Requirement Specification for a System

This document describes a textual requirement specification notation, called *ReqSpec. It* has been derived from the draft Requirements Definition and Analysis Language (RDAL) Annex for use with the SAE Architecture Analysis & Design Language (AADL).

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, i.e., stakeholder requirements, and requirements, i.e., system requirements. Goals are expressed by *goal* declarations and requirements by *requirement* declarations.

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

Goal documents are represented by files with the extension *goaldoc* and requirement documents by files with the extension *reqdoc*. A goal document contains a *document* declaration that contains *document section* declarations and *goal* declarations. A requirement document contains a *document* declaration that contains *document section* declarations and *requirement* declarations.

Goals organized according to the architecture structure are represented by files with the extension *goals*. Each file consists of a *stakeholder goals* declaration for a specific architecture component that contains a set of *goal* declarations.

Requirements organized according to the architecture structure are represented by files with the extension *reqspec*. Each file consists of a *system requirements* declaration for a specific architecture component that contains a set of *requirement* declarations, or it consists of a *global requirement* declaration that contains a set of reusable *requirement* declarations that can be included in different system requirement declarations.

The *stakeholder goals*, *system requirements, goals document*, *requirements document* constructs can have names of <dot> separated identifiers. These names can be used to qualify goals and requirement contained in them.

For *stakeholder goals* users are expected to use the file extension *goals*, for *system requirements* the extension *reqspec*, for a *goals document* the extension *goaldoc*, and for *requirements document* the extension *reqdoc*.

Goals and requirements can be referenced by their identifier or by qualifying them with their container name. Note that document section identifiers are not used for qualification of names.

Stakeholder Goals

ReqSpec has the *Goal* construct to represent individual stakeholder requirements.

Stakeholder goals can be organized in two ways:

- By the *StakeholderGoals* construct to represent a collection of goals for a particular system that is represented as an AADL component.
- By the *Document* construct that contains goals, possibly organized into (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 )?
[
( quality ( <Qualitylabel> )+ )?
( category ( <Categorylabel> )+ )?
( description  Description )?
( ConstantVariable )*
( 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>
DocReference ::= URI to an element in an external document

WhenCondition ::=
  when in modes <Mode> ( , <Mode> )*
  |
  when in error state <ErrorState> ( , <ErrorState> )*
  |
  <Java like 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 goals 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 *StakeholderGoals* container specifies the component classifier of the system of interest.
- *Quality:* list of quality attribute labels (without comma separation) that this verification method addresses. It is used when defining filter criteria for assurance tasks.
- *Category*: list of user defined category labels (without comma separation) in which this verification activity should be performed. It is used when defining assurance tasks.
- *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).

- *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 Goals Construct

The *StakeholderGoals* 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.

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

A *StakeholderGoals* declaration has the following elements:

- *QualifiedName*: name as a <dot> separated sequence of identifiers 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.
- *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 *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).
- 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 *SystemRequirements* declaration as a set or set individual requirements through an *include* statement. The system identified by the *SystemRequirements 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, SystemRequirements*, and *GlobalRequirements* 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 ( <Categorylabel> )+ )?
( description  Description )?
( Variable )*
( WhenCondition )?
( Predicate )?
( rationale String )?
( mitigates ( <Hazard> )+ )?
( refines ( <Requirement> )+)?
( decomposes ( <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 requirement specification container).
- *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.
- *Quality:* list of quality attribute labels (without comma separation) that this verification method addresses. It is used when defining filter criteria for assurance tasks.
- *Category*: list of user defined category labels (without comma separation) in which this verification activity should be performed. It is used when defining assurance tasks.
- *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).
- *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.
- *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 Requirements Construct

