Software Engineering Institute
Carnegie Mellon University

A Requirement Specification Language for AADL

Peter H. Feiler
Julien Delange
Lutz Wrage

June 2016

TECHNICAL REPORT
CMU/SEI-2016-TR-008

Software Solutions Division

http://www.sei.cmu.edu

This material is based upon work funded and supported by the Department of Defense under Contract No. FA8721-05-C-0003 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center.

This report was prepared for the
SEI Administrative Agent
AFLCMC/PZM
20 Schilling Circle, Bldg 1305, 3rd floor
Hanscom AFB, MA 01731-2125

DM-0003425

Table of Contents

List of Figures

Abstract

This report describes a textual requirement specification language, called *ReqSpec*, for the Architecture Analysis & Design Language (AADL). It is based on the draft Requirements Definition and Analysis Language Annex, which defines a meta-model for requirement specification as annotations to AADL models. A set of plug-ins to the Open Source AADL Tool Environment (OSATE) toolset supports the ReqSpec language. Users can follow an architecture-led requirement specification process that uses AADL models to represent the system in its operational context as well as the architecture of the system of interest. ReqSpec can also be used to represent existing stakeholder and system requirement documents. Requirement documents represented in the Requirements Interchange Format can be imported into OSATE to migrate such documents into an architecture-centric virtual integration process. Finally, ReqSpec is an element of an architecture-led, incremental approach to system assurance. In this approach, requirements specifications are complemented with verification plans. When executed, these plans produce evidence that a system implementation satisfies the requirements. This report introduces the ReqSpec notation and illustrates its use on an example.

1 Introduction

This report describes a textual requirement specification language, called *ReqSpec*, for the Architecture Analysis & Design Language (AADL). 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 of ReqSpec is to support the elicitation, definition, and modeling of requirements for real-time embedded systems in an iterative process. ReqSpec supports the refinement of requirements along with the system design; qualitative and quantitative analysis of the created requirements specification; and, finally, verification of the associated system architecture models to ensure that they meet the requirements.

The draft RDAL Annex defines a meta-model for concepts related to requirement specification. These concepts were drawn from the Requirements package of the Object Management Group (OMG) Systems Modeling Language (SysML) [OMG 2015]. In addition, we have added many other concepts to cover important aspects of requirements engineering methods not included in SysML; these additional concepts come from the Federal Aviation Administration (FAA) *Requirements Engineering Management Handbook* [FAA 2009], the KAOS[1] method [Lamsweerde 2009], and IEEE Standard 830-1998: Recommended Practice for Software Requirements Specifications [IEEE 2009].

ReqSpec distinguishes between stakeholder requirements, referred to as *goals*, and system requirements, referred to as *requirements*. Goals express stakeholder intent and may conflict with each other, while system requirements represent a contract that a system implementation is expected to meet.

The ReqSpec notation accommodates several capabilities. 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. The report *Requirements and Architecture Specification of the Joint Multi-Role (JMR) Joint Common Architecture (JCA) Demonstration System* introduced this process [Feiler 2015].

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

We proceed by first introducing the syntax of the ReqSpec notation in Section 2. In Section 3, we provide guidelines for using ReqSpec. Then, in Section 4, we illustrate its use in ALRS and describe the migration of existing requirement documents into an ALRS context.

[1] KAOS stands for both Knowledge Acquisition in Automated Specification and Keep All Objectives Satisfied [Lamsweerde 2009]. It is a goal-oriented approach to capturing software requirements.

2 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. Users 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 a 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.

2.1 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 *GoalsDocument* and *DocumentSection* constructs are described in Section 2.2.3.

2.1.1 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 ( String )? )?
  ( stakeholder ( <Stakeholder> )+ )?
  ( see goal ( <Goal> )+)?
  ( see document ( DocReference )+ )?
  ( issues (String)+ )?
  ( ChangeUncertainty )?
]

