of confidence and uncertainty. But the constraint $m(X, \overline{X}) = 0$, brings the main benefit of letting use mathematical tools, such as BBN. Hence, we proposed for most common safety case models in GSN, some transformation rules into a confidence network. We particularly introduce the use of noisy-or for alternative arguments, and an adapted version of noisy-and for complementary arguments. An experiment on a real case study of a rehabilitation robot has been carried out by Do Hoang (2015). A confidence

graph of 65 nodes has been identified with only two alternative arguments (all the others were complementary). The complete analysis is still under development but, we were able to compute the complete graph and get a tornado graph in few minutes with the AgenaRisk tool with consistent results. In this chapter, we only focus on the feasibility of a quantitative estimation of confidence, and its propagation in a confidence network. But this is obviously completely dependent on the determination of the confidence values themselves. As already mentioned, this important issue is part of our future work.

4

Fault tolerant planning

The work presented in this chapter has been initiated when I was recruited at LAAS after my PhD. I co-advise with Marc-Olivier Killijian and David Powell from the dependability group, and Felix Ingrand from the robotic group of LAAS, the PhD of Lussier (2007). This collaboration leads to publications in international conferences (Lussier et al., 2007b,c) and one workshop (Lussier et al., 2007a).

Despite successes in autonomous navigation, exemplified by Mars rovers and the clearing of the *DARPA Grand Challenge* (Monterlo et al., 2006) and *DARPA Urban Challenge* (Urmson et al., 2008), and more recently by the Google and Delphi experiment cars, fully autonomous systems are not yet accepted for real-life applications. Such systems should be able to choose and execute high-level actions without any human supervision, in practice using planners as a central decisional mechanism. However, one of the major stumbling blocks is the difficulty of verifying and validating the behavior of such decisional software in an open, unstructured and dynamic environment. One way to increase the confidence that can be placed in planners despite imperfect verification and validation is to consider a tolerance approach with regard to residual development faults (such as design faults and programming faults). We investigate such an approach in this chapter, focussing on faults in the planner's declarative planning models. To the best of our knowledge, very little work has been published on such an approach whereas we have shown in Lussier et al. (2007b,c) that such mechanisms can really improve the level of confidence that can be placed in an autonomous system. Possible reasons for this limited use may result from the following points:

- fault tolerant mechanisms must be implemented through redundancy in a context with limited resources (space, power, etc.)
- fault tolerance mechanisms proposed in control engineering are efficient for dealing with sensor or actuator faults, but they generally do not consider faults related to decisional levels.

The proposed approach in this chapter is based on the well known fault-tolerant software method called Recovery Block Randell (1975). It implements error detection and system recovery to avoid system failures. The Recovery Block pattern, as presented in Armoush (2010), includes N *diverse*, independent, and functionally equivalent software modules called "versions" from the same initial specification. These diverse blocks are usually obtained using different programming languages, compilers, etc. and developed by different teams. These blocks are classified into primary and $N-1$ secondary versions. The primary version (Block) is executed first and submitted to an error detection test. If an error is detected, a secondary alternate version (Recovery Block) is executed and tested. This last step is repeated until either one alternate passes is tested error-free, or all alternates are exhausted and an overall system failure is reported.

The main contribution of this chapter is a fault tolerant architecture targeting software faults in planners, extendable to most decisional mechanisms, and to validate it through fault injection experiments.

This chapter is structured as follows. Section 4.1 introduces basic concepts of dependability, robust planning, and current means for increase confidence in planers. In Section 4.2, we present a framework for developing planners that are able to tolerate development faults in their application-dependent knowledge, and an implementation example on an existing robot architecture. An experimental framework is proposed in Section 4.3 for evaluating the efficacy of the proposed fault-tolerance mechanisms. Section 4.4 presents our evaluation results and discusses the relevance of planning model diversification in our application. Finally, Section 4.5 concludes and suggests future research directions.

4.1 Background in robust planning

On-line real-time planning is essential for any system that claims to be autonomous and able to fulfill its goals in an unpredictable open environment. Planning is the activity of producing a plan to reach a goal from a given state (e.g., the mission goals for the upcoming day of an exploration rover), using a given *planning model*. Planning can be implemented in several ways but, in practice, two approaches predominate:

- *Search in a state space* manipulates a graph of actions and states. It explores different action sequences from an initial state to choose the most suitable one to achieve given goals.

- *Search in plan space* manipulates a graph of incomplete plans. It starts with an empty plan containing the initial state and the final state (the planner's objectives), then considers ways to refine it by adding possible and useful actions until the search comes up with a complete plan that satisfies the planner goals. Unlike the search in a state space, actions are not added sequentially to the plan: the first action added to the empty plan may be the second to be executed. CSP (Constraint Satisfaction Problem) solving is an iterative algorithm commonly used in this approach, assigning successively possible values to each of the system variables and verifying that constraints between the variables remain satisfied.

In both cases, the planner typically consists of two parts: (a) a declarative *planning model* describing the system, the objects it can interact with, the system's possible actions and the associated constraints, and (b) a planning search engine that can reason on the planning model and produce a plan of actions enabling goals to be reached. A planner typically need two inputs: the current state of the system (position, sensors and actuators status, etc.) and its mission objectives (*goals*). The planning model is specific to the application, while the planning engine may be independent from the application. However planning model and search engine are often tightly linked by heuristics, that are included within the model to guide the search of the engine. Moreover, the planning model must be written in a way exploitable by the planning engine, and is usually not easily translated to another engine, nor easily understood by human developers or testers.

The robustness of a planner, that is its ability to achieve the system's assigned goals despite adverse situations (lighting conditions, unexpected obstacles, etc.), may be attained through either implicit or explicit handling of adverse situations (Lussier et al., 2005).

Robustness through implicit handling of adverse situations is typically achieved by commitment strategy: planners seek to produce flexible plans that contain as much latitude and adaptability as possible Ghallab and Laruelle (1994); Muscettola et al. (2002). The plan produced is in fact a family of plans consistent with the constraints of the system. Inflexible plans, where all actions parameters are defined in advance, including the start and end dates of each action Chien et al. (2005), need to rely heavily on adaptation capabilities at lower system layers to tolerate small discrepancies in plan execution.

Robustness through explicit handling of adverse situations, i.e., when actions of the current plan fail, may be achieved through two strategies:

- *Re-planning*, which consists in developing a new plan from the current system state and still unresolved goals. Depending on the planning model complexity, replanning may be significantly time costly. Other system activities are thus generally halted during replanning.

- *Plan repair*, which attempts to reduce the time lost in re-planning by salvaging parts of the previous failed plan, and executing them while the rest of the plan is being

repaired. However, if reducing the salvaged plan conflicts with unresolved goals, plan repair is stopped and re-planning is initiated.

Note however that such approaches do not cover residual faults in the planning engine nor in the planning models.

4.2 An approach of fault tolerant planning: FTPlan

This section presents our approach using diversity to tolerate development faults in planning models and heuristics. We first introduce the general principles of our approach, before giving an implementation example.

As previously stated, we focus on development faults in planners, such as design faults (an incorrect model, improperly used heuristics, etc.) or programming faults (programming mistakes in the inference mechanism, faulty variable values, etc.).

4.2.1 General principles

The proposed fault tolerant planner architecture is based on the Recovery Block pattern. It uses only two diverse blocks, with an additional component, called FTPlan, in charge of detecting errors and performing the recovery. We first introduce this FTPlan component, then detail its error detection mechanisms. Finally, we propose a sequential policy for system recovery. A concurrent one has also been studied and is presented in Lussier et al. (2015).

FTplan component

From a dependability point of view, the fault tolerance mechanisms have to be as independent as possible from the planners. This is why we propose to handle both the detection and recovery mechanisms, and the services necessary for their implementation, in a middleware level component called FTplan, standing for *Fault-Tolerant PLANner coordinator*. This component has to integrate the fault tolerance mechanisms into the autonomous system architecture. This implies essentially communication, synchronization and coordination between the error detection mechanisms and the redundant planners.

FTplan is intended to allow tolerance of development faults in planners (and particularly in planning models). FTplan itself is *not* fault-tolerant, but being much simpler than the planners it coordinates, classic verification and testing (such as formal method or exhaustive testing) can be applied to check that it is fault-free.

Error detection

Implementing error detection for decisional mechanisms in general, and planners in particular, is complex Lussier et al. (2005). There are often many different valid plans, which can

be quite dissimilar. Therefore, error detection by comparison of redundantly-produced plans is not a viable option. Thus, we must implement error detection by independent means. Here, we propose four complementary error detection mechanisms:

1. A *watchdog timer* can be used to detect when the search process is too slow or when a critical failure such as a deadlock occurs. Timing errors detected in this way can be due to faults in the planning model, in its search engine, or ineffectiveness of the search heuristics.

2. A *plan analyzer* (i.e., an on-line plan oracle) can apply an acceptance test to the produced plan to check that it satisfies a number of constraints and properties. This set of constraints and properties can be obtained from the system specification and from domain expertise but it must be independent with respect to the planning model. A plan analyzer is able to detect errors due to faults in the planning model or heuristics, and in the planner itself.

3. A *plan failure detector* is a classic mechanism used in robotics for execution control. Failure of an action which is part of the plan, may be due to an unresolvable adverse environmental situation, or may indicate errors in the plan due to faults in the knowledge or in the planning engine. Usually, when such an action failure is raised, the planning engine tries to repair the plan. When this is not possible, it raises a plan failure. We can use these plan failure reports for detection purposes.

4. An *on-line goal checker* verifies whether goals are reached while the plan is executed. A plan can be declared as (partially) failed if every action of the plan has been carried out but not all goals have been achieved. The on-line goal checker can resubmit unfulfilled goals to the planner at the start of the next replanning.

Note that both watchdog timer and plan analyzer detect errors during planning and thus before plan execution, while the plan failure detector and the on-line goal checker monitor the plan execution itself.

System recovery

With the sequential mechanism, the planners are executed sequentially, one after another. The principle is given in Fig. 4.1. Basically, each time an error is detected, we switch to another planner until all goals have been reached or until all planners fail one after another when starting from the same initial system state. In the latter case, no models allow the planner to tackle the planning problem successfully: an exception must be raised to inform the operator of mission failure and to allow the system to be put into a safe state (line 29). When all planners have been used but some goals are still unsatisfied, we revert to the initial set of planners (while block: line 4 to 32). This algorithm illustrates the use of the four detection mechanisms presented in Section 4.2.1: watchdog timer (lines 9 and 25),

plan analyzer (line 14), plan failure detector (line 16 and 18), on-line goal checker (lines 4, 6 and 17).

Until all goals have been achieved, the proposed algorithm reuses planners that have previously been detected as failed (line 5). This makes sense for two different reasons: (a) a perfectly correct plan can fail during execution due to an adverse environmental situation, and (b) some planners, even faulty, can still be efficient for some settings since the situation that activated the fault may have disappeared.

It is worth noting that the choice of the planners, and the order in which they are used, is arbitrary in this particular example (line 7): we only chose to exclude the last planner that led to an execution failure. However, the choice of the planner could take advantage of application-specific knowledge about the most appropriate planner for the current situation or knowledge about recently observed failure rates of the planners.

4.2.2 Implementation on a real robot

We present in this section an implementation of the previously proposed sequential planning policy in the LAAS hierarchical software architecture for autonomous systems.

LAAS architecture

The LAAS architecture Alami et al. (1998); Lemai and Ingrand (2004) has been successfully applied to several mobile robots, some of which have performed missions in real situations (human interaction or exploration). It is composed of three main layers as presented in Fig. 4.2.

The functional layer is composed of a set of automatically generated *GenoM modules*, each of them offering a set of services, which perform computation (e.g., trajectory movement calculation) or communication with physical devices (sensors and actuators). A service request gives rise to the execution of an elementary action, the success or failure of which is reported to the requester, along with other action-specific information. Data exchange between modules is performed through the use of "posters", each of which is a shared memory space attached to a module, and readable by the others.

The procedural executive *OpenPRS* (*Open Procedural Reasoning System*), is in charge of decomposing and refining plan actions into lower-level actions executable by functional components, and coordinating their execution. This component links the decisional component (IxTeT) and the functional layer. During execution, OpenPRS reports any action failures to the planner, in order to re-plan or repair the plan. As several IxTeT actions can be performed concurrently, it has also to schedule sequences of refined actions.

IxTeT (*IndeXed TimE Table*) Ghallab and Laruelle (1994) is a temporal constraint planner, combining high-level actions to build plans. It uses CSP to search in plan space, as presented in Section 4.1. Its deliberations are based on piecewise constant functions called *attributes* that represent the evolution of the system state, of its resources, and of

```
1. begin mission
2.   exec_failure ← NULL;
3.   failed_planners ← ∅;
4.   while (attainable_goals ≠ ∅)
5.     candidates ←  planners;
6.     while (candidates ≠ ∅  & attainable_goals ≠ ∅)
7.       choose k such as (k ∈ candidates)
                  & (k ∉ failed_planners)
                  & ( (k ≠ exec_failure)
                  | (k ∪ failed_planners = candidates));
8.       candidates ← candidates \ k;
9.       init_watchdog(max_duration);
10.      send(plan_request) to k;
11.      wait % for either of these two events
12.        □ receive (plan_found) from k
13.          stop_watchdog();
14.          if analyze(plan)=OK then
15.             failed_planners ← ∅;
16.             res_exec ← k.execute_plan();
17.             update(attainable_goals);
18.             if res_exec ≠ OK then
19.                exec_failure ← k;
20.             end if
                % if the plan fails, then
                % attainable_goals != empty and the
                % online goal checker will loop line 3 or 5
21.          else
22.             log(k.invalid_plan);
23.             failed_planners ←  failed_planners ∪ k;
24.           end if
25.         □ watchdog timeout
26.           failed_planners ←  failed_planners ∪ k;
27.      end wait
28.      if failed_planners = planners then
29.        raise exception 'no valid plan found in time';
           % no remaining planner,
           % the mission has failed
30.      end if
31.    end while
32.  end while
33. end mission
```

Figure 4.1: Sequential Planning Policy

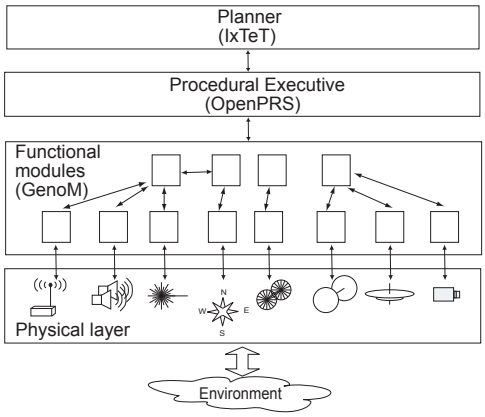


Figure 4.2: The LAAS architecture

its environment. The different actions are described in a *planning model* file as a set of constraints either on the system attributes (e.g., robot position, energy consumption, and environment evolution) or on temporal and numerical variables (e.g., action duration). A valid plan is a partially-ordered set of possibly concurrent, non-conflicting actions that together achieve the system goals.

Constraints (C) are defined using:

- classical mathematical operators for temporal and numerical variables (V),
- the *consume*, *produce* and *use* predicates for consumption, production or usage of a system resource,
- the *hold* (H) and *event* (E) predicates for the other system attributes. The hold predicate represents persistence of an attribute value over a given period of time (e.g., *hold(robot, position, (start,end))*), whereas the event predicate represents an instantaneous change of value (e.g., *event(photo, (to_ do, done), time)*).

Actions are modeled through IxTeT *tasks*. Fig. 4.3 gives the example of a high-level action that can be used to photograph a scientific object in an exploratory mission. Line 1 declares the task and its numerical and temporal parameters: `x` and `y` are the Cartesian coordinates of the scientific object to be photographed, while `t_start` and `t_end` are temporal constraints representing respectively the starting and ending time of the task. Lines 3 to 5 define the constraints on the system attributes required for all the task duration: lines 3 and 4 stipulate that the robot position must not change while taking a picture, whereas line 5 requires that the camera points down to the object that need to be photographed. Line 6 marks the photo as successfully taken when the task terminates. Line 7 presents an example of resource management: the resource CAMERA is used for the duration of the task. Line 8 presents an example of constraints on temporal values

```
1.  task TAKE_PHOTO(?x, ?y)(t _start, t_end) {
2.    ?x in [-oo,+oo]; ?y in [-oo,+oo];
3.    hold(POS_X():?x, (t_start, t_end));
4.    hold(POS_Y():?y, (t_start, t_end));
5.    hold(POS _CAMERA():down,(t_start, t_end));
6.    event(IMAGE(?x, ?y):(to_do,done),t_end);
7.    use(CAMERA():1, (t_start, t_end));
8.    (t_start - t_end) in [10,60];
9.  }nonPreemptive
```

Figure 4.3: An Action in the IxTeT Formalism

by specifying the possible duration of the task (from ten to sixty seconds). Finally, line 9 closes the task definition and states that executions of the task cannot be preempted.

Fault tolerant planner implementation

The fault tolerance principles presented in Section 4.2.1 have been implemented in a fault tolerant planner component as presented in Fig. 4.4. This component replaces the original component "Planner" presented in Fig. 4.2. The FTplan component is in charge of communicating with OpenPRS as the original planner does. To be consistent with the current implementation, FTplan uses the same technologies as OpenPRS and IxTeT for communication.

The current version of FTplan implements the sequential redundant planner coordination algorithm presented earlier (Section 4.2.1, Fig. 4.1) with two IxTeT planners. Currently, the plan analysis function is empty (it always returns *true*) so error detection relies solely on just three of the mechanisms introduced earlier: watchdog timer, plan failure detection, and on-line goal checker.

Fig. 4.5 presents an example of a fault tolerance scenario using the sequential policy: a first plan is produced and executed using the planner, but an action failure is detected during execution. To simplify the diagram, the plan failure detector service is represented with the message "executionFailure". The first planner is then re-initialized while the second one is asked for a new plan from the current situation. However this planning lasts too long (a model fault may have caused the planner to freeze) so the watchdog times out, FTplan reinitializes the second planner before switching back to the first planner and asking for a new plan. The plan is then produced and successfully executed.

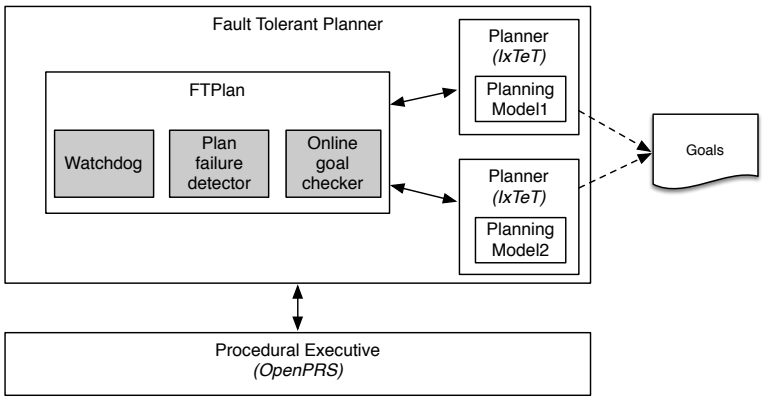


Figure 4.4: Fault tolerant planner

Model diversification

In our implementation, two different planning models are used with the IxTeT planning engine. The first model (which we will call $Model_1$) was originally developed to validate implementation of the LAAS architecture (and particularly the IxTeT component) on a real robot. It is the result of iterative efforts from a different team of researchers. It contains actions needed for a space exploration rover, such as moving to a designated position, photographing objects, and communicating with an orbiter.

