Requirement Specification for a System

This document describes a textual requirement specification notation, called *ReqSpec.* It draws on the draft Requirements Definition and Analysis Language (RDAL) Annex, which defines a meta-model for requirement specification as annotations to AADL models.

The objective is to support the elicitation, definition and modeling of requirements for real-time embedded systems in an iterative process, thus supporting the refinement of requirements along with the system design, as well as qualitative and quantitative analysis of the created requirements specification, and finally, the verification of the associated system architecture models to ensure that they meet the requirements.

The draft RDAL Annex defines a Meta model that reflects RDAL's core concepts. These concepts have been taken from the Requirements package of the OMG Systems Modeling Language (SysML). In addition, many other concepts from the FAA Requirements Engineering Management Handbook (REMH) [FAA 2009], the KAOS method [Lamsweerde 2009] and the IEEE Std. 830-1998 have been added to cover important aspects of RE methods not included in SysML.

We distinguish between stakeholder requirements, referred to as *goals*, and system requirements, referred to as *requirements*. Goals express stakeholder intent and may be in conflict with each other, while system requirements represent a contract that is expected to be met by a system implementation.

The *ReqSpec* notation accommodates two major use cases.

First, it supports an Architecture-led Requirement Specification (ALRS) process. In this process stakeholder goals are turned into verifiable system requirement specifications, by annotating an AADL model of the system of interest in its operational environment and – as appropriate – elements of the system architecture. This process has been introduced in [Feiler 2015].

Second, it supports the migration of existing stakeholder and system requirements documents into a set of *ReqSpec* files that become annotations to an AADL model of a system. For that purpose we have built a tool to import existing requirements documents via the Object Management Group (OMG) Requirements Interchange Format (*ReqIF*) as well as export ReqSpec based modifications.

Concepts of the ReqSpec Notation

ReqSpec allows users to define goals, or stakeholder requirements, and requirements, or system requirements. Goals are expressed by *goal* declarations and requirements by *requirement* declarations.

Goals and requirements can be organized according to the architecture structure, by associating them with AADL component types or implementations, or they can be organized according to a document structure, in terms of document sections.

A *stakeholder goal set* declaration represents goals for a specific architecture component and contains a set of *goal* declarations.

A *system requirement set* declaration represents requirements for a specific architecture component and contains a set of *system requirement* declarations. User can also declare a set of reusable requirement declarations through a *global requirement set* declaration. Such reusable requirements can then be included in system requirement set declarations.

A *goals document* contains a *document* declaration that includes *document section* declarations and

goal declarations.

A *requirements document* contains a *document* declaration that includes *document section* declarations and *requirement* declarations.

Summary of file extensions:

- For *goals document*, use the extension *goaldoc*.
- For *requirements document*, use the extension *reqdoc*.
- For *stakeholder goal set*, use the file extension *goals*.
- For *system requirement set* and *global requirement set*, use the extension *reqspec*.

The *stakeholder goal set*, *system requirement set*, *global requirement set*, *goal document*, and *requirement document* constructs represent goal and requirement containers. They can have names with <dot>-separated identifiers (e.g., aircraft.Autopilot). These names can be used to qualify goals and the requirements contained in them.

A *goal*, *system requirement*, or *global requirement* has an identifier as name.

Goals and requirements can be referenced by their identifiers within the same container or by qualifying them with their container (e.g., aircraft.Autopilot.Req1).

References are shown in the grammar as <Goal> or <Requirement>, indicating the type of element being referenced.

Optional elements are shown as ( )?. Elements repeated one or more times are shown as ( )+, and elements repeated zero or more times as ( )*. For example:

- `( dropped )?`
- `( DocReference )+`
- `( ConstantVariable )*`

The set of elements between square brackets, [ ], can appear in any order.

Finally, users should be aware that ReqSpec is case sensitive. This is different from AADL,, which is not case sensitive.

Stakeholder Goals

ReqSpec uses the *Goal* construct to represent individual stakeholder requirements. Stakeholder goals can be organized in two ways:

· by the *StakeholderGoalSet* construct, to represent a collection of goals for a particular system that is represented as an AADL component

· by the *GoalsDocument* construct that contains goals, possibly organized into a (nested) *DocumentSection* to reflect the structure of an existing textual stakeholder requirement document

We proceed by describing the *Goal* and *StakeholderGoals* constructs. The *Document* and *DocumentSection* constructs are also used for requirements and are described in Section Documents and Document Sections.