Title ::= String

TargetClassifier ::= <AADL Component Classifier>

TargetElement ::= <ModelElement>

CategoryReference ::= <CategoryType>.<CategoryLabel>

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

DocReference ::= URI to an element in an external document
```

```
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).
- *Title:* a short descriptor of the goal. This optional element may be used as a more descriptive label than the name.
- *For:* If present, it identifies the target of the goal within a system. The target is a model element within the classifier, such as 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 2.5) 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, it can be a sequence of strings, a reference to the classifier/model element identified by the *for* element (expressed by the keyword *this*), or references to variables (defined next).
- Set of *Constant:* Constants are used to parameterize goal and requirement specifications. Many changes to a goal or requirement appear 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 separately specified verification plans. See Section 2.4 for details on variables.
- *WhenCondition:* the condition under which the requirement applies. The condition is a set of AADL modes (operational modes), error behavior states (failure modes) specified by the Error Model Annex Version 2 (EMV2), or a general expression on model properties using the syntax of value predicate expressions (see the Appendix 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; it represents a more detailed specification of a goal.
- *Conflicts with:* references to other goals that this goal is in conflict with.
- *Evolves:* references to other goals that this goal evolves from. This allows for tracking of goals as they change over time.
- *Dropped*: If this keyword is present, the goal has been dropped and may be replaced by a goal that has evolved from this goal. Users can provide a rationale for dropping the 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 it expressed as a Uniform Resource Identifier (URI). This element 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. The concept of change uncertainty is based on the work of Nolan and colleagues [Nolan 2011]. See Section 2.7 for details on uncertainty specifications.

When a goal is used in a *GoalsDocument*, the *for* clause can consist of a target description string or a classifier reference, optionally followed by a target element reference within that classifier. These references allow goals found in existing stakeholder goals documents to be mapped into an architecture model so that users can identify different systems for different goals in the same document or document section.

2.1.2 The Stakeholder Goals Construct

The *StakeholderGoalSet* construct is a container for *Goal* declarations. It is typically used to group together stakeholder goals for a particular system, namely, all goals that are associated with an AADL component type or implementation.

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

QualifiedName ::= Identifier ( . Identifier )*
```

A *StakeholderGoalSet* declaration has the following elements:

- *QualifiedName:* a unique name as a <dot>-separated sequence of identifiers.
- *Title:* a short descriptor of the stakeholder goal set. This optional element may be used as a more descriptive label than the name.
- *For:* identifies the target of the set of stakeholder goals and references 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 the set can be referenced without qualification.
- *Description:* a textual description of the stakeholder goals for a specific system. In its most general form, it can be a sequence of strings, a reference to the classifier/model element identified by the *for* element (expressed by the keyword *this*), or references to constants.

- Set of *Constant:* Constants are used to parameterize goal and requirement specifications. Many changes to a goal or requirement appear 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 separately specified verification plans. See Section 2.4 for details on variables.
- Set of *Goal:* a set of goal declarations. All contained goals are intended to be associated with the system represented by the classifier.
- *See document:* reference to an external document. This element is used to record the fact that the origin of the stakeholder requirements in this container is the identified document.
- *Issues:* allows users to record issues that may be encountered as a set of textual notes (Strings).

2.2 System Requirements

ReqSpec uses 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 *RequirementsDocument* construct (see Section 2.2.3). When representing system requirements in an AADL model of the system and its operational context, users employ the *SystemRequirementSet* construct 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 a *GlobalRequirementSet* can then be included in a *SystemRequirementSet* declaration as a set or as individual requirements through an *include* statement.

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, such as *abstract* or *device*.

2.2.1 The System Requirement Construct

The *SystemRequirement* construct represents a requirement for a specific system.

```
SystemRequirement ::=
requirement Name ( :  Title )?
  ( for TargetElement )?
[
  ( category (CategoryReference )+ )?
  ( description Description)?
  ( Variable )*
  ( WhenCondition )?
  ( Predicate )?
  ( rationale String )?
  ( mitigates ( <Hazard> )+ )?
  ( refines ( <Requirement> )+)?
  ( decomposes ( <Requirement> )+)?
  ( inherits ( <Requirement> )+)?
  ( evolves ( <Requirement> )+)?
```

