

# A Requirement Specification Language for AADL

Peter H. Feiler  
Julien Delange  
Lutz Wrage

**August 2015**

**DRAFT TECHNICAL REPORT**  
CMU/SEI-2015-TR-XXX – PUBLIC RELEASE

Distribution Statement A:

<http://www.sei.cmu.edu>



Copyright 2015 Carnegie Mellon University

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.

Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the United States Department of Defense.

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

NO WARRANTY. THIS CARNEGIE MELLON UNIVERSITY AND SOFTWARE ENGINEERING INSTITUTE MATERIAL IS FURNISHED ON AN "AS-IS" BASIS. CARNEGIE MELLON UNIVERSITY MAKES NO WARRANTIES OF ANY KIND, EITHER EXPRESSED OR IMPLIED, AS TO ANY MATTER INCLUDING, BUT NOT LIMITED TO, WARRANTY OF FITNESS FOR PURPOSE OR MERCHANTABILITY, EXCLUSIVITY, OR RESULTS OBTAINED FROM USE OF THE MATERIAL. CARNEGIE MELLON UNIVERSITY DOES NOT MAKE ANY WARRANTY OF ANY KIND WITH RESPECT TO FREEDOM FROM PATENT, TRADEMARK, OR COPYRIGHT INFRINGEMENT.

Carnegie Mellon® is registered in the U.S. Patent and Trademark Office by Carnegie Mellon University.

DM-0001962

---

# Table of Contents

<b>Abstract</b>	<b>iii</b>
<b>1 Introduction</b>	<b>1</b>
<b>2 The ReqSpec Notation</b>	<b>2</b>
2.1 Stakeholder Goals	2
2.1.1 The Goal Construct	2
2.1.2 The Stakeholder Goals Construct	4
2.2 System Requirements	5
2.2.1 The Requirement Construct	5
2.2.2 The System Requirements Construct	7
2.3 The Document and Document Section Constructs	8
2.4 Variables and Predicates	9
2.4.1 Constant and Compute Variables	9
2.4.2 Global Constant Variables	10
2.4.3 Requirement Predicates	10
2.5 User-definable Requirement Categories	11
2.6 Stakeholders and Their Organizations	11
<b>3 General Guidelines for Use of ReqSpec</b>	<b>12</b>
3.1 ReqSpec Files	12
3.2 System Requirements and Stakeholder Goals	12
3.3 Classifier Extensions, Implementations, and System Requirements	12
3.4 Requirement Refinements	13
3.5 Requirement Decomposition	13
3.6 Requirement References	13
<b>4 Example Use of ReqSpec</b>	<b>15</b>
4.1 ReqSpec Declarations in OSATE	15
4.2 An Example System in ReqSpec	18
<b>5 ReqSpec and ReqIF</b>	<b>20</b>
5.1 Implementation Considerations and Requirements	20
5.2 Importing ReqIF Requirements to ReqSpec	20
5.3 Mapping Rules	21
5.3.1 Tailoring the Import Process	22
5.4 Exporting ReqSpec requirements to ReqIF	24
<b>6 Summary and Conclusion</b>	<b>26</b>
<b>References</b>	<b>27</b>

---

## List of Figures

Figure 1 Project with ReqSpec and Organization Files	15
Figure 2 Dialog to set Project References	16
Figure 3 Graphical Presentation of the ASSA System Interface (System Type)	16
Figure 4 Requirement Specification for the ASSA System	17
Figure 5 Requirement Predicate on Values	18
Figure 6 Goal set for ASSA Sensors	19
Figure 7 Example of Requirement Specification Aligned with AADL Model	19
Figure 8 ReqIF file Opened in RMF Tool	20
Figure 9 Selecting the ReqIF Import Command	21
Figure 10 Open the data types definition of the ReqIF document	22
Figure 11 View of Data Types in a ReqIF file	23
Figure 12 Mapping Between ReqSpec and ReqIF	24

---

## Abstract

This document describes a textual requirement specification language (ReqSpec) for AADL. It is based on the draft Requirement Definition and Analysis Language (RDAL) Annex, which defines a Meta model for requirement specification as annotations to AADL models. A set of plug-ins to the 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 ReqIF or in a table-based MS Word format can be imported into OSATE in order to migrate such documents into an architecture-centric virtual integration process (ACVIP). Finally, ReqSpec is an element of an incremental life cycle assurance approach. In this approach requirement specifications are complemented with verification plans. When executed these plans produce evidence that a system implementation satisfies the requirements. The focus of this document is on introducing the ReqSpec notation and tool support including the requirement document import/export capability.



