

Online Safety Monitoring Using Safety Modes

Jérémie Guiochet, David Powell and Étienne Baudin

Université de Toulouse

LAAS-CNRS

France

{name}@laas.fr

Jean Paul Blanquart

EADS Astrium

Toulouse, France

jean-paul.blanquart@astrium.eads.net

Abstract—Robotic systems have to carry out more and more complex tasks, including ones where humans can be endangered. Residual design faults in such systems, as well as the inevitability of physical faults and interaction faults during operation, motivate the use of safety monitors to prevent catastrophic failures. In this paper, we consider the design of such safety monitors for multi-functional robotic systems. We present an approach and a formalization of the process for determining safety rules. It consists in identifying *safety modes*, according to the different tasks carried out by the monitored system. In practice, each safety mode is related to one or several functional modes and is specified by a *permissiveness vector* that defines the authorized domains of variation of key physical variables. The set of safety modes can be partially ordered according to their authorization vectors and can thus be represented as a directed acyclic graph. This graph is used to automatically build a model representing safety modes and their transitions, which can be implemented in an independent safety monitor. A case study has been carried out on a mobile manipulator robot, working in a factory alongside humans.

Index Terms—Dependability, Safety, Online Monitoring, Robot, Autonomy

To carry out more and more complex tasks, robotic systems are being given increasing authority and autonomy. This raises major dependability concerns, particularly for systems operating in the presence of humans. Despite the use of fault removal and prevention techniques, it is impossible to guarantee avoidance of all development faults or, of course, of physical faults and interaction faults arising during deployment. Thus, it is necessary to design systems capable of fulfilling their mission (reliability) and avoiding catastrophic failures (safety) despite the presence of faults [1]. Following this logic of accepting the inevitability of faults, we propose the use of an independent subsystem to carry out online verification of global safety properties in order to provide end-to-end protection against faults activated or occurring at run-time. In this paper, we

will refer to the independent subsystem as the *safety monitor* [2], and to the functional system as the *monitored system*. Such safety monitors exist in many critical application domains: transportation [3], space [4], medical systems [5], [6], civil engineering [7], nuclear power plants [8], and multi-purpose robotics [9]. Many expressions are used to denote such a safety monitor: *Monitoring and Safing Unit* [10], *Protection System* [11], *Safety Manager* [7], *Checker* [9], [12], *Safety Bag* [3] *Guardian Agent* [6]. The safety monitor obtains information about the state of the monitored system and the environment either by directly reading the values of key variables maintained by the monitored system or by using dedicated additional sensors. It decides whether the observed situation is safe or not according to a set of monitoring rules. If a hazardous situation is detected, the safety monitor triggers a forward recovery procedure to put the monitored system in a safe state.

Most of the literature on safety monitors focusses on system architecture and on how assertion-checking can be integrated into the architecture [13]. But none we are aware of addresses the process leading to the identification, definition and expression of the safety assertions to be checked. Moreover, for the above cited systems, safety rules are checked continuously, without taking account the fact that systems can perform diverse tasks where the appropriate safety rules can change according to the tasks be carried out. This is particularly important with multi-functional robots, especially those including autonomous abilities.

The contribution presented in this paper aims to overcome these issues. We propose a structured approach based on the concept of *safety modes*. A systematic and formalized approach is proposed to improve the safety rule identification process, and to partially automate the production of safety rules. The use of modes integrates the fact that systems can carry out tasks with different safety rules. For each functional mode of the monitored

system, the safety monitor activates the corresponding safety mode, and checks a specific set of monitoring rules. An additional aspect of the proposed method is that it helps designers to specify reaction strategies that are more flexible than emergency stop, which leads to improved availability and efficiency.

In Section I of this paper, we formalize the concept of safety modes and define two types of safety rules: permissiveness rules and context rules. Section II discusses the management of safety modes and, in particular, the transitions between safety modes. Section III develops our preliminary case study on a mobile manipulator robot, working in a factory alongside humans. It demonstrates that safety modes do simplify the specification process, and that most hazardous situations can be handled by reaching a safety mode in which the monitored system remains partially functional. Some examples coming from this case study are used throughout the paper to illustrate theoretical and formal aspects. Finally, Section IV discusses lessons learnt and future work that needs to be carried out.

I. SAFETY MODES

The objective of safety modes is to describe the dynamics of the safety monitor, identifying different sets of safety rules activated according to the current tasks of the system. We propose a formal notation for mode specification, and also a partial order relation between modes that facilitates the production of safety rules.

A. Background

In the literature, the term *mode* is mainly used to describe different configurations of a system, in which different control laws are applied [14]. In particular, modes allow discrete changes from one control law to another. Relatively few works link the notions of safety rules and functional modes of operation.

In [15], four functional modes of a manipulator robot arm are defined from the combinations of two discrete variables, each with two possible values: the maximum speed of the arm (slow or fast), and the choice of motion control (automatic or guided by a human). The control laws are different in each mode. Some transitions between modes require the introduction of an intermediate mode, in order to respect safety criteria. In particular, the transitions from the automatic modes to the manual modes require an intermediate mode in which the arm is stopped. In [16], three modes are used to manage, within a *safety manager*, the in-flight

testing of an experimental neural network for fly-by-wire flight control. The three modes are: *nominal* (with conventional flight control), *research* (neural network flight control), and *failure* (research mode with injected faults). When the pilot pushes a button in the nominal mode, the research mode is engaged if the necessary conditions (about flight parameters, hardware, software, communication buses) are fulfilled. If any of these conditions does not hold, the safety manager automatically returns to the nominal mode. From the research mode, the pilot can engage the failure mode. The monitoring rule set of this mode is a superset of the research mode rule set with two additional rules.

Those two approaches illustrate that in many systems, operational and safety modes are tightly linked. Our approach is similar in that it consists in defining some discrete modes, in which is applied a specific set of safety rules that aims to prevent the system from reaching an unsafe state. The main difference is in that we introduce safety modes as a first-class concept distinct from functional aspects and propose a general method for specifying them.

B. Definitions

We define a *safety mode* as a state of the safety monitor, in which a specific set of monitoring rules is applied. The safety monitor obtains information about the monitored system and its environmental context, determines the current safety mode, and checks if the conditions of a safe execution are fulfilled. Functional mode changes that do not impact safety do not lead to a change in safety mode. Hence, there is a one to many binary association between safety modes and functional modes.

