Catalogue of System and Software Properties*

Victor Bos[1], Harold Bruintjes[2], and Stefano Tonetta[3]

[1] SSF victor.bos@ssf.fi
[2] RWTH h.bruintjes@cs.rwth-aachen.de
[3] FBK tonettas@fbk.eu

Abstract. The use of formal methods has been recognized in different domains as a potential means for early validation and verification. However, correctly specifying formal properties is very hard due to the ambiguity of the typical textual requirements and the complexity of the formal languages. To this end, we define the Catalogue of System and Software Properties. Starting from a taxonomy of requirements extracted from space standards, we derive a list of design attributes divided per requirement type. We map these design attributes to AADL system architectures and properties, for which we define formal semantics and properties. We exemplify the approach using AADL models taken from the space domain.

1 Introduction

In many sectors such as transportation, space and health, the criticality of the software systems requires a high level of confidence and operational integrity. Formal methods allow early discovery of potential issues which otherwise may be discovered only during the (software) system integration and validation. In formal methods, the correctness and validity of design models is expressed in terms of formal properties. Therefore, their proper definition becomes a cornerstone of the early validation. The process of adequate properties specification poses multiple challenges. Requirements, being the main source of the properties to be specified, are often not trivial to be formalized due to the current practice of using natural language specifications. Furthermore, the formal properties may be very complex, which hinders the specification and the use of the formal methods.

We start from the observation that, as the design is being expanded and refined, design attributes are being specified either to perform specific analysis such as schedulability or to configure standard functions such as the monitoring of critical values. However, these design attributes are not typically formalized in the language used for system-level early verification and validation and no consistency is therefore checked with the previous analysis. This is a missed opportunity, as formalizing such design attributes can increase the consistency of the models and decrease the cost of specifying formal properties.

In this paper, we describe the Catalogue of System and Software Properties (CSSP). This is a predefined set of design properties already formalized into a mixture of temporal, real-time, and probabilistic logics. We extracted the design properties by analyzing standards and requirements documents in the space domain and creating a taxonomy of requirements with a list of design attributes divided per requirement type. We defined a

* This work was supported by ESA/ESTEC (contract no. 4000111828 (CATSY))

property set for AADL (Architecture Analysis & Design Language [1]), which can be used to specify such design attributes on a system architecture. By assigning values to the design attributes in this way, a predefined formal property is specified. These formal properties can be then analyzed for consistency among different levels of the architecture, can be used as assumptions or guarantees of components for contract-based design, and can be model checked on a behavioral model of the components. We extended the COMPASS tool to support this property set, automatically associating design attributes to formal properties. We exemplify the approach on the AADL model of a space system.

Related work Similar approaches make use of predefined patterns, either by defining a fixed set of patterns with placeholders, or by defining a grammar to construct such patterns from well defined rules. Well-known approaches for patterns are proposed in [2] (qualitative properties), [3] and [4] (real-time properties), and [5] (probabilistic). Recently a combination of such patterns, derived by a grammar was published in [6]. More domain specific patterns are known as well, such as for security [7] and epistemic properties [8]. Patterns are used also in contract based approach: in [9], a number of predefined patterns are defined to specify contract assumptions and guarantees.

All these approaches have in common that patterns are defined with a number of placeholders which accept (formal) propositions describing a certain state of the system. This is in contrast with the CSSP, which rather assigns values to predefined design attributes of the system, from which then formal properties are derived. So, the CSSP distinguishes itself from pattern-based approaches in two ways: first, it contains a predefined set of formal properties systematically derived from requirement categories; second, the formal properties are not instantiated by replacing the pattern placeholders with arbitrary expressions, but rather by assigning values to various attributes of the system. Clearly, in this way, the CSSP is very limited in expressiveness. The advantage is that the designer does not need to choose a pattern and to invent an instantiation.

2 Scope and Known Limitations

In the paper, we analyze the typical types of system requirements and we list a set of design attributes that are often used to specify a related detailed characteristic on the design. In order to have a systematic approach, we list these design attributes using a classification of requirement types. The classification is derived from ECSS (European Cooperation for Space Standardization) documents related to requirements engineering, in particular from [10,11,12,13,14,15]. The classification intends to cover all different levels of abstraction in space missions: *Mission Level* defining the mission objectives, products, and services; *System-of-System Level* covering aggregates of ground segment, space segment, launch segment and support segment; *System Level* covering single satellites, launchers, and data processing centers; *Sub-System Level* covering, e.g., electrical power, attitude control, structure, thermal control, and software; and finally *Equipment Level* covering, e.g., valves, batteries and individual electronic boxes.