The Goal Construct

The *Goal* construct represents a stakeholder goal with respect to a particular system.

```
Goal ::=
goal Name ( :  Title )?
( for TargetElement )?
[
( category ( CategoryReference )+ )?
( description  Description )?
( Constant )*
( WhenCondition )?
( rationale String )?
( refines ( <Goal> )+ )?
( conflicts with ( <Goal> )+)?
( evolves ( <Goal> )+)?
( dropped )?
( stakeholder ( <Stakeholder> )+ )?
( see goal ( <Goal> )+)?
( see document ( DocReference )+ )?
( issues (String)+ )?
( ChangeUncertainty )?
]

Title ::= String
TargetClassifier ::= <AADL Component Classifier>
TargetElement ::= <ModelElement>
CategoryReference ::= <CategoryType>.<CategoryLabel>
DocReference ::= URI to an element in an external document

Description ::= String ( <Constant or Variable> | this | String )*

WhenCondition ::=
  when in modes <Mode> ( , <Mode> )*
  |
  when in error state <ErrorState> ( , <ErrorState> )*
  |
  when expression
```

A goal declaration has the following elements:

- *Name*: an identifier that is unique within the scope of a goal container (requirement document or stakeholder goal set container).
- *Title:* a short descriptor of the goal. This optional element may be used as more descriptive label than the name.
- *For:* if present it identifies the target of the goal within a system, i.e., a model element within the classifier, e.g., a port, end to end flow or subcomponent. The enclosing *StakeholderGoalSet* container specifies the component classifier of the system of interest.

- *Category*: list of category references without comma separation (see Section User-defined Categories) to characterize a stakeholder goal. Such labels can be used for specifying filtered views of stakeholder goals.
- *Description:* A textual description of the goal. In its most general form this can be a sequence of strings, a reference to the classifier/model element identified by the *for* element (expressed by the keyword *this*), as well as references to Variables (see below).
- Set of *ConstantVariable:* Constant variables are used to parameterize goal and requirement specifications (see Section Constant and Compute Variables). Many of the changes to a goal or requirement are in a value used in the goal or requirement specification. Variables allow users to define a requirement value once and reference it in the description, predicates, and in verification activities of verification plans expressed in the Verify notation (documented in a separate report).
- *WhenCondition:* the condition under which the requirement applies. The condition is a set of AADL2 modes (operational modes), EMV2 error behavior states (failure modes), or a general expression on model elements and properties using the syntax of value predicate expressions (see Appendix A for details).
- *Rationale:* the rationale for a stakeholder goal as string.
- *Refines:* one or more references to other goals that this goal refines. Refinement of a goal does not change the system for which the goal is specified, but represents a more detailed specification of a goal.
- *Conflicts with:* references to other goals this goal is in conflict with.
- *Evolves*: references to other goals this goal evolves from. This allows for tracking of goals as they change over time.
- *Dropped*: if keyword is present the goal has been dropped and may be replaced by a goal that has evolved from this goal.
- *Stakeholder:* Reference to a stakeholder. Stakeholders are grouped into organizations. Each organization is defined in a separate file using the *Organization* notation.
- *See goal:* reference to a stakeholder goal in an imported stakeholder requirement document.
- *See document:* reference to an external document and element within expressed as URI. This is used to record the fact that a stakeholder requirement is found in a document other than an imported requirement document.
- *Issues:* allows users to record issues that may be encountered as a set of textual notes (Strings).
- *ChangeUncertainty:* user specified indication of stakeholder goal uncertainty with respect to changes. See Section Change Uncertainty for details on uncertainty specifications.

Note that when a goal is used in a *GoalsDocument*, the *for* clause can consist of a target description string or a classifier reference followed by a target element reference within that classifier. This allows goals found in existing stakeholder goals documents to be mapped into an architecture model and support identifying different systems for different goals in the same document or document section.

The Stakeholder Goal Set Construct

The *StakeholderGoalSet* construct is a container for *Goal* declarations. It is typically used to group together stakeholder goals for a particular system. It represents a name scope for the goal declarations contained in it, i.e., a goal is referenced by the *StakeholderGoals* name and the *Goal* name – separated

by a dot.

```
StakeholderGoalSet ::=
stakeholder goals QualifiedName ( :  Title )?
for ( TargetClassifier | all )
( use constants <GlobalConstants>* )?
[
(description  Description )?
(see document ( DocReference )+ )?
( Constant )*
( Goal )+
( issues (String)+ )?
]
QualifiedName ::= Identifier ( . Identifier )*
```

A *StakeholderGoalSet* declaration has the following elements:

- *QualifiedName*: name as a <dot> separated sequence of identifiers.
- *Title:* a short descriptor of the stakeholder goal container. This optional element may be used as more descriptive label than the name.
- *For:* if present it identifies the target of the set of stakeholder goals. This is a reference to an AADL component classifier. The keyword *all* is used to indicate a set of goals that can be applied to any system.
- *Use constants*: set of references to global constant definitions. The constants within those set can be referenced without qualification.
- *Description:* A textual description of the Stakeholder goals for a specific system. In its most general form this can be a sequence of strings, a reference to the classifier/model element identified by the *for* element (expressed by the keyword *this*), as well as references to Variables (see below).
- *See document:* reference to an external document. This is used to record the fact that the origin of the stakeholder requirements in this container is the identified document.
- Set of *Constant:* Constants are used to parameterize goal and requirement specifications (see Section Constant and Compute Variables). Many of the changes to a goal or requirement are in a value used in the goal or requirement specification. Variables allow users to define a requirement value once and reference it in the description, predicates, and in verification activities of verification plans expressed in the Verify notation (documented in a separate report).
- Set of *Goal*: a set of goal declarations. All contained goals are intended to be associated with the system represented by the classifier.
- *Issues:* allows users to record issues that may be encountered as a set of textual notes (Strings).