```
  ( dropped ( String )? )?
  ( 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 a more descriptive label than the name.
- *For:* If present, it identifies the target of the requirement within a system. The target is a model element within the classifier, such as a port, end-to-end flow, or subcomponent. The enclosing *SystemRequirementSet* container specifies the component classifier of the system of interest.
- *Category:* list of category references without comma separation (see Section 2.5) 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, it can be a sequence of strings, a reference to the classifier/model element identified by the *for* element (expressed by the keyword *this*), or references to variables (defined next).
- Set of *Variable:* Constants and compute variables are used to parameterize requirement specifications (see Section 2.4). Many changes to a goal or requirement appear 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 separately specified verification plans. See Section 2.4 for details on variables.
- *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 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 set container. See Section 2.4.3 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. This element 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 that this goal evolves from. This element allows for tracking of goals as they change over time.
- *Dropped*: If this keyword is present, the requirement has been dropped and may be replaced by a goal that has evolved from this goal. Users can provide rationale for dropping the requirement.
- *Development Stakeholder:* reference to a stakeholder from the development team, such as 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 *StakeholderGoalSet* or a *GoalsDocument*.
- *See requirement:* reference to a system requirement in an imported system requirement document (*RequirementsDocument*).
- *See document:* reference to an external document and optional element within expressed as a URI. This element records 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.

When a requirement is declared in a *RequirementsDocument*, the *for* clause can consist of a target description string or a classifier reference followed by a target element reference within that classifier. These references allow requirements found in existing system requirements documents to be mapped into an architecture model so that users can identify different systems for different requirements within the same document or document section.

2.2.2 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 <GlobalConstantSet>* )?
[
  ( description  Description )?
  ( Variable )*
  ( SystemRequirement )*
  ( include <GlobalRequirementSet or global requirement> ( for ComponentCategory+ |
self )
```

```
  ( see document ( DocReference )+ )?
  ( see goals ( <StakeholderGoals or GoalsDocument> )+ )?
  ( issues (String)+ )?
]
```

A *SystemRequirementSet* declaration has the following elements:

- *QualifiedName:* a unique 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.
- *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, it can be a sequence of strings, a reference to the classifier/model element identified by the *for* element (expressed by the keyword *this*), or references to variables (defined below).
- *See document:* reference to an external document. This element 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 2.4). Many changes to a goal or requirement appear 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 separately specified verification plans. See Section 2.4 for details on variables.
- 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 requirement set or a global requirement inside a global requirement set. 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. *Self* indicates that the global requirement applies only to the component itself.
- *Issues:* allows users to record issues that may be encountered as a set of textual notes (Strings).

2.2.3 The Global Requirement Set and Global Requirement Constructs

The *GlobalRequirementSet* construct is a container for *GlobalRequirements* declarations. It is used to group together system requirements that can be applied to a number of systems; they then represent a reusable set of requirements that can be included with a *SystemRequirementSet* declaration.

```
GlobalRequirements ::=
global requirements QualifiedName ( :  Title )?
( use constants <GlobalConstantSet>* )?
[
```

```
  ( description  Description )?
  ( see document ( DocReference )+ )?
  ( see goals ( <StakeholderGoals or GoalsDocument> )+ )?
  ( Variable )*
  ( GlobalRequirement )*
  ( issues (String)+ )?
]
```

The *GlobalRequirement* construct represents a reusable requirement specification that is generally applicable, may be restricted to certain AADL component categories, or may applicable to all connections.

```
GlobalRequirement ::=
requirement Name ( :  Title )?
  ( for ComponentCategory+ | connection )?
[
  // Same as for SystemRequirement
]

ComponentCategory ::= abstract | system | <other AADL component categories>
```

2.3 Documents and Document Sections

The *Document* construct allows users to organize stakeholder goals or system requirements into document sections to mirror existing documentation. This construct supports the 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* contains only stakeholder goals, while a *RequirementsDocument* contains only system requirements.

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

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

RequirementsDocument ::=
document QualifiedName ( :  Title )?
[
```

```
  ( description  String )?
  ( Requirement | RequirementsDocumentSection )+
  ( issues (String)+ )?
]

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

GoalsDocument and *RequirementsDocument* declarations have the following elements:

- *QualifiedName:* a unique name as a <dot>-separated sequence of identifiers.
- *Title:* a short descriptor of the stakeholder goal container. This optional element may be used as a 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.
- *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 a 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.
- *Issues:* allows users to record issues that may be encountered as a set of textual notes (Strings).

2.4 Variables and Predicates

2.4.1 Constants and Computed 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; and 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 for expression syntax details. The type is inferred from the value when not explicitly declared. Users can optionally specify that the

value of a property identified by the **as** for the model element must be the same as the *constant* value.

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 (as <PropertyName> )?

ComputedVariable ::=
compute Name : TypeSpec

TypeSpec ::= BaseType | typeof <PropertyName>

BaseType ::= boolean | string | integer (units <UnitsTypeName> )?
    | real (units <UnitsTypeName> )? | model element | <PropertyTypeName>
```