The *SystemRequirements* construct is a container for *Requirement* declarations. It is typically used to group together system requirements for a particular system. It represents a name scope for the goal declarations contained in it, i.e., a requirement is referenced by the *SystemRequirements* name and the *Requirement* name – separated by a dot.

```
SystemRequirements ::=
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 *SystemRequirements* declaration has the following elements:

- *QualifiedName*: name as a <dot> separated sequence of identifiers 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 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 StakeholderGoals 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 Requirements and Requirement Constructs

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

GlobalRequirements ::=

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

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

[

(**description** String)?

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

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

(Variable)*

(GlobalRequirement)*

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

]

The *Requirement* 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 is intended to represent existing stakeholder goal or system requirement documents that are imported into a ReqSpec representation. In such documents goals and requirements are organized into document sections. Once an existing stakeholder requirement or system requirement document has been imported into ReqSpec, its goals can be associated with an AADL model enabling traceability and consistency checks on stakeholder goals and system requirements.

A *Document* contains a set of document sections, stakeholder goals, or system requirements. A

DocumentSection can recursively contain document sections, stakeholder goals, or system requirements. In the case of stakeholder goals the file extension *goaldoc* is used, while in the case of requirements the file extension *reqdoc* is used.

A *Document* represents a name scope for the *goal* and *requirement* declarations contained in it, i.e., a goal or requirement is referenced by the *Document* name and the *goal* or *requirement* name – separated by a dot. Note that document sections do not contribute to name scoping, i.e., *goal* and *requirement* declarations have to be unique within the requirement document.

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

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

A *RequirementDocument* 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

Constant and Compute Variables

ReqSpec allows the user to introduce variables with values to localize common changes to a stakeholder goal or system requirement. They act as parameters that can be referenced by Description elements in Goal and Requirement declarations and by Predicate elements in Requirement declarations. These variables are called *ConstantVariable*. Their values can be any numeric value with optional measurement unit, as well as Booleans, strings, and enumeration literals. Acceptable measurement units are any unit defined as Units literals in property sets of the AADL core language.

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 Description elements of verification activities to present verification results. Note that specification of verification activities are expressed by the *Verify* notation that is part of incremental life-cycle assurance.

```
Variable ::=
ConstantVariable | ComputedVariable

ConstantVariable ::=
val ( Type )? Name = Value

ComputedVariable ::=
compute Name

Type ::= <base type>
```

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

Global Constant Variables

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:

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

Requirement Predicates

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` The user specifies a predicate on expected values vs. computed values resulting from a verification activity. The expression is typically one or more comparison expressions that reference variables and specify constants.
- Full predicate expression: `predicate <general Java like expression>` Java like expression syntax alligns with the Xbase.

User-definable Categories

The *categories* notation allows users to define three categories of labels:

- *Quality*: to represent quality attributes. These labels can be used to categorize goals, requirements, and verification methods.
- *Phase*: to represent development phases. These labels can be used to categorize verification activities.
- *Category*: to represent selection categories. These labels can be used to categorize goals, requirements, verification methods, and verification activities.

These categories are declared with goals and requirements, as well as verification methods and verification activities. The latter are part of the *Verify* notation in support of incremental lifecycle assurance. For requirements the *quality* category is used to assess coverage. The categories are also used for filtering subsets of goals, requirements, verification methods and activities.

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