System Requirements

ReqSpec has the *SystemRequirement* construct to represent an individual requirement for a specific system. A system requirement is intended to be verifiable and not in conflict with other requirements. System requirement documents are modeled by the *Document* construct (see Section Documents and Document Sections). When representing system requirements in the context of an AADL model of the

system and its operational context the *SystemRequirements* construct is used to represent a collection of requirements for a particular system.

Users can also define requirements that are not specific to a particular system, but are applicable to any component or components of a specified set of component categories. Such *GlobalRequirements* can then be included in a *SystemRequirementSet* declaration as a set or set individual requirements through an *include* statement. The system identified by the *SystemRequirementSet for* statement determines the scope of applicability of the requirement, i.e., the requirement is applicable to that system and its subsystems down the hierarchy that match the category.

We proceed by describing the *SystemRequirement*, *SystemRequirementSet*, and *GlobalRequirementSet* constructs in turn.

Note that the term *system* in system requirements is not limited to the AADL *system* component category. A system may be represented by other categories as well, e.g., by *abstract* or *device*.

The System Requirement Construct

The *SystemRequirement* construct represents a system requirement.

```
SystemRequirement ::=
requirement Name ( :  Title )?
( for TargetElement )?
[
( quality ( <Qualitylabel> )+ )?
( category ( CategoryReference )+ )?
( description  Description )?
( Variable )*
( WhenCondition )?
( Predicate )?
( rationale String )?
( mitigates ( <Hazard> )+ )?
( refines ( <Requirement> )+)?
( decomposes ( <Requirement> )+)?
( inherits ( <Requirement> )+)?
( evolves ( <Requirement> )+)?
( dropped )?
(development stakeholder ( <Stakeholder> )+ )?
( see goal ( <Goal> )+)?
( see requirement ( <Requirement> )+)?
( see document ( DocReference )+ )?
( issues (String)+ )?
( ChangeUncertainty )?
]
```

A SystemRequirement declaration has the following elements:

- *Name*: an identifier that is unique within the scope of a requirement container (requirement document or system requirement set).
- *Title:* a short descriptor of the requirement. This optional element may be used as more descriptive label than the name.
- *For:* if present it identifies the target of the requirement within a system, i.e., a model element

within the classifier, e.g., a port, end to end flow or subcomponent. The enclosing *SystemRequirements* container specifies the component classifier of the system of interest.

- *Category*: list of category references without comma separation (see Section User-defined Categories) to characterize a requirement. Such labels can be used for specifying filtered views of system requirements.
- *Description:* A textual description of the requirement. In its most general form this can be a sequence of strings, a reference to the classifier/model element identified by the *for* element (expressed by the keyword *this*), as well as references to Variables (see below).
- Set of *Variable:* Constant and compute variables are used to parameterize requirement specifications (see Section Constant and Compute Variables). Many of the changes to a goal or requirement are in a value used in the requirement specification. Variables allow users to define a requirement value once and reference it in the description, predicates, and in verification activities of verification plans expressed in the Verify notation (documented in a separate report).
- *WhenCondition:* the condition under which the requirement applies. The condition is a set of AADL2 modes (operational modes), EMV2 error behavior states (failure modes), or a general expression on model elements and properties.
- *Predicate:* a formalized specification of the condition that must be met to indicate that the requirement is satisfied. The predicate may refer to variables defined as part of this requirement or the enclosing requirement specification container. See Section Requirement Predicates for details.
- *Rationale:* the rationale for a system requirement as a string.
- *Mitigates:* one or more references to hazards that this requirement addresses. The references are to an element in an EMV2 error model associated with the AADL model.
- *Refines:* one or more references to other requirements that this requirement refines. Refinement of a requirement represents a more detailed specification of a requirement for the same system. Requirements for a system are refined until they become verifiable.
- *Decomposes:* one or more references to requirements of an enclosing system that this requirement is derived from, i.e., it provides traceability across architecture layers.
- *Inherits:* one or more references to requirements of an enclosing system that is being inherited as a whole. For example, requirements on interfaces of an enclosing system can be inherited by those subsystems that directly take the input or produce the output of the enclosing system. This element provides traceability across architecture layers.
- *Evolves*: references to other goals this goal evolves from. This allows for tracking of goals as they change over time.
- *Dropped*: if keyword is present the goal has been dropped and may be replaced by a goal that has evolved from this goal.
- *Development Stakeholder:* Reference to a stakeholder from the development team, e.g., a security engineer or a tester. During architecture design, design choices may lead to new requirements, whose stakeholder is the developer making the choice. Stakeholders are grouped into organizations. Each organization is defined in a separate file using the *Organization* notation.
- *See goal:* reference to one or more stakeholder goals that the requirement represents. The goals

are assumed to be declared in a *StakeholderGoals* container or a *Document*.