A second variant (called $Model_2$) for the same target application has been developed by a different team. Diversification with respect to $Model_1$ has been also forced through specific design choices. For example, robot position is defined using Cartesian coordinates in $Model_1$ whereas $Model_2$ uses a symbolic representation, thus implementing fundamentally different algorithms to those of the Cartesian one. Overall, numerous modifications were carried out, such as pruning redundant system attributes and constraints, or "merging" complementary attributes (e.g., a system position attribute can be combined with a moving/still boolean attribute to give a single attribute that gives the system position when it is motionless, or else the value `MOVING`).

4.3 Framework for Validation

Our validation framework relies on fault injection at the decisional layer of a full stack of robot controller software, and simulation of robot hardware. Only the robot hardware is simulated, all the software components otherwise execute and interact in real time, in the same way as on a real robot. Although the considered robot controller software stack has been extensively used in demonstrations of a real robot, we preferred to resort to simulation because the behavior of a real robot may become hazardous when we inject faults and it

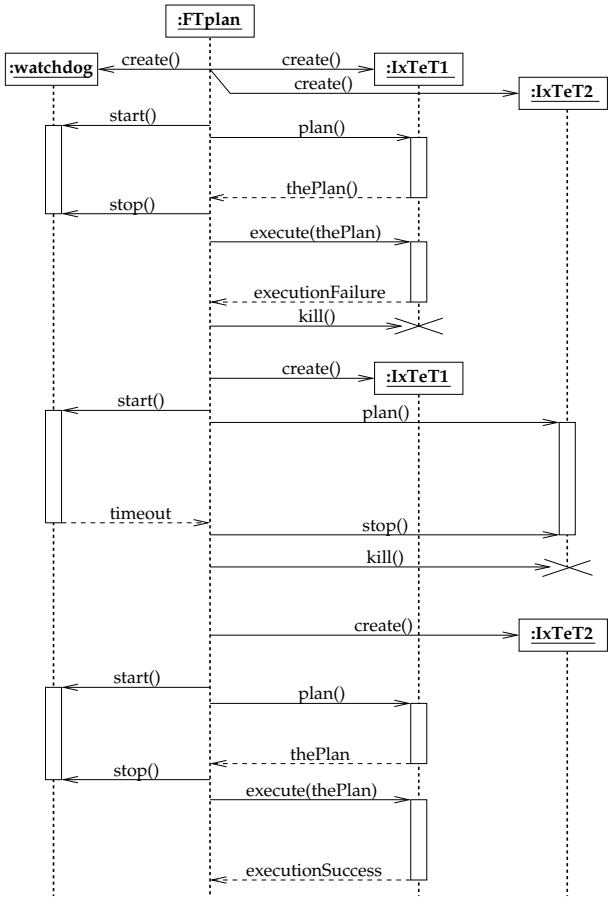


Figure 4.5: A fault tolerance scenario with the sequential planning policy

could cause damage to itself or its direct surroundings. A second reason is that numerous repetitive experiments on real robots would be both expensive and hard to automate.

Fault injection is used since it is the *only* way to test the proposed fault tolerance mechanisms with respect to their specific inputs, i.e., faults in planning knowledge. In the absence of any data regarding real faults in declarative models, there is no other practical choice than to rely on *mutations*[1], which have been found to efficiently simulate real faults in imperative languages Daran and Thévenod-Fosse (1996).

We present in this section the framework that has been used to validate the proposed fault tolerant mechanisms: its software architecture, the workload and faultload generated as experiment inputs, the data recorded for each basic experiment and, based on that data, the measurements that represent the output of each set of experiments.

4.3.1 Testing software architecture

The whole simulation environment is represented in Fig. 4.6. It incorporates three elements: an open source robot simulator named Gazebo, an interface library named Pocosim, and the components of the LAAS architecture already presented in Section 4.2.2.

The *robot simulator Gazebo*[2] is used to simulate the physical world and the actions of a mobile robot in that world. It generates realistic sensor feedback and physically plausible interactions between objects through a simulation of rigid-body physics in three dimensions.

The *Pocosim library* Joyeux et al. (2005) is a software bridge between the simulated robot executed on Gazebo and the software commands generated by the GenoM modules.

Our target autonomous system is an existing ATRV (All Terrain Robotic Vehicle) robot commercialized by iRobot, and employs GenoM software modules interfaced with the Gazebo simulated hardware (see Figure 4.7). The upper layer of the LAAS architecture executes as presented in the previous section. The functional layer consists of eight GenoM modules that can be categorized into three groups:

- The SICK and RFLEX modules both control hardware components through the Pocosim library: SICK controls a laser sensor whereas RFLEX controls wheel motions and an odometer.
- NDD, ASPECT and POM are software modules that use SICK and RFLEX to implement navigation and obstacle avoidance. POM establishes position data of the robot according to the odometer and other possible localization mechanisms. ASPECT uses this position and feedback from SICK to create a map of the robot's immediate surroundings, which is used by NDD to generate navigation commands using a nearness diagram algorithm.

[1] A mutation is a syntactic modification of an existing program.
[2] *The player/stage project*, http://playerstage.sourceforge.net

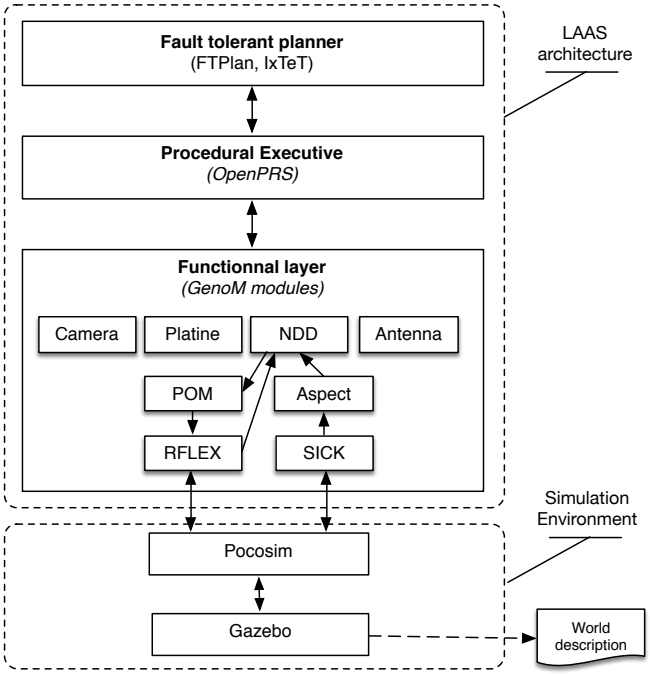


Figure 4.6: Simulation environment

Figure 4.7: LAAS-CNRS Dala robot with an ATRV base

- PLATINE, ANTENNA and CAMERA control hardware components that are not simulated by Gazebo but by software simulation in the GenoM modules themselves (respectively, a camera orientation device, a communication antenna, and two cameras).

4.3.2 Workload

Autonomous systems move in unpredictable, open and unknown environments. They do not know a priori the obstacles (static or dynamic) that they will encounter, the terrain configuration, the available roads, the presence of external perturbations (weather, lack of brightness, etc.).

Our workload mimics the possible activity of a space rover. The system is required to achieve three subsets of goals during a mission: *take science photos* at specific locations (in any order), *communicate* with an orbiter during specified visibility windows, and *be back at the initial position* at the end of the mission.

To partially address the fact that the robot must operate in an open unknown environment, we need to confront the system with several different missions, in several different "worlds". Each *mission* encompasses the number and location of photos to be taken, and the number and occurrence times of orbiter visibility windows. Each *world* is a set of static obstacles unknown to the robot. These unknown obstacles stress the robot's navigation and obstacle avoidance mechanism (see Section 4.3.1). At the plan execution level, the unknown obstacles create uncertainty as regards the outcome of action executions, and can possibly prevent the robot from achieving some of its goals.

We implemented four missions and four worlds, thus applying sixteen execution contexts to each fault situation. The missions and worlds are defined with respect to 12m x 12m environment. The initial position of the robot is set equal to the center of this square, at coordinates (0,0). Missions are referenced as gradually more difficult M1 to M4 (Fig. 4.8): M1 consists in three photos in nearby locations and two communication goals (shown as shaded intervals on the mission time axis), whereas M4 consists in four communications goals and five far apart photo locations. The maximum duration of a mission is 800 seconds, during which it is physically possible to achieve all the objectives. Worlds are referenced as W1 to W4 (Fig. 4.9). W1 is an empty world with no obstacles to hinder plan execution, while W2 and W3 contain small cylindrical obstacles that are avoidable by our robot navigation and obstacle avoidance mechanism. However, W4 includes larger rectangular obstacles that may be impossible for the navigation module to circumnavigate, and thus susceptible to irremediably block the robot path.

The experiments are inherently non-deterministic, due to asynchrony of the various robot subsystems and in the underlying operating systems. Task scheduling differences between similar experiments may degrade into task failures and possibly unsatisfied goals, even in the absence of faults. To address this non-determinacy, we execute each basic experiment three times, leading to a total of 48 experiments per fault scenario (3 executions

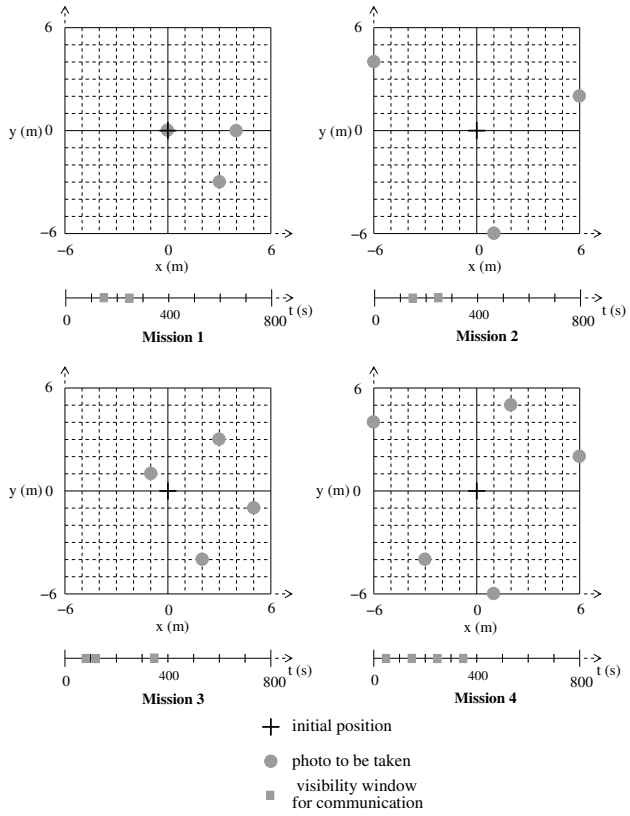


Figure 4.8: The Set of Missions: M1 to M4

$*$ 4 missions $*$ 4 worlds). Ideally More repetition would be needed for statistical inference on the basic experiments, but this would have led to a total number of experiments higher than that which could have been carried out with our available resources (including initialization and data processing, each basic experiment lasts about 20 minutes).

4.3.3 Faultload

To assess performance and efficacy of the proposed fault tolerance mechanisms, we inject faults in a planning model by random mutation of the model source code (i.e., in $Model_1$ of Fig. 4.4).

From a syntactical analysis of the IxTeT formalism, five types of possible mutations were identified:

i) Substitution of numerical values: each numerical value is exchanged with members of a set of real numbers that encompasses (a) all numerical variables in all the tasks of

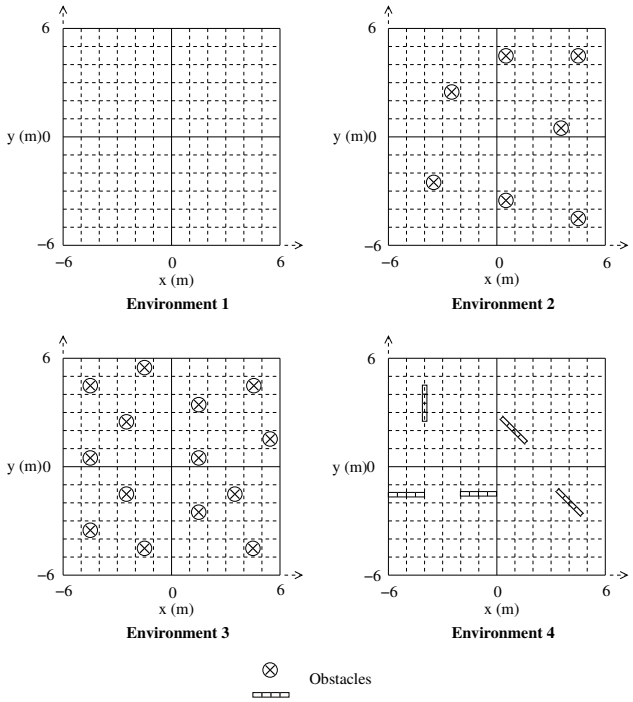


Figure 4.9: The Set of Worlds: W1 to W4

the model, (b) a set of specific values (such as 0, 1 or -1), and (c) a set of randomly-selected values.

ii) Substitution of variables: since the scope of a variable is limited to the task where it is defined, numerical (resp. temporal) variables are exchanged successively with all numerical (resp. temporal) variables of the same task.

iii) Substitution of attribute values: in the IxTeT formalism, attributes are the different variables that together describe the system state. Attribute values in the model are exchanged with other possible values in the range of the attribute.

iv) Substitution of language operators: in addition to classic numerical operators on temporal and numerical values, the IxTeT formalism employs specific operators, such as "nonPreemptive" (that indicates that a task cannot be interrupted by the executive).

v) Removal of a constraint relation: a randomly selected constraint on attributes or variables is removed from the model.

Substitution mutations were automatically generated using the SESAME tool Crouzet et al. (2006).

All in all, more than 1000 mutants were thus automatically generated from the 300 lines planning model. To improve representativeness of injected faults, we also chose to discard mutants where no plan is found in any mission (we consider that models that systematically fail would easily be detected during the development phase).

4.3.4 Recorded data and measurements

Numerous log files are generated by a single experiment: simulated data from Gazebo (including robot position and hardware module activity); output messages from GenoM modules, the OpenPRS procedural executive and FTplan; requests and reports sent and received by each planner, as well as outputs of the planning process. For each basic experiment, these uncompressed text files require from 4 to 16 Mb; the 48 experiments characterizing one mutant require nearly 320 Mb.

Condensing this amount of data into significant relevant measures is problematic. Moreover, contrary to more classic mutation experiments, the result of an experiment cannot be easily dichotomized as either failed or successful. As previously mentioned, an autonomous system is confronted with partially unknown environments and situations, and some of its goals may be difficult or even impossible to achieve in some contexts. Thus, assessment of the results of a mission must be graded into more than just two levels. Additionally, detection of equivalent mutants is becoming more complex due to the non-deterministic execution context of autonomous systems.

To address these issues, we chose to categorize the quality of the result of an experiment according to: (a) *mission dependability*, defined in terms of the goals that have been successfully achieved (or alternatively, through the inverse notion of mission *un*-dependability), (b) *mission performance* indicators such as the mission execution time and the distance covered by the robot to achieve its goals, and (c) internal measures of *planning behavior*.

Considering the sets of missions $\mathcal{M}$ given to the system, worlds $\mathcal{W}$ in which it evolved and faults $\mathcal{F}$ injected into the system, mission *un*-dependability for an elementary experiment is given by:

- $\overline{\varphi}_p(\mathcal{M}, \mathcal{W}, \mathcal{F})$, the average proportion of unachieved photo goals,
- $\overline{\varphi}_c(\mathcal{M}, \mathcal{W}, \mathcal{F})$, the average proportion of unachieved communication goals,
- $\overline{\varphi}_r(\mathcal{M}, \mathcal{W}, \mathcal{F})$, the average proportion of unachieved returns to the initial position,
- $\overline{\mu}(\mathcal{M}, \mathcal{W}, \mathcal{F})$, the average proportion of failed missions, where a mission is pessimistically defined as failed if any goal (photo, communication, return) is not achieved.

For example, $\overline{\varphi}_p(M3, W4, 39)$ represents the mean proportion of failed photos for the mission $M3$, in the world $W4$, with the injected fault 39, averaged over the several elementary experiments (currently three) carried out with each injected fault. We define $\mathcal{M}^* = \{M1, M2, M3, M4\}$ (and, similarly, $\mathcal{W}^* = \{W1, W2, W3, W4\}$), such that $\overline{\varphi}_p(\mathcal{M}^*, W4, \emptyset)$ represents the mean proportion of failed photos for all four missions in world $W4$, with no injected faults; it characterizes twelve elementary experiments.

We define two measures to characterize the performance of the rover during its mission:

- $\overline{D}(\mathcal{M}, \mathcal{W}, \mathcal{F})$, the average distance (in meters) covered by the rover,
- $\overline{T}(\mathcal{M}, \mathcal{W}, \mathcal{F})$, the average duration (in seconds) of rover activity, i.e., the time at which it performed its last action.

Finally, to characterize the internal behavior of plan execution, we define:

- $\overline{R}(\mathcal{M}, \mathcal{W}, \mathcal{F})$, the average number of replannings during an experiment.

4.4 Results

Experiments were executed on i386 Linux-based systems with a 3.2 GHz CPU and the Linux OS. We first study the performance cost of the proposed mechanisms, then give three examples of fault injection results, and finally present global results of the fault injection campaign.

We particularly focus on three systems: a non-redundant robot using $Model_1$ (referred to as $Robot_1$), another non-redundant robot using $Model_2$ (referred to as $Robot_2$) and a diversely redundant robot using FTplan with $Model_1$ and $Model_2$ (referred to as $Robot_{1/2}$).

Two further systems are considered when analyzing the performance impact of our mechanisms. Both implement a version of IxTeT with no plan repair capability, thus leading to increased numbers of replannings and model switches: $Robot_1^*$ uses $Model_1$, and $Robot_{1/1}^*$ uses FTplan with the same $Model_1$ for both planners.

Detailed results are given in Lussier et al. (2015). We only present here the main conclusions. We first analyse fault free (no faults are injected) performance to determine the overhead of the proposed sequential mechanism. We also consider the worst case, i.e. when the replanting functionality is deactivated (it then lead to more failed missions). This analysis show that the overhead for the execution time is 15% when the controller includes FTPlan. We deem these results to be quite acceptable considering the negative impact of discarding plan repair.

We analyses then the fault tolerance efficacy, with records as the one in Figure 4.10. We do not provide here the complete analysis and explanations, but we draw the two main conclusions:

- The redundant diversified models of the fault-tolerant $Robot_{1/2}$ provide a notable improvement to dependability in the presence of faults: in all cases, the proportions of failed goals decrease compared to the non-redundant $Robot_1$.
- Even when considering the pessimistic measure of the proportion of failed missions (recall that a mission is considered as failed even if only a single elementary goal is not achieved), the improvement procured by redundant diversified models is appreciable: 41% in worlds W1-W3, 29% when world W4 is also considered.