```
Categories ::=
( category [ ( SelectionCategoryLabel )+ ] )?
( quality [ ( QualityCategoryLabel )+ ] )?
( phase [ ( PhaseCategoryLabel )+ ] )?
```

Each category label must be globally unique.

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

The concept of change uncertainty is used to assess how likely requirements may change and what the impact of such change is. This is a common technique to prioritize entities. For example, in the Architecture Tradeoff Analysis Methods (ATAM®) criticality and difficulty of change are used to prioritize use cases. [Nolan 2011] uses pairs of likelihood and impact measures to let experts assess the change uncertainty in system requirements and by focusing on those with a high change uncertainty up to 50% reduction in requirement changes can be achieved. These measures have been proposed to be included in RDAL. Safety analysis practices, e.g., SAE ARP4761, use likelihood of occurrence and severity of impact to prioritize hazards.

In the context of ReqSpec we use a single likelihood measure called *volatility* and a single impact measure called *impact* as sufficient measures to identify high payoff opportunities for reducing requirement change and in the context of safety and security hazards for reducing safety and security risks.

Satisfiable Requirements

RDAL distinguishes between *verifiable* and *satisfiable* requirements. Verifiable requirements must be met (true/false result). Satisfiable requirements are quantified and are expected to be met to a certain degree. Satisfiable requirements are used in design tradeoffs where different architecture design decisions result different degrees to which a set of satisfiable requirements are met.

In Reqspec we support the specification of desirable target values that a system design is expected to satisfy. We have done so in the context of a *value predicate* for a requirement. The value predicate specifies the value that must be met (*verifyable* requirement). This predicate can be augmented with a desirable target value in the form of

`upto` `<value>`

or

`downto` `<value>`.

Guidelines for Use of ReqSpec

This section provides some general guidelines on the use of ReqSpec within the OSATE or similar modeling environment and assumes that you are familiar with the use of the environment.

ReqSpec Files

You create files that contain stakeholder goals using the extension *goals* and files containing system requirements using the extension *reqspec*. For stakeholder goal documents and system requirements documents you use the extensions *goaldoc* and *reqdoc*, respectively.

You can place all of these files in a single folder within a project that contains your AADL model, e.g., in a folder named *requirements* at the same level within a project as a folder that contains AADL packages. However, this is just a convention and, for example, you may prefer to place the *reqspec* files in a separate AADL project from that containing your AADL model. If they are in separate projects you must set the project references for the projects within OSATE/Eclipse.

You may place one or more sets of stakeholder goals, system requirements, and associated documents in a single file. As a convention, we recommend you create a separate file for each *system requirements* set or *stakeholder goals* set belonging to a system. In other words, for each AADL system type or other component type representing a system, e.g., device or abstract, you should create a separate file.

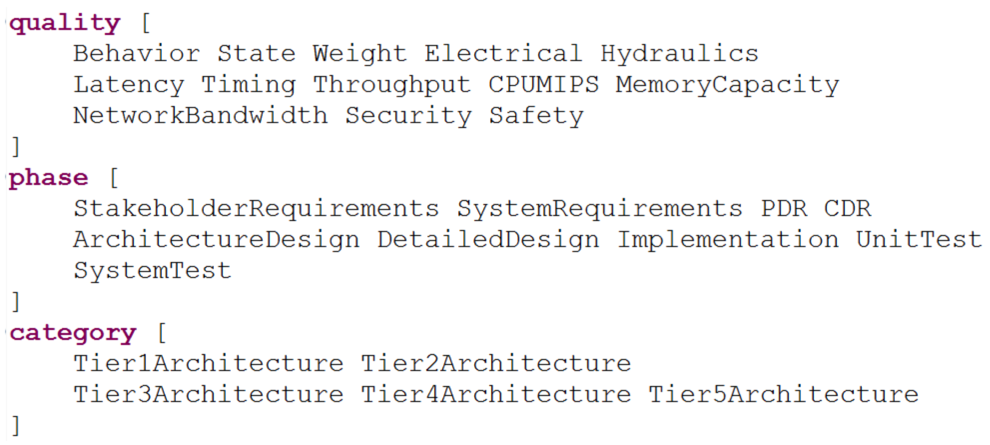
System Requirements and Stakeholder Goals

The names for system requirements sets must be globally unique among all system requirements sets within the scope defined in your workspace. For example, they must be unique within a project or across projects within a workspace, if project references are set for those projects. Similarly, stakeholder goals sets must have a globally unique name.

You can use <dot> separated IDs as the name *for a system requirements* set. By convention you can use names that are the equivalent of the qualified component classifier name (e.g. aircraft_Pkg::Autopilot) identifying the target system with "::" replaced by "." (e.g. aircraft_Pkg.Autopilot). Note that a *system requirements* set and a *stakeholder goals* set for a system can have the same name that is globally unique.

Categorizing Requirements

You can associate category labels with requirements and goals into groups using the *categories* notation. You assign categories to stakeholder goals and system requirements. For example you can define categories *latency* and *response_time*. Figure 1 shows an example of category declarations.



