



**Unifying Control and Verification
of Cyber-Physical Systems
(UnCoVerCPS)**

WP1 Modelling and conformance testing (Task 1.4)

Deliverable D1.1 – Assessment of languages and tools for the automatic formalisation of system requirements

WP1	Deliverable D1.1 – Assessment of languages and tools for the automatic formalisation of system requirements
Authors	Simone Schuler - GE Global Research Alexander Walsch - GE Global Research Matthias Woehrle - Bosch
Short Description	In this deliverable we show how system requirements of hybrid systems can be formalized. We introduce standard monitor automata that represent specification pattern templates. A graphical user interface is introduced for the easy capture of requirements. Additionally, an overview of existing requirements capture tools is given and typical requirements for the different UnCoVerCPS use cases are collected.
Keywords	specification pattern templates, hybrid automata, requirements capture, formal verification, standard monitor automata, structured English
Deliverable Type	Report
Dissemination level	Public
Delivery Date	31 Dec 2015
Contributions by	Goran Frehse, Jens Oehlerking, Daniel Hess, Olaf Stursberg, Geoff Pegman
Internal review by	Daniel Hess, Goran Frehse
External review by	–
Internally accepted by	Matthias Althoff
Date of acceptance	21 Dec 2015

Document history:

Versior	Date	Author/Reviewer	Description
1.0	12 Oct 2015	Simone Schuler	initial version
1.1	05 Nov 2015	Simone Schuler	revised version
1.2	03 Dec 2015	Simone Schuler	final version

Contents

1	Introduction	3
2	UnCoVerCPS tool chain	4
3	State of the art	5
4	Requirement formalisation	7
5	Graphical user interface	12
6	Representative example	13
7	Summary and outlook	14
	Appendix	17
A	Requirements from applications	17
A.1	Wind turbine	17
A.2	Smart grid	18
A.3	Automated driving	19
A.4	Human-robot collaborative manufacturing	20
A.5	Requirement categories	28
B	Tool evaluation	28
B.1	Evaluation criteria	29
B.2	Stimulus	34
B.3	AutoFocus3	34
B.4	Others	35
B.5	Conclusions on tool evaluation	36
	References	36

1 Introduction

One of the main goals of UnCoVerCPS is the reasoning about formal properties and, in particular, safety of cyber-physical systems (CPS) during both design (i.e., offline) and operation (i.e., online). In order to reason about safety of CPS it is necessary to collect the respective safety requirements in a format suitable for the analysis tool. In general this format is defined by a formal language, which is not necessarily understandable by non-experts.

In industry, however, requirements are usually written in English-like (so called controlled English) expressions such that they are understandable across different disciplines (e.g. technical, legal, marketing). This English-like language is subsequently translated into a formal language used along the development process. Currently this is done manually in UnCoVerCPS. However, this has significant drawbacks: Some requirements in hybrid dynamic systems can only be represented by complex formal language terms and therefore introduce a high risk of translation errors. Complex expressions are for example necessary for requirements, which explicitly mention time or uncertainties in the environment models.

In addition, many requirements are very similar in their structure and are easily representable with the help of templates. Thus, a template-based (semi-)automated translation is proposed. In this report, we therefore address the challenges of translating English-like requirements to a representation suitable for UnCoVerCPS. In order to reduce manual intervention in the formalisation of these requirements, a methodology based on the mapping of requirement templates to standard monitor automata is introduced. Representative requirements for hybrid cyber-physical systems are taken from the UnCoVerCPS use cases.

Standard monitor automata are hybrid automata expressing forbidden states of hybrid systems frequently found across application domains¹ (see also [25] for the one of the earliest works on monitor automata). Since translations can be reused across domains they are expected to add a good level of risk reduction when going from English-like expressions to a formal hybrid systems modelling language. In order to show applicability of this approach, requirements from the wind turbine use case are automatically translated from a natural language format into the formal modelling language compatible with the UnCoVerCPS verification tools.

The main contributions of this deliverable are: We present a new framework for the translation of requirements in the form of specification pattern templates into hybrid automata (Section 4). Additionally, a graphical user interface is presented that allows the user to enter requirements based on controlled-English templates and exports them as a machine readable formal language format suitable for the UnCoVerCPS verification tools (Section 5). An extensive evaluation

¹Forbidden states are states that the system is not allowed to enter because they compromise system safety.

of existing tools for requirements capture and (semi-) automatic formalisation is given in Appendix B.

2 UnCoVerCPS tool chain

Within the scope of UnCoVerCPS a tool chain for the development of cyber-physical systems has been proposed (see Figure 1). Based on SCADE and Simplorer from Esterel Technologies [20], a model of the considered cyber-physical system and its surrounding entities (e.g. human workers in human-robot collaborative tasks or other traffic participants in automated driving) is provided. The models are translated into hybrid automata, which is the common modelling formalism for the subsequent verification algorithms. SCADE is already able to formally verify discrete systems but lacks the ability to verify hybrid dynamic systems. Within UnCoVerCPS this will be complemented by the tools SpaceEx [23] developed at Université Joseph Fourier Grenoble 1 and CORA [11] developed at Technische Universität München. Within this report we focus on SpaceEx with respect to verification. It is assumed that a successful integration of “requirement capture activities” into SpaceEx is the first necessary step for a successful UnCoVerCPS tool chain integration.

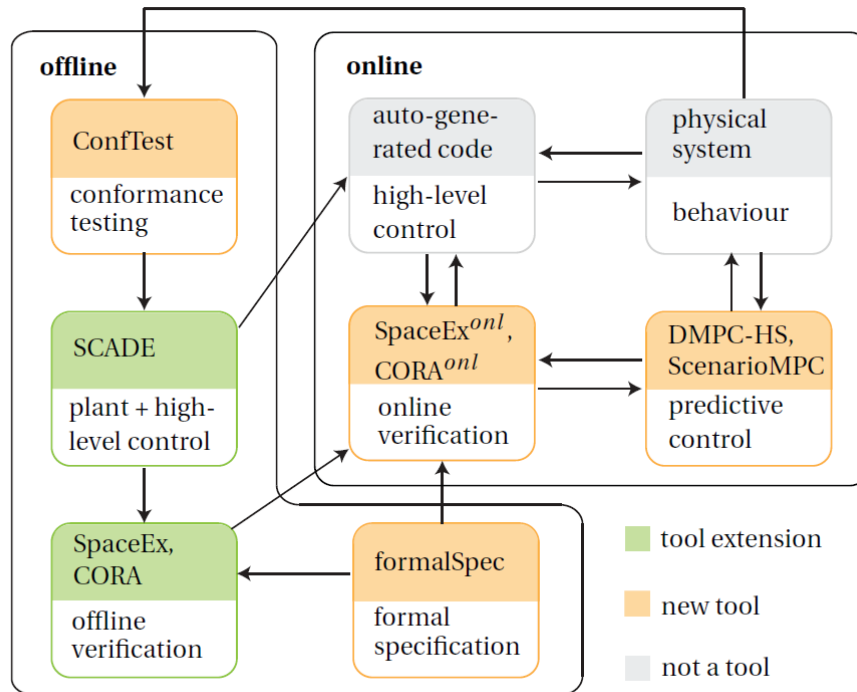


Figure 1: Tool chain of UnCoVerCPS.

SpaceEx is a verification platform for hybrid systems. The purpose is to verify (ensure beyond

reasonable doubt) that a given mathematical model of a hybrid system satisfies the desired safety properties. The current basic functionality is to compute sets of reachable states for hybrid systems with piecewise affine dynamics. Upon termination of the analysis, an over-approximation of the reachable states is displayed.

In order to use SpaceEx in its currently available stand-alone version, a system model and a configuration file have to be provided. The latter specifies the initial states along with some analysis options. SpaceEx comes with a model editor that allows graphical modelling of nested hybrid systems following standardised syntax and semantics [26]. The graphical model is also represented in a text based model description language using XML format.

To automatically detect undesired system behaviour, an observer can be added to the system model. An observer, following the same syntax and semantics as the system model, is also called monitor automaton. Since the monitor automaton encodes undesired states, which must not be reachable by the system, it is semantically equivalent to a set of safety requirements (something bad must never happen). The verification of a given set of requirements plus a given system is successful, if the intersection of the reachable states and the forbidden states is empty.

A monitor automaton can be added graphically to the system, when using the stand-alone SpaceEx version. However, in the industrial world safety requirements are usually entered in a human readable format which is close to the English language. Therefore, a methodology needs to be found that helps to maintain English-like text input whilst also providing a suitable input for SpaceEx (and subsequently UnCoVerCPS) and allows hybrid system analysis with minimum manual intervention. To this end, the main concern of this deliverable is to build a new tool called 'formalSpec' as a front end to SpaceEx that allows for easy capture of safety requirements and a (semi)-automatic formalisation in a SpaceEx readable format (see Figure 1, bottom). The new tool 'formalSpec' will automatically translate the system requirements into a formal language and export them in a machine readable SpaceEx compatible format.