Note, however, that in the presence of injected faults, the fault-tolerant $Robot_{1/2}$ is *less* successful than a single fault-free model. This apparent decrease in dependability is explained by the fact that incorrect plans are only detected when their execution has failed, possibly rendering one or more goals unachievable, despite recovery. This underlines the importance of plan analysis procedures to attempt to detect errors in plans *before* they are executed.

4.5 Conclusion and perspectives

Lack of dependability remains a severe impediment to the take-up and practical utilization of autonomous systems. At the software level, the dependability of decisional mechanisms such as planners, which are essential for truly autonomous operation, is particularly challenging. The difficulty of validating such autonomy software makes it very difficult to provide safety and reliability guarantees for autonomous systems.

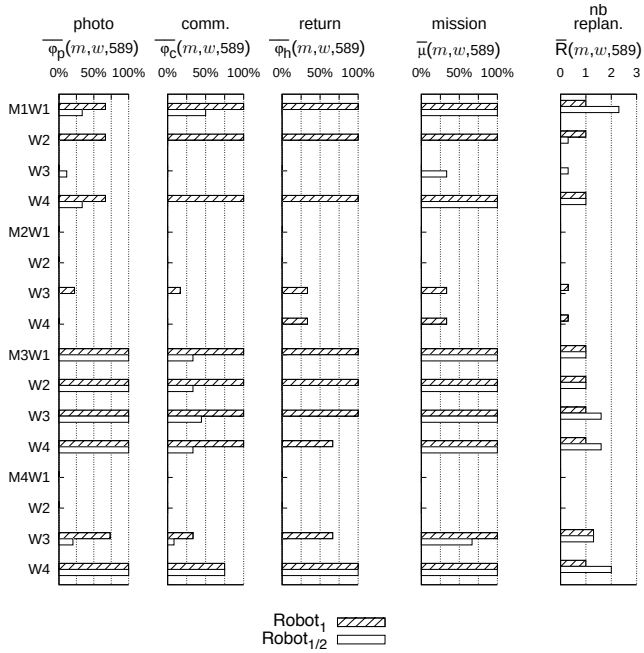


Figure 4.10: Results for mutation 589

In this chapter, we focused on the reliability aspect of the problem, proposing an innovative fault tolerance approach for temporal planners. The proposed approach aims to complement verification and testing by providing tolerance to residual design faults in coded domain-specific knowledge. To this end, we advocate the use of diversified planning models and search heuristics. We proposed a component providing error detection and recovery appropriate for fault-tolerant planning, and implemented it in the LAAS architecture. Our current implementation is that of sequential planning associated with the first three error detection mechanisms. To assess the performance overhead and the efficacy of the proposed mechanisms, we developed a validation framework that exercises the software on a simulated robot platform, and carried out what we believe to be the first ever mutation experiments on declarative models. These experiments were conclusive in showing that the proposed mechanisms do not severely degrade the system performance in the chosen scenarios, yet usefully improve the system behavior in the presence of model faults.

There are many directions for future research. First, implementation of a plan analyzer should allow much better goal success levels to be achieved in the presence of faults since it should increase error detection coverage and provide lower latency. Implementation of the concurrent planning policy and comparison with the sequential planning policy are also of interest. It would also be worthwhile to assess the impact of diversification in

planning heuristics rather than just models, and study the possible benefits of using more than two diversified versions. An interesting point would also be to keep traces of success scores of the planners to define a preferred order for planners (e.g., invoke less successful planners when the “best" planners have failed). In addition, the statistical relevance of the results would benefit from many more experiments. The use of a large computer grid would drastically improve the number of experiments that could be executed in reasonable time and eliminate the need for manual inspection to remove trivial mutants.

5

Active safety monitoring

This chapter presents the last results of a continuous and intensive work in my research activities. I started this work when I joined LAAS (Guiochet and Powell, 2006; Guiochet et al., 2008a) and initially collaborated with David Powell and then with Hélène Waeselynck, to supervise two PhD (Mathilde Machin in 2015 and Mekki-Mokhtar (2012)), and with Matthieu Roy to supervise the postdoc of Fanny Dufossé (2013-2014). This work led to several publications in international conferences (Mekki-Mokhtar et al., 2012; Machin et al., 2014a, 2015) and a submission to a journal. This work has been applied in several case studies in the context of a direct collaboration with Airbus-ASTRIUM and in two European projects (PHRIENDS (2006-2009) and SAPHARI (2011-2015)). A new PhD just started in 2015 (Lola Masson), to extend the method to more complex autonomous architectures.

To deal with residual faults and adverse situations in autonomous systems, fault tolerant mechanisms are one main alternative to increase safety. “Safety monitors" are a type of such mechanisms, and are in charge of assuring that the system, despite faults and adverse situations, stays in safe states. They are able to observe the system and its environment, and to react using a safety margin to keep the system in a safe state. Specification and design of such devices is usually done in an ad hoc manner, with very simple safety rules implemented (e.g., in case of contact with an obstacle, an bumper directly disconnect the power of the actuator of a mobile robot). We argue that in the future, versatile autonomous systems, will have to deal with complex safety rules, that might be activated or deactivated according to the tasks, and with the possibility of reacting with many different ways that might be non consistent. The work presented in this chapter

presents a framework to specify such rules, starting from a hazard analysis, and using formal verification techniques to synthesized them. This framework also integrates the issue of assuring safety while preserving functionality of the considered system.

Even if as presented in Chapter 1, many works deal with architecture and verification mechanisms of safety monitors, very few research has been carried out on the generation of safety rules with an explicit integration of the permissiveness issue. We propose a complete method and a tool available online (SMOF, 2015), which has been used in an industrial use case.

The reminder of the chapter is structured as follows. Section 5.1 introduces baseline and concepts around safety monitoring, and provides an overview of the proposed framework. Section 5.2 presents how the system and properties and modeled using formal logic. In Section 5.3, the safety rule synthesis algorithm using the NuSMV model-checker is explained. Section 5.4 deals with the last step of the process, which is the verification of the consistency between the synthesized safety rules. Section 5.5 presents an application to a real industrial case study, and the experiment results of a fault injection campaign. We conclude in Section 5.6, outlining the benefits and limitations of SMOF, and citing future directions.

5.1 Baseline and concepts

5.1.1 Concepts

Taking inspiration from the diverse monitor defined by IEC61508 (2010), we define a *safety monitor* as a device responsible for safety, in opposition to the main control channel which is responsible for all other functional and non-functional requirements of the system. The monitor is equipped with means for context observation (i.e., sensors) and able to trigger safety interventions. The safety monitor is required to protect against all faults that adversely affect safety, including interaction faults. In particular, the monitor must accommodate any variation of the environment and any dysfunction of the control channel, including arbitrary behavior of the latter, e.g., when it is faulty. To this end, the whole safety channel is assumed fault-free. For example we consider that the sensors available to the monitor are perfect, without uncertainty. In practice, of course, these assumptions must be enforced by appropriate use of self-checking, verification, redundancy and some degree of independence between the safety channel and the main channel.

Assuming such a perfect monitor, a system developer faces the problem of specifying the behavior of the monitor, i.e., which intervention should be applied, and when. Prerequisite elements to address this issue are the following :

A **safety invariant (SI)** is a necessary and sufficient condition to avoid a hazardous situation, resulting from a hazard analysis. If a safety invariant is violated, we assume that damage is immediate and irreversible, with no possible recovery. We refer to any state violating the safety invariant as a *catastrophic state*.

Example: "the robot speed shall not exceed 3 m/s*" (where* 3 m/s *is the speed beyond which harm is considered to be inevitable).*

A **safety intervention** is an ability of the monitor to constraint the system behavior in order to prevent the system from violating a safety invariant. An intervention is only effective when its *preconditions* are satisfied. We distinguish two types of interventions: inhibitions and actions.

A **safety inhibition** prevents a change in system state. When triggered, an inhibition is assumed to be immediately effective.

Example : "lock the wheels", with "robot stationary" as precondition.

A **safety action** triggers a change in system state.

Example : "apply emergency brake". The expected effect "the speed will not exceed 5 m/s*" occurs only if the intervention is applied while the precondition "the speed is less than* 2.5 m/s*" is true.*

Safety. The monitor is said safe if it ensures the safety invariant in the monitored system, i.e., it guarantees the non-reachability of the catastrophic states.

Permissiveness. The monitor is said permissive if the monitored system is able to freely move about in its state space.

For example, a monitor allowing the system to operate at high speed, manipulating a sharp object in human presence would be very permissive. As autonomous systems are particularly versatile, they are supposed to operate in many different states. To achieve the assigned tasks, the system need to reach a wide range of states. As soon as the monitor triggers intervention, it damages permissiveness.

Now, by nature, safety is ensured by reducing the possible behavior of the system. Safety and permissiveness are antagonistic. We take this antagonism into account by designing the monitor to be *maximally permissive with respect to safety*, i.e., to restrict functionality only to the extent necessary to ensure safety.

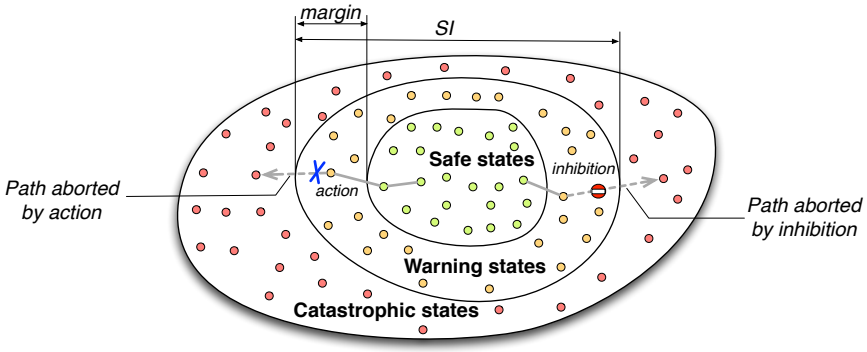


Figure 5.1: Partition of system states in catastrophic, warning and safe states

As illustrated in Figure 5.1, a safety invariant defines a partition into catastrophic and non-catastrophic states of the monitored system, and thus enable to assess whether the monitor ensures the safety of the system.

As recovery is not possible, interventions have to be applied before the catastrophe, i.e., in non-catastrophic states with some margin from the catastrophic border. Now, interventions add constraints to the system behavior, reducing the permissiveness. So the set of non-catastrophic states is partitioned into warning states, where interventions are applied, and safe states, in which the system operates without constraint. The warning states are defined such that every path from a safe state to a catastrophic state passes through a warning state. Applying interventions only in warning states is a first choice in the strategy production to ensure permissiveness.

In this paper, we address the issue of producing the high-level requirements of the monitor that ensure safety invariants and are maximally permissive with respect to safety. A **safety rule** defines a way of behaving in some potentially hazardous states. It is composed of a condition and an intervention, to apply when the condition is true. The condition is chosen such that it becomes true before the safety invariant is violated.

Example: "if the robot speed is greater than 2 m/s *then apply emergency brake."*

A safety strategy is a set of safety rules intended to ensure one safety invariant.

Example: "If the robot speed is greater than 2 m/s *then apply emergency brake. And, if the slope is greater than 10% then apply emergency brake."*

5.1.2 Process overview

Figure 5.2 presents the overall process to use the SMOF framework. The process starts with a HAZOP-UML hazard analysis, which outputs safety invariants expressed in natural language. We consider as a running example a mobile robot with a manipulator arm and the informal safety invariant: *The arm must not be extended beyond the platform when the platform velocity is greater than* V_0.

The safety invariant is then expressed formally with predicates on variables that are observable by the monitor. We focus for now only on predicates involving a variable compared to a fixed threshold. This type of safety threshold is amenable to formal verification and is used in many real systems. Considering the two monitor observations: the absolute speed v, and a Boolean observation of the arm position a ($true$ when the arm is above the base, $false$, when the arm is extended), the example safety invariant is formalized as $SI1 = (v < V_0) \vee (a = true)$. It is equivalent to define the catastrophic state as $v > V_0 \wedge a = false$.

To synthesize strategies, this invariant is modeled in a SMOF template, as detailed in Section 5.2. In order to keep models simple enough to be validated, each safety invariant is modeled separately. In this step, we determine the partition of non-catastrophic states into safe states and warning states by splitting variable value intervals or sets. This is done one variable after another. For example, the speed interval $[0, V_0[$ from the safety

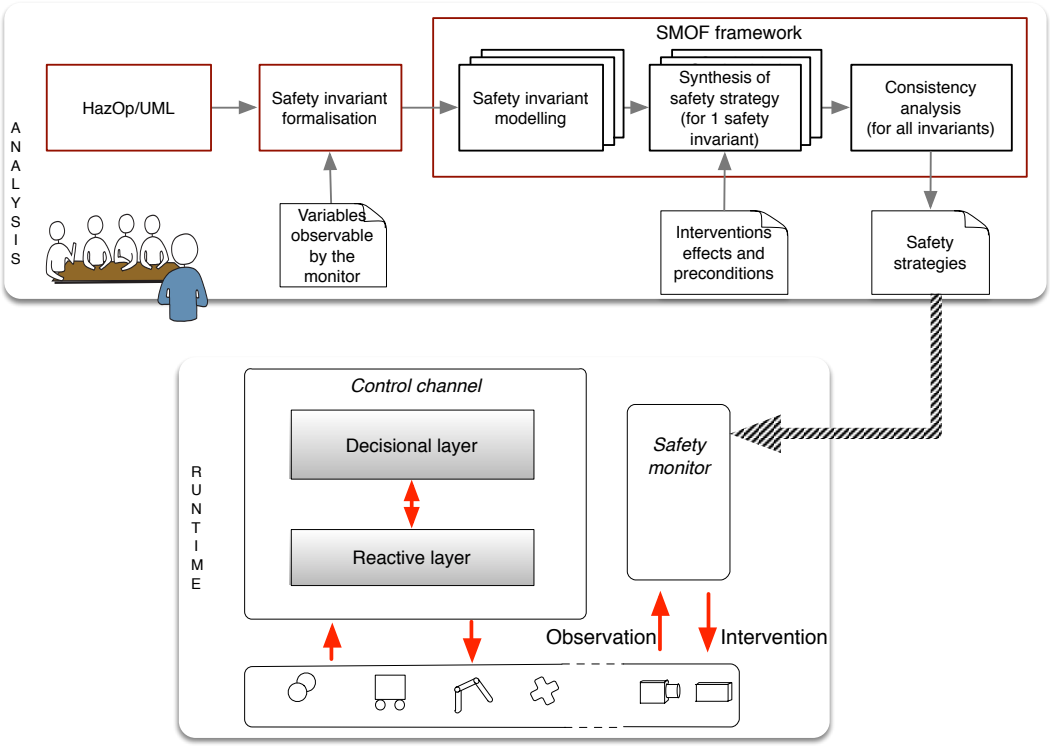


Figure 5.2: Overview of the process

invariant $SI1$ is partitionable according to a margin m in two intervals $[0, V_0 - m[$ and $[V_0 - m, V_0[$. In the case of arm position, the observation is Boolean. The singleton value set $\{true\}$ cannot be partitioned, hence no margin exists. The resulting states are shown in Figure 5.3 where only two possible interventions have been considered: the monitor is able to engage brakes (action) and to prevent the arm from extending (inhibition).

Then, the strategy synthesis can be done, as detailed in Section 5.3. Figure 5.4 illustrates a satisfying strategy, which applies braking in $s1$ and arm inhibition in $s2$ and $s3$. Additionally to the transitions leading directly to the catastrophic state, several other transitions are deleted, but the system can still carry out tasks. This strategy is safe and permissive with respect to safety. When the model does not admit satisfying strategies, the user has several choices: 1) add new interventions; 2) try to reduce permissiveness, for example accept that this warning state is no longer reachable and recompute the set of warning states; 3) modify the safety invariant.

As safety invariants are processed separately, the final step is to check the consistency between the strategies that ensure different safety invariants. This is addressed in Section 5.4.

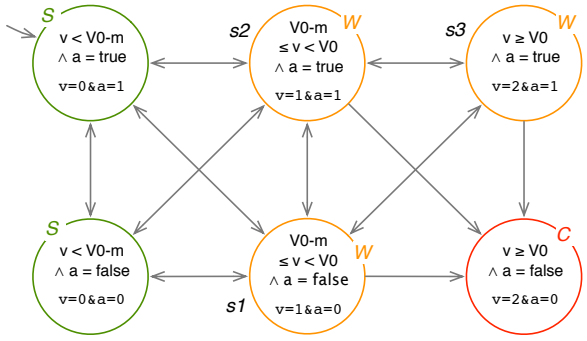


Figure 5.3: The example behavior from the partitions $\{true, false\}$ for arm position (encoded by `a` $\in \{$`0`,`1`$\}$ in the template), and $\{[0, V_0 - m[, [V_0 - m, V_0[, [V_0, V_{max}[\}$ for velocity (encoded by `v` $\in \{$`0`,`1`,`2`$\}$). The state that violates the safety invariant is labeled C for catastrophic. The warning states (W) are those that lead the system to the catastrophe in one step.

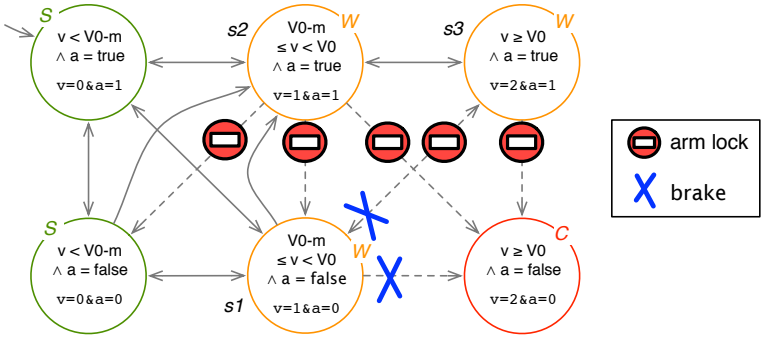


Figure 5.4: The example behavior, modified by a safety strategy

The final safety strategies are then implemented in a real time safety monitor, for on-line verification, as presented in the application Section 5.5.

5.2 Modeling with a SMOF template

A SMOF model is a model of the part of the system related to one safety invariant, seen from the monitor point of view. A SMOF model is based on one formalized safety invariant.

The SMOF model gathers all information necessary to produce strategies that ensure the safety invariant:

- Behavior: automaton of the system in absence of the monitor, which in particular contains all paths reaching a catastrophic state.

- Interventions: Abilities of the monitor to constrain the system behavior.
- Properties: Desired properties of the monitor action (e.g., safety and permissiveness).

Each item is modeled using the variables that are observable by the monitor, relevant to the (formalized) safety invariant and discretized. If the variables can be constrained by interventions, they are also controllable.

The SMOF model is seen from an abstract point of view in Fig 5.5. A strategy is composed of safety rules; each safety rule associates an intervention (or a combination of interventions) to a warning state, contained in the behavior. Safety and permissiveness are satisfied by the pair of the behavior and the strategy. Validity is a side-property linked to the intervention preconditions (see Section 5.2.2). During the synthesis process, the rules, and so the strategy, are modified whereas the rest of the SMOF model remains unchanged.