Some remarks follow. First, we do not consider this classification as complete but just a means to derive the design attributes. The taxonomy is biased by the very specific domain (space) of the documents. Moreover, despite the documents coming from the

same domain, we found conflicting descriptions of requirement types. Therefore, the resulting taxonomy has been influenced by our knowledge and experience.

Second, we focus the attention on technical requirements, which specify characteristics of systems (products or services) to be developed. In [11], the term Technical Requirement is defined as the "required technical capability of the product in terms of performances, interfaces and operations". So, we do not consider requirements such as verification, usage, portability, and usability requirements. We also exclude from the present analysis security requirements, which are left for future work.

Third, in the description of requirement classes below, the term system refers to the subject for which requirements specify characteristics. In this sense, a system covers products and services. Moreover, it should not be confused with the system abstraction level. The term system (as the subject of requirements) applies to all abstraction levels.

3 Requirements Taxonomy and Design Attributes

1 *Context Requirements* specify the context in which the system is supposed to work. They specify assumptions about external entities such as physical phenomena, external systems and resources. For instance, if the system is software, its context is usually specified in terms of resources like processing power and memory. Context Requirements include both Environmental Requirements and Physical Requirements, defined in [10] respectively as "Requirements that define the context in which the system operates" and "Requirements that establish the boundary conditions to ensure physical compatibility and that are not defined by the interface requirements, design and construction requirements, or referenced drawings".
Typical design attributes associated to context requirements are the *allocated processor and memory* units, the *processing capacity*, the *clock frequency*, *endianness*, *memory size*s, and *addressable memory units*.

2 *Configuration Requirements* specify the product's internal composition in terms of sub-systems and connections. For instance, for a satellite, the Configuration Requirements define the power system, the thermal control system, the attitude control system, the payload systems, and all connections between them. In [10], Configuration Requirements are defined as Requirements related to the composition of the product or its organization. In this case, typical design attributes are the *list of sub-systems*, the *type of sub-systems*, the *connections between sub-systems*, and the *redundancy scheme* for sub-systems (identification of nominal and redundant sub-systems).

3 *Interface Requirements* define the data or services that a system provides to or requires from other systems. Depending on the system being specified, such data/services can be physical, thermal, electrical, or software.
Typical physical design attributes are *area*, *volume*, *alignment*, *stiffness*, *tolerance*, *geometry*, *flatness*, *fixation*, *mass* and *inertia*; electrical design attributes are *voltage*, *current*, and *margins for electrically induced effects* on nearby components; thermal design attributes are *temperature* and *thermal resistance* (conductibility) of materials; software design attributes are *event*s or *function*s.

4 *Functional requirements* specify what a system should do. In [10], the following example of a functional requirement is given: "The product shall analyze the surface of Mars

and transmit the data so that it is at the disposal of the scientific community". Although, in [10], Mission Requirements are identified as separate class, we identify Mission Requirements as Functional Requirements at the Mission abstraction level, as they specify "what the mission should do".

4. 1 *Input/Output Functional Requirements* describe the relation between input/output of the system. Typical design attributes associated to input/output functional requirements are the output that is generated in *response* to an input of the component, the maximum *reaction time* that can elapse between the received input and the generated output, and the *input that is required to generate an output*.

4. 2 *Mode Requirements* specify modes and mode transitions of a system. The mode (also known as phase) represents a configuration of a system determining how the system shall respond to inputs in that configuration. In addition transitions allow systems to switch between modes. Mode Requirements can also specify mode invariants, i.e., constraints on the values of variables. For instance, a single star tracker unit has one or more tracking modes. During a tracking mode, the star tracker constantly computes its attitude based on locations of detected stars. A Mode Requirement specifies an invariant on the attitude so that when the difference between successive attitudes is too large, the star tracker is said to have lost its "tracking lock" and will have to switch to another mode.
Typical design attributes associated to mode requirements are the list of *modes*, *mode transitions*, *transition triggers*, and *mode invariants*.