3 State of the art

Using English-like expressions to specify system requirements and (semi-) automatic translation for verification has been proposed by other authors as well. Dwyer et al.'s [19] was one of the first papers to introduce qualitative specification pattern templates and their translation into different logic expressions (e.g. linear temporal logic). Among others, Konrad and Cheng [28] extended Dwyer's original patterns into the real-time domain and Post et al. [31] applied and extended those patterns for the automotive domain taking examples from Bosch. A

generalisation to probabilistic specification patterns was introduced in [24]. The specification templates as proposed by Konrad & Cheng are depicted in Figure 2.

Start	1: property	::= <i>scope</i> “,” <i>specification</i> “.”	
Scope	2: scope	::= “Globally” “Before ” R “After ” Q “Between ” Q “ and ” R “After ” Q “ until ” R	
General	3: specification	::= <i>qualitativeType</i> <i>realtimeType</i>	
Qualitative	4: qualitativeType	::= <i>occurrenceCategory</i> <i>orderCategory</i>	
	5: occurrenceCategory	::= <i>absencePattern</i> <i>universalityPattern</i> <i>existencePattern</i> <i>boundedExistencePattern</i>	
	6: absencePattern	::= “it is never the case that ” P “ holds”	
	7: universalityPattern	::= “it is always the case that ” P “ holds”	
	8: existencePattern	::= P “ eventually holds”	
	9: boundedExistencePattern	::= “transitions to states in which ” P “ holds occur at most twice”	
	10: orderCategory	::= “it is always the case that if ” P “ holds” (<i>precedencePattern</i> <i>precedenceChainPattern1-2</i> <i>precedenceChainPattern2-1</i> <i>responsePattern</i> <i>responseChainPattern1-2</i> <i>responseChainPattern2-1</i> <i>constrainedChainPattern1-2</i>)	
	11: precedencePattern	::= “, then ” S “ previously held”	
	12: precedenceChainPattern1-2	::= “ and is succeeded by ” S “, then ” T “ previously held”	
	13: precedenceChainPattern2-1	::= “, then ” S “ previously held and was preceded by ” T	
	14: responsePattern	::= “, then ” S “ eventually holds”	
	15: responseChainPattern1-2	::= “, then ” S “ eventually holds and is succeeded by ” T	
	16: responseChainPattern2-1	::= “ and is succeeded by ” S “, then ” T “ eventually holds after ” S	
	17: constrainedChainPattern1-2	::= “, then ” S “ eventually holds and is succeeded by ” T “, where ” Z “ does not hold between ” S “ and ” T	
	Real-time	18: realtimeType	::= “it is always the case that ” (<i>durationCategory</i> <i>periodicCategory</i> <i>realtimeOrderCategory</i>)
		19: durationCategory	::= “once ” P “ becomes satisfied, it holds for ” (<i>minDurationPattern</i> <i>maxDurationPattern</i>)
		20: minDurationPattern	::= “at least ” c “ time unit(s)”
21: maxDurationPattern		::= “less than ” c “ time unit(s)”	
22: periodicCategory		::= P “ holds ” <i>boundedRecurrencePattern</i>	
23: boundedRecurrencePattern		::= “at least every ” c “ time unit(s)”	
24: realtimeOrderCategory		::= “if ” P “ holds, then ” S “ holds ” (<i>boundedResponsePattern</i> <i>boundedInvariancePattern</i>)	
25: boundedResponsePattern		::= “after at most ” c “ time unit(s)”	
26: boundedInvariancePattern		::= “for at least ” c “ time unit(s)”	

Figure 2: Specification patterns taken from Konrad & Cheng [28].

The papers and studies mentioned above deal with requirements capture and translation into different discrete logics only. The translation from logic into a monitor automaton suitable for verification has to be done by the respective verification or model checking tool, e.g. NuSMV [17] amongst others. For most existing model checking tools, the equivalent to the monitor automaton has the form of a Büchi-automaton or a general ω -automaton. For the purpose of UnCoVerCPS, we deal with hybrid systems, i.e. systems that consist of discrete and continuous time states. SpaceEx and other tools for verification of hybrid systems cannot deal with specifications in the form of an ω -automaton.

Nevertheless, the initial idea of this project was to use an existing tool designed for requirements capture of discrete systems. This tool would allow to enter requirements in controlled English, similar to the specification patterns discussed previously. The output of the tool should be a formal representation of the requirement with well-specified semantics (non-ambiguity) along with a machine-readable format. We took into account that the output format would not

be directly suitable for SpaceEx due to the differences between Büchi automata and hybrid automata and that therefore additional translation steps would be needed (see Figure 3). The system model preparation as depicted Figure 3 is not part of the effort described in this report and therefore assumed to exist. In the course of UnCoVerCPS project system models will be generated as part of different activities.

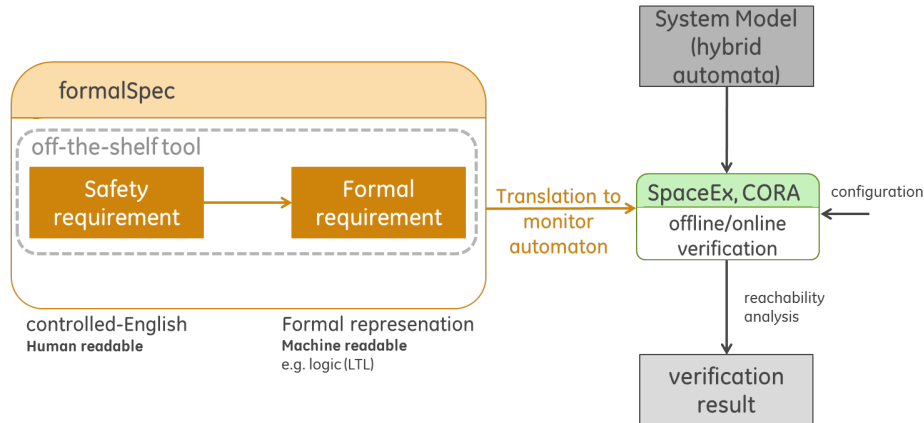


Figure 3: English-like safety requirement capture and flow-down to SpaceEx.

However, no representative end-to-end example similar to what is sketched in Figure 3 exists. This is caused by fundamental theoretical differences between the translation of text-based specification pattern templates into hybrid automata and the translation into logical expressions as used by most tools. Linear temporal logic (LTL) as used by the investigated tools (see Appendix B) does not cover the phenomena that can occur in hybrid systems. It is on the one hand more expressive than the hybrid automata suitable for the verification in SpaceEx and is on the other hand not able to express continuous time requirements which are fundamental to CPS. Additional problems are caused by tool specific limitations. A detailed evaluation of the different tools that were investigated within this effort is given in Appendix B.

4 Requirement formalisation

This section describes the major contribution of this report. Since none of the existing tools satisfy the criteria for the semi-automatic formalisation of system requirements suitable for the UnCoVerCPS tool chain, a novel mapping from controlled-English based specifications to hybrid automata is described. The controlled-English specification patterns are based on the specification pattern templates as described in Figure 2. In addition, their applicability has been cross checked against the requirements listed in Appendix A.

The basic idea is that requirements as found in hybrid systems can be expressed using controlled-English pattern templates and mapped to an equivalent hybrid automaton. Doing so does not require any knowledge of the theory of hybrid systems by the user and in addition is applicable to any development process in industry that starts with text based safety requirements. On a system level and in their controlled-English representation, specification pattern templates for hybrid dynamic systems do not differ from specification pattern templates available for discrete time systems. However, as explained in the previous section *formalisation* of the requirements differs significantly and existing approaches cannot be used.

Regarding Figure 1, the 'formalSpec' block is realised as depicted in Figure 4. Controlled-English pattern templates are used, which specify safety requirements. We then provide the translation of these pattern templates into hybrid automata. Our contributions are two-fold: First, we define a unique mapping from pattern templates into hybrid automata, so called standard monitor automata; second, we provide these hybrid automata in a text-based model description suitable for SpaceEx. The XML format used by SpaceEx is the de facto standard format for hybrid automata. Using a translation tool [15], it can be translated into input formats suitable for all major verification tools for hybrid systems [14].

One advantage of the direct translation from pattern templates into hybrid automata without logical expressions as an intermediate format is that the user has absolute control of what is used in the verification tool. Most of the existing tools for discrete systems only translate the pattern templates into logic expressions (e.g. linear temporal logic) and the verification or model checking tools translate the logic expression into a state automaton. The second translation might not be non-ambiguous between different tools and therefore a cause for non-repeatable results.