To formally described our model, we choose to use languages and tools available in the formal verification community, in particular model checking. To express the properties, propositional logic is too limited, whereas temporal logic, like CTL (Computational Tree Logic), is particularly well adapted to express our permissiveness properties (presented section 5.2.3). Our framework is based on the model-checker NuSMV2 from Cimatti et al. (2002). In the following, code and output of NuSMV are given in typewriter font.

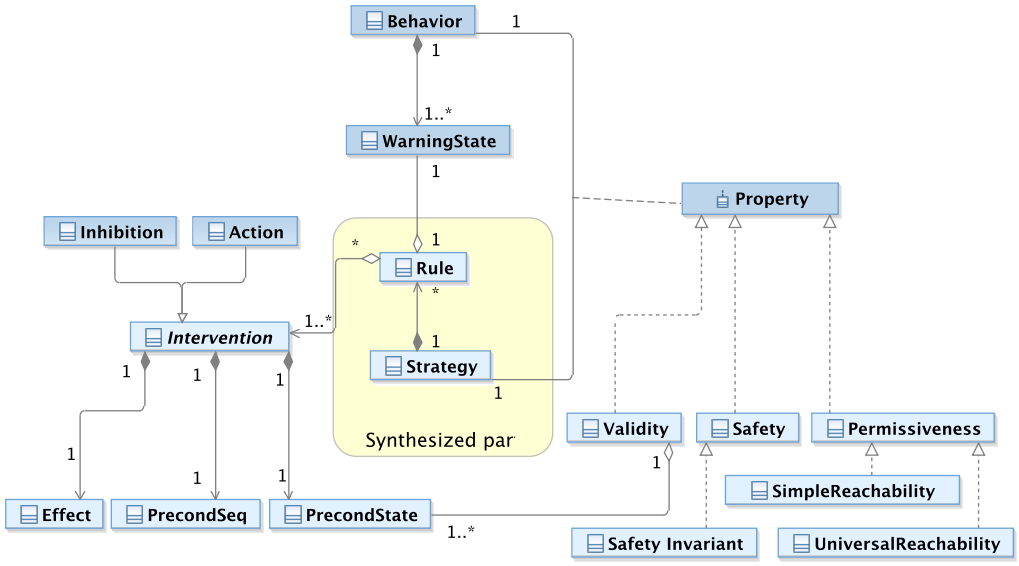


Figure 5.5: Meta-model of the SMOF modeling template

5.2.1 Behavior

The safety invariant, formalized with observable variables, partitions the variable ranges in two. Another partition is done to take a margin (formal conditions for the existence of a margin are studied in Mekki-Mokhtar et al. (2012)). To ease the modeling in NuSMV, the parts are numbered with integers, as illustrated in Figure 5.3. The behavior states result from the Cartesian product of the range partitions. NuSMV is well-suited to SMOF modeling as it transparently builds the Cartesian product of the ranges of all variables. When no constraint is declared, all the combinations of variable values (i.e., states) are possible and all transitions between each pair of states are implicitly declared. Declaring variables in NuSMV amounts to generate a complete graph. Constraints are then added to delete undesired states and transitions of the graph. Transitions are only identifiable by their starting and arriving states, using the NuSMV operator `next()`.

The most common constraint is the continuity of variable, e.g., the velocity cannot "jump" from `0` (standing for $[0, V_0 - m]$) to `2` ($[V_0, V_{max}]$). The module `Continuity` of the template declares a variable `x` with the constraint `next(x) = x | x+1 | x-1`, i.e., the next value of `x` can stay in the same interval or move to an adjacent interval, but it cannot jump from one interval to another that has no common boundary. The parameters of `Continuity()` are the maximal value (the minimal value is `0`) and the initial value.

5.2.2 Interventions

An intervention is modeled by its effect and its preconditions. The effect is a constraint that cuts some transitions from a state, to reduce the set of possible next states.

The effect is guaranteed only if the preconditions are satisfied. We distinguish two preconditions. State preconditions, further denoted by $Precond_{State}$, model the states in which the intervention can be applied. For example, it is not desirable to physically lock the wheels of a moving vehicle. The strategy is said valid if it applies always the intervention properly with respect with $Precond_{State}$ (*Validity* property). Intervention efficiency may also depends on the system history. We choose to take into account only the previous state, and model it by sequential preconditions denoted by $Precond_{Seq}$. They specify constraints on the originating state of the transition where the intervention is applied. An example with two observation variables (a and v) is given Figure 5.6 (in this example a is not a boolean, and can have 3 discrete values).

We propose to express these conditions in a SMV module, defined for the specification of the intervention. An example of its use is given below:

```
-- myInterv : Interv(precondState, precondSeq, flag_myInterv, effect)
brake1 : Intervention(TRUE, v=0 & next(v)=1, flag_brake1, next(v)!=v+1);
```

In this example, `brake1` has no $Precond_{State}$ (TRUE value is given). The $Precond_{Seq}$ parameter specifies that the effect is only guaranteed if the braking is triggered on a transition from the state `v=0` to state `v=1` (using the operator `next`). the `flag_myInterv`

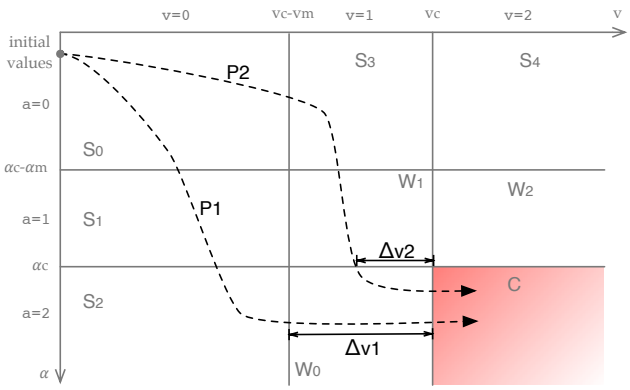


Figure 5.6: Example of sequential precondition. We consider the intervention "braking". The expected effect is "the velocity `v` cannot be greater than V_c". Let assume a rule that applies braking in W_0. If the system is on the path P_1, the braking is triggered when $v \in]v_c - v_m[$. Whereas on the path P_2, the braking is triggered at $v_c - v_\epsilon$. Let now consider the delay between the brake command and the brake effectiveness. This delay, taken into account in the choice of v_m (safety margin), avoid the system to reach v_c. On the contrary, in P_2, the system may reach $v = v_c$ because the required delay for brake effectiveness is not respected. Hence, the intervention is efficient when coming from S_2 but not when coming from W_1. As a consequence, a sequential precondition is added: "braking can be triggered only on a transition originating in a state satisfying $v < v_c - v_m$".

variable specify the safety rule condition. The considered effect is that next value of `v` cannot increase.

5.2.3 Properties

Safety and permissiveness properties are modeled in CTL (*Computation Tree Logic*), which is entirely supported by NuSMV without any syntax change. Time along paths is modeled by three operators: X for a property to hold in the next state, G to hold on the entire path, F to hold eventually. The branching aspect is modeled by A, all the branches, and E, there exists a branch. A CTL operator is composed of one branching operator and one time operator. It is applied on states, or more generally on statements on the system state.

To model safety, we use the atomic property *cata* to denote the catastrophic states. *cata* is the negation of the safety invariant, e.g., `cata := speed=2 & arm_pos=0`. Safety is modeled as the unreachability of the catastrophic states, i.e., in CTL, $AG\neg cata$.

Permissiveness is modeled by two reachability properties applied to any non-catastrophic state s_{nc}:

- Simple reachability $EF\ s_{nc}$
  The state is reachable from the initial state.

- Universal reachability $AG\ EF\ s_{nc}$
  The state is reachable from any reachable state.

For safety, we pessimistically consider that several independent variables may change their values simultaneously in the behavior. We call such simultaneous modifications *diagonal transitions* by reference to the two variable case (see Figure 5.3, transition from initial state to $s1$). Relying on those possible but unlikely transitions to ensure permissiveness is not desirable. A more complete definition of reachability properties that ignore diagonal transitions during permissiveness checking is provided in Machin et al. (2014b) and used by the tools we developed.

Permissiveness is encoded in the template through a module containing the simple and universal reachability for one state. The reachability instanciation of each non-catastrophic state is automatically generated. As a consequence, permissiveness is defaultly checked through universal reachability for every non-catastrophic state. For models where no safe and permissive strategies can be defined, it is also possible to reduce permissiveness to the simple reachability. Of course, this induce some modification to the resulting behavior of the robot and it should be validated by the robotic experts.

5.2.4 Strategies

A strategy is a function associating interventions to warning states. A strategy *applies* the interventions (e.g., control brake engagement and disengagement) according to the current and past states of the system as follows:

- No intervention is applied in the initial state.
- Within a path $P = (s_1, ..., s_k, s_{k+1}, ...)$, the intervention i is associated to s_{k+1} is applied if:
  - s_{k+1} satisfies the $Precond_{State}$ of i, and,
  - i was already applied in s_k, or the transition (s_k, s_{k+1}) satisfies the $Precond_{Seq}$ of i.

5.3 The strategy synthesis

From the theoretical SMOF model detailed previously, we have developed the associated algorithm and tools, based on the SMV model-checker, to assist the SMOF framework.

5.3.1 Synthesis overview

An overview of the synthesis inputs and outputs is given Figure 5.7. Behavior, properties and interventions are the inputs, and a strategy is then produced as output of the

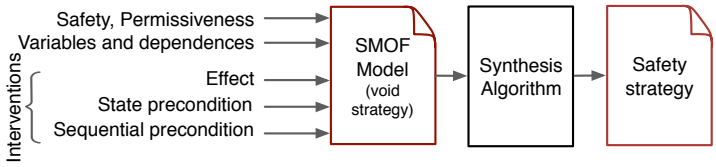


Figure 5.7: Production of safety strategies, for one safety invariant

algorithm. If we consider a system with n warning states, and m possible interventions by the monitor, the number of possible strategies is then 2^{mn} (all combinations of the interventions in the warning states). Among these strategies, the objective is to determine strategies *satisfying* properties. A basic synthesis algorithm could enumerate every possible strategy to check whether it satisfies the requested properties. But complete enumeration is not desirable due to the exponential number of strategies. Hence we build a strategy tree, enabling us to prune branches during the exploration (branch-and-bound algorithm). A formal description fo strategy sets and pruning criteria are presented hereafter.

5.3.2 Strategy set

Let I be the set of interventions, of size m, and $I_C = 2^I$ the set of *intervention combinations*. In the running example, $I = \{a, b\}$ and $I_C = \{a, b, ab, \emptyset\}$ (abbreviating the set notation). Let $S_w = \{s_1, \ldots, s_n\}$ be the set of warning states of the behavior, of size n. A strategy N is a function that associates to a warning state an intervention combination.

$$N : S_w \mapsto I_C$$

For a given SMOF model, if we consider n warning states, a strategy will be noted:

$$N = \{(s_1, i_1), \ldots, (s_j, i_j), \ldots, (s_n, i_n)\}$$

The potential results of the synthesis are the safe and permissive strategies. To restrict the set of solutions, we search for *satisfying* strategies, i.e., strategies that are valid, safe and permissive. Furthermore, we focus on *minimal* satisfying strategies, i.e., strategies for which each intervention is necessary to satisfy the requested properties. A satisfying strategy $N = \{(s_1, i_1), \ldots, (s_n, i_n)\}$ is minimal if there does not exist a different satisfying strategy $N' = \{(s_1, i'_1), \ldots, (s_n, i'_n)\}$ such that $\forall k \in [1, n], i'_k \subseteq i_k$. For instance, let $N_1 = \{(s_1, ab), (s_2, b)\}$ and $N_2 = \{(s_1, a), (s_2, \emptyset)\}$ be satisfying strategies, N_2 is minimal, as $a \subset ab$, and $\emptyset \subset b$.

To define the required set structure as a tree, we introduce the *undefined intervention*, denoted by $\perp$. To check the properties, it is interpreted in the model as no intervention ($\varnothing$ specify that the defined intervention is the void intervention). As illustrated in Figure 5.8, the tree root is the fully undefined strategy $N_0 = \{(s_1, \perp), \ldots, (s_n, \perp)\}$. Given a

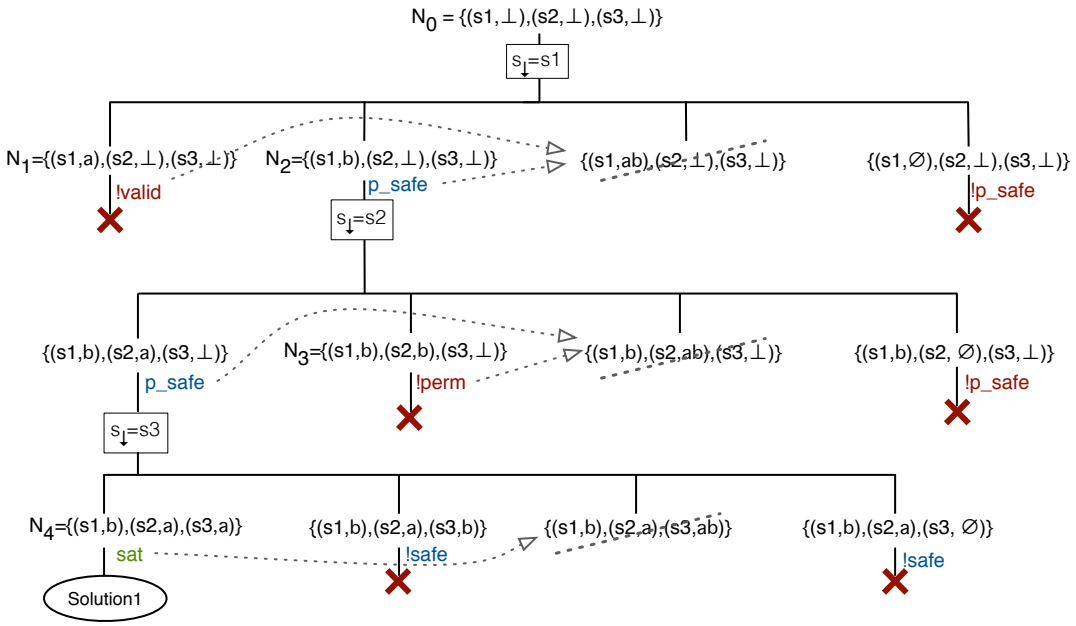


Figure 5.8: Search tree for our example. The choice of states $s_\downarrow$, which define the tree structure, are mentioned in frames. The nodes that are crossed out or unmentionned are not generated and checked. The dotted lines corresponds to the pruning of combined siblings. The results of the tree traversal is the only node labelled by `sat`.

partially defined strategy N, building its children requires to choose a state $s_\downarrow$ where the intervention to apply is not yet defined, i.e., $N(s_\downarrow) = \bot$. If such a state does not exist, N is a fully defined strategy and a tree leaf. The 2^m children of N, denoted N_i, are the nodes such as $\forall s \neq s_\downarrow,\ N_i(s) = N(s)$ and $N_i(s_\downarrow) = i$, with $i \in I_C$.

We define another relation between tree nodes. Let N and N' be children of N_p by the state $s_\downarrow$. N' is a *combined sibling* of N if $N'(s_\downarrow) \subset N(s_\downarrow)$. For example $\{(s_1, a), (s_2, ab), (s_3, \bot)\}$ is a combined sibling of $\{(s_1, a), (s_2, a), (s_3, \bot)\}$.

5.3.3 Pruning criteria

Table 5.1 gives an overview of the pruning criteria. The first pruning criterion applies to a strategy N that is `!valid`. For instance, in Figure 5.8, strategy $N_1 = \{(s_1, a), (s_2, \bot), (s_3, \bot)\}$ is `!valid` because the intervention that locks the arm folded is applied in a warning state where the arm is unfolded. Child strategies of N define interventions in other warning states, but this will not fix the problem in the first one. Either the first warning state becomes unreachable, and so the child strategy is not minimal, or the state is reachable and the child strategy remains invalid. So, the children of an invalid strategy are either

invalid or not minimal. Similarly, a combined sibling of N_1 define other interventions in the same warning state, and the first intervention still makes the strategy invalid. Recursively none of the children of the combined siblings are solutions to the problem, we can prune the subtrees of the combined siblings.

Consider now a partial strategy that is not permissive, e.g., N_3. All its children are `!perm` as well, because adding interventions can only cut transitions. In the same way, its combined siblings are `!perm`. This second criterion prunes descendants and combined siblings of non-permissive strategies.

The third criterion, `sat`, which includes safe, valide and permissive criteria, discards strategies that are not minimal. Assume N_4 is a satisfying strategy. Its descendants and combined siblings might be `sat` as well, but they involve additional interventions and are thus not minimal. This corresponding subtrees can be pruned. N is appended to the list of solutions returned by the search.

The fourth and fifth criteria are evaluated using a *subgraph* of the behavior, where the warning states with an undefined ($\perp$) intervention are removed. This subgraph focuses on reaching the catastrophic state via the warning states for which a decision has been taken, and specifically via the state s_p targeted by the most recent decision. If the strategy is `safe` in this subgraph we say that it is partially safe (`p_safe`). The evaluation of `p_safe` for N_2 is illustrated in Figure 5.9. The resulting subgraph is composed of all safe states and the warning state s_1 (s_2 and s_3 are removed because no intervention are associated). There is a path to the catastrophic state in this subgraph. The descendants of N_2 would add interventions that can only delete transitions exiting warning states that are outside the current subgraph. So, all descendants of N_2 are unsafe and can be pruned (fourth pruning criterion).

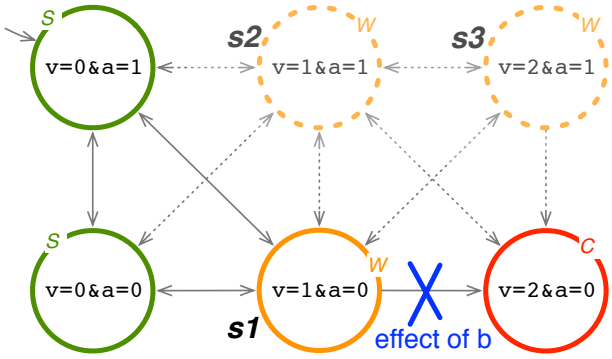


Figure 5.9: A view to check whether $(s_1, b), (s_2, \perp), (s_3, \perp)$ is `p_safe`. The states and transitions in plain line constitute the considered subgraph.

Additionally to the pruning criteria, an important parameter of the traversal is the choice of the warning state $s_\downarrow$ to deploy the children of a strategy N. Let remind the only constraint on this choice is $N(s_\downarrow) = \bot$, i.e., the interventions applied in $s_\downarrow$ are not yet determined. Assume N is `p_safe`. There is no path to the catastrophic state in the subgraph, either because the rules of N cut hazardous transitions, or removing states induced absence of transition to catastrophic states. The `p_safe` property is true but this does not provide valuable information for further strategy synthesis.

To avoid this situation, we add a constraint on the $s_\downarrow$ choice. It is now selected such that N' is `!p_safe`, with N' defined by:

$$N'(s_\downarrow) = \emptyset \; and \; \forall s \neq s_\downarrow \quad N'(s) = N(s)$$

Thus, `p_safe` is not trivially true, and the fourth criterion is more efficient.