2.4.2 Reusable Global Constants

In some cases, users might want to define a set of constants that they can reference within the system requirement specification of any system component. Such *global constants* are defined in *global constant sets* in files with the extension *constants*. The following syntax is used in those files:

```
GlobalConstantSet ::=
  constants QualifiedName
[ Constant+ ]
```

These global constant sets are then made accessible to a stakeholder goal set, system requirement set, or global requirement set through a *use constants* declaration. This allows users to reference these constants without qualification.

2.4.3 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 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 specify a predicate without needing to know the exact syntax of a particular notation.

- Value assertion: `value predicate Expression`
  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 *Computed-Variable*. Constants and computed variables are referenced by their names. AADL property and property constant references are prefixed by `#`. The expression language includes the operators **and**, **or**, **not**, ==, **!=**, >=, <=, >, <, >< (contained in range), +, -, *****, /, **div** (integer divide), and **mod**. It supports parentheses and functions such as min, max, round, and abs. See the Appendix for details.
  For example, a user specifies `ActualCPUBudget <= MaxCPUBudget`, where *MaxCPUBudget* is a constant and *ActualCPUBudget* is a computed variable.
- Behavioral assertion: A future version of ReqSpec will support behavioral predicate syntax. Meanwhile users can specify behavioral assertions through the *informal predicate* construct.

2.5 User-Definable and Predefined Category Types and Labels

ReqSpec allows users to associate category labels with goals and requirements. Users can also associate category labels 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, such as for verifying key quality attributes or 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 be unique among category types. Labels must be unique within a category type. A category is referenced by its type and label—for example, *Kind.Guarantee*.

The following category types have been predefined in 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 a safety hazard or security vulnerability that the requirement addresses.
  - *Constraint:* a constraint on the implementation of a system, typically on the subcomponents and their properties, states, and connectivity.
  - *Consistency:* a consistency constraint between information in ReqSpec and an AADL model or between models. For example, the values of ReqSpec constants must be 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*, and *ElectricalPower*.
- *Phase:* to represent development phases, including *SystemRequirements*, *ArchitectureDesign*, *PDR*, *CDR*, *DetailedDesign*, *Implementation*, *UnitTest*, and *SystemTest*.
- *Layer:* tier of a layered architecture, including *Tier1*, *Tier2*, *Tier3*, *Tier4*, and *Tier5*.

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

2.6 Stakeholders and Their Organizations

The *organization* notation allows users 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*. This example shows how to declare organization and stakeholder names and the optional elements users can include for each stakeholder.

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

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

2.7 Change Uncertainty

Various techniques are commonly used to prioritize change. 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] and derive design assurance levels (DALs) to focus on high-payoff reduction of safety risk.

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) or frequent changes in the operational environment.