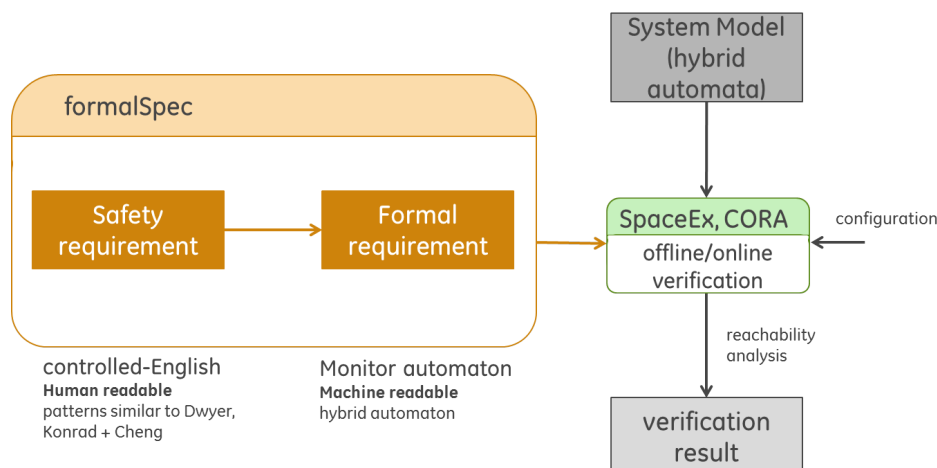


Figure 4: formalSpec: Safety requirements are mapped to monitor automata and subsequently analysed by SpaceEx.

In the following, we use the grammar introduced by Konrad and Cheng [28], since it is widely used within the academic world. However, in industry, project requirements are usually phrased differently, also depending on internal guidelines. We consider this as a minor issue, since syntax in general is replaceable and the focus is on the semantics.

We now provide the standard monitor automata that correspond to the specification pattern templates in Table 1. It is assumed that a system model which is going to be analysed already exists. Building the system model is out of scope for this report. The first column of the table, *Category*, categorizes requirements as introduced by Konrad and Cheng [28]. Only those requirements suitable for the analysis of hybrid systems have been considered. Along with the category a controlled-English text specification is provided in the second column (*Pattern Template*). This specification uses a symbolic form for predicates (e.g. p, q). Predicates are arbitrary mathematical expressions that evaluate to true or false. The text specification is formalised both in logic and as a hybrid automaton. The logic formula uses signal temporal logic (STL) as introduced by Maler and Nickovic [30], while the graphical representation of hybrid automata follows Henzinger [26] and is shown as it appears in the SpaceEx graphical editor. The hybrid automata are general in the sense that all mathematical expressions that evaluate to true or false can be used to substitute the predicates. However, which type of expression can actually be used may be limited by the specific verification tool. The scope *Globally* is only translated for the first specification pattern. Hybrid automata for the other templates with a global scope are omitted in this presentation since they are a subset of the scope *After q*, where *After q* always evaluates to true and the first state of the hybrid automaton is the `loc1` state.

Unless noted otherwise, all transitions in the monitor automata use so-called *may-semantics* (the standard for hybrid automata): As the system enters a guard it may take the transition, or delay taking the transition (or not take it at all) as long as the invariant condition of the source location is satisfied. Transitions with *must-semantics*, indicated with the label ‘urgent’, must be taken at the earliest point possible, i.e., the system cannot delay taking the transition once it satisfies the guard condition. For all monitor automata, the transition into the ‘error’ state is a may transition. This formulation is based on the assumption that the monitor automata are used within a reachability analysis computing approximative results (as in SpaceEx), where all *may* transitions have to be taken. When using the monitor automata in case of simulation, the proposed formulations might lead to ‘false negative’ results, since transitions do not have to be taken, even though the guards evaluate to true. To avoid these false negative results in simulation, the monitor automata have to be adjusted by changing the necessary transitions into *must* transitions.

Category	Pattern Template	STL	Hybrid Automaton
absence pattern (global)	Globally, it is never the case that p holds.	$G(!p)$	<pre> graph TD loc1[loc1] -- p --> error[error] </pre>
absence pattern	After q , it is never the case that p holds.	$G(q \rightarrow G(!p))$	<pre> graph TD before_activation[before_activation] -- q --> loc1[loc1] loc1 -- p --> error[error] </pre>
universality pattern	After q , it is always the case that p holds.	$G(q \rightarrow G(p))$	<pre> graph TD before_activation[before_activation] -- q --> loc1[loc1] loc1 -- !p --> error[error] </pre>

Category	Pattern Template	STL	Hybrid Automaton
minimum duration	After q , it is always the case that once p becomes satisfied, it holds for at least T time units.	$G(q \rightarrow G(p \rightarrow G_{\geq T}p))$	
maximum duration	After q , it is always the case that once p becomes satisfied, it holds for less than T time units.	$G(q \rightarrow G(p \rightarrow F_{< T}!p))$	
bounded recurrence	After q , it is always the case that p holds at least every T time units.	$G(q \rightarrow G(F_{\leq T}p))$	

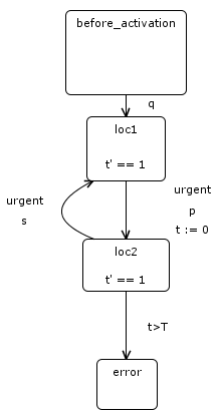
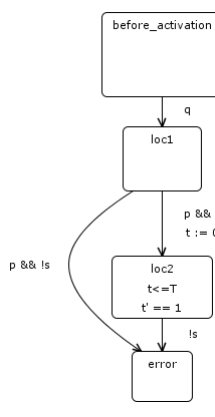
Category	Pattern Template	STL	Hybrid Automaton
bounded response	After q , it is always the case that if p holds, then s holds after at most T time units.	$G(q \rightarrow G(p \rightarrow F_{\leq T}s))$	 <p>The diagram shows a hybrid automaton with four states: 'before_activation', 'loc1', 'loc2', and 'error'. - 'before_activation' transitions to 'loc1' on event 'q'. - 'loc1' contains the invariant $t' == 1$. It transitions to 'loc2' on event 'p'. - 'loc2' contains the invariant $t' == 1$. It transitions to 'error' on the condition $t > T$. - There is a self-loop on 'loc1' labeled 'urgent s'. - There is a self-loop on 'loc2' labeled 'urgent p' and 't := 0'.</p>
bounded invariance	After q , it is always the case that if p holds, then s holds for at least T time units.	$G(q \rightarrow G(p \rightarrow G_{\leq T}s))$	 <p>The diagram shows a hybrid automaton with four states: 'before_activation', 'loc1', 'loc2', and 'error'. - 'before_activation' transitions to 'loc1' on event 'q'. - 'loc1' transitions to 'loc2' on event 'p && s' and 't := 0'. - 'loc2' contains the invariants $t \leq T$ and $t' == 1$. It transitions to 'error' on event '!s'. - There is a self-loop on 'loc1' labeled 'p && !s'.</p>

Table 1: Pattern templates for hybrid systems and translation to signal temporal logic and standard monitor automata.

Replacing the standard monitor automata with the real system requirements can now be done by replacing the predicates with the real values.

5 Graphical user interface

Within this project, a graphical user interface (GUI) for the requirements capture and formalisation has been developed. The GUI allows the user to select requirements from the previously defined pattern templates. The predicates can then be replaced by the user with the real specifications of the system. Export to SpaceEx or saving the requirements directly

is also included.

Selection of the pattern templates follows Konrad & Cheng [28], where first the *scope*, then the *specification type* and finally, the specification itself is selected (see Table 2).

Start	01: property	::= <i>scope</i> ” ,” <i>specification</i> ” .”
Scope	02: scope	::= ”Globally” ”After <i>q</i> ”
General	03: specification	::= qualitativeType realtimeType
Qualitative	04: qualitativeType	::= <i>absencePattern</i> <i>universalityPattern</i>
	05: absencePattern	::= ”it is never the case that” <i>p</i> ”holds”
	06: universalityPattern	::= ”it is always the case that” <i>p</i> ”holds”
Real-time	07: realtimeType	::= ”it is always the case that” (<i>minDurationPattern</i> <i>maxDurationPattern</i> <i>periodicPattern</i> <i>boundedResponsePattern</i> <i>boundedInvariancePattern</i>)
	08: minDurationPattern	::= ”once” <i>p</i> becomes satisfied, it holds for at least” <i>c</i> ”time unit(s)”
	09: maxDurationPattern	::= ”once” <i>p</i> becomes satisfied, it holds for less than” <i>c</i> ”time unit(s)”
	10: periodicPattern	::= <i>p</i> ”holds at least every” <i>c</i> ”time unit(s)”
	11: boundedResponsePattern	::= ”if” <i>p</i> ”holds, then” <i>s</i> ”holds after at most” <i>c</i> ”time unit(s)”
	12: boundedInvariancePattern	::= ”if” <i>p</i> ”holds, then” <i>s</i> ”holds for at least” <i>c</i> ”time unit(s)”

Table 2: Specification pattern templates for hybrid systems as used in the GUI.