The fifth pruning criterion exploits the, untrivial, piece of information `p_safe=TRUE`. When a strategy is `p_safe`, it means that the considered subgraph is `safe`. Intuitively, no more intervention are required in the subgraph. It is then useless to consider the combined siblings of such a strategy, and they are pruned by the fifth criterion. The children, corresponding to add interventions outside of the subgraph are explored. This reasoning is underlain by the assumption that the intervention application in one state does not depend on interventions applied in other states, and in particular on the states that are not part of the subgraph. As soon as the $Precond_{Seq}$ are used, this assumption is not verified. Hence, the fifth criterion may prune satisfying strategies. Conversely, this criterion guarantees that the returned solutions are minimal. An unminimal satisfying strategy is, either a descendant of a minimal satisfying strategy (and it is pruned by the third criteria), or a combined sibling of a minimal satisfying strategy, pruned by the fifth criterion.

	Node property	Pruned relative nodes
1	`!valid`	Descendants and combined siblings
2	`!perm`	Descendants and combined siblings
3	`sat`	Descendants and combined siblings
4	`!p_safe`	Descendants
5	`p_safe`	Combined siblings

Table 5.1: Pruning criteria

5.3.4 Synthesis tool

After the presentation of the algorithm, we present some features of the implementation. The tool has several variants of tree traversal. It has been implemented in a parallel manner to improve performance.

Variants of implementation We have developed two variants of the tree traversal. Of course, all the variants return only satisfying strategies.

Variant 1 Only the first four pruning criteria are used, then only unsatisfying or unminimal strategies are discarded. The returned set contains all the minimal strategies, and some unminimal strategies. No returned solution allows to conclude that no solution exists for the model, and that the model must be changed. The traversal is in absence of solution is very fast, notably thanks to the fourth criterion. Conversely, when there are many solutions, this version is very long to execute as it explores all of them.

Variant 2 The five criteria guarantee that the returned strategies are minimal. When a solution exists, this tree traversal is shorter than variant 1.

Implementation Before the tree traversal, an initialization is required and has been automated to:

- determine the warning states
- generate the default permissiveness properties
- generate the validity property
- check for non-safety and permissiveness of the initial model

We have implemented the tree traversal in an parallel manner. The structure of the implementation is the following. A master thread launches slave threads. Each slave has a local version of the SMOF model defined by the user. The master sends nodes to the slaves. The slaves generate the SMOF model corresponding to the node and check properties by calling NuSMV. From the results, pruning criteria are assessed. The nodes are sent back to the master, with the required deployment policy (i.e., deploy children, combined siblings or not). The master generates the corresponding nodes and send them to the slaves.

Performance A performance assessment of the SOMF synthesis tool has been carried out using artificial models. It allows us to consider search spaces ranging for a few thousands strategies to more than 10^{18}. The models are generated as follows. One variable has 3 values, the others have 2 values. The initial state has all variables at value 0, and the only catastrophic state has variables at their maximum values. There are three possible models for interventions. Suffix `_a` corresponds to *action* decreasing a variable value. An action is defined for each variable. Suffix `_i` is the same for *inhibition* : a freezing intervention is defined for each variable. Then, both actions and inhibitions are defined for each variable. For example, `3var_6` has 3 variables and 6 interventions (3 actions and 3 inhibitions).

Performance is assessed running the SMOF synthesis tool on a Intel Core I7-4770 processor running at 3.4GHz with 16 GB of memory. Results are presented in Table 5.2.

For each model, the number of complete strategies (i.e., of tree leaves) is given. It would be the number of steps of brute-force search. The two variants are assessed. To compare our algorithm to brute-force search, the first column gives the percentage of examined nodes (i.e., tree nodes), with respect to the number of complete strategies. These values are very low, which demonstrates the overall efficiency of the defined pruning criteria. The second column gives the number of solutions found. The results of variant 1 demonstrate that models only with *actions* (suffix `a`) have no solution. Intuitively, as the actions force the variables to change, no permissive strategy can be found. The third column gives the execution time. Its increase is kept reasonable, if one considers that there are orders of magnitude in size between the smallest and largest search spaces, and that we do not stop the algorithm after a first solution is found.

As can be seen in Table 5.2, Variant 1 is quite long to execute when the model size increase. Furthermore, it is not realistic to require from the end user to review 17106 strategies returned for `3var_6`, even if assisted by criteria computed on strategies. For the same model, Variant 2 returns around 10 times less solutions, with the guarantee of minimality and a shorter execution time. The only inconvenient of the Variant 2 is that it does not return all the minimal solutions and thus cannot allow to conclude that the model does not admit a solution. The synthesis on `_a` models show that the Variant 1 is very fast in case of non-existence of solutions.

It may be surprising that a model with few variables requires a 4-hours synthesis (for variant 1) or half an hour (with variant 2). One might wonder whether the approach is useful in realistic cases. Firstly, the number of variables is not unrealistic. A safety invariant models only one safety-relevant aspect of a system. In the real system studied by Mekki-Mokhtar et al. (2012), each invariant had no more than two variables. Secondly, the artificial models we used are generic, i.e., they have many interventions and no variable dependencies. It induces that there are numerous solutions to find (except the `_a` models), much more than in real cases. This case is supported by the performance results of the following industrial case study. Synthesis done with Variant 1 spends 0.15s for SI4, 0.32s for SI5 and 0.09s for SI6. Moreover, our tool is thrifty in memory. In all these synthesis, it uses maximum 15 MB.

5.4 Analysis of consistency

Within the strategy production process, each safety invariant is modeled separately by one SMOF model, and is ensured by a separate strategy. As a consequence, two strategies that ensure two different safety invariants may apply interventions at the same time. Now these two interventions can be incompatible (e.g., braking and accelerating).

To check that no incompatible interventions are applied concomitantly, the different SMOF models are turned into SMV modules and gathered in one SMV model. The

Table 5.2: Experimental pruning performance

Model	Number of strategies	**Variant 1** Examined nodes (%)	Solutions	Time	**Variant 2** Examined nodes (%)	Solutions	Time
2var_2a	64	4.7	0	5s	4.7	0	50ms
2var_2i	64	12.5	1	23s	12.5	1	150ms
2var_4	4096	0.56	9	21s	0.44	6	170ms
3var_3a	10^6	10^{-3}	0	20s	10^{-4}	0	200ms
3var_3i	10^6	10^{-2}	40	3min	10^{-3}	12	1.0s
3var_6	10^{12}	10^{-6}	17106	4h	10^{-8}	1128	18.3s
4var_4a	10^{18}	10^{-14}	0	3s	10^{-14}	0	2.7s
4var_4i	10^{18}	-	-	-	10^{-11}	12954	26min

main module of this model has to ensure consistent behaviors in SMOF models, as for interventions and observations. Then, the intervention concomitancy is checked.

Interventions applied for one invariant have consequences on the whole system. In models, as soon as an intervention is applied in one SMOF model, it is globally applied, i.e., in all the SMOF models in which it is modeled. Intervention application is done with the effect and preconditions defined locally in each SMOF model.

Some observations from different SMOF models may be dependent. The dependency has to be modeled in the same way as it is inside of a SMOF model. A particular case of dependency is to use the same observation in two different SMOF models. For example, the velocity may be used for an invariant of velocity limitation and for an invariant that required to observe whether the system is at stop or not. The two invariants share a common observation but they have different partition of that. Constraints are declared to put in consistency the two partitions, as illustrated in Figure 5.10.

The non-concomitancy properties for each pair of intervention (i, j) (two interventions are never simultaneously activated) can be formulated with:

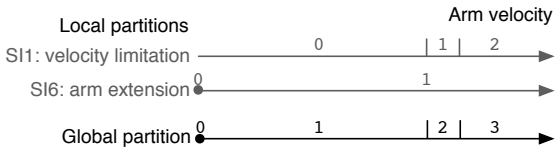
$$AG\neg\ (\ flagInterv_i\ \wedge\ flagInterv_j)$$

where $flagInterv_i$ represents the activation of intervention i.

Permissiveness properties of each SMOF model are re-checked, as the added constraint may reduce the permissiveness. The more the dependencies are modeled, the more accurate are the permissiveness results.

5.5 Industrial case study

To demonstrate our approach, we apply the whole process, from HAZOP to implementation, to a case study provided by Kuka Robotics within the framework of the European project Saphari (*Safe and Autonomous Physical Human-Aware Robot Interaction* SAPHARI (2011-2015)).



(a) Intersection of partitions

```
INVAR arm_vel=0 <-> si6.arm_vel=0 & si10.arm_vel=0;
INVAR arm_vel=1 <-> si6.arm_vel=1 & si10.arm_vel=0;
INVAR arm_vel=2 <-> si6.arm_vel=1 & si10.arm_vel=1;
INVAR arm_vel=3 <-> si6.arm_vel=1 & si10.arm_vel=2;
```

(b) Constraint on variables

Figure 5.10: Mapping of the arm velocity observation for velocity limitation (SI1) and standstill determination (in SI2). The partition from SI1 is divided in three intervals: normal `0`, in the margin `1`, greater than the limit `2`. The partition from SI2 has two values: standstill `0`, movement `1`. To make consistent the velocity values in SI1 and SI2, a global variable is created. Its partition is the intersection of the local partitions, as seen in 5.10a. Constraints are added in the model to map global and local partitions 5.10b.

The system is composed of a mobile platform and an articulated arm with 7 axis, as shown in Figure 5.11. It is an industrial co-worker in a manufacturing setting, intended to share its workspace with human workers. It takes and places boxes, which contain parts, on shelves, tables, or on the top of the robot platform in order to convey them. A restricted area is defined ; it is an area forbidden to the robot.

The system is equipped with a safety layer, a separate and redundant control, satisfying most of our assumptions about the theoretical safety monitor. In particular, the safety monitor specification associates interventions and logic formula of predicates on observation variables. Only thresholds are admitted to define predicates. The safety layer has only two interventions: engaging the arm brake and engaging the platform brake. The monitor can only observe a small subset of the system variables. All observations used in the following are available in the safety layer.

The system has been modeled in UML, resulting in 15 Use Cases with around 50 attributes and 15 sequence diagrams. The HAZOP analysis results in more than hundred HAZOP lines with a non-zero severity. Each line is intended to be modeled as a safety invariant. In practice, the HAZOP experience and the PHA (*Preliminary Hazard Analysis*) enable to fast group similar HAZOP lines. Thirteen safety invariants are formulated and presented in Table 5.3.

We have implemented strategies of SI1, SI2, SI3, SI4 and SI6. As the robot we use was not equipped with a gripper, it was not possible to implement SI5. To test our fault tolerance approach, we carry out fault injection experiments. Fault injection (see Arlat et al. (1989)) consists in adding intentionally faults in the system to test the fault tolerance device, i.e., the safety monitor. The consequences of the faults are assessed during an

Figure 5.11: Case study robot

SI1	The velocity of robot arm must not be greater than V_0.
SI2	The velocity of robot platform must not be greater than V_0.
SI3	The robot must not enter the restricted area.
SI4	The robot arm must not be extended beyond the platform footprint when the platform moves.
SI5	A gripped box must not be tilted more than α_0.
SI6	A collision between a human and the robot arm must not hurt the human.
SI7	The robot platform must not collide a human
SI8	The velocity of any point of the robot must not be greater than V_0.
SI9	The robot arm must not drop a box.
SI10	The robot arm must not clamp human parts.
SI11	The robot gripper must not clamp human parts.
SI12	The robot must not override boxes laid on tables, shelves and robot storage.

Table 5.3: Safety invariants, resulting from HAZOP-UML analysis

activity, or mission, of the system. Experiments outputs are read-outs that are used to compute measures.

The implemented strategies ensure the safety, and more precisely the safety invariants SI1 to SI6. The only safety problems were due to experimental bias, on more precisely on the calculation of the safety margins. This underlines the strong dependence of this method on the margin calculation. The system expertise is needed to compute margins that take into account all dynamic and control parameters. The monitor is assessed as permissive in all the cases that enable the assessment.

5.6 Conclusion

Active safety monitors for autonomous systems have to deal with complex safety rules, inducing several interventions that should be consistent. To develop such active monitors, we proposed a formalized process based on the definition of warning states linked to a safety margin. In these warning states it is possible to trig some interventions before the system goes in catastrophic states. We proposed a complete framework, SMOF, starting from hazard analysis and ending in safety rules synthesis. In this paper, we particularly focus on the safety rule generation algorithm, and on its validation on a real industrial case study.

A major benefit of SMOF, is that it provides a systematic and formal approach for the expression of safety rules, where it is usually done ad hoc and only based on the expertise of the analysts. The models are also of great importance to describe the monitor behavior in order to take it into account for the development of the functional layer. Indeed, safety margins and interventions need to be non-ambigus to determine controller reactions. Our approach is based on the use of a well-known formal language (CTL), and we proposed a template available online to simplify the use of our tool. Most of the complex verifications (like permissiveness) are automatically generated and checked. More precisely on the intervention selection, we propose in this paper to take into account the warning state where the system is, but also the path followed to reach this state. A last notable result, is that after application on a real use case, there is no combinatory explosion of the algorithm, and its performance is acceptable.

A main limitation of using our approach lies in the expression of dependencies or partitions of observable variables. Indeed, the efficiency of the generation could be highly increased when the analyst has a good level of expertise. Another, is the fact that the current version of SMOF does not include a mechanism to activate/deactivate the safety rules depending on the task performed by the system. In most of autonomous applications this would be an important issue as systems tends to be more and more versatile. The proposed approach is also limited to the functional level, with simple expression of the safety invariant using propositional logic. For instance, we do not consider interventions like blocking requests from decisional layer.

Future directions concern the extension of the framework to the definition of a several warning regions, in order to trig intervention with different level of efficiency. For instance, a soft intervention might reduce the speed, and if needed a second hard intervention could activate emergency stop. This approach is also linked to the implementation on different layers (hardware and software) with different integrity levels. We also plan to extend SMOF to the observation of the decisional layer (e.g., task or trajectory plans), and possible intervention as rejecting requests from the decisional layer. Finally, the SMOF is about to be transferred to the industry in the context of the European CPSE Lab project CPSELabs (2015-2018), for mobile robot systems in human environment.

6
Conclusion and perspectives

This chapter concludes this manuscript, but also presents some research works which are out of the scope of dependable robots. My research perspectives are presented in the last section.

6.1 What is here

The work presented here falls in two main research axes of dependable system development: safety analysis and assessment (Chapters 2 and 3) and fault tolerant architecture (Chapters 4 and 5).

In Chapter 2, the HAZOP-UML method for hazard identification for human robot interactions has been presented. This method is based on a system modeling language, and provides guidance for analysts with a list of guide words associated to potential deviations of interaction elements. Main benefits are that it is applicable in early steps of the development process, with a systematic approach, providing traceability and complete integration in the development (same models could be used for development and for safety analysis). This collaborative method has been successfully experimented on several real robotic applications in order to analyse human-robot interactions. It is now mature enough to be transferred to industrial projects. An additional effort should be done to completely integrate in the UML models a description of the context and induced adverse situations, which is part of my perspectives.

Chapter 3 on safety case confidence, is a more recent work. The initial motivation which was to apply safety case to a robotic application, led us to analyse how to deal with the uncertainties inherent to such systems. Indeed, while building a safety argument some uncertainties may exist due to non deterministic robot behavior or unknown human

reactions for instance. We proposed to use Dempster-Shafer theory for the definitions of confidence and uncertainty, and some Bayesian Networks formulas for confidence propagation in a safety case. For now, we only use this quantitative approach to perform sensitivity analysis. As presented in the perspective section, an interesting direction is to work on a real interpretation and identification of the confidence values.

Besides these works focusing on safety analysis and assessment, I also presented in chapter 4, a contribution to fault tolerance in an autonomous software architecture. For this work, we proposed a recovery mechanism in case of a detection of an error of a planner (timeout of the planner, property violation of the plan, failed planned action, or failed goal). We assessed one mechanism through an intensive fault injection campaign, and showed that it is possible without reducing performance, to increase dependability with a redundant task planner. We believe that such an approach is still seldom used in robotics mainly due to a resource limitation, an absence of dependability culture in this community, and thus not a priority considering issues still open in path or task planning for instance. Nevertheless, as for other safety critical applications, robotics will surely come to such techniques in a short term.

A second contribution in the field of fault tolerance, is presented in chapter 5. The main objective is to provide for autonomous systems an independent device able to guarantee that safety properties will be verified in case of a failure or an adverse situation that the system cannot manage. We called this component an active safety monitor, in the sense that it is able to observe the system and the environment but also can trigger some actions. The challenge for autonomous systems is to produce the safety properties that might be complex or inconsistent. For that we developed a framework SMOF (Safety Monitoring Framework), for safety rules synthesis. It is actually composed of a formalization of the safety rules concept, and a tool based on model checking technique, which permit to synthesize these rules. For now only one real system has been used as a case study, but first results show that our formalization and the associated tool are fully applicable. This framework (with a dedicated tool) was developed during two PhDs, and is now ready to be extended to more complex architectures and applications.

6.2 What is not here

Besides development of techniques for robot dependability, I was involved in several other researches which are presented here.

6.2.1 Game theory for safety rule synthesis

During the development of SMOF, I also coordinated an exploratory work on game theory applied to the synthesis of safety rules. Intuitively, this approach considers a 2-player game where the safety monitor plays against a malicious adversary that stands for a faulty controller or a disobliging environment. The adversary fires transitions in the system

model. Its aim is to reach a catastrophic state. To counteract the adversary's moves, the monitor may trigger safety interventions. It has a winning strategy if it always succeeds in maintaining both safety and universal liveness, whatever the moves of the adversary. The game theory tool used is the UPPAAL extension for game theory, TIGA. We have succeeded in modeling variable-oriented safety invariants and permissiveness in TIGA, which is state-oriented and supports a small part of CTL. Nevertheless, due to a limited set of CTL operators in TIGA, we had to add new variables, which led to some combinatory explosions, and decrease TIGA performances for the synthesis. In contrast, our tool based on NuSMV, presented in Chapter 5, provides relatively efficient performance at the price of having to develop a branch-and-bound algorithm to explore efficiently the set of possible strategies. We believe that when UPPAL or new tools will be published, game theory would be an interesting candidate for the safety rule synthesis. This work was done in collaboration with Matthieu Roy and Hélène Waselynck. I also co-supervises the post-doctoral of Fanny Duffosé. Results are presented in Machin et al. (2015).

6.2.2 A UML profile for robustness testing

I also contributed to a collaborative work between LAAS (H. Waeselynck and myself) and a professor of UNICAMP, Brazil (Regina Moraes), on the formalization of robustness testing. It is a specific form of black-box testing that complements conformance testing by studying whether erroneous or stressful input conditions (e.g., faults, or attacks) may alter the system's regular behavior. The dependability community has a long tradition of robustness testing based on fault injection. A recurring problem is the lack of approaches to document experiments, which might be one of the reasons why studies reusing tests to compare and to consolidate results are seldom available. Our research investigates whether the Unified Modelling Language (UML) could be used for documenting robustness testing. We developed a specialization and extension of UML profile (U2TP) that addresses the documentation of robustness testing experiments. In this collaboration I was in charge of developing the UML extensions and responsible of the UML semantics conformance. This work led to the publication Moraes et al. (2014), and will be reused in the prospective of robot testing in virtual words.