---

# 1 Introduction

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

The objective of RDAL is to support the elicitation, definition and modeling of requirements for real-time embedded systems in an iterative process, thus supporting the refinement of requirement 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 it meets its 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.

RDAL distinguishes 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 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. This process has been introduced in [JMR-SR2 2014].

Second, it supports the migration of existing stakeholder and system requirement 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 requirement documents via the Object Management Group (OMG) Requirements Interchange Format (*ReqIF*) as well as export ReqSpec based modifications.

We proceed by first introducing the syntax of the ReqSpec notation. Then we illustrate its use in ALRS, followed by a description of the migration of existing requirement documents into an ALRS context.

---

## 2 The ReqSpec Notation

ReqSpec allows users to define *goals*, i.e., stakeholder requirements, and *requirements*, i.e., system requirement specifications. *Goals* are placed into *stakeholder goals* containers that are associated with a particular system (AADL component classifier). Similarly *requirements* are placed into *system requirements* containers.

Alternatively, goals can be organized into *goals documents* with *document sections* to reflect an existing stakeholder goals document, and requirements similarly into *requirement documents*,

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.

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 ()\*.

The set of elements between [ ] can be in any order.

### 2.1 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 *GoalsDocument* 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 *GoalsDocument* and *DocumentSection* constructs are also used for requirements and are described in Section 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 ( <ReqCategory> )+ )?
  ( description Description )?
  ( ConstantVariable )*
  ( rationale String )?
  ( refines ( <Goal> )+ )?
  ( conflicts with ( <Goal> )+ )?
  ( evolves ( <Goal> )+ )?
  ( dropped )?
  ( stakeholder ( <Stakeholder> )+ )?
  ( see document requirement ( <Requirement> )+ )?
  ( see document ( DocReference )+ )?
  ( issues (String)+ )?
  ( ChangeUncertainty )?
]

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

```

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.
- *Category*: reference to a requirement category. Requirement categories are user-definable, e.g., to categorize requirements as safety, security, or performance requirements. The goals can be filtered by category. This is an optional element.
- *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. 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).
- *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 document requirement*: reference to a stakeholder requirement 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. The concept of change uncertainty is based on [Nolan 2011]. See Section xxx for details on uncertainty specifications.

Note that when a goal is used in a *GoalsDocument* then 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 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.

### 2.1.2 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 NestedName ( : Title ) ?
for ( TargetClassifier | all )
( use constants <Global Constants>* ) ?
[
  ( description Description ) ?
  ( see document ( DocReference )+ ) ?
  ( ConstantVariable ) *
  ( Goal )+
  ( issues (String)+ ) ?
]
```

A *StakeholderGoals* declaration has the following elements:

- *NestedName*: 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. 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).

## 2.2 System Requirements

ReqSpec has the *Requirement* construct to represent system requirements. A system requirement is intended to be verifiable and not in conflict with other requirements. System requirement documents are modeled by the *RequirementDocument* construct (see Section 2.3). 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. We proceed by describing the *Requirement*, and *SystemRequirements* 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*.

### 2.2.1 The Requirement Construct

The *Requirement* construct represents a system requirement.

```

Requirement ::=
requirement Name ( : Title )?
  ( for TargetElement )?
[
  ( category ( <ReqCategory> )+ )?
  ( description Description )?
  ( Variable )*
  ( Predicate )?
  ( rationale String )?
  ( mitigates ( <Hazard> )+ )?
  ( refines ( <Requirement> )+ )?
  ( decomposes ( <Requirement> )+ )?
  ( evolves ( <Requirement> )+ )?
  ( dropped )?
  ( development stakeholder ( <Stakeholder> )+ )?
  ( see goal ( <Goal> )+ )?
  ( see document goal ( <Goal> )+ )?
  ( see document requirement ( <Requirement> )+ )?
  ( see document ( DocReference )+ )?

```

```

( issues (String)+ )?
( ChangeUncertainty )?
]

```

A Requirement 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.
- *Category*: reference to a requirement category. Requirement categories are user-definable, e.g., to categorize requirements as safety, security, or performance requirements. The goals can be filtered by category. This is an optional element.
- *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*: Variables are used to parameterize requirement specifications. Many of the changes to a 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 and assertion.
- *Set of Variable*: Constant and compute variables are used to parameterize requirement specifications. 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).
- *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 2.4.2 for details.
- *Rationale*: the rationale for a system requirement as 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 in the development process, e.g., a security engineer or a tester. 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.
- *See document goal*: reference to a stakeholder goal in an imported stakeholder requirement document.
- *See document requirement*: reference to a system requirement in an imported system requirement 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).
- *Uncertainty*: user specified indication of stakeholder goal uncertainty. See below for details on uncertainty specifications.