```
quality [
    Behavior State Weight Electrical Hydraulics
    Latency Timing Throughput CPUMIPS MemoryCapacity
    NetworkBandwidth Security Safety
]
phase [
    StakeholderRequirements SystemRequirements PDR CDR
    ArchitectureDesign DetailedDesign Implementation UnitTest
    SystemTest
]
category [
    Tier1Architecture Tier2Architecture
    Tier3Architecture Tier4Architecture Tier5Architecture
]
```

Figure 1: Category, Phase, and Quality Labels Example

Individual category labels can be referenced in a goal, requirement, or in sorting or filter conditions for a tool.

Classifier Extensions, Implementations, and System Requirements

System requirements are inherited by type extensions, by implementations and their extensions, and by instances expressed by subcomponent declarations. In other words, system requirements must be satisfied by all instances of component types or implementations, whose classifiers can be traced back through the *extends* hierarchy (as defined by core AADL2) to the system requirements set.

A component classifier that is an extension of another component classifier with a system requirements set, can also have a system requirements set. The requirements associated with the extension must either be new requirements, or requirements that evolve (expressed by the *evolves* reference) from a requirement associated with the original component classifier. With the *evolves* reference, the original requirement is overwritten for the extension, but still holds for the original.

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.

Verification plans (in the Verify notation) associate verification activities with requirements – clearly documenting their verifiability. Every leaf requirement in a refinement hierarchy must have associated verification activities. Enclosing requirements in the hierarchy may have – but are not required to have – verification activities. In the former case satisfaction of the leaf requirement claims is sufficient for the enclosing requirement, while in the case where an enclosing requirement has verification activities, the verification satisfaction is necessary but still requires the passing of verification activities associated with the enclosing requirement.

Requirement Decomposition

When a system architecture is elaborated by defining a component implementation, i.e., a blue print, requirements for a system may be decomposed into requirements for its subsystems. It may be desirable to provide traceability of this decomposition. This is accomplished by *decomposes* references to the original requirement.

The decomposed requirement can be recorded in two ways:

- As a requirement associated with the component classifier of the subcomponent. The requirement is placed in the system requirements set of the subcomponent classifier. There may be multiple instances of the same subcomponent classifier, i.e., different use context, each with a system requirements set. Each of these use contexts imposes decomposed (derived) requirements that must all be met by instances of one type of component. The resulting requirements associated with the subcomponent classifier represent the common requirements across these use contexts. For example, suppliers may provide both a component type (spec sheet) that must meet the specified requirements (refinement substitution rules of core AADL2 apply), and component implementations (blue prints) that can be verified to satisfy the requirements.
- As a requirement associated with the subcomponent itself. In this case the requirement is specified as an implementation constraint on the subcomponent by placing the requirement with the enclosing classifier, typically the component implementation. This is done by identifying the subcomponent as target element in the requirement *for* clause. In this case, a separate decomposed requirement is maintained for each use context and treated as implementation constraint that the classifier supplied for the subcomponent (e.g. a spec sheet from a supplier) must meet.

Requirement References

Requirements (and goals) can be referenced by just their name if the context uniquely identifies them. This is the case when the referenced requirement is 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 (decomposes) or from system requirements to stakeholder goals. For qualified references the system requirements set actually containing the requirement must be identified.

References

[FAA 2009] Federal Aviation Administration. *Requirements Engineering Management Handbook* DOT/FAA/AR-08/32. 2008. http://www.faa.gov/aircraft/air_cert/design_approvals/air_software/media/AR-08-32.pdf.

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

[Feiler 2015] Peter Feiler, *Requirements and Architecture Specification of the Joint Multi-Role (JMR) Joint Common Architecture (JCA) Demonstration System*. Special Report CMU/SEI-2015-SR-031, Software Engineering Institute. Oct 2015.

[Nolan 2011] Nolan, A.J.; Abrahao, S.; Clements, P.; Pickard, A.; *Managing requirements uncertainty in engine control systems development*, Requirements Engineering Conference (RE), 2011 19th IEEE International, vol., no., pp.259-264, Aug. 29 -Sept. 2 2011. DOI: 10.1109/RE.2011.6051622 URL: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6051622&isnumber=6051621