For each safety mode and each transition between safety modes, the safety rules that are checked may be different. We distinguish two types of rules. The first type are *permissiveness rules*, that define the functional capabilities allowed for the current safety mode. This type of rule checks that some functional variables do not violate the domains of variation authorized in the current safety mode. For example, speed ranges $[0, 1] m.s^{-1}$ and $[0, 2] m.s^{-1}$ for a mobile robot are two possible authorized domains. However, permissiveness rules do not deal with context or environment. The second type of rules, *context rules*, are intended for monitoring the system with respect to hazardous situations, either related to the environmental conditions, or the state of critical resources.

C. Permissiveness rules

Safety modes are defined on the basis of safety-relevant functional variables that have ranges of authorized variations. We define the *authorization variable* A_f , associated to a functional variable f , to denote the *authorized domain* of variation of f . A_f may take on different values $A_f^{(i)}$ at different times, thus leading to a variable constraint on the functional variable f . A domain that can be authorized is called an *admissible domain*. Each admissible domain of f is a member of the powerset (set of all possible subsets) of the domain of f , noted $\mathcal{P}(\text{dom}(f))$. The set $\mathcal{A}_f = \{A_f^{(1)}, \dots, A_f^{(k)}\}$ of admissible domains of f is thus a family of sets over $\text{dom}(f)$.

Definition 1: The **set of admissible domains** for a functional variable f is defined as $\mathcal{A}_f = \{A_f^{(1)}, \dots, A_f^{(m)}\}$, where $\forall i, A_f^{(i)} \in \mathcal{P}(\text{dom}(f))$

A property of a family of sets is that it is partially ordered under the inclusion relation, so the elements of \mathcal{A}_f can be represented as a directed acyclic graph, which we call the *permissiveness graph*. Two nodes $A_f^{(i)}, A_f^{(j)}$ in the permissiveness graph are linked by a directed arc (from $A_f^{(i)}$ to $A_f^{(j)}$) if $A_f^{(i)}$ includes $A_f^{(j)}$ as a sub-domain. In this case, we say that $A_f^{(i)}$ is more *permissive* than $A_f^{(j)}$, since it allows a wider variation of safety-relevant functional variable f . Conversely, $A_f^{(j)}$ is said to be more *restrictive* than $A_f^{(i)}$.

Definition 2: An admissible domain $A_f^{(i)}$ is more **permissive** (or, less **restrictive**) than $A_f^{(j)}$ if $A_f^{(i)} \supseteq A_f^{(j)}$.

Paths on the permissiveness graph represent possible reaction strategies in the face of detected hazardous situations, under the premise that more restricted domains of safety-relevant functional variables are safer than more permissive ones. To allow feasible changes of a functional variable, the permissiveness graph must be weakly connected. Thus, if two admissible domains are not ordered by the inclusion relation, they must either include, or be included by, another admissible domain.

Definition 3: A set of admissible domains for f is said to be **feasible** if

$$\forall i, j, \left(A_f^{(i)} \not\subseteq A_f^{(j)} \right) \wedge \left(A_f^{(j)} \not\subseteq A_f^{(i)} \right) \\ \Rightarrow \exists k, \left(A_f^{(k)} \supseteq A_f^{(i)} \cup A_f^{(j)} \right) \vee \left(A_f^{(k)} \subseteq A_f^{(i)} \cap A_f^{(j)} \right)$$

Let us consider a few simple examples. Let *BaseSpeed* (type `real`) be the speed of the base of a mobile robot, such that $\text{dom}(\text{BaseSpeed}) \subseteq \mathbb{R}^+$. Let us consider three admissible domains (in this case, intervals) for *BaseSpeed*:

$$A_{\text{BaseSpeed}} = \begin{cases} A_{\text{BaseSpeed}}^{(1)} = [0, 0]m.s^{-1} \\ A_{\text{BaseSpeed}}^{(2)} = [0, 1]m.s^{-1} \\ A_{\text{BaseSpeed}}^{(3)} = [0, 2]m.s^{-1} \end{cases}$$

The first possible value of the authorization variable $A_{\text{BaseSpeed}}$ defines a constraint of no movement ($\text{BaseSpeed} = 0$), whereas the other two define two different upper speed limits. In this case, the admissible domains form a totally-ordered set: $A_{\text{BaseSpeed}}^{(3)} \supset A_{\text{BaseSpeed}}^{(2)} \supset A_{\text{BaseSpeed}}^{(1)}$, ranging from the most permissive to the most restrictive constraint on *BaseSpeed*.

As a second example, consider a functional variable *GripperState* (type `enum`) representing the state of a gripper at the tip of a robot arm, such that $\text{dom}(\text{GripperState}) = \{\text{Open}, \text{Closed}\}$. We can consider three admissible domains for *GripperState*:

$$A_{\text{GripperState}} = \begin{cases} A_{\text{GripperState}}^{(1)} = \{\text{Open}\} \\ A_{\text{GripperState}}^{(2)} = \{\text{Closed}\} \\ A_{\text{GripperState}}^{(3)} = \{\text{Open}, \text{Closed}\} \end{cases}$$

The first and second values of $A_{\text{GripperState}}$ denote obligatory positions of the the gripper, whereas the third value indicates that both positions are authorized. Here, $A_{\text{GripperState}}$ is only partially-ordered with $A_{\text{GripperState}}^{(3)}$ representing the most permissive domain and $A_{\text{GripperState}}^{(1)}$ and $A_{\text{GripperState}}^{(2)}$ being alternative less permissive domains.

Alternatively, authorized gripper actions might be designated by means of an authorization variable $A_{\text{GripperTransition}}$ pertaining to a functional variable *GripperTransition* representing transition events between gripper states where a possible set of values for $A_{\text{GripperTransition}}$ might be:

$$A_{\text{GripperTransition}} = \begin{cases} A_{\text{GripperTransition}}^{(1)} = \emptyset \\ A_{\text{GripperTransition}}^{(2)} = \{\text{open}, \text{close}\} \end{cases}$$

In this third example, the first value of $A_{\text{GripperTransition}}$ forbids any change in state of the gripper, whereas the second value allows both possible transitions. The domain $A_{\text{GripperTransition}}^{(2)}$ is evidently more permissive than $A_{\text{GripperTransition}}^{(1)}$.

All the three examples lead to sets of admissible domains that can be represented as weakly-connected

directed acyclic graphs, and are thus feasible sets in the sense of Definition 3.

The notion of a set of admissible domains for a single safety-relevant functional variable f can be generalized to consider *vectors* of n safety-relevant variables, $\vec{f} = (f_1, \dots, f_n)$ such that $\mathcal{A}_{\vec{f}} = \{A_{\vec{f}}^{(1)}, \dots, A_{\vec{f}}^{(m)}\}$ denotes a set of m admissible domains for \vec{f} . $\mathcal{A}_{\vec{f}}$ is a subset of the Cartesian product of the sets of admissible domains for each element of \vec{f} : $\mathcal{A}_{\vec{f}} \subseteq \mathcal{A}_{f_1} \times \dots \times \mathcal{A}_{f_n}$.