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

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

2.8 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> )+
```

3 Guidelines for Using ReqSpec with AADL Models

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

3.1 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 extensions. Users can place these files in folders within a project that contains the AADL model; for instance, users 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, users must set the project references for the projects within OSATE/Eclipse.[2]

3.2 Defining Stakeholder Goal and System Requirement 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 users name these goal sets or requirement sets with the same name as the qualified name of the component classifier using "." instead of "::" to separate identifiers.

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 to 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.

3.3 Requirement Sets and Component Extension Hierarchy

AADL allows users to define a component type and to 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

[2] Set the project references using the pull-down menu Project → Properties → Project References, as shown in Figure 2.

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. For 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, so the implementation is expected to satisfy these requirements.

3.4 Requirement Refinement

A requirement may be refined into subrequirements to provide a more precise specification and to make the requirement verifiable. Users do this 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.

3.5 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* the target element, or as a 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. The constraint is declared in a requirement set associated with a component implementation, which allows the *for* reference to the subcomponent as the target element. When a supplier provides a 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.

3.6 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.

3.7 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, for example, to focus on safety and performance requirements. Section 2.5 introduced the predefined category types and labels.

The categorization also allows users to assess requirements coverage and verification early and throughout the development lifecycle. 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 have been specified, and 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.

4 Example Use of ReqSpec

This section describes the use of ReqSpec in OSATE. First, we describe how to create ReqSpec files in OSATE. Then we illustrate several use scenarios on an example.

4.1 Installing ReqSpec and ALISA in OSATE

The most recent release of OSATE (2.2.1) includes ReqSpec. Users can extend an installation of OSATE [OSATE 2016] with the ALISA extension [ALISA 2016]. Users can also include a copy of an Eclipse project called AlisaBasics,[3] which contains the predefined category types and a verification method registry for the analysis plugins available in OSATE. This project and example projects using ReqSpec and ALISA are available at https://github.com/osate/alisa-examples.

4.2 ReqSpec Declarations in OSATE

In this section, we describe how ReqSpec files are created, updated, and analyzed through an Xtext-based textual editor. A navigator, forms, and a graphics-based user interface are currently in development.

Figure 1 shows the AADL Navigator on the left. The SituationalAwarenessSystem project contains AADL model packages organized into subfolders. In this example, we put the ReqSpec files into a separate folder called *Requirements*, where the requirements folder is at the same level as the folder packages that contain the AADL model. Note the different extensions used to distinguish between different types of ReqSpec files.

The right-hand side shows a specification of system requirements. The editor understands the syntax of the *ReqSpec* notation. It provides syntax coloring and ensures that each element of a stakeholder specification, such as the phone number, is specified at most once. It also supports content assist. When the user types <Ctrl> <spacebar>, the editor provides syntactically legal choices.

[3] In the near future, AlisaBasics will be included automatically.

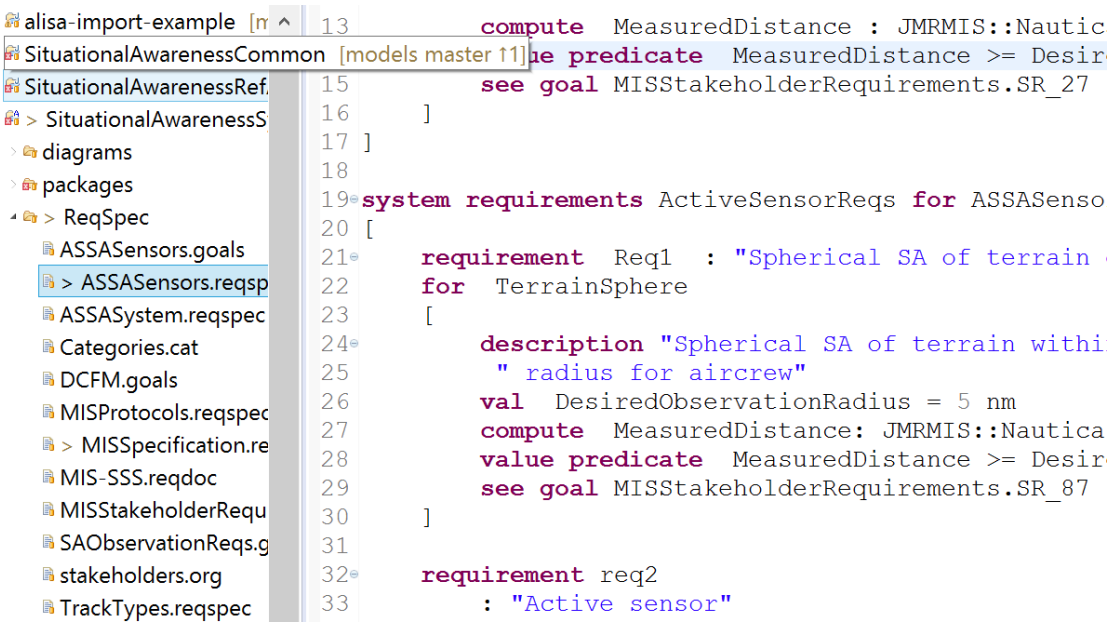

Figure 1: A Project with ReqSpec and Organization Files

The ReqSpec files could be placed in a separate project if desired. In that case, the user will have to add a Project Reference into the project containing the ReqSpec files to reference the project containing the AADL models. This tells the ReqSpec tool where to find the AADL model.

Use the properties dialog to set the project references for the project containing the ReqSpec files. To do this, select the project in the AADL navigator and invoke the properties dialog through the context menu. An example dialog is shown in Figure 2.

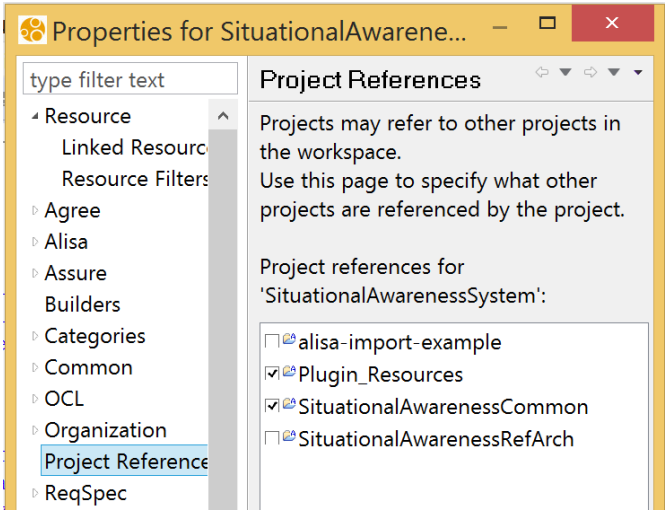

Figure 2: Dialog to Set Project References

New ReqSpec (reqspec, goals, reqdoc, goaldoc), Category (cat), or Organization (org) files are created by invoking *File/New/File* and specifying a file name with the appropriate extension.

Figure 3 shows a set of requirement specifications for the ASSA system. These requirements originally came from a requirement document; using the import tool, we migrated them into a ReqSpec annotation in an AADL model.