A detailed example on how to enter safety requirements and the export to a SpaceEx readable format is given in Section 6.

6 Representative example

In order to show the applicability of the presented approach, one exemplary requirement from the wind turbine use case is formalised. Further wind turbine requirements and requirements from the other use cases have been summarised in Appendix A. We show the complete tool chain, starting with a requirement in controlled-English, the mapping to a specification

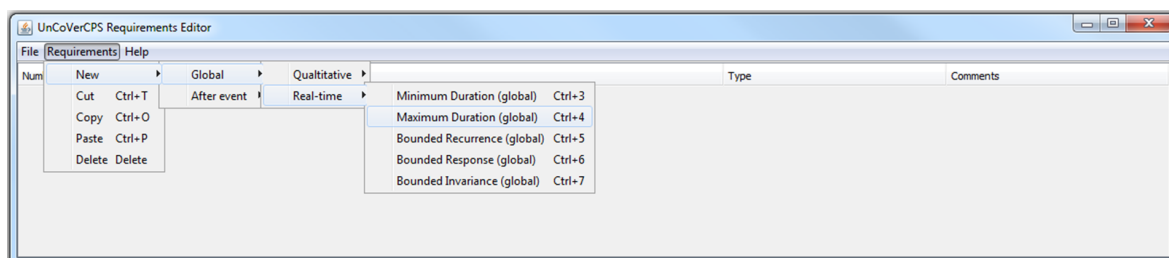


Figure 5: Selection of a new requirement template in the GUI.

template by the user and the automatic formalisation into a monitor automaton readable by SpaceEx using the graphical user interface.

We look at one of the fault detection requirements for the wind turbine:

The absolute difference between the commanded pitch angle and the measured pitch angle shall be larger than the maximum difference for less than c time units.

We can identify this as a real-time *maxDurationPattern* with a global scope. Reformulating it using the templates presented in Table 2 with (01,02,03,08), we receive

Globally, it is always the case that once p becomes satisfied, it holds for less than c time unit(s).

The predicates p and c then have to be replaced by $p = \text{abs}(\text{theta}_c - \text{theta}) > \text{max_diff}$ and $c = c$, where theta_c and theta are variables and theta_max and c are real-valued constants.

We will show next, how this can be entered using the graphical user interface and then exported to SpaceEx. Figure 7 shows how a new pattern template can be selected. After selection, the pattern templates (including the predicates) appear in the requirements list (see Figure 6). The symbolic predicates p and c can now be replaced by the real requirement specifications (Figure 7). After the requirement capture, export to SpaceEx (Figure 8) or saving the requirements directly is possible. Upon exporting to SpaceEx, the symbolic predicates in the standard monitor automata are replaced by the predicates entered in the GUI. It is then possible to load these monitor automata in SpaceEx and couple them to the system model for verification (see Figure 9).

7 Summary and outlook

In this report we proved an assessment of languages and tools for the automatic formalisation of system requirements suitable for cyber-physical systems and the UnCoVerCPS tool chain. Following standard specification patterns which are well known in the literature, a new translation into hybrid automata suitable for verification within the UnCoVerCPS tool chain

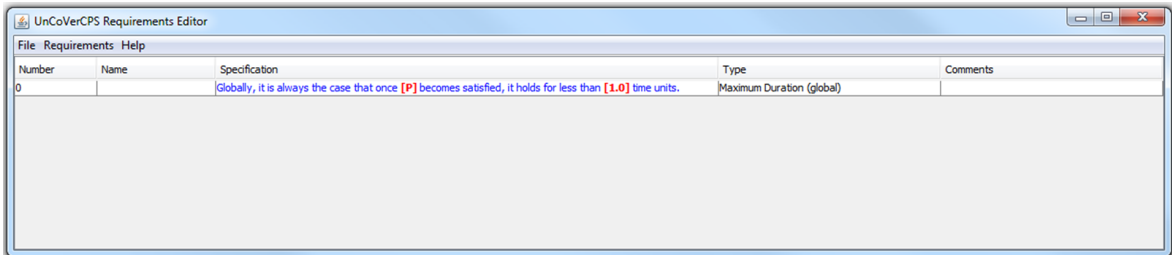


Figure 6: Requirement template with predicates.

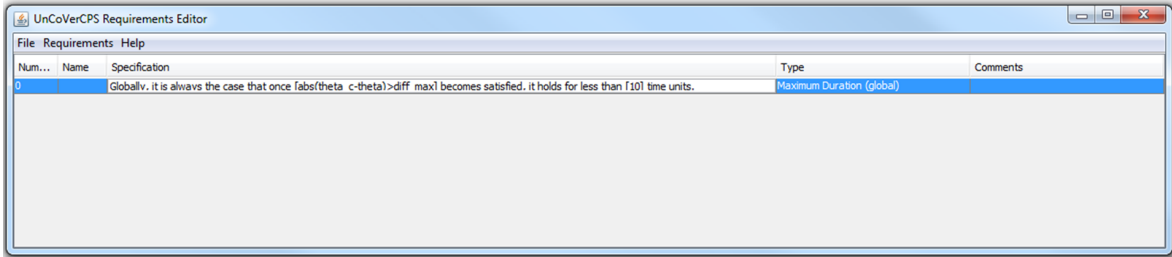


Figure 7: Wind turbine requirement with replaced predicates.

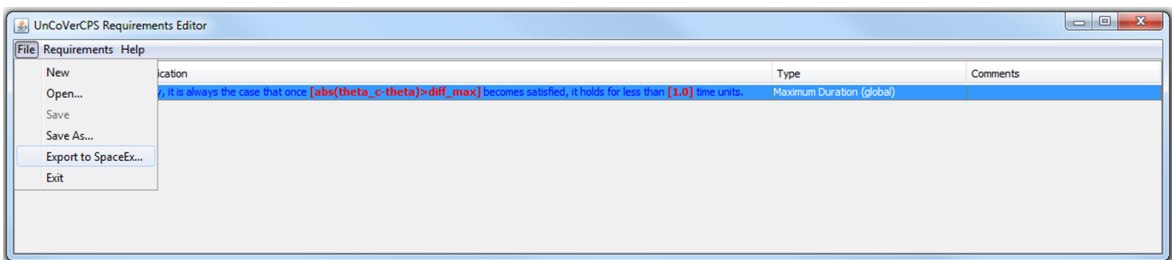


Figure 8: Requirements export to SpaceEx.

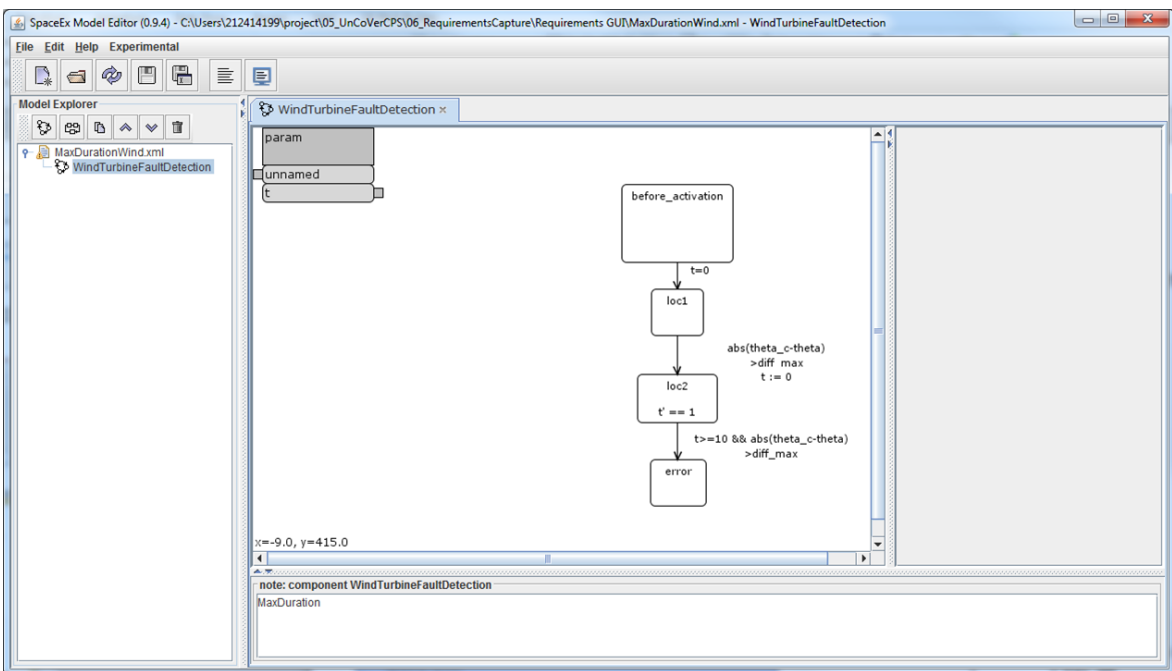


Figure 9: Exported monitor automaton in SpaceX.

was presented. Available specification patterns have been cross checked with the safety requirements from the UnCoVerCPS use cases. Finally, a graphical user interface has been developed to allow for automatic formalisation. This allows an easy capture of the system requirements without the necessity of the user to be familiar with hybrid systems and formal languages.