Note that when a requirement is used in a *RequirementsDocument* then 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 support identifying different systems for different requirements in the same document or document section.

## 2.2.2 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 NestedName ( : Title )?
for ( TargetClassifier | all )
( use constants <Global Constants>* )?
[
( description String )?
( see document ( DocReference )+ )?
( Variable )*
( Requirement )*
( issues (String)+ )?
]
```

A *SystemRequirements* declaration has the following elements:

- *NestedName*: 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*: It 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 set 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.
- Set of *Variable*: Constant and compute variables are used to parameterize requirement specifications. 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. If the *SystemRequirements* container identifies an AADL classifier in the *for* element, then all requirements are intended to be associated with the entity represented by the classifier or with an item within the classifier as specified by the *item* clause in the requirement.
- *Issues*: allows users to record issues that may be encountered as a set of textual notes (Strings).

## 2.3 The Document and Document Section Constructs

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 them with an AADL model and perform 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.

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).

## 2.4 Variables and Predicates

### 2.4.1 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 ::=
  computed Name

Type ::= <any type from the Java type system>
```

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 actual execution.

### 2.4.2 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:

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

### 2.4.3 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 Xbase expression> Xbase is a Java like expression language used by Xtend and available for inclusion in Domain Specific Languages (DSL) through Xtext. It supports OCL like constraint expressions on AADL models similar to Lute/Resolute.

## 2.5 User-definable Requirement Categories

The *categories* notation allows user to define categories of requirements. These categories can then be assigned to stakeholder goals and system requirements. Each category must be globally unique.

Categories can have several subcategories. A category can be a subcategory of one or more other categories. Categories with subcategories can themselves be subcategories.

When a category with subcategories is specified for a requirement or a filter, it acts as shortcut for listing all the subcategories.

When it is used in a filtering condition, then requirements with any of the listed categories are included. In other words, a filter match occurs when the categories are the same or one is a subcategory of the other.

Requirement categories are declared in a separate file with the extension *cat*.

In support of incremental life cycle assurance the categories notation also support the definition of verification categories and selection categories.

```
RequirementCategories ::=
requirement categories
[
  ( RequirementCategory )+
]

RequirementCategory ::=
Name ( { <RequirementCategory>+ } )?
```