We define \vec{F} to be the vector of *all* the safety-relevant variables of a given system S . Each admissible domain $A_{\vec{F}}^{(i)}$ defines a *safety mode* m_i of system S .

Definition 4: A **safety mode** m_i of a system S with a vector of safety-relevant functional variables $\vec{F} = (f_1, \dots, f_{|\vec{F}|})$ is defined by an affectation of authorized domains for each element of \vec{F} , $A_{\vec{F}}^{(i)} = (A_{f_1}^{(i)}, \dots, A_{f_{|\vec{F}|}}^{(i)})$. We call $A_{\vec{F}}^{(i)}$ the **permissiveness vector** associated with mode m_i .

Safety modes can be partially ordered by permissiveness by direct extension of Definition 2, applied to the modes' permissiveness vectors.

Definition 5: A safety mode m_i is more **permissive** (respectively, less **restrictive**) than m_j if $A_{\vec{F}}^{(i)} \supseteq A_{\vec{F}}^{(j)}$, i.e., if $\forall k \in [1, |\vec{F}|], A_{f_k}^{(i)} \supseteq A_{f_k}^{(j)}$. We use the notation $m_i \succ m_j$ to denote the permissiveness relation between safety modes.

The permissiveness vector of a given safety mode defines the domains that must be respected in that mode by the set of safety-relevant functional variables of the system. Formally, we express this as the *permissiveness rule* associated with the safety mode:

Definition 6: Permissiveness rule: in safety mode m_i with associated permissiveness vector $A_{\vec{F}}^{(i)}$, the permissiveness rule is defined as the boolean function: $P(m_i) = (\forall k \in [1, |\vec{F}|], f_k \in A_{f_k}^{(i)})$

The notion of a set of *feasible safety modes* follows by direct extension of Definition 3.

Definition 7: A set of safety modes for system S is said to be **feasible** if:

$$\forall i, j, \left(A_{\vec{F}}^{(i)} \not\subseteq A_{\vec{F}}^{(j)} \right) \wedge \left(A_{\vec{F}}^{(i)} \not\supseteq A_{\vec{F}}^{(j)} \right) \\ \Rightarrow \exists k, \left(A_{\vec{F}}^{(k)} \supseteq A_{\vec{F}}^{(i)} \cup A_{\vec{F}}^{(j)} \right) \vee \left(A_{\vec{F}}^{(k)} \subseteq A_{\vec{F}}^{(i)} \cap A_{\vec{F}}^{(j)} \right)$$

An admissible set of safety modes is one in which it is possible for the system to change safety modes without necessarily falsifying a permissiveness rule, i.e., there are no discontinuities in the admissible domains of the system's safety-relevant functional variables.

As a simple example, consider a mobile robot equipped with a gripper on a manipulator arm. We consider four safety-relevant functional variables: *BaseSpeed*, *GripperTransition* (defined as previously), *ArmStatus* (type enum) where $\text{dom}(\text{GripperTransition}) = \{\text{folded}, \text{unfolded}\}$, and *ArmForce* (type real) where $\text{dom}(\text{BaseSpeed}) = \mathbb{R}^+$.

We wish to define the following six safety modes for this system:

- *FastMove*: the robot can move at maximum speed, assuming that no human beings are in its vicinity. While moving fast, its arm must be in the folded position.
- *SlowMove*: the robot can move at reduced speed, even if there are human beings in its vicinity.
- *FastWork*: the robot can use the full functionality of its manipulator arm, as long as its base is stationary and there are no human beings in its vicinity.
- *CollaborativeWork*: the robot can use its arm at reduced speed in the vicinity of or in collaboration with a human being, as long as its base is stationary.
- *MoveAndWork*: the robot can use its arm at reduced speed while moving, as long as no human beings are in its vicinity and any load it manipulates is not dangerous.
- *Stop*: the base and the arm of the robot are stationary; any load in the gripper must not be dropped.

Table I defines the permissiveness vectors over the four safety-relevant variables defined previously.

Figure 1 shows the permissiveness graph that can be generated automatically from the set of safety modes defined in Table I. It can be seen that the set of safety modes is *feasible* (the permissiveness oriented graph is weakly connected) and that safety mode *Stop* is the most restrictive safety mode (it appears as a sink node on the graph).

D. Context rules

As previously presented, permissiveness rules do not include external conditions, and particularly hazardous situations induced by non-controllable variables. We

TABLE I
EXAMPLE OF SAFETY MODES AND ASSOCIATED PERMISSIVENESS VECTORS

Safety-related functional variables	Safety Modes					
	FastMove	SlowMove	Stop	CollaborativeWork	FastWork	Move&Work
BaseSpeed	[0,2]	[0,1]	[0,0]	[0,0]	[0,0]	[0,1]
ArmStatus	{folded}	{folded}	{folded}	{folded,unfolded}	{folded,unfolded}	{folded,unfolded}
ArmForce	[0,10]	[0,10]	[0,10]	[0,50]	[0,120]	[0,120]
GripperTransition	{∅}	{∅}	{∅}	{open, close}	{open, close}	{open, close}

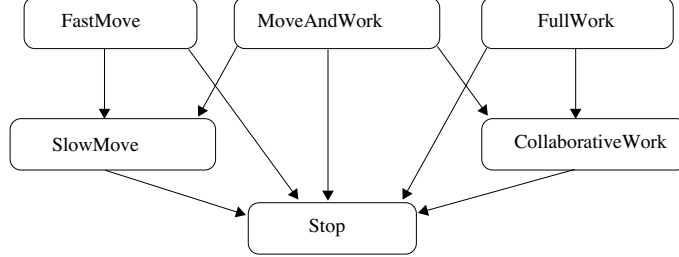


Fig. 1. Graph of safety modes with permissiveness relation order (a → b: a is more permissive than b)

introduce the notion of *context rules* to define external conditions that must be respected to ensure safety. For instance, the presence of a human in the robot's vicinity can be considered as a *context variable* that could be used in such a context rule. Context rules can be defined using the same formal notation as for permissiveness rules. For instance, let us consider the context variable *HumanDistance*, which is the distance between the robot and the closest human. This information is safety-relevant for instance in the *FastMove* mode of the previous example where the robot can reach speeds that could cause injury in case of collision. Context rules for different safety modes can be defined in terms of different *HumanDistance* ranges such as $[0, +\infty]$, $[50, +\infty]$, and $[200, +\infty]$ cm. For example, the context rule for the *FastMove* mode in Figure 1 would be $C(\text{FastMove}) = (\text{HumanDistance} \in A_{\text{HumanDistance}}^{(3)})$, where $A_{\text{HumanDistance}}^{(3)} = [200, +\infty]$, i.e., we specify that in the safety mode *FastMove* the distance to any humans cannot be under 200 cm. In this example, only one context variable is used in the rule, but in general, the context rule could include constraints on many other context variables (e.g., natural light intensity or cluttering level of environment, which both have a high impact on the robot's abilities to sense its environment and plan a safe trajectory).