Going forward, we want to formally prove the equivalence of the signal temporal logic (STL) formulation with hybrid automata and check consistency of both using model checking techniques. Additionally, some of the monitor automata have equivalent formulations, e.g. without using the 'urgent' semantic. We would like to study the efficiency of the different automata for the same specification pattern templates in the analysis tools.

Related to the use cases within UnCoVerCPS, we want to formalise the requirements we have received for the use cases and verify them together using the models developed within the other tasks of this project.

Appendix

A Requirements from applications

To get a clear picture of the type of requirements appearing in hybrid cyber-physical systems, we collected requirements of all four use cases (wind turbine, smart grid, automated driving and human-robot collaborative manufacturing) considered within UnCoVerCPS. Due to this broad range of application domains and the current level of model development, the system requirements are formulated differently. Additionally, this is only an initial assessment of the final system requirements. Requirements may change, be further specified, and additional requirements might be added during the period of UnCoVerCPS.

A.1 Wind turbine

Within UnCoVerCPS, a simplified model of a wind turbine is developed, that exhibits the main nonlinear characteristics of a real wind turbine. When operating wind turbines, usually a trade-off has to be made between achievable power extraction from the wind and loads or damage to the structural components of the turbine. Additional constraints come from actuator limitations and the power grid operators.

Wind turbines operate differently depending on the current wind conditions. Below the so-called rated wind speed, the turbine tries to maximize the power extraction from the wind (partial loads) while above rated wind speed, power extraction is constant, to minimize the damage to the structural components (full load). Requirements differ depending on the current operating condition.

In addition to the requirements that hold during the actual operation of the wind turbine, other requirements concern the whole life-time of a turbine such as damage equivalent loads or extreme events.

A preliminary overview of the requirements for the wind turbine use case is given below:

Full load

- The pitch rate shall be smaller than the maximal pitch rate.
- The pitch acceleration shall be smaller than the maximal pitch acceleration.
- The generator torque shall be between the minimum and the maximum torque.
- The generator torque rate shall be between the minimum and the maximum torque rate.

- The rotor speed shall be smaller than the maximal rotor speed.
- The tower base moment shall be between the minimal and the maximal tower base moment.
- Damage equivalent loads (DEL) of tower base moment shall be less than the maximal DEL.

Partial load

- Requirements from full load.
- The pitch angle shall be larger than the stall pitch angle.

Fault detection

- The absolute difference between the commanded pitch angle and the measured pitch angle shall be larger than the maximum difference for less than c time units.
- The absolute difference between two individual pitch angles shall be larger than the maximum difference for less than c time units.

A.2 Smart grid

Power grids are large-scale systems consisting of varying numbers of generating, transmitting, and consuming nodes. For larger power grids, a possible control approach developed by the project partners within UnCoVerCPS is to use a two-layer structure:

- The upper layer aims at coordinating the generating nodes of the grid by adapting the active powers of the nodes to momentary loads and by controlling the secondary frequency.
- The lower layer establishes a local (decentralized) controller for any generating node (i.e. synchronous generators, wind power plants, and solar power plants); the aim of these controllers is to control active power, voltage, secondary frequency, and/or rotor angle.

The specification of control goals for the two layers is as follows:

Upper layer

- Minimize deviations from power-load balances.
- Satisfy input and state constraints.

Lower layer

- Stabilize the respective grid node by placing the closed-loop poles in the left half-plane.
- Establish a satisfactory damping by choosing a suitable cone (ratio of imaginary and real parts) for the closed loop poles.
- Minimize an \mathcal{H}_∞ -criterion to reduce the effect of disturbances.
- Optionally: eliminate steady state errors.

These criteria are defined with different quantitative parameters for nominal operation and for failure scenarios (such as temporary unavailability of a link). For the lower layer, the control goals are typically specified in relation to expected intervals for the quantities which physically couple the nodes of the power grid.

A.3 Automated driving

Two of the project partners within UnCoVerCPS investigate automated driving. Mainly two different challenges arise here: Cars are moving based on human interaction and subject to environmental conditions. Besides developing safe autonomous behaviour, the human factor has also to be included in the design of automated driving to study effects when the grade of automation changes. This is especially important in dangerous situations. The consideration of other traffic partners is necessary for a holistic safety design process.

For this use case, system and safety requirements are already available in a detailed format. They mainly concern the communication between different vehicles and safety requirements for the vehicle itself. A representative selection of the requirements is given below.

Communication requirements (selection)

- The largest communication sequence flow duration shall be less than *TBD* seconds.
- The maximal waiting time for an *ACK* message should be 20 ms.
- When an acknowledgeable message arrives, an *ACK* message shall be sent to the sender by the receiver within the maximal waiting time.
- If an *ACK* message exceeds the maximal waiting time, the message being acknowledged shall be considered as lost.
- If a message is considered as lost, it shall be resent.

- A vehicle shall send `MVR.FINISHED` messages to its session partner after finishing successfully its planned manoeuvre.

Vehicle requirements (selection)

- Two or more automated vehicles with longitudinal and lateral control, with up to 2m/s^2 acc. controllable in lat and lon direction.
- Each vehicle shall be able to drive autonomously up to 40km/h.
- Each vehicle shall have 360° object detection.
- Each vehicle shall be able to center in the lane.
- Steering wheel actuator must react in *TBD* ms.
- Brake pedal actuator must react in *TBD* ms.
- Each vehicle shall be equipped with differential GPS with SAPOS reference signals.
- Each vehicle shall be able to track a time-parametrised trajectory of XY coordinates, which is at least C^3 smooth and of a function type *TBD*.
- Each vehicle will adhere to the tracking-error bounds provided during manoeuvre negotiation for the cooperative manoeuvre.
- Each vehicle will execute the trajectory tracking controller at a fixed rate of *TBD*.
- Each vehicle will execute the cooperative manoeuvre planning module at a fixed rate of *TBD*.

A.4 Human-robot collaborative manufacturing

The GRAIL Robot System is a food assembly system (e.g. for sandwiches or pizzas) based upon a four DoF food grade robot combined with a vision system to provide information about the position and orientation of food components. The robot is expected to operate close to human (and other robot) co-workers on the production line. One aspect of safety is to keep the inertia of the robot down so as to minimise any potential injury in the event of collision. However, the main safety mitigation is to monitor the intrusion of co-workers into the planned workspace of the robot and, in the event of such an intrusion, slow the robot down such that it could perform a “safe to stop” action prior to colliding with the person. Any safe to stop action leaves the robot in an operable condition such that it can restart without

further intervention once the co-worker exits its workspace. The robot does not actively avoid a collision with a co-worker solely caused by the motion of the co-worker, i.e. when the robot is stationary.

Requirements for this use case include system requirements, functional requirements, safety requirements and legislative requirements. Additionally, a risk assessment is included.

System requirements

- The robot will have a maximum mass of 20kg.
- The system will be hygienic.
- The system will meet IP 67 ingress protection requirements.
- The system should have a payback working in a two shift system of 1 year or less.
- The system will have a payback working in a two shift system of 2 years or less.
- The system will have an availability of at least 99%
- The Mean Time Between Failure of the system will be greater than 10,000 hours.
- The success rate of tasks should be greater than 90% during an 8 hour shift.
- The system will have a minimum operating life of 40,000 hours.
- The stay-out zone for people will be no more than the operating volume of the robot .
- The robot should be back-driveable.
- The effective inertia of the final link of the robot should be low to minimise any potential impact injury or damage.
- The robot will be capable of operating in either floor mounted or ceiling mounted mode (with suitable mounting spacers).
- The system should be capable of being moved without significant recalibration, i.e. the set up time after a mover should be less than 5 minutes.
- The system should perform auto-recognition or auto-calibration of new tools wrt the tool centre and action points.
- The system should operate from a 110V AC supply.