6.2.3 Geo-privacy risk assessment

An on-going collaboration is in the field of geo-privacy. In this context, I used my expertise in risk analysis to develop a method for geo-privacy risk estimation. In the context of Location-Based Services (LBS), for instance find a point of interest using a mobile phone, there exist many and various challenges for their deployment. The need for identifying the risk related to the processing of personal data before determining the appropriate means to reduce them, is without doubt, one of the most important in the domain of LBS. Unfortunately, to date there is a lack of methodologies to adequately address this

problem. The risk assessment proposals found in security standards are so generic, that they are really difficult to map to Privacy and, even more to the domain of LBS. This work makes a step forward to provide useful tools for the geo-privacy assessment of services and applications of future IoT. In particular, the goal pursued is to provide a geo-privacy Risk Assessment Methodology for LBS (we called it G-PRIME). This work has been done in the context of an ANR project AMORES, led by Marc-Olivier Killijian. We co-supervised a post-doctoral (Jesus Friginal), and we published the results in an international conference and in a journal (Friginal et al. (2014, 2013)).

6.3 Perspectives

This section points out the main perspectives I propose to develop in the field of dependable robots. The first one is the extension of safety monitoring framework presented in Chapter 5, the second one is a new direction in the field of robot testing, and the last one is an extension of the safety argumentation work presented in Chapter 3.

6.3.1 Multi-layered safety monitoring for autonomous architecture

The work presented in Chapter 5 on safety monitoring is limited to combinatory safety rules based on observations of the functional level (e.g., speed, acceleration). We propose to extend this work towards two dimensions. The first one is to adapt the monitoring framework to observations and interventions at the decisional or executive levels of an autonomous architecture. This would imply that the considered monitor (or several monitors), may observe decisional layer outputs such as plans or requests from the executive level. Interventions may also concern both layers (e.g. block a request, reject a plan). While exploring this multi-layered monitoring, we believe that some interventions may be ordered and applied according to their efficiency and impact on the system. For instance, in case of a rule violation, a monitor may first trig a "soft" intervention at the executive level (like blocking a request), and in case of a second violation stop the robot engaging the brakes. The new monitoring framework should be able to take into account the possibility to trig intervention at several levels of the autonomous architecture, according to different levels fo observation.

This work will be carried out in close collaboration with robotics experts to determine the possible observations and interventions, before modeling them. We plan to develop new concepts for a monitoring and also an improvement of the current tool based on NuSMV. We just started to collaborate in 2015, through a co-supervision with Hélène Waeselynck of a PhD student (Lola Masson). A longer term investigation will be to take into account the fact that safety rules may be changed or adapted due to history of the system. Learning, which is part of artificial intelligence techniques, is an interesting candidate, but for now too uncertain to be a core function in a safety channel.

6.3.2 Testing robots in virtual worlds

An autonomous robot should perform a variety of tasks without any human supervision, in various, complex and uncertain environments. In order to deploy such systems in human or critical environments, rigorous validation is of paramount importance. The tests, for safety and cost constraints, may be first realized using simulation of the robot moving in virtual worlds. The objective of this work is to develop a test method that exploits such simulation facilities to place the robotic system in a wide variety of operational situations and observe how it faces them. A few studies have focused on the intensive testing of autonomous systems in realistic simulation environments. This topic was addressed in the European project R3COP (2015). In this project, an inventory of relevant features for vision algorithms has been prepared, and some work explored methods of generating random 3D scenes including these characteristics (Zendel et al., 2013). Other works of this project addressed the automatic analysis of mobile robots tests (Horányi et al., 2013b), using a formal language developed at LAAS to analyze test traces (Waeselynck et al., 2010). Apart from this project, we can also cite the work using the Perlin noise to generate rudimentary 2D worlds and test the robot navigation (Arnold and Alexander, 2013).

In our perspective, we plan to develop an automatic generation of virtual worlds, like worlds generated in video games (e.g., procedural generation). The main contribution will be to synthesize the virtual worlds in such a way that they exhibit the safety-relevant cases, and ensure adequate coverage of the functionalities deployed on the robot. We also plan to explore how to automatically change the virtual worlds according to the results of the tests. This work should be done in close collaboration with robotics experts, particularly for the simulation tool. We plan to use MORSE (Modular OpenRobots Simulation Engine) Morse (2015) which is developed at LAAS-CNRS to serve that purpose. It provides the right level of simulation (perception/action) to validate the execution of complex plans. A first proposal would be to apply this method to the navigation functionality developed in Genom (2015), which is deployed on real mobile robots at LAAS. A final objective would be to deploy a method and models independent from the simulator and the robot architecture. Hence, the implementation of such an approach aimed at providing the following contributions:

- a model of the worlds, missions and uncertainties, related to safety analysis;
- identify a set of relevant criteria to select the tests;
- test input generation algorithms, by combining generation processes from the field of the "classical" test and procedural methods of generating worlds from the field of video games;
- analyze simulation traces, with both qualitative aspects (properties of satisfaction) and quantitative measures;

- a model-driven approach to facilitate the reuse of the building blocks of the approach.

A PhD, which I co-advise with Helene Waeselynck (dependability group) and Felix Ingrand (robotic group) on those issues started in 2014. But it is obvious that a complete generic framework development will require more resources than a single PhD.

6.3.3 Certification of collaborative autonomous systems

Certification of robotic systems is today a major issue for their deployment. Guarantees of safe behavior of such systems cannot indeed be provided to regulatory bodies. We proposed in Chapters 2 and 3, two approaches that may contribute to the certification process: hazard identification and safety argument production. We still need to improve and extend them. First, it is important to extend HAZOP-UML to take into account the system context (light, surfaces, noises, etc.) in UML models in order to analyse their impact on safety. One direction is to model the context as it is done by Horányi et al. (2013b) for obstacles. For now, we only integrate this aspect in the HAZOP tables. Such a contribution may increase traceability through the use of models, but also help to perform a more systematic analysis of adverse situations. Another improvement is to link HAZOP-UML with safety arguments, using for example the result HAZOP hazard list to instantiate a safety case pattern. Such an approach is equivalent to the one proposed by Denney and Pai (2012), but our approach starting from system modeling and ending in a safety case would cover more steps of the development process.

Once a safety argument built, we expect to still have many uncertainties in the inferences and evidences in the context of a collaborative and autonomous robot. A main issue is how to model and estimate residual uncertainties in an argument. We will extend previous work for confidence assessment of safety cases to take into account uncertainties specific to collaborative autonomous system. It would be particularly interesting to be fully compliant with the belief theory as it is done by Pichon et al. (2012). This may lead to integrate that disbeliefs may exist in elements of a safety argument. This would release the hypothesis done in Chapter 3 (we assumed that if an element of an argument has a disbelief different from zero, then this element was removed from the safety case). For this task we wish to collaborate with experts in belief theory particularly in France, from IRIT at Toulouse or UTC in Compiegne. I co-supervise with Gilles Motet, a PhD student (Rui Wang), which started in 2014, partly on this subject (focusing on the belief theory aspects). We just started to work on certification in aeronautics, but we expect that the results might be applicable to systems such as collaborative autonomous robots.

Bibliography

2006/42/EC (2006). Council directive 2006/42/ec on machinery. Official Journal of the European Union (JOCE) L157.

93/42/EEC (1993). Council directive of the 14th of june 1993 concerning medical devices. Journal Officiel des Communautés Européennes (JOCE) L169.

Aguirre, F., Sallak, M., Schön, W., and Qiu, S. (2013). On the distinction between aleatory and epistemic uncertainty and its implications on reliability and risk analysis. In *European Safety and Reliability Conference, ESREL 2013*.

Alami, R., Albu-Schaeffer, A., Bicchi, A., Bischoff, R., Chatila, R., Luca, A. D., Santis, A. D., Giralt, G., Guiochet, J., Hirzinger, G., Ingrand, F., Lippiello, V., Mattone, R., Powell, D., Sen, S., Siciliano, B., Tonietti, G., and Villani, L. (2006). Safe and Dependable Physical Human-Robot Interaction in Anthropic Domains: State of the Art and Challenges. In *report IROS'06 Workshop on Physical Human-Robot Interaction in Anthropic Domains*.

Alami, R., Chatila, R., Fleury, S., Ghallab, M., and Ingrand, F. (1998). An Architecture for Autonomy. *International Journal of Robotics Research*, 17(4), pages 315–337.

Albu-Schaffer, A., Eiberger, O., Grebenstein, M., Haddadin, S., Ott, C., Wimbock, T., Wolf, S., and Hirzinger, G. (2008). Soft robotics. *Robotics Automation Magazine, IEEE*, 15(3), pages 20–30.

Albus, J. S., Huang, H.-M., Messina, E. R., Murphy, K., Juberts, M., Lacaze, A., Balakirsky, S. B., Shneier, M. O., Hong, T. H., Scott, H. A., Proctor, F. M., Shackleford, W. P., Michaloski, J. L., Wavering, A. J., Kramer, T. R., Dagalakis, N. G., Rippey, K. A. S., William G. and, and Legowik, S. (2002). *4d/rcs version 2.0: A reference model architecture for unmanned vehicle systems*. Technical report, National Institute of Standards and Technology (NIST), U.S. Department of Commerce.

Alemzadeh, H., Chen, D., Lewis, A., Kalbarczyk, Z., and Iyer, R. (2015). Systems-theoretic safety assessment of robotic telesurgical system. In *34th International Conference on Computer Safety, Reliability and Security*.

Alexander, R., Gorry, B., and Kelly, T. (2010). Safety lifecycle activities for autonomous systems development. In *5th SEAS DTC Technical Conference*.

Alexander, R., Hallmay, M., and Kelly, T. (2007). Certification of Autonomous Systems under UK Military Safety Standards. In *Proceedings of the 25th International System Safety Conference (ISSC'07)*. Washington DC, USA.

Alexander, R., Hawkins, H., and Rae, D. (2015). *Situation coverage–a coverage criterion for testing autonomous robots*. Technical report YCS-2015-496, University of York, Department of computer science.

Alexander, R., Herbert, N., and Kelly, T. (2008). Structuring safety cases for autonomous systems. In *Proceedings of 3rd IET International System Safety Conference*. Birmingham, UK.

Alexander, R., Herbert, N., and Kelly, T. (2009). Deriving safety requirements for autonomous systems. In *SEAS DTC Technical Conference*.

Allenby, K. and Kelly, T. (2001). Deriving safety requirements using scenarios. In *Requirements Engineering, 2001. Proceedings. Fifth IEEE International Symposium on*, pages 228–235.

Anaheed, A., BaekGyu, K., Insup, L., and Oleg, S. (2012). A systematic approach to justifying sufficient confidence in software safety arguments. In *Computer Safety, Reliability, and Security Lecture Notes in Computer Science*, edited by O. Frank and D. Peter, volume 7612, pages 305–316. Springer Berlin Heidelberg.

Anaheed, A., Jian, C., Oleg, S., and Insup, L. (2013). Assessing the overall sufficiency of safety arguments. In *21st Safety-critical Systems Symposium (SSS'13), Bristol, United Kingdom*.

Arlat, J., Crouzet, Y., and Laprie, J.-C. (1989). Fault injection for dependability validation of fault-tolerant computing systems. In *International Symposium on Fault-Tolerant Computing (FTCS)*, pages 348–355.

Arlow, A., Duffy, C., and McDermid, J. (2006). Safety specification of the active traffic management control system for english motorways. In *The First Institution of Engineering and Technology International Conference on System Safety*.

Armoush, A. (2010). *Design Patterns for Safety-Critical Embedded Systems*. PhD thesis, Embedded Software Laboratory - RWTH Aachen University.

Arnold, J. and Alexander, R. (2013). Testing autonomous robot control software using procedural content generation. In *Computer Safety, Reliability, and Security*, edited by F. Bitsch, J. Guiochet, and M. Kaâniche, volume 8153 of *Lecture Notes in Computer Science*, pages 33–44. Springer Berlin Heidelberg.

Avižienis, A., Laprie, J., Randell, B., and Landwehr, C. (2004). Basic concepts and taxonomy of dependable and secure computing. *IEEE Transactions on Dependable and Secure Computing*, 1(1), pages 11–33.

Avižienis, A., Laprie, J.-C., Randell, B., and Landwehr, C. (2004). Basic concepts and taxonomy of dependable and secure computing. *IEEE Transactions on Dependable and Secure Computing*, 1(1), pages 11–33.

Bader, K., Lussier, B., and Schön, W. (2014). A fault tolerant architecture for data fusion targeting hardware and software faults. In *The 20th IEEE Pacific Rim International Symposium on Dependable Computing (PRDC 2014)*, p. 10. Singapore.

Bensalem, S., Gallien, M., Ingrand, F., Kahloul, I., and Nguyen, T.-H. (2009). Toward a more dependable software architecture for autonomous robots. *IEEE Robotics and Automation Magazine*, 16, pages 1–11.

Bensalem, S., Havelund, K., and Orlandini, A. (2014). Verification and validation meet planning and scheduling. *International Journal on Software Tools for Technology Transfer*, 16, pages 1–12.

Bensalem, S., Silva, L. d., Ingrand, F., and Yan, R. (2011). A verifiable and correct-by-construction controller for robot functional levels. *Journal of Software Engineering for Robotics*, 1(2).

Bernard, D. E., Gamble, E. B., Rouquette, N. F., Smith, B., Tung, Y. W., Muscettola, N., Dorias, G. A., Kanefsky, B., Kurien, J., Millar, W., Nayal, P., Rajan, K., and Taylor, W. (2000). Remote Agent Experiment DS1 Technology Validation Report. Ames Research Center and JPL.

Bishop, P., Bloomfield, R., and Guerra, S. (2004). The future of goal-based assurance cases. In *DSN Workshop on Assurance Cases : Best Practices, Possibles Obstacles, and Future Opportunities*. Florence, Italy.

Bjørn Axel Gran, R. F. and Thunem, A. P.-J. (2004). An approach for model-based risk assessment. In *23rd International Conference, SAFECOMP 2004, Potsdam, Germany*, pages 311–324. Springer Berlin / Heidelberg.

Blanquart, J.-P. (2010). *Survey of state of the art and of the practice in safety and diagnosability*. Technical report D_SP1_R5.8_M2, EADS Astrium Satellites, CESAR European Project.

Blender3D (2015). www.blender.org. Accessed July 2015.

Böhm, P. and Gruber, T. (2010). A novel HAZOP study approach in the RAMS analysis of a therapeutic robot for disabled children. In *Computer Safety, Reliability, and Security*, pages 15–27. Springer.

Boiteau, M., Dutuit, Y., Rauzy, A., and Signoret, J.-P. (2006). The AltaRica data-flow language in use: modeling of production availability of a multi-state system. *Reliability Engineering & System Safety*, 91(7), pages 747 – 755.

Borrelly, J. J., Coste-Manière, E., Espiau, B., Kapellos, K., Pissard-Gibollet, R., Simon, D., and Turro, N. (1998). The ORCCAD Architecture. *The International Journal of Robotics Research*, 17(4), pages 338–359.

Bouguerra, A., Karlsson, L., and Saffiotti, A. (2008). Monitoring the execution of robot plans using semantic knowledge. *Robotics and Autonomous Systems*, 56(11), pages 942 – 954.

BRICS (2009-2013). Best of robotics. Project supported by the European Commission under the 7th Framework Programme, www.best-of-robotics.org. Accessed: 2015-04-30.

Brodskiy, Y., Wilterdink, R. J. W., Stramigioli, S., and Broenink, J. F. (2014). Fault avoidance in development of robot motion-control software by modeling the computation. In *4th International Conference on Simulation, Modeling, and Programming for Autonomous Robots, SIMPAR2014, Bergamo, Italy*, volume LNCS8810, pages 158–169. Bergamo, Italy Berlin: Springer International Publishing.

Bruyninckx, H. (2001). Open robot control software: the orocos project. In *IEEE International Conference on Robotics and Automation (ICRA)*, volume 3, pages 2523–2528 vol.3.

Cantrell, S. and Clemens, P. (2009). Finding all the hazards how do we know we are done? *Professional Safety, American Society of Safety Engineers*, 54(11).

CARLOS (2012-2014). Cooperative mobile robotics. Project supported by the European Commission under the 7th Framework Programme, carlosproject.eu/who. Accessed: 2015-04-30.

Carlson, J. and Murphy, R. R. (2003). Reliability Analysis of Mobile Robots. In *Proceedings of the 2003 IEEE International Conference on Robotics & Automation*, pages 274–281. Taipei, Taiwan.

Cesta, A., Finzi, A., Fratini, S., Orlandini, A., and Tronci, E. (2010). Validation and verification issues in a timeline-based planning system. *The Knowledge Engineering Review*, 25(Special Issue 03), pages 299–318.

Chen, I. R. (1997). Effects of Parallel Planning on System Reliability of Real-Time Expert Systems. *IEEE Transactions on Reliability*, 46(1), pages 81–87.

Chen, I. R., Bastani, F. B., and Tsao, T. W. (1995). On the Reliability of AI Planning Software in Real-Time Applications. *IEEE Transactions on Knowledge and Data Engineering*, 7(1), pages 14–25.

Chien, S., Sherwood, R., Tran, D., Cichy, B., Rabideau, G., Castano, R., Davis, A., Mandl, D., Trout, B., Shulman, S., and Boyer, D. (2005). Using autonomy flight software to improve science return on earth observing one. *Journal of Aerospace Computing, Information, and Communication*, 2(4), pages 196–216.

CHRIS (2008-2012). Cooperative Human Robot Interaction Systems. Project supported by the European Commission under the 7th Framework Programme, www.chrisfp7.eu/. Accessed: 2015-04-30.

Cimatti, A., Clarke, E., Giunchiglia, E., Giunchiglia, F., Pistore, M., Roveri, M., Sebastiani, R., and Tacchella, A. (2002). Nusmv 2: An opensource tool for symbolic model checking. In *International Conference on Computer Aided Verification (CaV)*, pages 359–364.

Cook, D., Vardy, A., and Lewis, R. (2014). A survey of auv and robot simulators for multi-vehicle operations. In *IEEE/OES Autonomous Underwater Vehicles (AUV),*, pages 1–8.

CORAS (2014). A platform for risk analysis of security critical systems. coras.sourceforge.net.

CPSELabs (2015-2018). Cyber-Physical Systems Engineering Labs. Project funded by the European Union, Horizon2020 Programme, www.cpse-labs.eu, accessed 2015-05-17.

Crestani, D., Godary-Dejean, K., and Lapierre, L. (2015). Enhancing fault tolerance of autonomous mobile robots. In *Journal of Robotics and Autonomous Systems*. Elsevier.

Crouzet, Y., Waeselynck, H., Lussier, B., and Powell, D. (2006). The SESAME Experience: from Assembly Languages to Declarative Models. In *Proceedings of the 2nd Workshop on Mutation Analysis*. Raleigh, NC.

Cyra, L. and Górski, J. (2011). Support for argument structures review and assessment. *Reliability Engineering and System Safety*, 96(1), pages 26–37.