- *See requirement:* reference to a system requirement in an imported system requirement document (*Document*)..
- *See document:* reference to an external document and optional element within expressed as URI. This is used to record the fact that a system requirement is found in a document other than an imported requirement document.
- *Issues:* allows users to record issues that may be encountered as a set of textual notes (Strings).
- *ChangeUncertainty:* user specified indication of stakeholder goal uncertainty.

Note that when a requirement is declared in a *RequirementsDocument*, the *for* clause can consist of a target description string, or a classifier references followed by a target element reference within that classifier. This allows requirements found in existing system requirements documents to be mapped into an architecture model and supports identifying different systems for different requirements within the same document or document section.

The System Requirement Set Construct

The *SystemRequirementSet* construct is a container for a set of *SystemRequirement* declarations. It is used to group together system requirements for a particular system, namely all requirements that are associated with an AADL component type or implementation.

```
SystemRequirementSet ::=
System requirements QualifiedName ( :  Title )?
for TargetClassifier
( use constants <GlobalConstants>* )?
[
( description  String )?
(see document ( DocReference )+ )?
(see goals ( <StakeholderGoals or GoalDocument> )+ )?
( Variable )*
( SystemRequirement )*
( include <GlobalRequirements or requirement> ( for ComponentCategory | self )*
( issues (String)+ )?
]
```

A *SystemRequirementSet* declaration has the following elements:

- *QualifiedName*: name as a <dot> separated sequence of identifiers.
- *Title:* a short descriptor of the system requirement set. This optional element may be used as a more descriptive label than the name.
- *For:* identifies the target of the set of contained system requirements by a reference to an AADL classifier. The keyword *all* is used to indicate a set of requirements that can be applied to any system.
- *Use constants*: set of references to global constant definitions. The constants within those sets can be referenced without qualification.
- *Description:* A textual description of the system requirements for a specific system. In its most general form this can be a sequence of strings, a reference to the classifier/model element identified by the *for* element (expressed by the keyword *this*), as well as references to Variables (see below).

- *See document:* reference to an external document. This is used to record the fact that the origin of the system requirements in this container is the identified document.
- *See goals:* reference to StakeholderGoalSet or GoalsDocument.
- Set of *Variable:* Constant and compute variables are used to parameterize requirement specifications (see Section Constant and Compute Variables). Many of the changes to a goal or requirement are in a value used in the requirement specification. Variables allow users to define a requirement value once and reference it in the description, predicates, and in verification activities of verification plans expressed in the Verify notation (documented in a separate report).
- Set of *Requirement*: a set of requirement declarations. By default all requirements are associated with the entity represented by the classifier. A requirement declaration may specify a model element within the classifier as its target in *for.*
- *Include*: reference to a global requirements or a requirement inside a global requirements declaration. The given component is the root of the component hierarchy in which the global requirement(s) apply. The *for* indicates the component categories to which the requirement applies to. *Self* indicates that the global requirement only applies to the component itself.
- *Issues:* allows users to record issues that may be encountered as a set of textual notes (Strings).

The Global Requirement Set and Requirement Constructs

The *GlobalRequirementSet* construct is a container for *Requirement* declarations. A *GlobalRequirementSet* declaration differs from *SystemRequirementSet* in that it contains *GlobalRequirement* declarations instead of *SystemRequirement* declarations, and it does not have a *for* statement, not an *include* statement.

GlobalRequirementSet ::=

global requirements QualifiedName (**:** Title)?

(**use constants** <GlobalConstants>*)?

[

(**description** String)?

(**see document** (DocReference)+)?

see goals (<StakeholderGoals or GoalDocument>)+)?

(Variable)*

(GlobalRequirement)*

(**issues** (String)+)?

]

The *GlobalRequirement* construct represents a global requirement. It application may be restricted to certain component categories through the *for* statement. The only difference to a *SystemRequirement* construct is the *for* statement.

```
GlobalRequirement ::=
requirement Name ( :  Title )?
( for ComponentCategory+ )?
[
```

```
// Same as for SystemRequirement
]
```

Documents and Document Sections

The *Document* construct allows users to organize stakeholder goals or system requirement into document sections to mirror existing documentation. This supports import of existing stakeholder requirement or system requirement documentation into ReqSpec.

A *Document* contains a set of document sections, and stakeholder goals or system requirements. A *DocumentSection* can recursively contain document sections, and stakeholder goals or system requirements.