```
system requirements ASSASystemReqs for ASSASystem::ASSASystem
[
    requirement req1 : "support offboard and onboard mission planning"
    for AMPSInterface
    [
        description "support offboard and onboard mission planning and replanning"
        see goal  MISStakeholderRequirements.SR_73 MISStakeholderRequirements.SR_74
        MISStakeholderRequirements.SR_75 MISStakeholderRequirements.SR_76 MISStakehold
    ]

    requirement req2 : "exchange planning information" for AMPSInterface
    [
        description "Planning information is communicated through the AMPSInterface."
    ]

    requirement req3 for ThreatAlerts
    [   description "alert other manned and unmanned systems"
        see goal MISStakeholderRequirements.SR_81
    ]
```

Figure 3: Requirement Specification for the ASSA System

The top-level requirement specification (e.g., `for ASSASystem::ASSASystem`) identifies the classifier of the ASSA system. The reference is qualified by the package name containing the classifier. These references are hyperlinked to their target. When the user holds down the <Ctrl> key while pausing the cursor over the reference, it appears as a hyperlink (i.e., underlined) that can be followed by clicking on it. Navigation by hyperlink is tracked in a navigation history. Users can return to the reference origin via navigational commands or toolbar buttons: 

The first requirement indicates that it is associated with an interface feature of the ASSA system called the AMPSInterface. This association reflects the fact that it is a requirement for the interaction between the ASSA system and an Aviation Mission Planning System (AMPS). The *See goal* elements identify several stakeholder goals that reflect the need for an interaction between the ASSA system and AMPS.

The second requirement is for the same interface feature and in its original text indicates the name of the interface for the interaction with a mission planning system.

The third requirement is associated with a different interface feature of the ASSA system.

Figure 4 illustrates a requirement with a parameterized value. The value of the desired observation radius is captured in the variable called *DesiredObservationRadius*. This variable is used in the requirement description and in the requirement predicate. The requirement predicate assures that any *MeasuredDistance* result from a verification activity is at least as large as the desired observation radius. Finally, the last line in this figure shows that the stakeholder requirement for this system requirement can be found as a goal in an imported requirement document.