- The system operation will be specified graphically with respect to the product to be produced, not the motions to be carried out.
- The system will operate with source and target components located on moving conveyor belts.
- The system will be able to operate with both source and target components placed at random positions and orientations on the conveyors.
- The system will be required to operate in conjunction with other robots and people on an assembly line performing collaborative assembly.

Functional requirements

- The robot will have a reach of 720mm over a minimum radius of $\pm 120^\circ$
- The robot will have 4 degrees of controlled freedom, i.e. 3 positional and 1 rotational (roll).
- The system will be capable of performing pick and place tasks.
- The system will have a maximum operating payload of 2kg or more.
- The system will have the following transfer speeds following a standard 25mm x 300mm x 25mm H cycle:
 - With a 2kg payload a minimum of 40 picks per minutes.
 - With a 1kg payload a minimum of 75 picks per minute.
 - With a 0.5kg payload a minimum of 90 picks per minute.
- The system will have a minimum repeatability of 0.2mm.
- The system will have a minimum accuracy of 2cm.
- Product change-over time between two existing products will be a maximum of 120 seconds.
- A new product installation should be less than 4 hours where no new tooling is required.
- The system will not collide with any fixed objects in its environment.
- The system will perform a safe stop to avoid causing a collision with any human body part entering its workspace.
- The system will be capable of re-starting after a safe stop without a system reset.

Legislative requirements

The primary purpose of the UnCoVerCPS system is to improve the safety of the human when operating in a collaborative manner with the GRAIL Sandwich Making Robot. Clearly, therefore, understanding the safety requirements for Human-Robot Collaboration (HRC) is very important for further implementation of this application.

In this subsection, the legislative requirements as set out in various European Standards are discussed. The requirements identified here are closely related to the requirements listed in the previous sections, with a description of how this is applied to the GRAIL sandwich making robot and the UnCoVerCPS project. It also contains a risk assessment of the additional hazards arising due to HRC in GRAIL sandwich making system. Note that only the additional hazards associated with HRC are considered here and other standard hazards such as electrical hazards are not included.

Any robot system put into service in Europe must bear a CE mark. Although not mandatory, conformity with standards listed in the Official Journal of the European Union [33] offers the benefit of presumption of conformity to the Essential Requirements of the CE marking directives and so is an easy way to demonstrate compliance. Within the European Standards, the safety requirements for HRC are spelled out in the EN ISO 10218-1:2011 [3], section 5.10 and EN ISO 10218-2:2011 [4], section 5.11. These standards describe the different ways in which humans and robots may interact and, due to the potential for physical contact between the robot and human, the additional technical requirements that must be met in order to make these special operating modes safe. Note that new standards are currently in development (particularly ISO/TS 15066) that may influence the requirements for safety in HRC systems which may be published within the time scale of the UnCoVerCPS project.

Four types of collaborative operation with their appropriate safety mitigations are described. These are:

- Safety-rated monitored stop: The robot stops whenever a human enters the collaborative workspace. The means by which the robot is caused to stop must have sufficient reliability.
- Hand guiding: The robot is operated by a hand guiding device situated close to robot's end effector. To ensure safety, an Emergency Stop button and an *\enabling device*", commonly called a dead-man's-handle, must be employed. These devices must have suitable reliability.
- Speed and separation monitoring: The robot is required to maintain a pre-determined speed and separation from the operator at all times. The means by which the speed

and separation is monitored must have sufficient reliability. The acceptable speeds and separations are determined by means of a risk assessment. The standards also refer to additional information contained in ISO/TS 15066, however, this is the standard referred to above that is still under development and has not yet been published. It is expected that this new standard will indicate how the required separation distance should be calculated depending on robot speed and stopping time, etc. and may come into force during the execution of the project.

- Power and force limiting by inherent design or control: The design of the robot or its control is such that that, in the event of a collision between the robot and the human, insufficient force can be applied to cause harm. Again, the power or force limiting mechanism must have sufficient reliability. This is expected to be defined in more detail in the as yet unpublished standard, ISO/TS 15066, but early drafts suggest that the speeds and masses associated with the GRAIL robot, although designed to have low mass and some degree of mechanical compliance, may exceed the limits deemed to be safe without the additional safeguards provided by the UnCoVerCPS system. In the meantime, a number of studies have been performed to evaluate the level of harm caused by impacts between a robot and human which can be used as guidance. Each of these types of collaboration ensures safety through the use of additional monitoring processes which limit the behaviour of the robot to a safe subset of possible behaviours. In the standards, these additional processes which limit the behaviour of the robot are referred to as protective devices, and so, for example, the speed and separation monitoring device referred to above would be a protective device. Explicit requirements for the reliability of such protective devices are presented in EN ISO 10218-2, Section 5.2.2 [3]. In particular, the protective device must meet the following:

- A single fault must not result in loss of the safety function.
- Whenever reasonably practicable, the single fault must be detected at or before the next time the safety function is relied upon.
- When the single fault occurs, the safety function of the protective device is performed (i.e. the robot is stopped or slowed as required) and a safe state is maintained until the detected fault is corrected.
- All reasonably foreseeable faults must be detected.

EN ISO 10218-2 [3] also places requirements on the manner in which the robot is stopped when an appropriate condition occurs. A number of different stop categories are defined in

EN 60204-1, Section 9.2.2 [2] as follows:

Category 0: Stop by immediate removal of power.

Category 1: A controlled stop with power available to the machine actuators to achieve a stop. Power is then removed.

Category 2: A controlled stop with power left available to the machine actuators.

For the GRAIL sandwich making robot, it is highly desirable that Category 2 stops can be used since this causes minimum disturbance to the continuous operation of the production line. Category 0 and 1 Stops will typically require a restart procedure to be followed once the hazardous situation has been cleared which may significantly reduce throughput. EN ISO 10218-2 [3] requires that any faults detected in the protective devices result in a Category 0 or 1 stop function. For the case when collaborative operation uses a safety-rated monitored stop mode, stop Category 2 is permissible. In addition, various general requirements are included in the standards which do not in themselves influence the technical direction of the UnCoVerCPS project. These general requirements include:

- The need to properly label equipment capable of collaborative operation.
- The need to indicate whether the system is operating in fully autonomous or collaborative operating modes. If the system can only operate in collaborative modes, then this requirement may be relaxed although this is not yet definitive. It is expected that ISO/TS 15066 will offer more guidance on this when it is published.
- The need to indicate the operating zone of the collaborative robot system by signage or floor marking.
- The need for the safe design of switching devices that switch between collaborative and non-collaborative modes of operation. Again, if the system can only operate in collaborative modes, then this requirement may be relaxed.
- The need to consider end effector design and the work piece to insure that no additional hazards are created (sharp edges, etc.).

Safety requirements

This section describes the safety requirements for the UnCoVerCPS system applied to the GRAIL Sandwich Making Robot System when it is put into production, i.e. when it is a finished system ready to be sold to customers. The safety requirements of the GRAIL Robot System are primarily driven by requirements set out in the legislature which were described in the previous section

The safety requirements during the current research and development phase, although just as important, can be considered differently and may be achieved using restrictive procedures and physical separation between humans and robots. Additionally, the personnel exposed to the hazards during the research and development phase will have robotics expertise and will be more familiar with the hazards than personnel working on a production line. Although the safety requirements described here will not be implemented during the UnCoVerCPS project, it is important that these future requirements are known and analysed so that the final productionisation step can be made as easy as possible. The following specific requirements are distilled from the applicable safety legislature:

- A single fault must not result in loss of the safety function.
- Whenever reasonably practicable, the single fault must be detected at or before the next time the safety function is relied upon.
- All reasonably foreseeable faults must be detected.
- Any failures detected in the safety system must cause the robot to stop and power to be removed from the actuators.
- The safety system must have a reliability level of “PL=d” according to EN ISO 13849-1:2006 [1].

Risk assessment

Hazards are categorised according to their severity (slight or serious), frequency of exposure to hazard (seldom or frequent) and the possibility of avoiding the hazard (possible or scarcely possible). The classification in each of the three categories for a hazard determines the Performance Level requirement for the mechanism that reduces the hazard according to EN ISO 13849-1, Annex A [1] with the hazard list taken from EN ISO 12100 (10). The hazards present in the GRAIL robot system appear in Table 3 which shows that impact is the biggest risk and that it requires protective measures with Performance Level d (PL=d).

Table 3: GRAIL robot system hazard assessment.