Formally, we define $\vec{G} = (g_1, \dots, g_{|\vec{G}|})$ to be the vector of safety-relevant context variables related to the environment of system S .

Definition 8: The **set of admissible domains** for a context variable g_k is defined as $\mathcal{A}_{g_k} = \{A_{g_k}^{(1)}, \dots, A_{g_k}^{(m)}\}$, where $\forall i, A_{g_k}^{(i)} \in \mathcal{P}(\text{dom}(g_k))$

Definition 9: A **context vector** for safety mode m_i is an affectation of authorized domains for each element of \vec{G} , $A_{\vec{G}}^{(i)} = (A_{g_1}^{(i)}, \dots, A_{g_{|\vec{G}|}}^{(i)})$.

Definition 10: Context rule: in safety mode m_i , it must be the case that $C(m_i)$ is true, with $C(m_i) = \forall k \in [1, |\vec{G}|], g_k \in A_{g_k}^{(i)}$

E. Safety mode automaton

Permissiveness and context rules are used to detect if the system is entering a hazardous state. These rules can be used both for checking conditions that must be maintained in a safety mode (*mode conditions*) and for checking conditions that must be fulfilled to allow transitions between safety modes (*guard conditions*). Mode conditions and guard conditions are obtained from the permissiveness and context vectors of the safety modes. Practically, the guard condition on a transition between safety modes m_1 and m_2 depends on the permissiveness relation between m_1 and m_2 (defined in Definition 5) which implies three types of transition (an example is presented in Figure 2):

- 1) To a more permissive safety mode: $m_2 \succ m_1$. The monitored system increases its functional abilities, but some contextual conditions have to be fulfilled (e.g., absence of humans). According

to the permissiveness relation, $A_{\bar{F}}^{(2)} \supseteq A_{\bar{F}}^{(1)}$, thus, reaching m_2 from m_1 , the $P(m_1)$ rule still remains *true*. In that case, only the $C(m_2)$ rule has to be checked for this transition (consider, for instance, a transition from *SlowMove* to *FastMove* in Figure 1).

- 2) To a less permissive safety mode: $m_2 \prec m_1$. As functional abilities decrease the permissiveness rule of the targeted mode (m_2) must be checked. For example, if the monitored system has to enter the *Stop* mode, it has to stop its base and arm before the acceptance of the transition by the safety monitor.
- 3) To an incomparable safety mode: $m_2 \not\prec m_1$ and $m_2 \not\supseteq m_1$. In this case the guard condition is defined by identifying a path in the permissiveness graph, passing through less permissive intermediate modes. The resulting guard condition is a logical *and* of all guard conditions along the path to the final mode.

With these three types of transition, all the transition conditions can be determined. However, all the transitions are not functionally interesting. For that reason, some transitions may be manually specified as forbidden. In Figure 2, transitions to more or less permissive safety modes are represented by solid arrows, whereas transitions between incomparable safety modes are represented by dashed arrows. An illustration is the case of transition from *CollaborativeWork* to *SlowMove* mode. Here, the intermediate mode *Stop* is used to evaluate the final guard condition: $P(\textit{Stop}) \wedge C(\textit{SlowMove})$. The transitions that are not represented are forbidden.

To ease readability, the request of the monitored system $\textit{changeMode}(m_i)$ on each transition condition is not represented on the automaton of Figure 2. For example, using statechart notation, the transition from *Stop* to *SlowMove* should be annotated with: $\textit{changeMode}(\textit{SlowMove})[P(\textit{SlowMove})]$, i.e., a transition is activated on event $\textit{changeMode}(\textit{SlowMove})$, guarded by the condition $P(\textit{SlowMove})$.

II. FROM SAFETY MODES TO SAFETY RULES

Once safety modes, permissiveness and context vectors have been specified, it is necessary to analyze how the safety monitor will react if the induced conditions are not fulfilled. Whereas previous sections can be applied in a generic way, this section is highly linked with the design of the system, because it requires

knowledge about which reaction strategies are possible. However, we present in this section some guidelines to implement safety modes and safety rules.

A. Activation of the safety modes

Since the safety rules to be ensured by the safety monitor depend on the current safety mode, one fundamental issue remains the identification of the current safety mode. This can be done by the observation of the physical attributes of the monitored system to deduce the corresponding safety mode. This approach has a major disadvantage: in some cases the safety monitor cannot identify the current safety mode. This issue is studied in the diagnosis community, where graph algorithms are used to identify system state given a set of observation variables. Of course our approach can integrate this approach, but we decided to first focus on the monitoring of safety rules, and to simplify the approach, we make some assumptions. First, we assume that the monitored system has been designed with a set of functional modes, linked with a many-to-one relation to the safety modes. Second, we have chosen to be notified by the monitored system about its mode change requests. This assumption avoids any ambiguity. The drawback of this solution is the required confidence given to the monitored system, which may send erroneous information and particularly wrong mode change requests. In that case, it should be demonstrated that in any case of mismatch, the safety monitor will put the system in a safe state. This may lead to a lower availability but guarantees a higher safety.