```
system requirements ActiveSensorReqs for ASSASensors::ActiveTerrainSensor
[
    requirement  Req1  : "Spherical SA of terrain distance"
    for  TerrainSphere
    [
        description "Spherical SA of terrain within " DesiredObservationRadius
         " radius for aircrew"
        val  DesiredObservationRadius = 5 nm
        compute  MeasuredDistance: JMRMIS::NauticalDistance
        value predicate  MeasuredDistance >= DesiredObservationRadius
        see goal MISStakeholderRequirements.SR_87
    ]
```

Figure 4: Requirement Predicate on Values

The ReqSpec/ALISA workbench performs consistency checks, such as confirming traceability of a goal or requirement to a stakeholder and ensuring that every component and interface feature has at least one requirement associated with it.

4.3 An Example System in ReqSpec

We used ReqSpec in three ways for the ASSA system.

First, we imported the contents of the stakeholder requirement document and the system/subsystem specification for a system called the Modular Integrated Survivability (MIS) system into the OSATE environment. We named these files *MISStakeholderRequirements.goaldoc* and *MIS-SSS.reqdoc*. In this case, the requirements are initially not associated with an AADL model. Once we have imported the contents, users can create an AADL model and manually associate the requirements from the requirement document with the model. In the process, users may associate different requirements from the same document section with different components in the AADL model. The ReqSpec tool has an analysis feature that identifies document sections that span multiple system components.

Second, we created *stakeholder goals* sets and *system requirements* sets that are associated with different systems in the architecture. We then created a separate file for each of the AADL packages. The files contain sets of *goal* and *requirement* specifications, one for each component specification in the AADL package.

Figure 5 shows an example of a *stakeholder goals* set specified for a component called the ASSA-Sensor. The keyword *stakeholder goals* introduces a name for a set of goals associated with the *ASSASensor*. Each *goal* specification has a unique name within the goal set. In our example, it includes a *title*, *description*, *stakeholder* reference, and list of references to the MIS stakeholder requirement document.

```
stakeholder goals SensorGoals for ASSASensors::ASSASensor
[ goal goal1
 title: "Passive ASE (ASSA sensor type)";
 [ description: "MIS shall support passive SA sensors (ASE)";
   stakeholder mrj.ab
   see requirement: MISStakeholderRequirements.SR_13,
   MISStakeholderRequirements.SR_69, MISStakeholderRequirements.SR_15;
 ]
]
```

Figure 5: A Goal Set for ASSA Sensors

Third, we illustrated requirement specifications that use variables to parameterize the requirement and specify that a property in the AADL model should have the same value as the variable or a particular value. This practice ensures that a verification activity operating on the model utilizes the correct values when performing the verification. In Figure 6, we show two example scenarios. One uses a constant in a *value predicate* to indicate that the value of the variable and a specific AADL property must be the same. In the other, the variable value is passed as a parameter to a verification activity.

In our first example, the user has developed the model with a property *JMRMIS::EnergyLevel*. In this case, we specify in a *value predicate* that the constant value is consistent with the property value.

In the second example, the value of the requirement is defined by a constant; in our example, it is called *DesiredObservationRadius*. This value will then be used in a verification plan associated with the requirements to indicate that its value is to be passed to a verification method via a property in the AADL model. In this case, the AADL model is automatically annotated with the appropriate property value. Note that specifications of verification activities are expressed by the *Verify* notation, which is part of the incremental lifecycle assurance tool environment.