Hazard	Note	Severity	Frequency	Possibility of Avoidance	Performance Level Required
Crushing	Crushing the co-workers hand between the end-effector and the conveyor belt if the human and the robot should attempt to place an ingredient on the same target piece of bread	Slight	Frequent	Possible	b
Shearing	Shearing points exist at e.g. robot elbow	Slight	Frequent	Possible	b
Cutting or Severing	Not likely due to robot design	n/a	n/a	n/a	n/a
Entanglement	Not likely due to robot design	n/a	n/a	n/a	n/a
Drawing-in or Trapping	Not likely due to robot design	n/a	n/a	n/a	n/a
Impact	The robot has a relatively low mass but operates at high speed	Serious	Frequent	Possible	d
Stabbing or Puncture	Not likely due to robot design. There are no sharp points	n/a	n/a	n/a	n/a
Friction or Abrasion	Not likely due to robot design	n/a	n/a	n/a	n/a
High pressure Fluid / Gas Injection or Ejection	The gripper uses a pneumatic supply	Slight	Frequent	Possible	b

A.5 Requirement categories

The previous sections summarize representative safety requirements for the use cases considered within UnCoVerCPS. We will now categorise these requirements and describe specification templates that generalize different types of requirements.

The first distinction in the requirements is the scope of the requirement. A requirement holds either *globally* or *after an event*. An example for a global specification from the smart grid use case is "Satisfaction of input and state constraints." A requirement that holds only after an event is e.g. `After an acknowledgeable message arrives, ...`. Following this, we can distinguish between qualitative and real-time requirements (see also [19] and [28] for discrete systems). Qualitative requirements are occurrence patterns like *absence of universality*. They appear in all use cases in the form of state or input constraints caused by undesired system behaviour for certain areas of the state space or actuator limitations. Exemplary, we can consider the wind turbine requirement:

`The pitch rate shall be smaller than the maximal pitch rate for the wind turbine.`

Real-time requirements are also important for hybrid systems. However, they are often specified at a later point in the development (e.g. when shut down procedures or extreme event handling are defined). From the use cases we identified *minimum duration patterns*, *maximum duration patterns*, *bounded response patterns* and *bounded invariance patterns*. An example for a maximum duration pattern is `The maximal waiting time for an ACK message should be 20ms.`

Additionally, e.g. in the case of the wind turbine, requirements that are concerned with the life-time of a component or system, e.g. damage equivalent loads in the case of the wind turbine arise. From a top-level point of view, these requirements do not differ from the previously defined requirements, since they also evaluate to true or false (a certain probability is reached or not). However, they might be more difficult for the verification tools to analyse.

B Tool evaluation

As described in Section 4, our initial approach was to use an available tool to translate pattern templates into a logic expression to be used as an intermediate format. Unfortunately, as will be seen in this section, output formats of the tools are either not available or not suitable for a subsequent translation into a hybrid automaton. Nevertheless, we think that these tools are very interesting if slightly out of scope for the UnCoVerCPS tool chain. They offer for discrete

systems what we would like for hybrid systems at the end of the UnCoVerCPS project period.

B.1 Evaluation criteria

At first a general requirement capture tool survey has been performed based on publicly available information. Evaluation criteria have been selected based on the needs of the UnCoVerCPS tool chain. All existing tools focus on discrete dynamic systems. The initial evaluation criteria have been as follows:

- tool maturity, e.g. commercial, academic prototype
- input format, e.g. controlled English, graphical
- output format, e.g. none (closed tool), logic, graphical
- suitability to example requirements
- integration into SpaceEX, i.e. the output format can be translated to a format usable by SpaceEX

An initial assessment has been made for eleven tools as summarized in Tables 4, 5, 6 7. After this initial inspection, four tools have been selected for closer evaluation: Stimulus [12], AutoFocus3 [21], Embedded Specifier [16] and Compass [18].

	AF3	RAT/ RATS	STIMULUS
supplier	Fortiss	Graz University of Technology and Fondazione Bruno Kessler, Trento, Italy	Argosim
contact	[21]	[9]	[12]
tool type	free, high TRL	open -LGPL, medium TRL	commercial
tool description	Tool suite from requirements engineering to deployment (auto test case generation, schedulability, etc.)	RATS includes the capability to synthesize reactive systems from their temporal specification (input language). It includes a game-based approach for debugging specifications.	STIMULUS can express natural language requirements using formalized language templates, state machines, and block diagrams (also with auto test case generation capability)
input format	natural language using templates	PSL (process specification language)	natural language using templates
representation	–	–	–
exchange format	ReqIF for requirements, interface to additional AF3 functions	no output format	no computer readable model. Output are test cases automatically generated from requirements. Test oracle - can be used to verify implementation.
functionality	Structure, analyse and verify requirements. Formal validation of requirements (NuSMV model-checker and Yices SMT solver)	Property simulation, property assurance, property realizability and synthesis, and property debugging using Games	user extendable templates, graphical entry (state machines, blocks), detect errors in requirements by simulation, include environmental assumptions
analysis method	model checking, theorem proving	model checking (NuSMV)	–

Table 4: Overview of requirements capture tools.

	VARED	CESAR/ CRYSTAL ²	MBAT ³
supplier contact	NASA - Johnson Space Center [13]	FP7 /ARTEMIS project [5]	FP7 project [8]
tool type	unclear if still available	unclear (log in needed)	unclear (log in needed)
tool description	Design tool for requirement engineers. Requirements capture, formalization and analysis.	CRYSTAL takes up the challenge to establish and push forward an interoperability specification and a reference technology platform as a European standard for safety-critical systems	Combined model-based analysis and testing of embedded Systems
input format	natural language (processed by an NLP tool)	unclear	unclear, seems to cover all of the V-model
representation	–	–	–
exchange format	stand-alone tool, no specified output format. Can import C/Simulink files	–	–
functionality	natural language processing, translation into LTL (restrictions) and successive verification against a system model	–	–
analysis method	Semantic text analysis tool, translation into LTL, verification of a state space model against LTL requirements. For this InVeriant, a symbolic model checker, that is capable of doing safety checking on a class of linear hybrid automata is used.	–	–

Table 5: Overview of requirements capture tools continued.

²Some of the results of this project are available within Embedded Specifier [16].

³see 2

	SPES 2020	HyTech	COMPASS
supplier	TUM/BMBF funded project	UC Berkeley	ESA project
contact	[10]	[27],[6]	[18]
tool type	unclear (log in needed)	unclear if still maintained	unclear (log in)
tool description	general project around methodology for embedded systems development. One of the work packages is around requirements engineering (seems to be based on AF3)	HyTech is an automatic tool for the analysis of embedded systems. HyTech computes the condition under which a linear hybrid system satisfies a temporal requirement.	Correctness, modelling, and per-formability of aerospace systems
input format	requirements engineering seems to be based on AF3	includes a scripting language, which in principle allows to implement custom algorithms for checking other properties	Specification patterns and graphical input
representation	–	–	–
exchange format	–	–	–
functionality	–	–	requirements capture and formal analysis of hybrid systems
analysis method	–	reachability analysis	model checking (NuSMV) + counterexample generation, requirement analysis (RAT), safety and FTA (FSAP), simulation techniques

Table 6: Overview of requirements capture tools continued.

	KeYmaera & Sphinx	Embedded Specifier
supplier	CMU	BTC
contact	[7]	[16]
tool type	open source (high TRL)	commercial
tool description	Sphinx: textual and graphical modelling tool to describe hybrid systems KeYmaera: hybrid systems verification tool	creation and management of semi-formal and formal requirements of any kind
input format	dL (a logic for specifying and reason on hybrid dynamic systems). No natural language but front end (sphinx) that is built around xtext.	textual requirements
representation	dL input, hybrid program representation	machine-readable specification
exchange format	–	C-Observer
functionality	–	automatic test case generation, export of stand-alone requirement observers, continuous traceability between all artefacts
analysis method	theorem proving	interface to verification tools

Table 7: Overview of requirements capture tools continued.

B.2 Stimulus

Stimulus is a requirement modelling and simulation tool by Argosim [12]. Once requirements have been entered, Stimulus can generate simulation traces that satisfy a given set of requirements or produces an arbitrary set of traces and the requirement set can act as an observer. In the first case any inconsistency within a requirement set (e.g. a conflict) is reported, in the latter case any inconsistency between the traces and the constraints specified by the requirements is signalled.

In the course of this work, a two-week evaluation version has been provided, that allowed us to go over examples and become familiar with the tool. Stimulus offers a convenient way of specifying the execution semantics of a system by entering mathematical formulae and state transition diagrams. Semantics can be mapped to English sentences allowing a high level of expressiveness. By nesting existing or user generated patterns even more complex patterns can be created.

Currently Stimulus does not offer an output format that represents the execution semantics of requirements. In fact, the only possible output format, JSON [34], uses a hierarchical concept where specification information is not accessible when parsing the file.

Argosim are planning on extending Stimulus with an interface to model checkers like NuSMV [17]. Stimulus uses binary decision diagrams to represent information internally.