B. Mode condition violation

In every safety mode, the monitor should be able to detect if there is a violation of the corresponding mode conditions, of type P or C. When it is not possible for the system to ensure both, the system cannot remain in the same safety mode, so the safety monitor must force a transition towards a safe state, i.e., a less permissive safety mode. To do this, we propose the concept of a *fall-back mode*, in which actions are undertaken to reach conditions of a less permissive safety mode (for example, a fall-back mode might correspond to activation of emergency braking). A limit can be set on the time spent in the fall-back mode. If that time limit is exceeded, the safety monitor attempts to force a transition towards an even less permissive mode, e.g., a *Stop* mode. As a last resort, the safety monitor should put the monitored system in

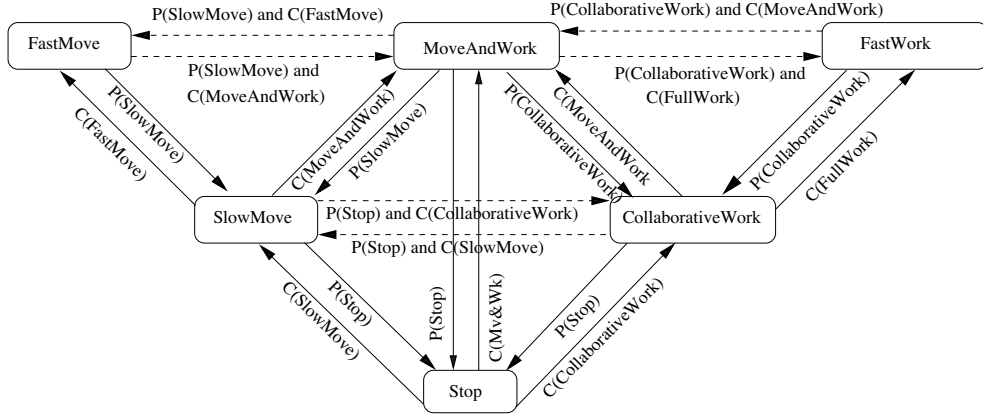


Fig. 2. Generated safety modes automaton (only guard conditions are shown as transition labels)

an ultimate fall-back mode, such as *EmergencyStop*, from which it may not be possible to recover, but which guarantees safety. Many levels of fall-back modes can be considered, but particular attention should be given to reaction time constraints (stopping the robot has to be fast in case of a hazardous situation).

The statechart in Figure 3 is an example. Only three safety modes are represented (*FastMove*, *SlowMove*, and *Stop*), and three fall-back modes are introduced (*ControlledMovement*, *ControlledStop*, *EmergencyStop*). As a first reaction, the safety monitor tries to control the speed. If it is not possible to reach the *SlowMove* mode, the *ControlledStop* mode is activated. As a last resort, *EmergencyStop* is activated.

A protocol must be defined to allow communication between the monitored system and the safety monitor. Indeed, in case of activation of a fall-back mode, the actions initiated by the safety monitor must be taken into account by the monitored system. To ensure full independence between the safety monitor and the monitored system, actions such as *reduceSpeed()* (it is not specified here how speed can be reduced), should be engaged by the safety monitor and the monitored system should be aware of this action to integrate it in its motion planning for instance.

C. Transition condition violation

We now analyze potential violations of a guard condition. As before, three types of transition are identified: a transition to a more permissive mode, to a less permissive mode, and to a mode that is not comparable. Indeed, in case of a *changeMode()* request, the guard condition can include a permissiveness rule (P rule), a

context rule (C rule), or both, and the reaction of the safety monitor is described in a generic way.

1) *Transition towards a more permissive safety mode (C rule)*: A transition towards a more permissive mode is allowed if the environment has changed in such a way that it respects more restrictive contextual conditions (for instance there are no humans, no hazardous obstacles). If C rules are not fulfilled, the safety monitor should keep the system in the same mode, and reject the request for changing the mode. This implies that the monitored system does not switch to its desired functional mode, and can integrate this rejection in its future plans. Again, as for mode conditions (previous section), a protocol needs to be defined for the monitored system to receive and react to mode change requests rejected by the safety monitor.

2) *Transition towards a less permissive safety mode (P rule)*: Switching to a less permissive mode is guarded by P rules, i.e., functional variables need to be restricted (for instance, speed has to be reduced) to satisfy the P rules before mode switching is allowed. If those conditions are not verified, this means that the monitored system wants to reach such a mode, but is unable to do so. Then, the safety monitor has to react to impose the conditions mandated by the more constrained environment. This is again done through the notion of fall-back modes, as described in Section II-B. For example, consider the transition from *fastMove* to *SlowMove* on Figure 3. If the guard condition $P[SlowMove]$ is not fulfilled when the mode change is requested, the safety monitor can engage the *ControlledMovement* fall-back mode previously defined for handling the violation of the mode condition of *FastMove*. For example, considering the

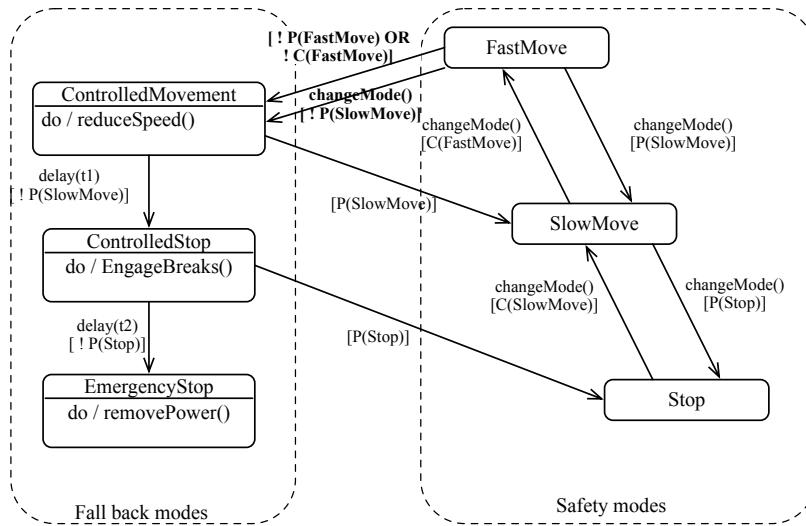


Fig. 3. Statechart representation of safety modes and fall-back modes

transition from *FastMove* to *SlowMove* on Figure 3. If the guard condition $P(\text{SlowMove})$ is not fulfilled when the mode change is requested the safety monitor can engage the *ControlledMovement* fall-back mode previously defined for handling a violation of the mode condition of *FastMove*

3) *Transition towards a non comparable safety mode (P and C rules)*: In the third case, a distinction should be made between P and C rules. Indeed, if a C rule is not verified, this means that the environment conditions do not fulfill the requirement, so the system should stay in its current mode (as previously presented). If a P rule is not satisfied, then the safety monitor will take over to force the monitored system through one or several fall-back modes until a more restrictive P rule is fulfilled.

III. APPLICATION TO A MOBILE MANIPULATOR ROBOT

This section presents an application of the proposed method on a paper case study based on a use case defined in the PHRIENDS project¹. The system and its environment are first defined, then the method is applied.