A *GoalsDocument* only contains stakeholder goals, while a *RequirementsDocument* only contains system requirements.

```
GoalsDocument ::=
document Name ( :  Title )?
[
(description  String )?
( Goal  | DocumentSection )+
(issues (String)+ )?
]

GoalsDocumentSection ::=
section Name ( :  Title )?
[
(description  String )?
( Goal  | DocumentSection )+
(issues (String)+ )?
]
RequirementsDocument ::=
document Name ( :  Title )?
[
(description  String )?
( Requirement  | DocumentSection )+
(issues (String)+ )?
]

RequirementsDocumentSection ::=
section Name ( :  Title )?
[
(description  String )?
( Requirement  | DocumentSection )+
(issues (String)+ )?
]
```

A *RequirementsDocument* declaration has the following elements:

- *Name*: an identifier that is globally unique (within the workspace of OSATE).
- *Title:* a short descriptor of the stakeholder goal container. This optional element may be used as more descriptive label than the name.
- *Description:* A textual description of the requirement document content.
- Set of *Goal, Requirement,* or *DocumentSection*: a set of goal, requirement, or document section declarations that reflect the content of a requirement document. This may be an external document that has been imported, or a set of stakeholder or system requirements developed in ReqSpec in a traditional document format.
- *Issues:* allows users to record issues that may be encountered as a set of textual notes (Strings).

A *DocumentSection* declaration has the following elements:

- *Name*: an identifier that is unique within the enclosing container. Section names are not involved in referencing goals or requirements contained in a document section.
- *Title:* a short descriptor of the document section container. This optional element may be used as more descriptive label than the name.
- *Description:* A textual description associated with a requirement document section.
- Set of *Goal, Requirement,* or *DocumentSection*: a set of goal, requirement, or document section declarations that reflect the content of a requirement document. This may be an external document that has been imported, or a set of stakeholder or system requirements developed in ReqSpec in a traditional document format.
- *Issues:* allows users to record issues that may be encountered as a set of textual notes (Strings).

Variables and Predicates

Constants and Compute Variables

ReqSpec allows the user to introduce *Constants* to localize common changes to a stakeholder goal or system requirement. Constants act as parameters that can be referenced by Description elements in goal and requirement declarations and by Predicate elements in requirement declarations. Their values can be expressions that result in numeric values with an optional measurement unit, numeric value ranges, as well as Booleans, strings, references to model elements, and values of any user defined property type. Acceptable measurement units are any unit defined as Units literals in property sets of the AADL core language. See Appendix A for expression syntax details. The type is inferred from the value when not explicitly declared.

A predicate for a requirement typically compares an expected value against a value that has been computed or measured during a verification activity. The *ComputedVariable* declaration allows the user to introduce the name of such variables explicitly. They can then be referenced in predicate declarations. They can also be referenced in verification plans that complement requirement specifications in the architecture-led incremental system assurance (ALISA) workbench [Delange 2016].

```
Variable ::=
Constant | ComputedVariable

Constant ::=
val Name (: TypeSpec )? = Expression
```

```
ComputedVariable ::=
compute Name : TypeSpec
TypeSpec ::= BaseType | typeof <PropertyName>
BaseType ::= boolean | string | integer (units <UnitsTypeName> )?
| real (units <UnitsTypeName> )? | model element | <PropertyTypeName>
```

Reusable Global Constants

In some cases it is desirable to define a set of constants that can be referenced within the *ReqSpec* model of any system. Global constants are defined in files with the extension *constants*. The following syntax is used in those files:

```
GlobalConstants ::=
constants QualifiedName
[ ConstantVariable+ ]
```

Requirement Predicates

ReqSpec supports the specification of predicates as a formalization of a requirement. Predicates must be satisfied as part of a verification activity in a verification plan to produce evidence that the requirement is met. In many verification activities, an actual value from a system implementation is verified is compared against an expected value. The actual value may be computed by an analysis or measured in a simulation, test execution, or operation.

Users can specify predicates in one of several forms:

- Free form: `informal predicate "informal specification"` The user informally specifies a predicate as text. This allows users to quickly attempt to specify a predicate without getting hung up about syntax of a particular notation.
- Value assertion: `value predicate ActualBudget <= MaxBudget`
  Expressions compare actual values against expected values. This is done by comparing ReqSpec constant values, AADL property constants, AADL property values associated with the system component in an AADL model, and computed values represented by a *ComputedVariable*. Constants and computed variables are referenced by their name. AADL property and property constant references are prefixed by **#**. The expression language includes the operators: and, or, not, ==, !=, >=, <=, >, <, >< (contained in range), +, -, *, /, div (integer divide), mod. It supports parentheses and functions such as min, max, round, and abs. See Appendix A for details.
  For example, a user specifies `ActualCPUBudget <= MaxCPUBudget`, where *MaxCPUBudget* is a constant and *ActualCPUBudget* is a computed variable.
- Behavioral assertion: Behavioral predicate syntax will be supported in a future version of ReqSpec. Meanwhile users can specify behavioral assertions through the *informal predicate* construct.