4. 3 *Data-Handling Requirements* specify the operations the system has to perform on data. This includes acquisition, processing, generation, and storage of data as well as refreshing and deletion of old data. For instance, for an Attitude and Orbit Control Subsystem (AOCS), data-handling requirements specify that the system estimates the spacecraft's current attitude and orbit from sensor inputs, determines the desired future attitude and orbit, and generates control commands for the actuators.
Typical design attributes associated to data-handling requirements are *input data rates*, *output data rates*, *volatile data*, *persistent data*, *processing steps*, *output data generation*.

4. 4 *Monitoring Requirements* specify the system parameters to be checked, the values against which to check, and the frequency at which to perform the check. In addition, monitoring requirements specify all actions needed depending on the outcome of a check.
Typical design attributes associated to monitoring requirements are *monitored parameters* (the parameters that have to be checked), *monitor range* (values against which to check, e.g., ranges or enumerated values), *check frequency*, *parameter check response actions* (specify what the system shall do after a check has been performed), *parameter check response result* (success or failure), and *monitoring state* (enabled or disabled).

4. 5 *Operational Requirements* specify rules according to which connected systems shall communicate. This includes requirements specifying what commands an operator shall issue to a system and what feedback is given in return. In [10], Operational Requirements are defined as: "Requirements related to the system operability".
Operational requirements include requirements on the observability of the system, on the commanding of the system, and on the protocols for system-to-system communications.
Typical design attributes related to observability are the *report format* (i.e., the format of the data the system has to generate), the *data frequency* (when data has to be generated at a certain frequency), *event data* (when data has to be generated only when a certain event occurs), and *observation mode* (enabled or disabled); associated to commandability,

there are the *list of commands*, the *type of commands*, *command conditions* (i.e., the conditions under when commands can be invoked), and *command responses* (i.e., the response a system shall send to the operator upon reception of a command); related to protocol, there are the *type of messages* (commands to the system, information from the system, acknowledgment), the *format of the messages* (headers, payload, source, target, ...), *message rate* (minimum, maximum, average), *response time* of acknowledgment (maximum time between reception of the original input message and the transmission of the acknowledgment), and *communication windows* (i.e., the period during which communication via a particular interface is possible).

5 *Quality Requirements* specify the manner in which a system must perform as well as the characteristics it should have in order for developers, maintainers, and users to be able to perform their tasks involving the system.

5. 1 *Performance Requirements* specify how well the system is supposed to perform with respect to certain indicators.
Typical design attributes are *jitter*, *latency*, *response time*s, *deadline*s, *throughput*, *processing capacity*, *CPU load*, *communication capacity*, and *memory capacity*.

5. 2 *Dependability Requirements* specify the degree to which the system can reasonably be relied upon. In [16], dependability is defined as "the extent to which the fulfillment of a required function can be justifiably trusted." An important concept related to dependability, especially in the space domain, is Fault Detection, Isolation, and Recovery (FDIR). The FDIR requirements describe what failures must be detected and what to do when a failure has been detected. For instance, if a failure occurs in equipment which has redundancy, it might be possible to switch from the nominal to redundant equipment configuration and continue with nominal operations.
We identify therefore the following design attributes: the *kind of failures* that might occur, the *tolerance of failures* (how often the failure may occur), the *failure detection delay*, and the *recovery actions*.

5. 3 *Reliability Requirements*. Reliability is defined as the probability that a system (i.e., product or service) will perform a required function under stated conditions for a stated period of time. In [16], reliability is defined as "the ability of an item to perform a required function under given conditions for a given time interval." For instance, the description of a space mission may state that after reaching a correct orbit, a 1 year nominal mission starts during which scientific data is gathered. Consequently, at the start of the nominal mission, the reliability shall be sufficiently high to ensure a 1 year nominal operation.
A measure for reliability of a system is the Mean Time To Failure, *MTTF*, which is the mean operational time (up time) of a system before any failure occurs.

5. 4 *Availability Requirements*. Availability is defined as the proportion of time for which the system (product or service) is able to perform its function in its intended environment. Availability takes into account both the operational time and the repair time.
Consequently, even if a system is not very reliable (i.e., it has a high probability of failure), short repair times of the system might be sufficient to achieve the desired availability. In [16], availability is defined as the "ability of an item to be in a state to perform a required function under given conditions at a given instant of time or over a given time interval, assuming that the required external resources are provided".

In general, *availability* can be seen as the fraction of time the system spends on average in an operational (i.e. *not failed*) state along its life cycle.