A. Definition of the system and its environment

The considered system is a mobile robot with a manipulator arm. The user can order the robot to pick up a specific object from a specific location and then to carry it to another location and to place it there,

¹PHRIENDS (Physical Human Robot Interaction: Dependability and Safety) is a project supported by the European Community under the 6th Framework Program, <http://www.phriends.eu>

or give it to him. Considering these tasks, we assume a high-level of interaction between the robot and the human. First, they both work in the same work space, which is often avoided: robots are usually enclosed in a specific area, or must follow a dedicated trajectory. Second, collaborative work is possible: the robot is able to take an object from the hand of a human, and the user can physically stop the robot during a task by catching any part of the robot arm. The considered environments are workshops or factories. Tasks can be summed up with the following UML use cases where the robot can:

- move to a location (holding or not a load),
- give an object to the user,
- take an object from the user's hand,
- place an object in a specified location,
- take an object from a specified location,

and the user can:

- physically guide the robot arm to a location,
- pause and resume a task by physically stopping and releasing the arm,
- abort a task by physically stopping the arm.

In its first version, the chosen application will not be able to work with the arm while moving the base. The arm for this application is the LWR (LightWeight Robot) developed by the DLR (German Aerospace Center) and built by KUKA, which is a seven degree of freedom arm composed of torque and motor position sensors [17]. In this paper and in [18], authors present different control laws, and more particularly, the ones that have been implemented for this arm in the low-level controller: position control, low impedance con-

trol and zero gravity control. The first one is a classical robotic control law, whereas the second one is based on a position-force law allowing the system to propose a variable stiffness around a fixed position. The latter control law compensates on each joint the effect of gravity so that a human can guide the robot as if it had no weight. The mobile base will be considered as a wheel base platform, able to navigate in an area where there are other mobile objects such as humans (as described in [19]).

B. Application of the method and results

The first step is to identify safety modes and safety-relevant functional variables considering all scenarios. We use UML sequence diagrams [20] as in Figure II. In a sequence diagram, interactions can be represented with messages as well as activities (for the robotic system for instance). This diagram can be used at a very first step of a project with a low level of detail. Only one scenario is presented here.

Safety modes are identified by a cross analysis of the possible deviations based on this diagram such as in [21], and of a preliminary risk analysis identifying hazardous situations. Figure 4 presents the main human robot interactions and robot activities during execution of the use case “take an object from a specified location”. In the presented scenario, the human gives an order to the robot to pick up an object which is at a specified location. Design choices are not presented on this diagram (such as the means to locate the position), but the level of description is sufficient to find the safety modes. During this scenario, a user wants to interrupt the task by physically stopping the arm (catching any part of the robot arm). Three actions are then possible, the user can: physically guide the robot arm to a location, abort the robot task, or pause the task and resume it when he wants. In our case, the user chooses to physically guide the robot arm to a location (which can be different from the first one). Finally, the robot holds the object with the gripper and moves the arm in the transportation position (arm folded is required for moving the mobile base).

For each step of the sequence diagram, we identify the corresponding safety modes which are presented on the right side of figure 4. Identification of the safety-relevant functional variables is based on this diagram, but also on a preliminary risk analysis. Eight such variables have been identified: *BaseSpeed*, *BaseAcceleration*, *BaseForce*, *ArmSpeed*, *ArmAcceleration*, *ArmForce*,

GripperTransition, *ArmStatus*. Permissiveness vectors are then associated with each safety mode by defining authorized domains for each these variables, as presented in the Table II (only a subset of variables is presented here).

Authorized domains for speed and acceleration of the robot base are obtained from expert studies (for instance the speed is limited at $0.25m/s$ in standard [22], but in [23], the authors show that up to $2m/s$ the LWR robot cannot provoke any damage). We use here k_1 and k_2 as proportionally factors to limit the speed of the robot, where $k_2 > k_1 > 0$, and $dist$ is the difference between the real distance to the closest human and a constant safety distance.

ArmForce and *BaseForce* are two variables related the torque of the joints and the wheel motors. For *ArmForce*, we keep the standard recommendation which is a maximum force of $125N$. The appropriate domain for *BaseForce* is the subject of future work.

For boolean or enumerated variables, we use the same notation as presented previously in Section I-C. For *ArmStatus*, values are *folded* and *unfolded*. Two admissible domains for *ArmStatus* are defined: $A_{ArmStatus}^{(1)} = \{folded\}$, which means that the arm has to be folded, and $A_{ArmStatus}^{(2)} = \{folded, unfolded\}$ for when the arm is free to move between folded and unfolded positions. As previously presented, $A_{GripperTransition}$ has two values: $\{\emptyset\}$ and $\{open, close\}$, which mean respectively that any transition of gripper state is forbidden, or that both transitions are allowed.

As presented in Table III, two safety-relevant context variables have been identified to detect hazardous situations. The first one is the safe distance with respect to the closest human, *HumanDistance*, which has currently been fixed at $2m$ in *FastMove* safety mode, and $0.5m$ in *SlowMove* safety mode. The second one, is the hazardous nature of the load, which impacts the functional abilities of the robot (for instance, holding a hazardous load will constraint the mobile base and the arm to move slowly).

The resulting permissiveness graph is the same as that in Figure 1, without the mode *MoveAndWork*, and is then used to build the mode automaton in Figure 5. As previously presented, this automaton is composed of transitions with plain lines which are directly derived from the permissiveness and context vectors, and transitions with dotted lines for induced transition conditions. Once this automaton is built, the next objective is to build the final automaton showing transitions



Fig. 4. Sequence diagram of the use case “take an object from a specified location”

TABLE II
CASE STUDY SAFETY MODES AND ASSOCIATED PERMISSIVENESS VECTORS

Safety-related functional variables	Safety Modes				
	FastMove	SlowMove	Stop	CollaborativeWork	FastWork
BaseSpeed	$[0, k_2 * dist]$	$[0, k_1 * dist]$	$[0, 0]$	$[0, 0]$	$[0, 0]$
ArmStatus	{folded}	{folded}	{folded}	{folded, unfolded}	{folded, unfolded}
ArmForce	$[0, 10]$	$[0, 10]$	$[0, 10]$	$[0, 50]$	$[0, 120]$
GripperTransition	$\{\emptyset\}$	$\{\emptyset\}$	$\{\emptyset\}$	{open, close}	{open, close}

TABLE III
CASE STUDY SAFETY MODES AND ASSOCIATED CONTEXT VECTORS

Safety related context variables	Safety Modes				
	FastMove	SlowMove	Stop	CollaborativeWork	FastWork
HazardousLoad	{False}	{False}	{True, False}	{True, False}	{False}
HumanDistance	$[2, +\infty]$	$[0.5, +\infty]$	$[0, +\infty]$	$[0, +\infty]$	$[0, +\infty]$