User-definable and Predefined Category Types and Labels

ReqSpec allows users to associate category labels with goals and requirements. These category labels can also be associated with verification methods and verification activities in verification plans.

Users can then define filters on those category specifications to focus on subsets of requirements and

verification activities, e.g., on verifying key quality attributes, or on verification activities relevant to certain development phases.

Categories are declared in a separate file with the extension *cat* using the following syntax:

```
Categories ::= ( CategoryType )+
CategoryType ::=
Name [ (CategoryLabel )+ ]
```

The name of each category type must unique among category types. Labels must be unique within a category type. A category is referenced by its type and label, e.g., *Kind.Guarantee*.

The following category types have been predefined within the ALISA workbench:

- *Kind:* to indicate the kind of requirement
- *Guarantee:* guarantee made by a system to its environment, typically about its output.
- *Assumption:* assumption made by a system about its environment, typically about its input.
- *Exception:* exceptional condition such as safety hazard or security vulnerability that the requirement addresses.
- *Constraint:* a constraint on the implementation of a system, typically, on the subcomponents, their properties, state, and connectivity.
- *Consistency:* a consistency constraint between information in ReqSpec and an AADL model or between models. For example, that the values of ReqSpec constants are consistent with property values in the AADL model.
- *Quality:* to represent operational quality attributes that the requirement addresses. The following category literals are included: *Behavior*, *State*, *Timing* (schedulability), *Latency* (response time), *Safety*, *Security*, *Reliability*, *Availability*, *CPUUtilization*, *MemoryUtilization*, *NetworkUtilization*, *Mass*, *ElectricalPower*
- *Phase:* to represent development phases: SystemRequirements, ArchitectureDesign, PDR, CDR, DetailedDesign, Implementation, UnitTest, SystemTest.
- *Layer:* Tier of a layered architecture: *Tier1*, *Tier2*, *Tier3*, *Tier4*, *Tier5*.

Users can define their own category types. Users can also extend predeclared category types by defining additional category labels using the *CategoryType* declaration.

Stakeholders and Their Organizations

The *organization* notation allows user to define organizations and stakeholders that belong to organizations. Stakeholder names must be unique within an organization. Stakeholders are referenced by qualifying them with the organization name.

Each organization is declared in a separate file with the extension *org*.

```
Organization::=
organization Name
( Stakeholder )+

Stakeholder ::=
```

```
stakeholder Name
[
( full name String )?
( title String )?
( description String )?
( role String )?
( email String )?
( phone String )?
( supervisor <Stakeholder> )?
]
```

Change Uncertainty

Various techniques are commonly used to prioritize entities. For example, in the Architecture Tradeoff Analysis Method® (ATAM®) criticality and difficulty of change are used to prioritize use cases during an architecture evaluation. Safety analysis practices such as SAE ARP4761 use likelihood of occurrence and severity of impact to prioritize hazards [SAE 1996] to derive design assurance levels (DAL) to focus on high payoff safety risk reduction.

We introduce the concept of change uncertainty to assess the volatility to change and the impact of change.

Volatility represents the likelihood of change to a requirement or architecture design. Volatility may reflect several indicators, such as familiarity with a system, i.e., whether such a system has been developed before, frequent changes in the operational environment,

Impact represents the effort involved in performing the change and addressing its impact other parts of a system. It may reflect indicators such as system complexity, precedence in technology use.

These measures can identify high-payoff opportunities for reducing requirement change. [Nolan 2011] has demonstrated that reduction of up to 50% in requirement changes can be achieved based on expert assessment of such categorical measures.

Design Goals

RDAL distinguishes between verifiable and satisfiable requirements. Verifiable requirements must be met, and testing will provide a true/false result. In ReqSpec, all system and global requirements must be verifiable. Satisfiable requirements are quantified and must be met to a certain degree.

ReqSpec supports the specification of desirable target values that a system design is expected to satisfy. It does so in the context of a value predicate for a requirement. The value predicate specifies the value or value range that the system must meet (a verifiable requirement). This predicate can optionally be augmented with a desirable target value that is above or below the required value or value range (a satisfiable requirement). It is specified by optionally adding the following to value predicates:

```
with ( <constant> upto | downto <value> )+
```

Guidelines for Use of ReqSpec

This section provides some general guidelines on using ReqSpec with AADL models. ReqSpec is supported in OSATE by the workbench extension ALISA[1] that supports architecture-led incremental system assurance throughout the life cycle [Delange 2016]. Section 4.1 provides detail on installing ReqSpec and ALISA in OSATE.

[1] ALISA as a whole or ReqSpec as a separate Eclipse feature are available through an update site at www.github.com/osate/alisa.