```
system requirements PassiveSensorReqs for ASSASensors::PassiveTerrainSensor
[
requirement Req4 : "Passive sensor"
 [
  val EnergyLevel = 0
  description "Passive sensor radiates " EnergyLevel " energy"
  value predicate #JMRMIS::EnergyLevel == EnergyLevel
  see goal MISStakeholderRequirements.SR_27
 ]
 requirement Req1 : "Spherical terrain awareness for aircrew"
 for  TerrainSphere
 [
  description "Spherical SA of terrain within " DesiredObservationRadius " radius for
aircrew"
  val DesiredObservationRadius = 5 nm
  compute  measuredDistance : JMRMIS::NauticalDistance
  value predicate  measuredDistance >= DesiredObservationRadius
  see goal MISStakeholderRequirements.SR_27
 ]
]
```

Figure 6: Example of Requirement Specification Aligned with an AADL Model

5 Summary and Conclusion

This report introduced a textual notation called ReqSpec to specify stakeholder and system requirements and associate them with AADL models. ReqSpec supports an architecture-led requirement specification process that utilizes AADL models to specify requirements of a target system in its operational context, safety requirements derived from identified hazards, and derived requirements for subsystems as the system architecture evolves. It draws on goal-oriented requirements engineering concepts to distinguish between stakeholder requirements that may conflict with each other and system requirements that must be verifiable and may have satisfiable design goals. Verification plans, expressed in a separate notation, specify how the user intends to verify that designs and implementations meet the requirements. As such, the ReqSpec notation is a key element of an incremental lifecycle assurance approach to developing critical software-reliant systems.

Appendix Expression Support for ReqSpec

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 in the most recent OSATE release for current capabilities.

Table 1: 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 `&lt=`, `>`, `>=`
		`<range> < <range>` also `&lt=`, `>`, `>=`
		`<numeric> >< <range>` (value included in range)
		`<range1> >< <range2>` (range1 included in range2)
		Numeric or range expressions on the left- and right-hand sides must use the same unit 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 sides must use the same unit 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 unit type.
		For division, if the right-hand argument has a unit, it must be of the same type as the unit on the left-hand side.

7	Unary	`+ <numeric>`
		`- <numeric>`
		`not <Boolean>`

Primary Expressions

1. Unit operations for numeric expressions:
   a. Unit assignment to a unitless 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 a 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 `2000 ns`
2. Conditional expression:
   `if <Boolean> then <expression1> else <expression2> endif`
   Both `expression1` and `expression2` must have the same type.

3. Reference to a 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 a property value in a 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 required 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 a decimal point) to a *constant* of type `real`. Similarly, addition of an `integer` value and a `real` value results in a `real` value.
7. 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, and closest `integer` values for a given `real` value

References

[ALISA 2016]
Architecture-Led Incremental System Assurance (ALISA) Workbench. Computer software. Software Engineering Institute, Carnegie Mellon University. 2016. https://github.com/osate/alisa

[Delange 2016]
Delange, J., Feiler, P., and Ernst, N. Incremental Life Cycle Assurance of Safety-Critical Systems. In Proceedings of the Eighth European Congress on Embedded Real Time Software and Systems. Toulouse, France. January 2016. http://www.erts2016.org/inc/telechargerPdf.php?pdf=paper_13

[FAA 2009]
Federal Aviation Administration. *Requirements Engineering Management Handbook*. DOT/FAA/AR-08/32. FAA. 2009. 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]
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), Version 2. Computer software. Software Engineering Institute, Carnegie Mellon University. 2016. 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.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
(Leave Blank)	June 2016	Final

4. TITLE AND SUBTITLE	5. FUNDING NUMBERS
A Requirement Specification Language for AADL	FA8721-05-C-0003

6. AUTHOR(S)

Peter H. Feiler, Julien Delange, and Lutz Wrage

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)	8. PERFORMING ORGANIZATION REPORT NUMBER
Software Engineering Institute Carnegie Mellon University Pittsburgh, PA 15213	CMU/SEI-2016-TR-008

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSORING/MONITORING AGENCY REPORT NUMBER
AFLCMC/PZE/Hanscom Enterprise Acquisition Division 20 Schilling Circle Building 1305 Hanscom AFB, MA 01731-2116	n/a

11. SUPPLEMENTARY NOTES

12A DISTRIBUTION/AVAILABILITY STATEMENT	12B DISTRIBUTION CODE
Unclassified/Unlimited, DTIC, NTIS	

13. ABSTRACT (MAXIMUM 200 WORDS)

This report describes a textual requirement specification language, called *ReqSpec*, for the Architecture Analysis & Design Language (AADL). It is based on the draft Requirements Definition and Analysis Language Annex, which defines a meta-model for requirement specification as annotations to AADL models. A set of plug-ins to the Open Source AADL Tool Environment (OSATE) toolset supports the ReqSpec language. Users can follow an architecture-led requirement specification process that uses AADL models to represent the system in its operational context as well as the architecture of the system of interest. ReqSpec can also be used to represent existing stakeholder and system requirement documents. Requirement documents represented in the Requirements Interchange Format can be imported into OSATE to migrate such documents into an architecture-centric virtual integration process. Finally, ReqSpec is an element of an architecture-led, incremental approach to system assurance. In this approach, requirements specifications are complemented with verification plans. When executed, these plans produce evidence that a system implementation satisfies the requirements. This report introduces the ReqSpec notation and illustrates its use on an example.

14. SUBJECT TERMS	15. NUMBER OF PAGES
AADL, architecture-centric virtual integration, model-based engineering, OSATE, requirements specification, system assurance	34

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	UL

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-89) Prescribed by ANSI Std. Z39-18
298-102