5. 5 *Maintainability Requirements* specify the acceptable efforts needed to restore a system after a failure has occurred. In [16], maintainability is defined as the "ease of performing maintenance on a product." It can be expressed as the probability that a maintenance action on a product can be carried out within a defined time interval, using stated procedures and resources. A measure for maintainability is the Mean Time To Repair, *MTTR*, that indicates the average time needed to restore the system operation after a failure has occurred.

5. 6 *Safety Requirements* specify system hazards and the acceptable risk of such hazard occurring. Hazards can be, e.g., loss of lives, injuries, loss of the mission, loss systems, and loss of equipment. Safety Requirements define all hazards relevant to the system. A typical design attribute is the *tolerance of failures* of the system.

4 The CSSP

The CSSP is based on models specified in AADL [1]. In AADL, a system is described in terms of components, which may contain other systems as subcomponents. Each system may specify a number of event, data or event data ports on its interface. Communication between systems and subsystems occurs via port connections.

The configuration of the system is determined by modes and mode transitions. At any given moment, each system is in a given mode, and based on events may move to another mode as specified by mode transitions.

The CSSP is specified as an AADL property set, which provides the language constructs to associate values to specific model's elements. For example, the CSSP contains the AADL property **Period**, which is applicable to event or event data ports; thus, every port of this kind in the architectural model can have a value associated with it that represents its period. The formal properties provided by the CSSP are determined by the value associated to the corresponding AADL properties.

A design attribute is then specified in terms of these property values or can be formalized by means of using certain structures in the AADL model (such as subcomponents and port connections).

4.1 Formalization of the CSSP

The formal semantics of CSSP formal properties relies on the behavioral semantics which have been defined for the SLIM language [17], a subset of AADL extended with behavioral models. In these semantics, event ports allow instantaneous event synchronization and data ports allow the continuous synchronization of data. Event data ports provide a mix where an event may be fired with an associated data value. Finally, SLIM models are described either by a real-time automaton, with *clock* values continuously increasing over time, or a probabilistic Markov chain, based on exponential distributions associated with events.

The logic used to define most CSSP formal properties is variant of MTL [18], where the atoms are predicates over the event and data ports, and modes of the AADL model.

In particular, we use the following notations: $F_{\leq u}\phi$ is true when ϕ is true within the following u time units; $O_{\leq u}\phi$ is true when ϕ is true within the preceding u time units; $\triangleright_{[a,b]}\,\phi$ is true when the next time ϕ is true is within $[a,b]$ time units; $\Cap\triangleright_{[a,b]}\phi$ is the strict version of $\triangleright_{[a,b]}\,\phi$ that does not consider the occurrence of ϕ at the current time; $change(x)$ is true when the value of x changes; $rise(b)$ is true when the expression b becomes true; the variable mode refers to the active mode of the current state; for an event data port p, $\mathtt{data}(p)$ holds the value of data passed with p after p occurs; for an event port p, $\sharp p$ is the number of occurrences of p in the past history. A more formal definition of the semantics can be found in [17].

Apart from MTL properties, we have ExpectedTime and Long-run average (abbreviated to LRA), which are described in [19] and rely on the probabilistic semantics of SLIM. The expected time is derived from the average sojourn time of a state in the system, and generates the mean time until reaching any state given as its parameter. The long-run average gives the fraction of time spent in states given as its parameter.

The list of formal properties that form the CSSP is shown in the following Table. The first column shows the CSSP formal properties. Each property is parametrized with an element from the input AADL model, indicated by the second column. In the third column the formalization is shown, which is parametrized by one or more AADL properties specified for the specified element. The formal property is defined if and only if all AADL properties inside the formal definition have a value in the model. For time intervals, if the upper bound is not set, it is assumed to be ∞.

Some properties can be expressed as an arbitrary arithmetic expression over data ports, data components and uninterpreted functions. This is a language feature specific to COMPASS. In the `CSSP` property set, they are encoded as strings, see [17] for details.