Organizing ReqSpec Files

Users create files that contain stakeholder goal sets, system requirement sets, global requirement sets, goals and requirements in document structured format, global constants, stakeholders in organizations, and category types by creating files with the appropriate extension. Users can place these files in folders within a project that contains the AADL model; for instance, you can create a folder named *requirements* at the same level within a project as a folder called *packages* that contains AADL packages. Users can also place these files in a project separate from the AADL model of a system. In this case, you must set the project references for the projects within OSATE/Eclipse.[1]

---

[1] Set the project references using the pull-down menu Project → Properties → Project References, as shown in Figure 3.

Defining System Requirement and Stakeholder Goal Sets

When users define stakeholder goals and system requirements in an architecture-led fashion they define stakeholder goal sets and system requirement sets for an AADL component type or implementation. It is recommended, but not required, that you name these goal set or requirement set with the same name as the qualified name of the component classifier using "." instead of "::" as identifier separator.

When users define stakeholder goals and system requirements in a document format, goals and requirements can be organized into document sections. There is no restriction as whether two goals or requirements in one section are associated with the same or different system components.

In the following sections we describe usage in terms of requirements. The same principals apply to goals.

Requirement Sets and Component Extension Hierarchy

AADL allows users to define a component type and define extensions that add or refine features and other type elements. Similarly, users can associate one or more implementations with a component type, and component implementations can be extensions of other component implementations.

Users define a separate requirement set for the original component type and a separate requirement set for the component type extension. The requirements in a system requirement set are associated with the component classifier identified in the *for* reference of the system requirement set. Users can target a requirement to a specific element in a component type or implementation by a *for* reference in the requirement declaration.

Requirements defined for the original component type are inherited by the extension. This means that a requirement set of the extension can focus on requirement declaration for additions or refinements of the component type. In the case of refinement, a requirement declaration associated with the original

component type may need to be rephrased. In this case, the rephrased requirement can be linked to the original requirement with an *evolves* reference.

Similarly users may define requirements on component implementations. These represent requirements for the particular component variant and requirements that represent implementation constraints. Note that requirements associated with a component type apply to implementations of that type, i.e., the implementation are expected to satisfy these requirements.

Requirement Refinement

A requirement may be refined into subrequirements in order to make it verifiable. This is done by placing the refined requirement in the same system requirements set as the original, and by identifying the original in a *refines* reference.

In the ALISA workbench users indicate that requirements are verifiable by associating verification plans with requirement sets. For each requirement the verification plan contains a claim that specifies a set of verification activities to demonstrate that the requirement is met. The result of performing or executing a verification activity represents evidence that the requirement is met or not met. If all refined requirements are met, then the requirement being refined is considered verified as well.

Requirement Decomposition

When a system architecture is elaborated by defining a component implementation—that is, a blueprint—requirements for a system may be decomposed into requirements for its subsystems. Users might want to provide traceability of this decomposed requirement to the original by adding *decomposes* references to the original requirement.

Users can record a decomposed requirement in two ways: as a requirement associated with the subcomponent – identified as *for* target element; or as requirement declared for the component classifier referenced by the subcomponent.

In the first case, the decomposed requirement represents an implementation constraint from a particular use context, which is declared in a requirement set associated with a component implementation, which allows the *for* reference the subcomponent as target element. When a component is provided by a supplier as subcomponent, this use context requirement must be verified on the provided component implementation.

In the second case, the user accumulates requirements from different use contexts within their design in a single location, namely the component type referenced by all subcomponent declarations.

Requirement References

Users can reference requirements (and goals) by just their name if the context uniquely identifies them. This is true when the referenced requirement appears in the same system requirements set or when the requirement is contained in a system requirements set that is associated with a classifier in the *extends* hierarchy of the target classifier.

In some cases, requirements must be qualified with the name of the enclosing system requirements set. This is the case for references from system requirements of a subsystem to requirements of a system (decomposed requirements) or from system requirements to stakeholder goals. For qualified references, the system requirement set that contains the requirement must be identified.

Categorizing Goals and Requirements

Users can associate category labels of different category types with requirements and goals. This allows users to create filtered views of requirements and verification plans, e.g., focus on safety and

performance requirements. Predefined category types and labels have been introduced in Section Category Types and Labels.

The categorization also allows us to assess requirements coverage and verification early and throughout the development life cycle. For example, the ALISA workbench can assess whether every feature of a component type has a requirement, whether requirements regarding the state, e.g., in the form of AADL modes, and behavior has been specified, whether quality attributes of interest and exceptional conditions leading to safety hazards or security risks have been covered. Similarly, categorization of verification activities according to phase allows the ALISA workbench to ensure that potential issues in a system design are discovered as early as possible through appropriate verification activities.

Appendix A

This Appendix describes the initial expression support for ReqSpec in the OSATE 2.2.1 maintenance release of May 2015. The expression notation will be aligned with the emerging AADL Constraint Annex. Please check the online help for the latest capabilities.