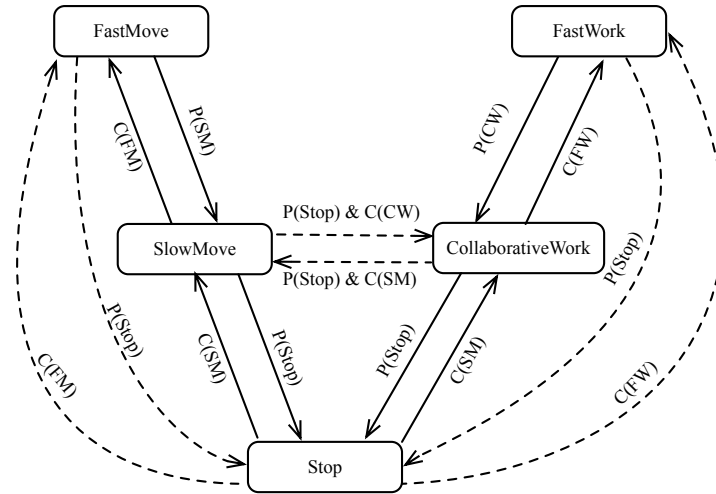


Fig. 5. Safety mode automaton without *changeMode()* request events (only guard conditions are shown as transition labels)

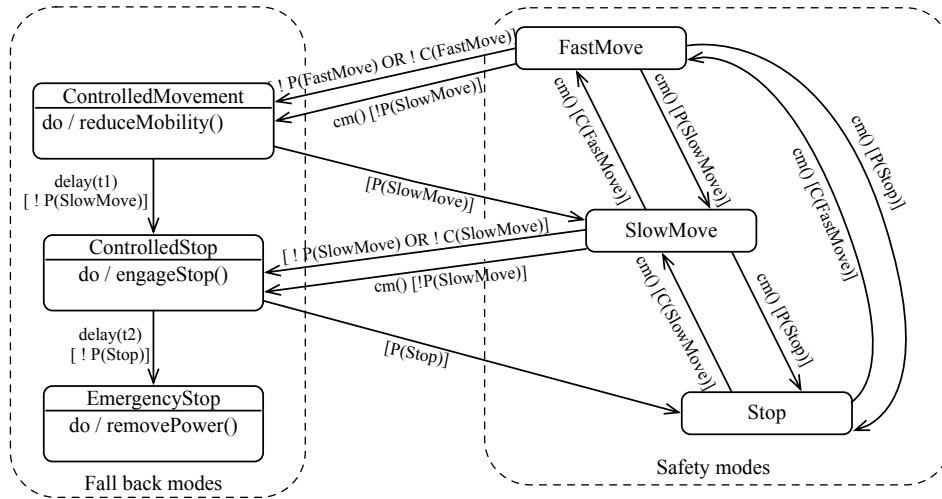


Fig. 6. Partial safety mode automaton with fall-back modes. *cm(x)*: abbreviation of request to change mode to "x" ("x" omitted avoid cluttering the figure)

and fall-back modes in case of condition violations. We only present in Figure 6 a subset of the whole resulting statechart. As previously presented, when P or C conditions are violated the monitor switches to a fall-back mode in which actions are executed to impose conditions of a less permissive mode. For instance, in the *FastMove* mode, if P or C conditions are violated, or if a request to change mode to *SlowMove* occurs but conditions are not fulfilled, we have defined a fall-back mode *ControlledMovement*, which is a transitional mode to reach the *SlowMove* conditions. After a fixed delay (here t_1), if the conditions $P(SlowMove)$ are still not fulfilled then the monitor switches to another

fall-back mode, *ControlledStop*, to stop all movement of the robot.

In each fall-back mode, actions are performed to fulfill conditions. For instance, in the *ControlledMovement* mode, the action *reduceMobility()* can act on base speed, base acceleration but also stop the arm if any movement has been engaged (which is forbidden because the arm must be folded in this mode). Hence, in every fall-back mode, actions depend on which constraint is violated and are design-dependent. Ideally, the monitor should be able to observe and to act completely independently from the functional channel. Nevertheless, this is

rarely possible as the robot cannot be fully redundant in terms of sensors and actuators. Hence, this step of the safety mode approach should be taken in account during design of the functional system.

IV. DISCUSSION

The application of the approach on the case study gives evidence to the applicability of the proposed formalism. The terminology and the notation have been systematically applied. An important point is the difference between functional and context variables that has been proposed. This segregation is fundamental for building a safety mode automaton with a partial order permissiveness relation based both on P and C conditions, and for determining transition conditions. This is a key point in our approach. Determination of safety modes, and functional and context variables, is a process that can be used to supplement risk analysis methods. In our case, we use sequence diagrams, and expert reviews to determine hazards. We are currently studying a systematic approach to link risk analysis to our safety mode approach.

Another point that needs to be further developed is the consistency of safety limits between the safety monitor and the functional system. Indeed, the safety monitor should engage reactions in case of hazardous situations, but it should let the functional system react first. For instance, when a human enters in the robot trajectory, the functional system should react, and only if the situation does not change, then the monitor has to react. This can be done defining different limits for human distance, for example, or by using a timeout to trigger mode switching.

Some limitations have been identified during the last step of the approach, which is the definition of the reactions of the safety monitor. First, we need to consider in more detail the perception capabilities of the safety monitor. Indeed, speed monitoring can be done through sensors, but detection that the system is about to open the gripper while that is forbidden cannot be done with a sensor. This means that the safety monitor should be informed about internal requests of the system, which can decrease independence between the two channels. Second, if reactions from the monitor are performed (for instance, a forced change of control law, or forced stop of all movement), this should be made known to the functional system, to rebuild a plan for instance. Both of these current limitations point to the need for further work on the architecture level [13] and on the protocol between monitored system and

safety monitor. This is also related to our proposal for the use of *fall-back modes*, which are transitional modes where actions are performed to impose less permissive conditions. Such modes depend on the observation and reaction means available for the monitor (which are constrained by the architecture and the inter-channel protocol).

Currently, a small number of safety modes has been considered, and simple authorization domains have been proposed. For instance, all the domains of continuous variables (speed, force, etc.) are totally ordered ($[0, 0] \subset [0, 1] \subset [0, 2]$). This implies that it is possible to switch to a more permissive mode without checking any permissiveness condition (if speed is in $[0, 1]$, it is also true that speed is in $[0, 2]$). Nevertheless, we should consider that sometimes, domains will not be totally ordered, and have for instance unordered (but overlapping) domains such as $[0, 20]$ and $[15, 30]$. In order to determine transition conditions, intermediate-modes might be added, such as in this case, the interval $[15, 20]$. Transition conditions are then more complex and are difficult to determine manually.