Name	**element**	**Formal property**
PersistentProperty(p)	data	$\mathrm{G}(change(p) \rightarrow \mathbf{Change}(p))$
Specifies that the value of p changes only on the **Change**(p) event.		
ModeInhibitedProperty(m)	mode	$\mathrm{G}(\mathsf{mode} = m \rightarrow \bigwedge_{e \in \mathbf{ModeInhibited}(m)} !e)$
Specifies that the events in **ModeInhibited**(m) cannot occur in mode m.		
ModeInvariantProperty(m)	mode	$\mathrm{G}(\mathsf{mode} = m \rightarrow \mathbf{ModeInvariant}(m))$
Specifies a generic invariant for mode m.		
MonitorProperty(p)	event data	$\mathrm{G}((p \wedge \mathsf{mode} \in \mathbf{MonitorEnabled}(p) \wedge (data(p) \notin \mathbf{MonitorRange}(p))) \rightarrow \mathrm{F}_{\leq u}\,\mathbf{MonitorResponse}(p))$
where $u = \mathbf{MonitorDelay}(p)$. Specifies that the event **MonitorResponse**(p) is fired if the value of p falls outside the specified **MonitorRange**(p).		
CompleteAlarm-Property(p)	event (data)	$\mathrm{G}(rise(\mathsf{mode} \in \mathbf{FailureCondition}(p)) \rightarrow \mathrm{F}_{\leq u}\,p)$
where $u = \mathbf{AlarmDelay}(p)$. Specifies that if failure configuration **FailureCondition**(p) is entered, the alarm event p follows.		
CorrectAlarmProperty(p)	event (data)	$\mathrm{G}(p \rightarrow \mathrm{O}_{\leq u}\,rise(\mathsf{mode} \in \mathbf{FailureCondition}(p)))$
where $u = \mathbf{AlarmDelay}(p)$. Specifies that if the alarm event p occurs, it was preceded by entering the failure configuration **FailureCondition**(p).		
RecoveryProperty(p)	event (data)	$\mathrm{G}(p \rightarrow \mathrm{F}_{\leq u}\,\mathsf{mode} \notin \mathbf{FailureCondition}(p))$
where $u = \mathbf{RecoveryDelay}(p)$. Specifies that upon event p, eventually the failure configuration **FailureCondition**(p) is recovered.		

CompleteTimeout-Property(p)	event (data)	$\mathrm{G}(F_{\leq u}(\textbf{TimeoutCondition}(p) \rightarrow p \vee \textbf{Timeout-Reset}(p)))$
where $u = \textbf{Timeout}(p)$. Specifies that if p does not occur within **Timeout**(p), the alarm **TimeoutReset**(p) must occur		
CorrectTimeout-Property(p)	event (data)	$\mathrm{G}(\textbf{TimeoutCondition}(p) \wedge \mathrm{O}_{\leq u}\, \textbf{TimeoutReset}(p) \rightarrow\ !p)$
where $u = \textbf{Timeout}(p)$. Specifies that if the alarm **TimeoutReset**(p) occurs, the event p did not occur.		
FunctionProperty(p)	data	$\mathrm{G}(p = \textbf{Function}(p))$
	event data	$\mathrm{G}(p \rightarrow \mathit{data}(p) = \textbf{Function}(p))$
Specifies the value of p remains within the associated function **Function**(p) (an expression).		
InvariantProperty(p)	data	$\mathrm{G}(p \in \textbf{InvariantRange}(p))$
	event data	$\mathrm{G}(p \rightarrow \mathit{data}(p) \in \textbf{InvariantRange}(p))$
Specifies the value of p remains within the associated range of values.		
ReactionProperty(p)	event (data)	$\mathrm{G}((p \wedge \mathsf{mode} \in \textbf{ReactionCondition}(p)) \rightarrow \mathrm{F}_{\in I}\, \textbf{Reaction}(p))$
where $I = [\textbf{ReactionMinDelay}(p), \textbf{ReactionMaxDelay}(p)]$. Specifies the event p is followed by **Reaction**(p) provided the mode is in **ReactionCondition**(p).		
PrecededByProperty(p)	event (data)	$p \rightarrow \mathrm{O}_{\in I}\, \textbf{PrecededBy}(p))$
where $I = [\textbf{PrecededMinDelay}(p), \textbf{PrecededMaxDelay}(p)]$. Specifies the event p is preceded by **PrecededBy**(p)		
PeriodProperty(p)	event (data)	$(\mathrm{F}_{\leq v}!\mathit{enabled} \vee \triangleright_{[v,v+j]}\, p) \wedge \mathrm{G}(\mathit{rise}(\mathit{enabled}) \rightarrow (\mathrm{F}_{\leq v}!\mathit{enabled} \vee \triangleright_{[v,v+j]}\, p)) \wedge \mathrm{G}((p \wedge \mathit{enabled}) \rightarrow (\mathrm{F}_{\leq u}!\mathit{enabled} \vee \not\triangleright_{[u,u+j]}\, p))$
where $u = \textbf{PeriodInterval}(p)$, $v = \textbf{PeriodOffset}$, $j = \textbf{PeriodJitter}$, $\mathit{enabled} = \mathsf{mode} \in$ **PeriodEnabled**. Specifies the event p occurs within the specified period and optional offset.		
ThroughputRatio(p)	event (data)	$\textbf{PeriodInterval}(p) = \textbf{ThroughputRatio}(p) * \textbf{Period-Interval}(\textbf{ThroughputInput}(p))$
Specifies the throughput of event p as an ratio of the throughput of **ThroughputInput**(p).		
ToleranceProperty(p)	port	$\mathrm{G}(\sharp p <= \textbf{Tolerance}(p))$
Specifies the tolerated number of failure events **Tolerance**(p). This is generally an assumption for other properties.		
MTTF(x)	event, component	ExpectedTime($\mathsf{mode} \in \textbf{FailureCondition}(x)$)
The expected time (mean time) until **FailureCondition**(x) holds.		
MTTR(x)	event, component	ExpectedTime($\mathsf{mode} \notin \textbf{FailureCondition}(x)$) with starting state = **FailureCondition**(x)
The expected time (mean time) until **FailureCondition**(x) no longer holds.		
Availability(s)	system	LRA($\mathsf{mode} \notin \textbf{FailureCondition}(s)$)
The availability specified as the *long-run average* of s being in a nominal mode.		