Daran, M. and Thévenod-Fosse, P. (1996). Software Error Analysis: a Real Case Study Involving Real Faults and Mutations. In *Proceedings of the 1996 ACM SIGSOFT International Symposium on Software Testing and Analysis*. San Diego, California.

Dardenne, A., Fickas, S., and van Lamsweerde, A. (1993). Goal-directed requirements acquisition. In *Science of Computer Programming*, volume 20, pages 3–50.

DefStan 00-56 (2004). Defence standard 00-56 issue 3: Safety management requirements for defence systems. UK Ministry of Defence.

DefStan00-58 (2000). HAZOP studies on systems containing programmable electronics. Defence Standard, Ministry of Defence, UK.

Delgado, N., Gates, A. Q., and Roach, S. (2004). A taxonomy and catalog of runtime software-fault monitoring tools. *Transactions on Software Engineering*, 30(12), pages 859–872.

Denney, E., Habli, I., and Pai, G. (2011). Towards measurements of confidence in safety cases. In *Proceedings of the 5th International Symposium on Empirical Software Engineering and Measurement (ESEM'11)*. Banff, Canada.

Denney, E. and Pai, G. (2012). A lightweight methodology for safety case assembly. In *Computer Safety, Reliability, and Security, SAFECOMP2012*, edited by F. Ortmeier and P. Daniel, volume 7612, pages 1–12. Springer Berlin Heidelberg.

Dhillon, B. (1991). *Robot reliability and safety*. Springer-Verlag.

Dhillon, B. and Anude, O. (1993). Robot safety and reliability: a review. *Microelectronics and Reliability*, 33(3), pages 413–429.

Dhillon, B. and Fashandi, A. (1997). Safety and reliability assessment techniques in robotics. *Robotica*, 15, pages 701–708.

Díez, F. J. and Druzdzel, M. J. (2007). *Canonical probabilistic models for knowledge engineering*. Technical report, Research Center on Intelligent Decision-Support Systems. UNED. Madrid, Spain.

Do Hoang, Q. (2015). *Analyse et justification de la sécurité de systèmes robotiques en interaction physique avec l'humain (in French)*. PhD thesis, INP Toulouse, LAAS-CNRS.

Do Hoang, Q. A., Guiochet, J., Powell, D., and Kaâniche, M. (2012). Human-robot interactions: model-based risk analysis and safety case construction. In *Embedded Real Time Software and Systems (ERTS2 2012)*. Toulouse, France.

Dogramadzi, S., Giannaccini, M. E., Harper, C., Sobhani, M., Woodman, R., and Choung, J. (2014). Environmental hazard analysis-a variant of preliminary hazard analysis for autonomous mobile robots. *Journal of Intelligent & Robotic Systems*, 76(1), pages 73–117.

Durand, B., Godary-Dejean, K., Lapierre, L., Passama, R., and Crestani, D. (2010). Fault tolerance enhancement using autonomy adaptation for autonomous mobile robots. In *International Conference on Control and Fault Tolerant Systems (SysTol)*, pages 24–29.

Echeverria, G., Lassabe, N., Degroote, A., and Lemaignan, S. (2011). Modular open robots simulation engine: Morse. In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pages 46–51. IEEE.

Ertle, P., Gamrad, D., Voos, H., and Soffker, D. (2010). Action planning for autonomous systems with respect to safety aspects. In *IEEE International Conference on Systems Man and Cybernetics (SMC)*, pages 2465–2472.

Fenton, N. and Neil, M. (2012). *Risk Assessment and Decision Analysis with Bayesian Networks*. CRC Press, Taylor and francis Group.

Filippini, R., Sen, S., and Bicchi, A. (2008). Toward soft robots you can depend on. *Robotics Automation Magazine, IEEE*, 15(3), pages 31–41.

Flacco, F., De Luca, A., Sardellitti, I., and Tsagarakis, N. G. (2012). On-line estimation of variable stiffness in flexible robot joints. *Int. J. Rob. Res.*, 31(13), pages 1556–1577.

Fleury, S., Herrb, M., and Chatila, R. (1997). Genom: A tool for the specification and the implementation of operating modules in a distributed robot architecture. In *International Conference on Intelligent Robots and Systems (IROS)*, volume 2, pages 842–849.

Fox, J. and Das, S. (2000). *Safe and sound - Artificial Intelligence in Hazardous Applications*. AAAI Press - The MIT Press.

Frantz, I., Andy, G., John, M., and Toyn, I. (2007). Integrating safety and formal analyses using UML and PFS. *Reliability Engineering and System Safety*, 92(2), pages 156–170.

Friginal, J., Gambs, S., Guiochet, J., and Killijian, M.-O. (2014). Towards privacy-driven design of a dynamic carpooling system. *Pervasive and mobile computing*, 14, pages 71–82.

Friginal, J., Guiochet, J., and Killijian, M.-O. (2013). Towards a privacy risk assessment methodology for location-based systems. In *International Conference on Mobile and Ubiquitous Systems: Computing, Networking and Services*.

Gat, E. (1998). Three-layer architectures. In *Artificial Intelligence and Mobile Robots*, edited by D. Kortenkamp, R. P. Bonasso, and R. Murphy, pages 195–210. Cambridge, MA, USA: MIT Press.

Gazebo (2015). gazebosim.org/. Accessed July 2015.

Genom (2015). https://www.openrobots.org/wiki/genom. Accessed July 2015.

Ghallab, M. and Laruelle, H. (1994). Representation and control in IxTeT, a temporal planner. In *The Second Artificial Intelligence Planning Systems Conference*, pages 61–67. Chicago, IL, USA: AAAI Press.

Glauser, D., Flury, P., Burckhardt, C., and Kassler, M. (1993). Mechanical concept of the neurosurgical robot Minerva. *Robotica*, 11(6), pages 567–575.

Goldberg, A., Havelund, K., and McGann, C. (2005). Runtime verification for autonomous spacecraft software. In *Aerospace Conference, 2005*, pages 507–516.

Goodenough, J., Weinstock, C., and Klein, A. (2013). Eliminative induction: A basis for arguing system confidence. In *35th International Conference on Software Engineering (ICSE2013)*, pages 1161–1164.

Goodenough, J. B., Weinstock, C. B., and Klein, A. Z. (2012). *Toward a Theory of Assurance Case Confidence*. Technical report, Software Engineering Institute, Carnagie Mellon University.

Goodloe, A. E. and Pike, L. (2010). Monitoring distributed real-time systems: A survey and future directions. Rapport technique, NASA/CR-2010-216724.

Gorski, J. and Jarzebowicz, A. (2005). Development and validation of a HAZOP-based inspection of UML models,. In *3rd World Congress for Software Quality, Munich, Germany*.

GSN-Standard (2011). GSN Community Standard version 1. www.goalstructuringnotation.info [Online; accessed Decembre 18th 2014].

Gspandl, S., Podesser, S., Reip, M., Steinbauer, G., and Wolfram, M. (2012). A dependable perception-decision-execution cycle for autonomous robots. In *International Conference on Robotics and Automation (ICRA)*, pages 2992–2998.

Guiochet, J., Do Hoang, Q. A., and Kaâniche, M. (2015). A model for safety case confidence assessment. In *The 34rd International Conference on Computer Safety, Reliability and Security (SAFECOMP2015)*. Springer.

Guiochet, J., Do Hoang, Q. A., Kaâniche, M., and Powell, D. (2012). Applying existing standards to a medical rehabilitation robot: Limits and challenges. In *Workshop FW5: Safety in Human-Robot Coexistence & Interaction: How can Standardization and Research benefit from each other?, IEEE/RSJ Intern. Conference Intelligent Robots and Systems (IROS2012), Portugal*.

Guiochet, J., Do Hoang, Q. A., Kaâniche, M., and Powell, D. (2013). Model-based safety analysis of human-robot interactions: The MIRAS walking assistance robot. In *Rehabilitation Robotics (ICORR), 2013 IEEE International Conference on*, pages 1–7.

Guiochet, J., Martin-Guillerez, D., and Powell, D. (2010). Experience with model-based user-centered risk assessment for service robots. In *IEEE International Symposium on High-Assurance Systems Engineering (HASE'2010)*, pages 104–113. San Jose, CA, USA: IEEE Computer Society.

Guiochet, J., Motet, G., Baron, C., and Boy, G. (2004). Toward a human-centered uml for risk analysis - application to a medical robot. In *Proc. of the 18th IFIP World Computer Congress (WCC), Human Error, Safety and Systems Development (HESSD04)*, edited by C. Johnson and P. Palanque, pages 177–191. Kluwer Academic Publisher.

Guiochet, J. and Powell, D. (2006). *Etude et analyse de systèmes indépendants de sécurité-innocuité de type safety bag*. Technical report 05551, LAAS-CNRS.

Guiochet, J., Powell, D., Baudin, E., and Blanquart, J.-P. (2008a). Online safety monitoring using safety modes. In *6th IARP - IEEE/RAS - EURON Joint Workshop on Technical Challenges for Dependable Robots in Human Environments*. Pasadena, CA, USA.

Guiochet, J., Powell, D., and Sarr, R. (2009). Deliverable d1.8 report on risk reduction approach for a roblt manpulator in close interaction with a person with respect to identified use cases.

Guiochet, J., Powell, D., Taofifenua, O., Guhl, T., and Bischoff, R. (2008b). Preliminary report on risk reduction strategy for a robot manipulator in close interaction with a person with respect to identified use cases. d1.7.

Guiochet, J., Powell, D., Taofifenua, O., Guhl, T., and Bischoff, R. (2008c). Report on analysis methods including safety integrity requirements for safe physical human-robot interaction. d1.6.

Haddadin, S. (2014). *Towards Safe Robots, Approaching Asimov's 1st Law*, volume 90 of *Springer Tracts in Advanced Robotics*. Springer.

Haddadin, S. (2015). Physical safety in robotics. In *Formal Modeling and Verification of Cyber-Physical Systems*, edited by R. Drechsler and U. Kühne, pages 249–271. Springer Fachmedien Wiesbaden.

Haddadin, S., Suppa, M., Fuchs, S., Bodenmüller, T., Albu-Schäffer, A., and Hirzinger, G. (2011). Towards the robotic co-worker. In *The 14th International Symposium on Robotics Research (ISRR2011)*, edited by C. Pradalier, R. Siegwart, and G. Hirzinger, pages 261–282. Springer Berlin Heidelberg.

Hansen, K. M., Wells, L., and Maier, T. (2004). HAZOP analysis of UML-based software architecture descriptions of safety-critical systems. In *Proceedings of NWUML*.

Hawkins, R., Kelly, T., Knight, J., and Graydon, P. (2011). A new approach to creating clear safety arguments. In *Proceedings of 19th Safety Critical Systems Symposium*. Southampton, UK.

HAZOP-UML website (2015). LAAS-CNRS, https://www.laas.fr/projects/HAZOPUML. Accessed July 2015.

Hitchcock, D. (2005). Good reasoning on the toulmin model. *Argumentation*, 19(3), pages 373–391.

Hobbs, C. and Lloyd, M. (2012). The application of bayesian belief networks to assurance case preparation. In *Proceedings of the 20th Safety-Critical Systems Symposium, Bristol, UK*, pages 159–176. Springer London.

Horányi, G., Micskei, Z., and Majzik, I. (2013a). Scenario-based automated evaluation of test traces of autonomous systems. In *Workshop on Dependable Embedded and Cyber-physical Systems (DECS) in the International Conference on Computer Safety, Reliability and Security (SafeComp)*.

Horányi, G., Micskei, Z., and Majzik, I. (2013b). Scenario-based automated evaluation of test traces of autonomous systems. In *SAFECOMP 2013-Workshop DECS (ERCIM/EWICS Workshop on Dependable Embedded and Cyber-physical Systems) of the 32nd International Conference on Computer Safety, Reliability and Security*, p. NA.

Howey, R., Long, D., and Fox, M. (2004). VAL: Automatic Plan Validation, Continuous Effects and Mixed Initiative Planning using PDDL. In *ICTAI*. Boca Raton, Florida.

HSE (2012). *Collision and injury criteria when working with collaborative robots*. Technical report, Prepared by the Health and Safety Laboratory for the Health and Safety Executive (HSE), UK.

Huang, H., Pavek, K., Novak, B., Albus, J., and Messina, E. (2005). A framework for autonomy levels for unmanned systems (alfus). In *Proceedings of The AUVSI's Unmanned Systems North America*.

Huang, H.-M. (2008). Autonomy levels for unmanned systems (alfus) framework - volume i: Terminology. Special Publication 1011-I-2.0, National Institute of Standards and Technology (NIST).

IEC61508 (2010). Functional safety of electrical/electronic/programmable electronic safety-related systems. Édition 2. International Electrotechnical Commission.

IEC61508-5 (2010). Functional safety of electrical/electronic/programmable electronic safety-related systems: Part 5: Examples of methods for the determination of safety integrity level. International Electrotechnical Commission.

IEC61508-7 (2010). Functional safety of electrical / electronic / programmable electronic safety-related systems - part 7: Overview of techniques and measures. International Organization for Standardization and International Electrotechnical Commission.

IEC61882 (2001). Iec61882 hazard and operability studies (hazop studies) - application guide.

IEC62304 (2006). Medical device software - software life cycle processes. International Electrotechnical Commission.

Ingrand, F. and Ghallab, M. (2014). Deliberation for autonomous robots: A survey. *Artificial Intelligence*.

Ingrand, F., Lacroix, S., Lemai-Chenevier, S., and Py, F. (2007). Decisional autonomy of planetary rovers. *Journal of Field Robotics*, 24(7), pages 559–580.

Ishimatsu, T., Leveson, N., Thomas, J., Katahira, M., Miyamoto, Y., and Nakao, H. (2010). Modeling and hazard analysis using STPA. In *4th Conference of the International Association for the Advancement of Space Safety (IAASS)*.

ISO10218-1 (2011). Robots and robotic devices – safety requirements for industrial robots – part 1: Robots. International Organization for Standardization.

ISO10218-2 (2011). Robots and robotic devices – safety requirements for industrial robots – part 2: Robot systems and integration. International Organization for Standardization.

ISO12100 (2010). Safety of machinery - general principles for design - risk assessment and risk reduction. International Standard Organisation.

ISO13482 (2014). Robots and robotic devices – safety requirements for personal care robots. International Organization for Standardization.

ISO13849-1 (2006). Safety of machinery – safety-related parts of control systems – part 1: General principles for design. International Organization for Standardization.

ISO31000 (2009). Risk management - Principles and guidelines. International Organization for Standardization.

ISO/FDIS14971 (2006). Medical devices - Application of risk management to medical devices. International Standard Organisation.

ISO/IEC-Guide51 (1999). Safety aspects - Guidelines for their inclusion in standards. International Organization for Standardization.

Jarzebowicz, A. and Górski, J. (2006). Empirical evaluation of reading techniques for UML models inspection. *International Transactions on Systems Science and Applications*, 1(2), pages 103–110.

Johannessen, P., Grante, C., Alminger, A., Eklund, U., and Torin, J. (2001). Hazard analysis in object oriented design of dependable systems. In *2001 International Conference on Dependable Systems and Networks, Göteborg, Sweden*, pages 507–512.

Johnson, B. and Kress-Gazit, H. (2015). Analyzing and revising synthesized controllers for robots with sensing and actuation errors. *I. J. Robotic Res.*, 34, pages 816–832.

Joyeux, S., Lampe, A., Alami, R., and Lacroix, S. (2005). Simulation in the LAAS Architecture. In *Software Development in Robotics Workshop, International Conference on Robotics and Automation (ICRA05)*. Barcelona, Spain.

Kane, A., Fuhrman, T., and Koopman, P. (2014). Monitor based oracles for cyber-physical system testing: Practical experience report. In *International Conference on Dependable Systems and Networks (DSN)*, pages 148–155.

Kazanzides, P. (2009). Safety design for medical robots. In *Annual International Conference of the IEEE Engineering in Medicine and Biology Society*, pages 7208–7211.

Kelly, T. and McDermid, J. (1997). Safety case construction and reuse using patterns. In *16th International Conference on Computer Safety and Reliability (SAFECOMP97)*.

Kelly, T. P. (1998). *Arguing Safety – A Systematic Approach to Managing Safety Cases*. PhD thesis, University of York.

Khatib, L., Muscettola, N., and Havelund, K. (2001). Mapping Temporal Planning Constraints into Timed Automata. In *TIME*, pages 21–27. Cividale del Friuli, Italy.

Klein, P. (1991). The safety-bag expert system in the electronic railway interlocking system Elektra. *Expert Systems with Applications*, 3, pages 499 – 506.

Kraetzschmar, G., Shakhimardanov, A., Paulus, J., Hochgeschwender, N., and Reckhaus, M. (2010). *Specifications of Architectures, Modules, Modularity, and Interfaces for the BROCRE Software Platform and Robot Control Architecture Workbench*. Technical report, BRICS FP7 project deliverable D-2.2.

Kruse, T., Pandey, A. K., Alami, R., and Kirsch, A. (2013). Human-aware robot navigation: A survey. *Journal of Robotics and Autonomous Systems*, 61(12), pages 1726–1743.

Lemai, S. and Ingrand, F. (2004). Interleaving Temporal Planning and Execution in Robotics Domains. In *The National Conference On Artificial Intelligence*, pages 617–622. San Jose, California.

Leucker, M. and Schallhart, C. (2009). A brief account of runtime verification. *Journal of Logic and Algebraic Programming*, 78(5), pages 293–303.

Leveson, N. G. (2011). *Engineering a Safer World, Systems Thinking Applied to Safety*. The MIT Press.

Lipaczewski, M., Ortmeier, F., Prosvirnova, T., Rauzy, A., and Struck, S. (2015). Comparison of modeling formalisms for safety analyses: SAML and AltaRica. *Reliability Engineering & System Safety*, (0), pages –.

Littlewood, B. and Wright, D. (2007). The use of multilegged arguments to increase confidence in safety claims for software-based systems: A study based on a BBN analysis of an idealized example. *IEEE Trans. Software Eng.*, 33(5), pages 347–365.

Lussier, B. (2007). *Fault Tolerance in Autonomous Systems*. PhD thesis, Institut National Polytechnique de Toulouse (in French).

Lussier, B., Gallien, M., Guiochet, J., Ingrand, F., Killijian, M. O., and Powell, D. (2007a). Experiments with Diversifies Models for Fault-Tolerant Planning. In *Proceedings of the 5th IARP/IEEE-RAS/EURON Joint Workshop on Technical Challenge for Dependable Robots in Human Environments*. Roma, Italia.

Lussier, B., Gallien, M., Guiochet, J., Ingrand, F., Killijian, M.-O., and Powell, D. (2007b). Fault tolerant planning for critical robots. In *37th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN07), Edinburgh, UK*.

Lussier, B., Gallien, M., Guiochet, J., Ingrand, F., Killijian, M.-O., and Powell, D. (2007c). Planning with diversified models for fault-tolerant robots. In *Proc. of The International Conference on Automated Planning and Scheduling (ICAPS07), Providence, Rhode Island, USA*, pages 216–223.

Lussier, B., Guiochet, J., Ingrand, F., Killijian, M.-O., and Powell, D. (2015). *Fault tolerant planning: towards dependable autonomous robots*. Technical report.