Three different requirements have been entered into Stimulus within the evaluation period. One simple but representative example along with the necessary steps is presented here. The requirement is a global bounded response pattern using the controlled English specifications by Konrad and Cheng [28]:

```
Globally, it is always the case that if x is less or equal 0.5 holds, then y
must be greater than 1 after at most 4 seconds.
```

First the semantics of this specification has been entered into Stimulus. This has been done by specifying a state transition diagram.

For this evaluation it was less important to specify an execution semantic that will meet a specific use case requirement but more to see how nested basic patterns can be mapped to a controlled English expression.

B.3 AutoFocus3

AutoFocus3 (AF3) is a development tool for embedded systems from Fortiss [21] based on Eclipse. The whole tool chain including requirements engineering, formal verification and test case generation is covered. AF3 can be interfaced with the model checker NuSMV [17], and

the results are directly incorporated into AF3.

Within the UnCoVerCPS project, we are not interested in this complete tool chain, but rather in the requirements capture tool. Requirements can be entered into AF3 using controlled English specifications as defined by Dwyer et. al [19]. Real time requirements cannot be entered directly, however, AF3 has a notion of time in the sense that time steps can be counted.

Exemplarily, we look at the pattern template: `Globally it is always the case that p holds`. Since AF3 directly interfaces with NuSMV there is usually no output format available that shows the translation of the text based requirements into a logical expression. However, it is possible to work with the developer version of AF3 [22] instead of the compiled version. Then, an exchange file called *.smv is generated as an input file for NuSMV. This input file includes the translation of the requirements into linear temporal. From here on, we have to proceed manually, using one of the many online conversion tools (i.e. Spot [32] or LTL2BA [29]) to translate the LTL formulae into an ω -automaton or Büchi-automaton. Beside the problem that this is a cumbersome procedure and involves many steps that have to be done by hand, the final Büchi-automaton cannot be read by SpaceEX, since Büchi-automata are more expressive than the hybrid automata SpaceEX can interpret.

B.4 Others

Embedded Specifier

Embedded Specifier by BTC [16] was partially developed within FP7 EU-projects. From the initial assessment and the documentation received from BTC it is the most developed tool for discrete systems in terms of requirements capture and automatic formalisation. Automatic test case generation is possible, such that requirements violation can be tested. A machine-readable observer is generated, to monitor requirements in hardware-in-the-loop tests. Unfortunately it was not possible to evaluate the tool in more detail, since it is commercial.

COMPASS

Within the COMPASS project [18] a System-Level Integrated Modeling Language (SLIM) has been developed. From the documentation received it was concluded that an extension to the AADL system modelling language has been developed. SLIM supports modelling of hybrid dynamic systems along with specifying erroneous behaviour. These models can be analysed with respect to safety requirements. From the documentation available we have not found any evidence that safety requirements have been entered in a human readable form.

B.5 Conclusions on tool evaluation

Concluding this appendix on tool evaluation, none of the available tools can provide the main functionality required for the UnCoVerCPS tool chain. All tools under closer consideration supported input of requirements in natural language or specification patterns. Some of the tools are closed and do not provide an exchange format. The tools that provide exchange formats, translate the requirements into discrete logics or Büchi-automata. Both formats are not suitable for further translation into an monitor automaton as required by SpaceEx. Therefore, the main part of this deliverables focuses on the development of new theory and tools specifically suited for the capture of system requirements in natural language and their verification for hybrid systems.

References

- [1] *EN ISO 13849-1:2008*. Brussels: CEN, 2008.
- [2] *EN 60204-1:2006+A1:2009*. Brussels: CEN, 2009.
- [3] *EN ISO 10218-1:2011*. Brussels: CEN, 2011.
- [4] *EN ISO 10218-2:2011*. Brussels: CEN, 2011.
- [5] CRYSTAL - critical system engineering acceleration. <http://www.crystal-artemis.eu/>, accessed on Oct. 12th, 2015.
- [6] HyTech - the Hybrid TECHnology tool. <http://embedded.eecs.berkeley.edu/research/hytech/>, accessed on Oct. 12th, 2015.
- [7] KeYmaera - a hybrid theorem prover for hybrid systems. <http://symbolaris.com/info/KeYmaera.html>, accessed on Oct. 12th, 2015.
- [8] MBAT - combined model-based analysis and testing of embedded systems. <http://www.mbat-artemis.eu/home/>, accessed on Oct. 12th, 2015.
- [9] RATS Y - requirements analysis tool with synthesis. <http://rat.fbk.eu/ratsy/>, accessed on Oct. 12th, 2015.
- [10] SPES 2020 - software plattform embedded systems. <http://spes2020.informatik.tu-muenchen.de/>, accessed on Oct. 12th, 2015.
- [11] M. Althoff. An introduction to CORA 2015. In *Proc. 2nd Workshop on Applied Verification of Continuous and Hybrid Systems (ARCH)*, 2015.

- [12] Argosim. Stimulus. <http://argosim.com/>, 2015.
- [13] J. Badger, D. Throop, and C. Claunch. VARED: verification and analysis of requirements and early designs. In *22nd Int. Requirements Engineering Conference (RE)*, 2014.
- [14] S. Bak, S. Bogomolov, and T. T. Johnson. HyST: A source transformation and translation tool for hybrid automaton models. <http://verivital.uta.edu/hyst/>, accessed on Sep. 30th, 2015.
- [15] S. Bak, S. Bogomolv, and T. T. Johnson. HYST: a source transformation and translation tool for hybrid automaton models. In *Proc. Conf. Hybrid Systems Computation and Control (HSCC)*, pages 36–42, 2015.
- [16] BTC. Embedded specifier. <http://www.btc-es.de>, 2015.
- [17] A. Cimatti, E. M. Clarke, E. Giunchiglia, F. Giunchiglia, M. Pistore, M. Roveri, R. Sebastiani, and A. Tacchella. NuSMV 2: An opensource tool for symbolic model checking. In *Proc. Int. Conf. Computer-Aided Verification (CAV)*, 2002.
- [18] COMPASS. Correctness, modeling and performance of aerospace systems. <http://compass.informatik.rwth-aachen.de/>. accessed on July 31st, 2015.
- [19] M. B. Dwyer, G. S. Avrunin, and J. C. Corbett. Patterns in property specifications for finite-state verification. In *Proc. 1999 Int. Conf. on Software Engineering*, pages 411–420, 1999.
- [20] Esterel. Homepage. <http://www.esterel-technologies.com/>. accessed on September 9th, 2015.
- [21] Fortiss. AutoFocus3. <http://af3.fortiss.org/>, 2015.
- [22] Fortiss. AutoFocus3 - developer version. https://af3-developer.fortiss.org/projects/autofocus3/wiki/Developers_documentation, 2015.
- [23] G. Frehse. An introduction to SpaceEX v0.8. <http://spaceex.imag.fr/documentation/user-documentation/introduction-spaceex-27>. accessed on September 9th, 2015.
- [24] L. Grunske. Specification patterns for probabilistic quality properties. In *Proc. 30th International Conference on Software Engineering (ICSE)*, pages 31–40, 2008.

- [25] N. Halbwachs, F. Lagnier, and P. Raymond. Synchronous observers and the verification of reactive systems. In *3rd Int. Conf. on Algebraic Methodology and Software Technology, AMAST'93*. Springer Verlag, 1993.
- [26] T. Henzinger. The theory of hybrid automata. In *Proc. 11th Symp. on Logic in Computer Science (LICS)*, pages 278–292, Jul 1996.
- [27] T. A. Henzinger, P.-H. Ho, and H. Wong-Toi. Hytech: A model checker for hybrid systems. *Software Tools for Technology Transfer*, 1:110–122, 1997.
- [28] S. Konrad and B. H. Cheng. Real-time specification patterns. In *Proc. 27th Int. Conf. Software engineering*, pages 372–381. ACM, 2005.
- [29] LTL2BA. <http://www.lsv.ens-cachan.fr/~gastin/ltl2ba/>, accessed on Oct. 6th, 2015.
- [30] O. Maler and D. Nickovic. Monitoring temporal properties of continuous signals. In *Formal Techniques, Modelling and Analysis of Timed and Fault-Tolerant Systems*, pages 152–166. Springer, 2004.
- [31] A. Post, I. Menzel, and A. Podelski. Applying restricted english grammar on automotive requirements: Does it work? a case study. In *Proceedings of the 17th International Working Conference on Requirements Engineering: Foundation for Software Quality, REFSQ'11*, pages 166–180, Berlin, Heidelberg, 2011. Springer-Verlag.
- [32] Spot. <https://spot.lrde.epita.fr/trans.html>, accessed on Oct. 6th, 2015.
- [33] E. Union. Official journal of the european union. <http://eur-lex.europa.eu/oj/direct-access.html>.
- [34] Wikipedia. Javascript object notation. <https://en.wikipedia.org/wiki/JSON>, 2015. accessed on Sep. 4th, 2015.