4.2 Coverage of the Design Attributes

In the next Table we summarize how the design attributes are formalized using AADL and the CSSP property set. For each design property, the associated CSSP parameter(s) or AADL structure are listed that allows its formalization.

Design Attributes	**CSSP parameters or structure**
allocated processor and memory	Connected component of category **processor** or **memory**.
processing capacity, *clock frequency*, *endianness*, *memory size*, *addressable memory units*	**InvariantRange** or **Function** property associated to corresponding connected feature. E.g., the system as an input port called processing capacity connected to the processor and **InvariantRange**=1(MHz)..5(MHz) applied to such a port.
list of sub-systems, *type of sub-systems*, *connections between sub-systems*	Modeled as AADL subcomponents and port connections between components.
redundancy scheme, *memory protection mechanism*	The sub-system architecture establishes the available redundancy. System modes identify the current sub-system configuration. E.g., the system has 2 identical CPUs, CPU A and CPU B, fully cross-strapped with 2 identical memory modules, mem A and mem B. In nom-nom mode, CPU A and mem A are used. In nom-red mode, CPU A and mem B are used.
area, *volume*, *alignment*, *stiffness*, *tolerance*, *geometry*, *flatness*, *fixation*, *mass*, *inertia*, *voltage*, *current*, *margins for electrically induced effects*, *temperature* and *thermal resistance*.	A (constant) **Function** property associated to corresponding connected feature. E.g., the system type has an output port called voltage and has set 50(Volts) as **Function** applied this port.
response to inputs and *reaction time*	**Reaction** and **ReactionLatency** associated to input event or event data ports.
*event*s or specific *function*s	Modeled as AADL event ports.
The *input that is required to generate an output*	**PrecededBy** associated to output event or event data ports.
modes, *mode transitions*, *transition triggers*	Modeled as AADL modes, mode transitions and transition triggers (event ports).
mode invariants	**ModeInvariant** associated with an AADL mode.
input data rates, *output data rates*	**PeriodInterval** associated to input/output event data ports.
volatile data	Default semantics (no formalization is required).
persistent data	**Change** event associated to a data port of an AADL component modeling a persistent memory.
processing steps	Modeled as AADL modes and mode transitions.
output data generation	Modeled as event data ports.
monitored parameters, *monitor range*	The parameters are modeled as event ports, the range is specified with **MonitorRange** for the parameter ports.
monitoring state check frequency	**MonitorEnabled** associated to the parameter port. **MonitorDelay** associated to parameter port (an upper bound on the response time implies a lower bound in the check frequency).
parameter check response actions	**MonitorResponse** associated to parameter port.
parameter check response result	**Function** associated to the output event port set as **MonitorResponse** of the parameter port.