Lussier, B., Lampe, A., Chatila, R., Ingrand, F., Killijian, M. O., and Powell, D. (2005). Fault Tolerance in Autonomous Systems: How and How Much? In *Proceedings of the 4th IARP/IEEE-RAS/EURON Joint Workshop on Technical Challenge for Dependable Robots in Human Environments*. Nagoya, Japan.

Machin, M., Dufossé, F., Blanquart, J., Guiochet, J., Powell, D., and Waeselynck, H. (2014a). Specifying safety monitors for autonomous systems using model-checking. In *The 33rd International Conference on Computer Safety, Reliability and Security (SAFECOMP2014)*, edited by A. Bondavalli and F. D. Giandomenico, pages 262–277. Springer International Publishing.

Machin, M., Dufossé, F., Blanquart, J.-P., Guiochet, J., Powell, D., and Waeselynck, H. (2014b). Specifying safety monitors for autonomous systems. In *SAFECOMP*. LNCS.

Machin, M., Dufossé, F., Guiochet, J., Powell, D., Roy, M., and Waeselynck, H. (2015). Model-checking and game theory for synthesis of safety rules. In *16th IEEE International*

Symposium on High Assurance Systems Engineering, HASE 2015, Daytona Beach, FL, USA, January 8-10, 2015, pages 36–43.

Mainprice, J., Sisbot, E. A., Siméon, T., and Alami, R. (2010). Planning safe and legible hand-over motions for human-robot interaction. *IARP Workshop on Technical Challenges for Dependable Robots in Human Environments*, 2(6), p. 7.

Mallet, A., Pasteur, C., Herrb, M., Lemaignan, S., and Ingrand, F. (2010). Genom3: Building middleware-independent robotic components. In *International Conference on Robotics and Automation (ICRA)*, pages 4627–4632.

Malm, T., Viitaniemi, J., Latokartano, J., Lind, S., Venho-Ahonen, O., and Schabel, J. (2010). Safety of interactive robotics : Learning from accidents. *International Journal of Social Robotics*, 2(3), pages 221–227.

Marchand, E., Rutten, E., Marchand, H., and Chaumette, F. (1998). Specifying and Verifying Active Vision-Based Robotic Systems with the SIGNAL Environment. *The International Journal of Robotics Research*, 17(4), pages 418–432.

Martin-Guillerez, D., Guiochet, J., Powell, D., and Zanon, C. (2010). UML-based method for risk analysis of human-robot interaction. In *2nd International Workshop on Software Engineering for Resilient Systems (SERENE2010), London, UK*. ACM.

Mekki-Mokhtar, A. (2012). *Processus d'identification de propriétés de sécurité-innocuité vérifiables en ligne pour des systèmes autonomes critiques*. PhD thesis, Université Paul Sabatier-Toulouse III.

Mekki-Mokhtar, A., Blanquart, J.-P., Guiochet, J., Powell, D., and Roy, M. (2012). Safety trigger conditions for critical autonomous systems. In *18th IEEE Pacific Rim International Symposium on Dependable Computing (PRDC 2012), Niigata, Japan*.

Mendoza, J. P., Veloso, M., and Simmons, R. (2012). Mobile robot fault detection based on redundant information statistics. In *Workshop at IROS'12 on "Safety in human-robot coexistence and interaction: How can standardization and research benefit from each other?", Vilamoura, Portugal*.

Menzies, T. and Pecheur, C. (2005). Verification and validation and artificial intelligence. *Advances in Computers*, 65, pages 153 – 201.

Micskei, Z., Szatmári, Z., Oláh, J., and Majzik, I. (2012). A concept for testing robustness and safety of the context-aware behaviour of autonomous systems. In *Conference on Agent and Multi-Agent Systems. Technologies and Applications (AMSTA)*, pages 504–513. Springer.

MIRAS (2009-2013). Multimodal Interactive Robot for Assistance in Strolling. Project supported by the French ANR (National Research Agency) under the TecSan (Healthcare Technologies) Program (ANR-08-TECS-009-04), www.miraswalker.com/index.php/en, accessed 2015-05-17.

Monterlo, M., Thrun, S., Dahlkamp, H., Stavens, D., and Strohband, S. (2006). Winning the DARPA Grand Challenge with an AI Robot. In *American Association of Artificial Intelligence 2006 (AAAI06)*. Boston, MA.

Moraes, R., Waeselynck, H., and Guiochet, J. (2014). Uml-based modeling of robustness testing. In *IEEE International Symposium on High Assurance Systems Engineering*, pages 168–175.

Morse (2015). www.openrobots.org/morse. Accessed July 2015.

Muscettola, N., Dorais, G. A., Fry, C., Levinson, R., and Plaunt, C. (2002). IDEA: Planning at the Core of Autonomous Reactive Agents. In *AIPS 2002 Workshop on On-line Planning and Scheduling*. Toulouse, France.

Muscettola, N., Nayak, P. P., Pell, B., and Williams, B. C. (1998). Remote Agent: To Boldly Go Where No AI System Has Gone Before. *Artificial Intelligence*, 103(1-2), pages 5–47.

Nesnas, I., Wright, A., Bajracharya, M., Simmons, R., Estlin, T., and Kim, W. S. (2003). CLARAty: An Architecture for Reusable Robotic Software. In *Proceedings of the SPIE Aerosense Conference on Unmanned Ground Vehicle Technology*. Orlando, Florida.

NREC (2015). National Robotic Engineering Center, Carnegie Mellon University. www.nrec.ri.cmu.edu/capabilities/safety_ops/. Accessed: 2015-07-30.

O'Brien, M., Arkin, R. C., Harrington, D., Lyons, D., and Jiang, S. (2014). Automatic verification of autonomous robot missions. In *International Conference on Simulation, Modeling, and Programming for Autonomous Robots (SIMPAR)*, pages 462–473. Springer.

OMG-ARM (2013). Structured assurance case metamodel (SACM), version 1. Object Management Group.

OMG-UML2 (2007). OMG unified modeling language (OMG UML), superstructure, v2.1.2. Object Management Group, formal/2007-11-02.

OMG-UML2 (2011). Unified Modeling Language (UML), Superstructure, V2.4.1, formal/2011-08-06. Object Management Group.

Orocos (2015). www.orocos.org. Accessed July 2015.

Pace, C. and Seward, D. (2000). A safety integrated architecture for an autonomous safety excavator. In *International Symposium on Automation and Robotics in Construction*.

Parasuraman, R., Sheridan, T., and Wickens, C. D. (2000). A model for types and levels of human interaction with automation. *IEEE Transactions on Systems, Man and Cybernetics, Part A: Systems and Humans*, 30(3), pages 286–297.

Pasqui, V., Saint Bauzel, L., Zong, C., Clady, X., Decq, P., Piette, F., Michel-Pellegrino, V., El Helou, A., Carre, M., Durand, A., Do Hoang, Q. A., Guiochet, J., Rumeau, P., Dupourque, V., and Caquas, J. (2012). Projet MIRAS: robot d'assistance à la déambulation avec interaction multimodale. *Ingénierie et Recherche BioMédicale (IRBM)*, 33(2), pages 165–172.

Pathak, S., Pulina, L., Metta, G., and Tacchella, A. (2013). Ensuring safety of policies learned by reinforcement: Reaching objects in the presence of obstacles with the icub. In *International Conference on Intelligent Robots and Systems (IROS)*, pages 170–175.

Pearl, J. (1988). *Probabilistic reasoning in intelligent systems: networks of plausible inference*. Morgan Kaufmann Publishers Inc. San Francisco, USA.

Pecheur, C. (2000). *Verification and Validation of Autonomy Software at NASA*. Technical report, NASA.

Penix, J., Pecheur, C., and Havelund, K. (1998). Using Model Checking to Validate AI Planner Domain Models. In *SEW*. Greenbelt, Maryland.

Pettersson, O. (2005). Execution monitoring in robotics: A survey. *Robotics and Autonomous Systems*, 53(2), pages 73 – 88.

PHRIENDS (2006-2009). Physical Human-Robot Interaction: Dependability and Safety. Project supported by the European Commission under the 6th Framework Programme (STReP IST-045359), www.phriends.eu. Accessed: 2015-04-30.

Pichon, F., Dubois, D., and Denœux, T. (2012). Relevance and truthfulness in information correction and fusion. *International Journal of Approximate Reasoning*, 53(2), pages 159 – 175.

Pollock, J. (2008). Defeasible reasoning. *Reasoning: Studies of Human Inference and Its Foundations*, pages 451–469.

Povse, B., Koritnik, D., Bajd, T., and Munih, M. (2010). Correlation between impact-energy density and pain intensity during robot-man collision. In *2010 3rd IEEE RAS and EMBS International Conference on Biomedical Robotics and Biomechatronics (BioRob)*, pages 179–183.

Powell, D., Arlat, J., Chu, H. N., Ingrand, F., and Killijian, M.-O. (2012). Testing the input timing robustness of real-time control software for autonomous systems. In *European Dependable Computing Conference (EDCC)*, pages 73–83.

Py, F. and Ingrand, F. (2004). Dependable execution control for autonomous robots. In *International Conference on Intelligent Robots and Systems (IROS)*, pages 1136–1141.

Quigley, M., Conley, K., Gerkey, B., Faust, J., Foote, T., Leibs, J., Wheeler, R., and Ng, A. Y. (2009). Ros: an open-source robot operating system. *International Conference on Robotics and Automation (ICRA), Workshop on open source software*, 3(3.2), pages 5–11.

R15.06-2012, A. (2012). American national standard for industrial robots and robot systems - safety requirements (revision of ansi/ria r15.06-1999).

R3COP (2015). http://www.r3-cop.eu/. Accessed September 2015.

Ramadge, P. J. and Wonham, W. M. (1987). Supervisory control of a class of discrete event processes. *Society for Industrial and Applied Mathematics (SIAM) journal on control and optimization*, 25(1), pages 206–230.

Randell, B. (1975). System Structure for Software Fault Tolerance. *IEEE Transactions on Software Engineering*, 1, pages 220–232.

ROBOSAFE (2013). Trustworthy Robotic Assistants. EPSRC-funded project, UK, www.robosafe.org/. Accessed: 2015-07-30.

ROBOT-PARTNER (2013-2016). Seamless Human-Robot Cooperation for Intelligent, Flexible and Safe Operations in the Assembly Factories of the Future. Project supported by the European Commission under the 7th Framework Programme, www.robo-partner.eu/. Accessed: 2015-04-30.

Robotics-VO (2013). *A Roadmap for U.S. Robotics – 2013 edition*. Technical report, Robotics Caucus Advisory Committee of the U.S. Congress.

Roderick, S., Roberts, B., Atkins, E., and Akin, D. (2004). The ranger robotic satellite servicer and its autonomous software-based safety system. *IEEE Intelligent Systems*, 19(5), pages 12–19.

ROS (2015). www.ros.org. Accessed July 2015.

ROSETTA (2009-2013). RObot control for Skilled ExecuTion of Tasks in natural interaction with humans; based on Autonomy, cumulative knowledge and learning. Project supported by the European Commission under the 7th Framework Programme, www.fp7rosetta.org/. Accessed: 2015-04-30.

Royakkers, L. and van Est, R. (2015). A literature review on new robotics: Automation from love to war. *International Journal of Social Robotics*, pages 1–22.

Rushby, J. (1989). Kernels for safety. pages 210–220.

Rutten, E. (2001). A framework for using discrete control synthesis in safe robotic programming and teleoperation. In *IEEE International Conference on Robotics and Automation (ICRA2011)*, volume 4, pages 4104–4109.

SAFROS (2009-2013). Patient Safety in Robotic Surgery. Project supported by the European Commission under the 7th Framework Programme, www.safros.eu/safros/. Accessed: 2015-04-30.

SAPHARI (2011-2015). Safe and Autonomous Physical Human-Aware Robot Interaction. Project supported by the European Commission under the 7th Framework Programme, www.saphari.eu, accessed 2015-05-17.

Schaefer, K. E. (2013). *The Perception and Measurement of Human-Robot Trust*. PhD thesis, University of Central Florida, College of Sciences, Orlando, US.

Scherer, S., Lerda, F., and Clarke, E. M. (2005). Model checking of robotic control systems. In *International Symposium on Artificial Intelligence, Robotics and Automation in Space (SAIRAS)*.

Schneider, S. A., Chen, V. W., Pardo-Castellote, G., and Wang, H. H. (1998). ControlShell: A Software Architecture for Complex Electromechanical Systems. *The International Journal of Robotics Research*, 17(4), pages 360–380.

SIMERO (2003). Safety strategies for human-robot cooperation. Partially funded by the German Research Foundation, www.ai3.uni-bayreuth.de/projects/simero/. Accessed: 2015-07-30.

Simmons, R., Pecheur, C., and Srinivasan, G. (2000). Towards automatic verification of autonomous systems. In *International Conference on Intelligent Robots and Systems (IROS)*, volume 2, pages 1410–1415.

SMOF (2015). Safety Monitoring Framework. LAAS-CNRS Project, https://www.laas.fr/projects/smof. Accessed 2015-07-01.

Srivatanakul, T. (2005). *Security Analysis with Deviational Techniques*. PhD thesis, University of York.

Steinbauer, G. (2013). A survey about faults of robots used in robocup. In *RoboCup 2012: Robot Soccer World Cup XVI*, pages 344–355. Springer.

Sulaman, S. M., Abbas, T., Wnuk, K., and Hö st, M. (2014). Hazard analysis of collision avoidance system using stpa. In *11th International Conference on Information Systems for Crisis Response and Management (ISCRAM)*.

Suwoong, L. and Yamada, Y. (2012). Risk assessment and functional safety analysis to design safety function of a human-cooperative robot. In *Human Machine Interaction - Getting Closer*, edited by M. Inaki. Intech.

Täubig, H., Frese, U., Hertzberg, C., Lüth, C., Mohr, S., Vorobev, E., and Walter, D. (2012). Guaranteeing functional safety: design for provability and computer-aided verification. *Journal Autonomous Robots*, 32(3), pages 303–331.

Thomas, J. and Leveson, N. G. (2011). Performing hazard analysis on complex, software and human-intensive systems. In *29th ISSC Conference about System Safety*.

Tiwari, A. and Sinha, P. (2003). Issues in v&v of autonomous and adaptive systems. In *Canadian Conference on Electrical and Computer Engineering*, volume 2, pages 1339–1342.

Tomatis, N., Terrien, G., Piguet, R., Burnier, D., Bouabdallah, S., Arras, K. O., and Siegwart, R. (2003a). Designing a Secure and Robust Mobile Interacting Robot for the Long Term. In *Proceedings of the 2003 IEEE International Conference on Robotics & Automation*, pages 4246–4251. Taipei, Taiwan.

Tomatis, N., Terrien, G., Piguet, R., Burnier, D., Bouabdallah, S., Arras, K. O., and Siegwart, R. (2003b). Designing a secure and robust mobile interacting robot for the long term. In *International Conference on Robotics and Automation (ICRA)*, pages 4246–4251.

Toulmin, S. (1958). The uses of argument. *Cambridge University Press*.

Ulrich, K. T., Tuttle, T. T., Donoghue, J. P., and Townsend, W. T. (1995). *Intrinsically Safer Robots*. Technical report, Barrett Technology Inc.

UR5-Robot (2015). *UR5 Technical specifications*. Technical report, Universal robots.

Urmson, C., Anhalt, J., Bae, H., Bagnell, J. A., Baker, C. R., Bittner, R. E., Brown, T., Clark, M. N., Darms, M., Demitrish, D., Dolan, J. M., Duggins, D., Ferguson , D., Galatali, T., Geyer, C. M., Gittleman, M., Harbaugh, S., Hebert, M., Howard, T., Kolski, S., Likhachev , M., Litkouhi, B., Kelly , A., McNaughton, M., Miller, N., Nickolaou, J., Peterson, K., Pilnick, B., Rajkumar, R., Rybski, P., Sadekar, V., Salesky, B., Seo, Y.-W., Singh, S., Snider, J. M., Struble, J. C., Stentz, A., Taylor , M., Whittaker, W. R. L., Wolkowicki, Z., Zhang, W., and Ziglar, J. (2008). Autonomous driving in urban environments: Boss and the urban challenge. *Journal of Field Robotics Special Issue on the 2007 DARPA Urban Challenge, Part I*, 25(8), pages 425–466.

Verifiable Robotics Research Group (2015). Sibley School of Mechanical and Aerospace Engineering, Cornell University. verifiablerobotics.com/. Accessed: 2015-07-30.

Visinsky, L., Cavallero, J., and Walker, I. (1994). Robotic fault detection and fault tolerance: A survey. *Reliability Engineering and System Safety*, 46, pages 139–158.

Volpe, R., Nesnas, I., Estlin, T., Mutz, D., Petras, R., and Das, H. (2000). *CLARAty: Coupled Layer Architecture for Robotic Autonomy*. Technical report D-19975, NASA - Jet Propulsion Laboratory.

Volpe, R., Nesnas, I., Estlin, T., Mutz, D., Petras, R., and Das, H. (2001). The CLARAty Architecture for Robotic Autonomy. In *Proceedings of the 2001 IEEE Aerospace Conference*. Big Sky, Montana. Citeseer.nj.nec.com/volpe01claraty.html.

Waeselynck, H., Micskei, Z., Rivière, N., Hamvas, A., and Nitu, I. (2010). Termos: A formal language for scenarios in mobile computing systems. In *MOBIQUITOUS 2010 : Mobile and Ubiquitous Systems: Computing, Networking, and Services*, volume 73 of *Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering*, pages 285–296. Springer Berlin Heidelberg.

Walker, I. and Cavallero, J. (1996). Failure mode analysis for a hazardous waste clean-up manipulator. *Reliability Engineering and System Safety*, 53, pages 277–290.

Wonham, W. M. (2005). Supervisory control of discrete event systems. URL www.control.utoronto.ca/DES/.

Woodman, R., Winfield, A. F., Harper, C., and Fraser, M. (2012). Building safer robots: Safety driven control. *Internatioanl Journal of Robotics Research*, 31(13), pages 1603–1626.

Yanco, H. and Drury, J. (2004). Classifying human-robot interaction: an updated taxonomy. In *IEEE International Conference on Systems, Man and Cybernetics, 2004*, volume 3, pages 2841–2846.

Zaman, S., Steinbauer, G., Maurer, J., Lepej, P., and Uran, S. (2013). An integrated model-based diagnosis and repair architecture for ROS-based robot systems. In *Robotics and Automation (ICRA), 2013 IEEE International Conference on*, pages 482–489.

Zendel, O., Herzner, W., and Murschitz, M. (2013). Vitro - model based vision testing for robustness. In *Robotics (ISR), 2013 44th International Symposium on*, pages 1–6.

Zhao, X., Zhang, D., Lu, M., and Zeng, F. (2012). A new approach to assessment of confidence in assurance cases. In *SAFECOMP Workshops*, pages 79–91.

Zinn, M., Khatib, O., Roth, B., and Salisbury, J. (2004). Playing it safe [human-friendly robots]. *Robotics Automation Magazine, IEEE*, 11(2), pages 12–21.

Zou, X., Alexander, R., and McDermid, J. (2014). Safety validation of sense and avoid algorithms using simulation and evolutionary search. In *International Conference on Computer Safety, Reliability, and Security (SafeComp)*, pages 33–48.