Finally, the safety monitor approach implies an extension of system with additional possibilities to change the system state. New risks can then be introduced, so particular attention needs to be paid to integrity of the monitor itself. Our study is based on the assumption that the monitor will not fail dangerously, but we also have to prove that there is not any inconsistency between the monitor and the system, which could lead to hazardous situations.

V. CONCLUSION

We have presented in this paper a formal framework to facilitate the specification of safety rules used by an independent safety monitor. Our approach is based on the safety mode concept, which associates a specific rule set to each functional behavior of the monitored system. A graph representing the partial order between safety modes according to a permissiveness relation allows the automatic determination of transition conditions. In addition, each safety mode can be associated with a fall back mode aimed at enforcing safe execution conditions.

A case study was carried out on a mobile manipulator robot. This did not reveal any inconsistencies and confirmed that the formal framework is indeed useful. However, a real implementation has yet to be done. Moreover, some issues presented in Section IV are still open and are the subject of ongoing work. The main

issue to be clarified is the protocol between the safety monitor and the monitored system for observation, reaction and mode synchronization. In the near future, we aim to apply our approach to another robotic system and, in another domain, to an autonomous satellite.

ACKNOWLEDGMENT

This work was partially supported by Astrium Satellites France and by the PHRIENDS Specific Targeted Research Project, funded under the 6th Framework Programme of the European Community under Contract IST-045359. The authors are solely responsible for its content. It does not represent the opinion of the European Community and the Community is not responsible for any use that might be made of the information contained therein.

REFERENCES

- [1] A. Avizienis, J.-C. Laprie, B. Randell, and C. Landwehr, "Basic concepts and taxonomy of dependable and secure computing," *IEEE Transactions on Dependable and Secure Computing*, vol. 1, no. 1, pp. 11–33, 2004.
- [2] S. Roderick, B. Roberts, E. Atkins, and D. Akin, "The ranger robotic satellite servicer and its autonomous software-based safety system," *IEEE Intelligent Systems*, vol. 19, no. 5, pp. 12–19, 2004.
- [3] P. Klein, "The safety-bag expert system in the electronic railway interlocking system Elektra," *Expert Systems with Applications*, vol. 3, pp. 499–506, 1991.
- [4] J. Blanquart, S. Fleury, M. Hernerk, and C. Honvault, "Software safety supervision on-board autonomous spacecraft," in *Proceedings of the 2nd European Congress Embedded Real Time Software (ERTS'04)*, 2004.
- [5] K. Wika and J. Knight, "A safety kernel architecture," University of Virginia - Department of Computer Science, Tech. Rep. CS-94-04, 1994.
- [6] J. Fox and S. Das, *Safe and sound - Artificial Intelligence in Hazardous Applications*. AAAI Press - The MIT Press, 2000.
- [7] C. Pace and D. Seward, "A safety integrated architecture for an autonomous safety excavator," in *International Symposium on Automation and Robotics in Construction*, 2000.
- [8] S. Daly and S. Orme, "The reliability of the sizewell 'b' reactor protection system," *Electrical and Control Aspects of the Sizewell B PWR, 1992.*, *International Conference on*, pp. 208–214, 1992.
- [9] F. Py and F. Ingrand, "Dependable execution control for autonomous robots," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Sendai, Japan*, 2004.
- [10] D. Bertheliet, C. Chicher, M. Narmada, D. Dalemagne, O. Boudillet, C. Veltz, R. Chemel, M. Yu, and G. De Rivals Mazeres, "Automated Transfer Vehicle (ATV) critical software overview," in *53rd International Astronautical Congress of the International Astronautical Federation (IAF), Houston, TX; USA*, 2002.
- [11] D. Essame, J. Arlat, and D. Powell, "Tolérance aux fautes dans les systèmes critiques," LAAS-CNRS, Tech. Rep. 00151, 2000.
- [12] M. Kim, I. Lee, U. Sammapun, J. Shin, and O. Sokolsky, "Monitoring, checking, and steering of real-time systems," in *2nd International Workshop on Run-time Verification*, 2002.
- [13] E. Baudin, J.-P. Blanquart, J. Guiochet, and D. Powell, "Independant safety systems for autonomy," LAAS-CNRS, Toulouse, France, Tech. Rep. 07710, 2007.
- [14] F. Maraninchi and Y. Rémond, "Mode-automata: About modes and states for reactive systems," *Lecture Notes in Computer Science*, vol. 1381, pp. 185–200, 1998.
- [15] D. Henrich and S. Kuhn, "Modeling intuitive behavior for safe human/robot coexistence cooperation," in *Proceedings of the International Conference on Robotics and Automation*, 2006.
- [16] M. Perhinschi, M. Napolitano, G. Campa, B. Seanor, J. Burken, and R. Larson, "Design of safety monitor schemes for a fault tolerant flight control system," *IEEE Transactions on Aerospace and Electronic Systems*, vol. 42, no. 2, pp. 562–571, 2006.
- [17] A. Albu-Schaffer, C. Ott, and G. Hirzinger, "A unified passivity-based control framework for position, torque and impedance control of flexible joint robots," *The International Journal of Robotics Research*, vol. 26, no. 1, pp. 23–39, 2007.
- [18] G. Hirzinger, A. Albu-Schaffer, M. Hahnle, I. Schaefer, and N. Sporer, "On a new generation of torque controlled lightweight robots," in *Proceedings 2001 ICRA. IEEE International Conference on Robotics and Automation*, vol. 4, 2001, pp. 3356–3363.
- [19] R. Alami, M. Krishna, and T. Siméon, "Provably safe motions strategies for mobile robots in dynamic domains," in *Autonomous Navigation in Dynamic Environment: Models and Algorithms*. in C. Laugier, R. Chatila (Eds.), Springer Tracts in Advanced Robotics, 2007.
- [20] OMG, "2nd revised submission to OMG RFP ad/00-09-02 - Unified Modeling Language : Superstructure - version 2.0," Object Management Group, 2003.
- [21] J. Guiochet, G. Motet, C. Baron, and G. Boy, "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)*, C. Johnson and P. Palanque, Eds. Kluwer Academic Publisher, 2004, pp. 177–191.
- [22] ISO10218-1, "Robots for industrial environments – safety requirements – part 1: Robot," International Organization for Standardization, Tech. Rep., 2006.
- [23] S. Haddadin, A. Albu-Schäffer, and G. Hirzinger, "Dummy crash-tests for the evaluation of rigid human-robot impacts," in *IARP-IEEE/RAS-EURON Workshop on Technical Challenges for Dependable Robots in Human Environments, Roma, Italy*, 2007.