event data	Either by modeling the data port as event data port, or by specifying **PrecededBy** if data generation is triggered by a previous event.
report format	An AADL data component type is used as type of an output event data port of the system and the AADL data component implementation is used to specify the format.
data frequency	Specified by **PeriodInterval** associated to output event data port.
observation mode	**PeriodEnabled** mode associated to an event data port that models the observation event.
list of commands	Modeled as AADL data component types.
type of commands	Modeled as the AADL data type of ports.
command conditions	**PeriodEnabled** mode associated to an event data port that models the command event.
command responses	**Response** associated to command port.
type of messages	AADL implementation of the command data component.
format of the messages	AADL implementation of the command data component.
message rate, *jitter*	**PeriodInterval** and **Period Jitter** associated to the command port.
response time, *latency*, *deadline*	**ResponseLatency** associated to the command port.
communication windows	**PeriodEnabled** mode associated to an event data port that models the AADL modes in which communication can occur.
throughput	Specify **ThroughputInput** and **ThroughputRatio** to define throughput in terms of input events.
CPU load, *communication capacity*, *memory capacity*	AADL connection with input port of processor and **MonitorRange** property associated to connected output port of the system.
kind of failures	Specify **FailureCondition** to determine the failure configuration.
the *failure detection delay*	Specify **AlarmDelay**.
recovery actions	Specify **Reaction** to indicate the recovery event after an error event.
MTTF, *MTTR*	Specify **FailureCondition** to determine the time to reach the failed state.
availability	Specify **FailureCondition** in which the system is not available.
tolerance of failures	**ToleratedError** on some input events representing faults or **InvariantRange** on an input data (e.g. the number of good pixels in the CCD of a star tracker must be higher than a certain threshold).

4.3 COMPASS tool support

The CSSP has been implemented in the COMPASS toolset [20,21], where it is possible for a user to specify the CSSP property values for a SLIM model (the input language of COMPASS). Such properties are automatically translated by COMPASS into their formal counterparts, which are then analyzed by the toolset. Furthermore, it is possible to specify contracts based on these properties, for which consistency and refinement checks can be made [22].

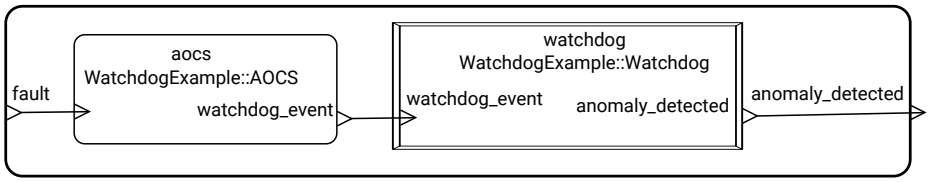

Fig. 1. Watchdog example

4.4 Example

To provide a better understanding of the use of the CSSP, we give an example based on a standard watchdog. This example and a more complete case study, modeling additionally an AOCS and a Startracker, using different kinds of CSSP properties is available at `compass.informatik.rwth-aachen.de/publications/safecomp2016`.

The AADL architecture is shown in Figure 1. The AOCS process periodically sends a signal to the watchdog, which raises an alarm if it does not receive the signal. The AOCS has a failure mode "dead" in which the signal is not sent. To formalize this, we can set the following AADL properties:

Timeout(anomaly_detected) => 1 Sec;
TimeoutReset(anomaly_detected) => reference(watchdog_event);
PeriodInterval(watchdog_event) => 1 Sec;
PeriodEnabled(watchdog_event) => (reference(alive));
ModeInhibited(dead) => (reference(watchdog_event));
Reaction(fault) => reference(anomaly_detected);
ReactionMaxDelay(anomaly_detected) => 3 Sec;

These properties automatically specify the formal properties *CompleteTimeoutProperty*(watchdog.anomaly_detected), *PeriodProperty*(aocs.watchdog_event), *ModeInhibitedProperty*(aocs.watchdog_event), and *ReactionProperty*(fault), as defined in Section 4.1. Moreover, COMPASS can automatically verify that *ModeInhibitedProperty*(aocs.watchdog_event) and *CompleteTimeoutProperty*(watchdog.anomaly_detected) entail (logically) *ReactionProperty*(anomaly_detected).