Operators and their precedence in ReqSpec expressions

Precedence	**Category**	**Operator**
1 (lowest)	Logical OR	`<Boolean> or <Boolean>`
2	Logical AND	`<Boolean> and <Boolean>`
3	Equality	`<expression> == <expression>` `<expression> != <expression>`

4	Relational	`<numeric> < <numeric>` also `<=, >, >=`
		`<range> < <range>` also `<=, >, >=`
		`<numeric> >< <range>` (value included in range)
		`<range1> >< <range2>` (range1 included in range2)
		Numeric or range expressions on the left and right hand side must use the same units type, if any.

5	Additive	`<numeric> + <numeric>` also `-` `<range1> + <range2>` (smallest range containing both ranges) Numeric or range expressions on the left and right hand side must use the same units type, if any.

6	Multiplicative	`<numeric> * <numeric>`
		`<real> / <real>`
		`<integer> div <integer>` also `mod`
		`<range> * <range>` (range intersection)
		For multiplication at most one argument may have a units type.
		For division, if the right hand side argument has a unit it must be of the same type as the unit the left hand side.

7	Unary	`+ <numeric>` `- <numeric>`

		not <Boolean>

Primary expressions:

1. Unit operations for numeric expressions

    a. Unit assignment to a unit-less expression
    <primary expression> <unit name>
    Example: (x + 1) ms, where X is an integer or real value without a unit.

    b. Conversion to numeric value without unit
    <primary expression> **in** <unit name>
    Example: (2.0ms) **in** ns, evaluates to 2000

    c. Conversion to different unit
    <primary expression> **%** <unit name>
    Example: (2ms) **%** ns, evaluates to 2000ns

2. Conditional expression
if <Boolean> **then** <expression1> **else** <expression2> **endif**
Both expression1 and expression2 must have the same type.

3. Reference to model element
this.<element name>.<element name>. … .<element name>
The keyword **this** refers to the target classifier of the requirement or requirement set.

4. Reference to property value in model
<model element>#<property name>
#<property name> (short form of **this**#<property name>)
The property name must be a property or a property constant, the model element must be either a literal model element reference or a value of type model element.

5. Literals with examples

    a. Boolean literal: true, false

    b. Integer literal, optionally with unit: 2000, 20ms

    c. Real literal: 12.5, 2.5ms

    d. String literal: "strings are enclosed in double quotes"

    e. Range literal: [1 .. 5], [500ms .. 2s]
    Note that a space character is needed before the two dots.

6. Automatic type conversion between Real and Integer occurs to match the target type. For example users can assign an Integer value (numeric value without decimal point) to a *constant* of type Real. Similarly, addition of an Integer value and a Real value results in a Real value.

7. Built-in functions: the following built-in functions are supported

    a. min, max: minimum or maximum value of a range

    b. abs: absolute value

    c. floor, ceil, round: next lower, higher, closest Integer value for a given Real value

References

[ALISA 2016]

Architecture-Led Incremental System Assurance (ALISA) Workbench. https://github.com/osate/alisa

[Delange 2016]

Delange, J., Feiler, P., Ernst, N., *Incremental Life Cycle Assurance of Safety-Critical Systems*, Proc. 8th European Congress on Embedded Real Time Software and Systems, Jan 2016. http://www.erts2016.org/inc/telechargerPdf.php?pdf=paper_13

[Eclipse 2015]

Eclipse. Xtend. 2015. http://www.eclipse.org/xtend

[FAA 2009]

Federal Aviation Administration. *Requirements Engineering Management Handbook*. DOT/FAA/AR-08/32. FAA. 2008. http://www.faa.gov/aircraft/air_cert/design_approvals/air_software/media/AR-08-32.pdf

[Feiler 2015]

Feiler, Peter. *Requirements and Architecture Specification of the Joint Multi-Role (JMR) Joint Common Architecture (JCA) Demonstration System*. CMU/SEI-2015-SR-031. Software Engineering Institute, Carnegie Mellon University. 2015. http://resources.sei.cmu.edu/library/asset-view.cfm?assetid=447184

[IEEE 2009]

Institute of Electrical and Electronics Engineers. IEEE Standard 830-1998: Recommended Practice for Software Requirements Specifications. IEEE Standards Association. 2009.

[Lamsweerde 2009]

van Lamsweerde, Axel van. *Requirements Engineering: From System Goals to UML Models to Software Specifications*. Wiley. 2009.

[Nolan 2011]

Nolan, A. J.; Abrahao, S.; Clements, P.; and Pickard, A. Managing Requirements Uncertainty in Engine Control Systems Development. 259–264. *19th IEEE International Requirements Engineering Conference (RE)*. Aug. 29–Sep. 2, 2011. http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=6051622&tag=1

[OMG 2015]

Object Management Group. OMG Systems Modeling Language. OMG. 2015. http://www.omgsysml.org

[OSATE 2016]

Open Source AADL Tool Environment (OSATE). https://wiki.sei.cmu.edu/aadl#OSATE.

[SAE 1996]

SAE International. ARP4761: Guidelines and Methods for Conducting the Safety Assessment Process on Civil Airborne Systems and Equipment. SAE. 1996.