5 Conclusions and Future Work

This paper provides an important contribution to close the gap between formal methods and the standard practices in system and software design. It describes an extension of AADL that defines a catalogue of AADL properties with a precise formal semantics and provides a mapping from standard design attributes typically used in system and software design to the AADL properties. This allows the designer to easily apply model checking techniques to verify the component behaviors or to check the consistency among the properties specified at different abstraction levels. To cover a wise range of properties we collaborated with space engineers analyzing standards and requirement documents taken from the space domain. For future work, we will extend the catalogue with documents of different domains. Moreover, this predefined set of properties opens up new research directions to customize and improve verification and synthesis techniques.

References

1. As-2 Embedded Computing Systems Committee SAE: Architecture Analysis & Design Language (AADL). SAE Standards n^o AS5506B (September 2012)
2. Dwyer, M.B., Avrunin, G.S., Corbett, J.C.: Property specification patterns for finite-state verification. In: Formal methods in software practice, ACM (1998) 7–15
3. Bellini, P., Nesi, P., Rogai, D.: Expressing and organizing real-time specification patterns via temporal logics. Journal of Systems and Software **82**(2) (2009) 183–196
4. Konrad, S., Cheng, B.H.: Real-time specification patterns. In: Software Engineering, ACM (2005) 372–381
5. Grunske, L.: Specification patterns for probabilistic quality properties. In: Software Engineering, IEEE (2008) 31–40
6. Autili, M., Grunske, L., Lumpe, M., Pelliccione, P., Tang, A.: Aligning qualitative, real-time, and probabilistic property specification patterns using a structured English grammar. Software Engineering, IEEE Transactions on **41**(7) (2015) 620–638
7. Cheng, B., Konrad, S., Campbell, L., Wassermann, R.: Using security patterns to model and analyze security requirements. In: RHAS. (2003) 13–22
8. Bozzano, M., Cimatti, A., Gario, M., Tonetta, S.: Formal design of fault detection and identification components using temporal epistemic logic. In: Tools and Algorithms for the Construction and Analysis of Systems. Springer (2014) 326–340
9. Gafni, V., Benveniste, A., Caillaud, B., Graph, S., Josko, B.: Contract specification language (CSL). Speeds D2 (2008)
10. ECSS Std ECSS-E-ST-10-06-C Space Engineering - Technical requirements specification. Technical Report Third issue, ESA-ESTEC, Requirements & Standards Division (March 2009)
11. ECSS Std ECSS-E-ST-10C Space Engineering - System engineering general requirements. Technical Report Third issue, ESA-ESTEC, Requirements & Standards Division (March 2009)
12. ECSS Std ECSS-E-ST-40C Space Engineering - Software. Technical Report Third issue, ESA-ESTEC, Requirements & Standards Division (March 2009)
13. ECSS Std ECSS-E-HB-40A Space Engineering - Software Engineering Handbook. Technical Report First issue, ESA-ESTEC, Requirements & Standards Division (December 2013)
14. ECSS Std ECSS-E-ST-60-30C Space Engineering - Satellite attitude and orbit control system (AOCS) requirements. Technical Report First issue, ESA-ESTEC, Requirements & Standards Division (August 2013)
15. ECSS Std ECSS-E-HB-10-02A Space engineering - Verification guidelines. Technical Report First issue, ESA-ESTEC, Requirements & Standards Division (December 2015)
16. ECSS Std ECSS-S-ST-00-01C ECSS System—Glossary of terms. Technical Report Third issue, ESA-ESTEC, Requirements & Standards Division (October 2012)
17. Bozzano, M., Bruintjes, H., Nguyen, V.Y., Noll, T., Tonetta, S.: SLIM 3.0 – syntax and semantics. Technical report, RWTH Aachen, Fondazione Bruno Kesseler (2016)
18. Koymans, R.: Specifying real-time properties with metric temporal logic. Real-time systems **2**(4) (1990) 255–299
19. Guck, D., Han, T., Katoen, J.P., Neuhäußer, M.R.: Quantitative timed analysis of interactive markov chains. In: NASA Formal Methods. Springer (2012) 8–23
20. COMPASS Project. `compass.informatik.rwth-aachen.de` Accessed 11-03-2016.
21. Noll, T.: Safety, dependability and performance analysis of aerospace systems. In: Formal Techniques for Safety-Critical Systems. Springer (2014) 17–31
22. Cimatti, A., Tonetta, S.: A property-based proof system for contract-based design. In: Software Engineering and Advanced Applications, IEEE (2012) 21–28