## 2.6 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 )?
]
```

---

## 3 General Guidelines for Use of ReqSpec

This section provides some general guidelines on the use of ReqSpec.

### 3.1 ReqSpec Files

Users create files with the extension *reqspec* that will contain stakeholder goals and system requirements. These files can be placed in a separate folder, e.g., in a folder named *requirements* at the same level as a folder called *packages* that contains AADL packages. This is just a convention that the user can change. For example, the user may place the *reqspec* files in a separate project from the AADL model.

*Reqspec* files can contain one or more *system requirements* sets and/or one or more *stakeholder goals* sets. As convention we recommend that a separate *reqspec* file is created 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, we create a separate *reqspec* file.

### 3.2 System Requirements and Stakeholder Goals

The *system requirements* set uses <dot> separated IDs as its name that must be globally unique among system requirements set. Similarly stakeholder goals sets must have a globally unique name among themselves. By convention users can use names that are the equivalent of the component classifier identifying the target system with “::” replaced by “.”. Note that a *system requirements* set and a *stakeholder goals* set for the same system can have the same name that is globally unique.

Note: in the rest of this section we only discuss rules for system requirements. However, they apply to stakeholder goals as well. Only when they differ we will make an explicit distinction.

### 3.3 Classifier Extensions, Implementations, and System Requirements

System requirements are inherited by type extensions, as well as 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 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. In other words, the original requirement has been overwritten for the extension, but still holds for the original.

### 3.4 Requirement Refinements

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.

Note that 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 requirement to have – verification activities. In the former case satisfaction of the leaf requirement claims is sufficient for the enclosing requirement, while in the latter case this satisfaction is necessary but still requires the passing of verification activities associated with the enclosing requirement.

### 3.5 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 requirement associated with the component classifier of the subcomponent. The requirement is placed in the system requirements set of the subcomponent classifier. Note that there may be multiple instances of the same subcomponent classifier, i.e., different use context. 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 represents the common requirements across these use contexts. Suppliers may provide both a component type (spec sheet) and 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 requirement associated with the subcomponent itself. In this case the requirement is specified as implementation constraint on the subcomponent by placing the requirement with the enclosing classifier, typically the component implementation – 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 and representing the spec sheet from a supplier must meet.

### 3.6 Requirement References

In the initial draft version of ReqSpec requirement references had to be qualified with the name of the enclosing system requirements set. Given that requirements may reside in the system requirements set of its own classifier or in the system requirements set of a classifier higher in the extends and implementation hierarchy, users would have to be aware of the appropriate set.

To simplify dealing with references we have introduced scoping of requirements (as well as goals) such that the user can reference a requirement by just its name. The target classifier extends and implementation hierarchy is used to resolve the requirement reference contextually without qualifier.

## 4 Example Use of ReqSpec

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

### 4.1 ReqSpec Declarations in OSATE

In this section we describe how ReqSpec files are created, updated, and analyzed through an Xtext-based textual editor. Currently, a Sirius-based (<http://www.eclipse.org/sirius/>) navigational table and graphics based user interface is also in development.

Figure 1 shows the AADL Navigator on the left. The SituationalAwarenessSystem project is shown as containing AADL model packages organized into subfolders. In this example we have chosen to put the ReqSpec files into a separate folder called *Requirements*.

The right hand side shows a specification of stakeholders. The editor understands the syntax of the *Organization* notation. It provides syntax coloring and ensures that each element of a stakeholder specification, e.g., the phone number, is specified at most once. It also supports content assist. When the user types <control> <spacebar> the editor provides syntactically legal choices.

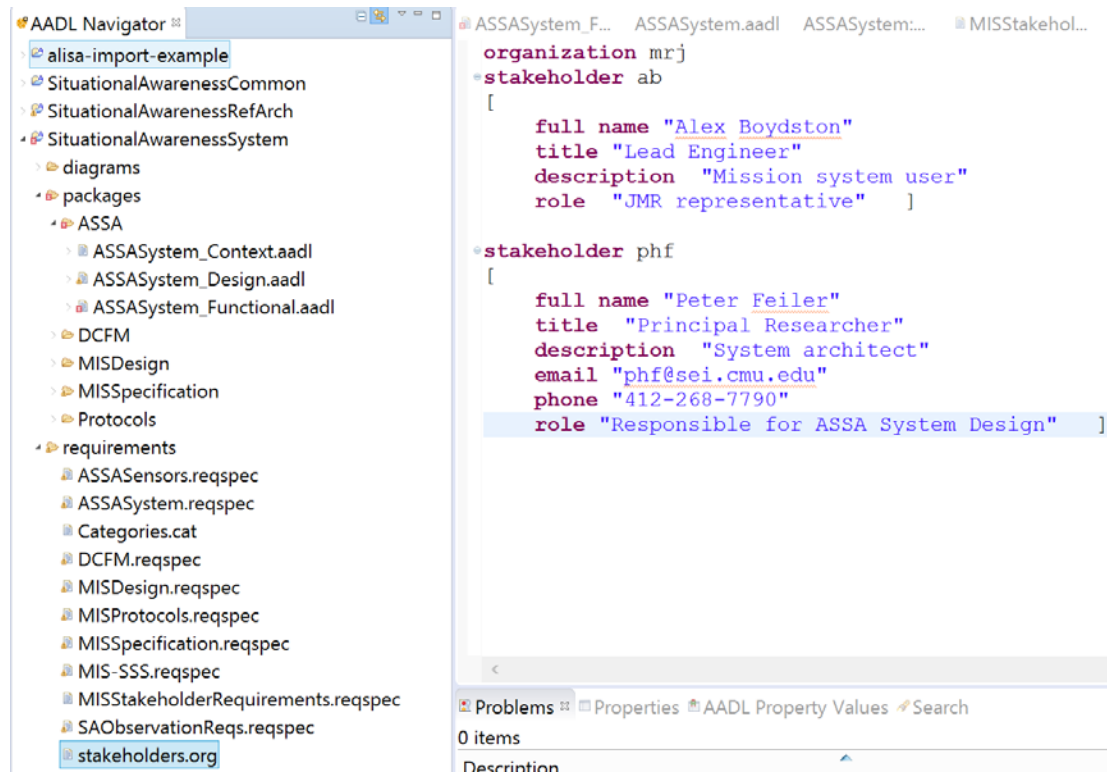


Figure 1 Project with ReqSpec and Organization Files

The ReqSpec files could be placed in a separate project if desirable. 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.

Project references are set in the properties dialog for the project containing the ReqSpec files. It can be invoked by selecting the project in the AADL navigator and invoking it through the context menu. An example dialog is shown in Figure 2.

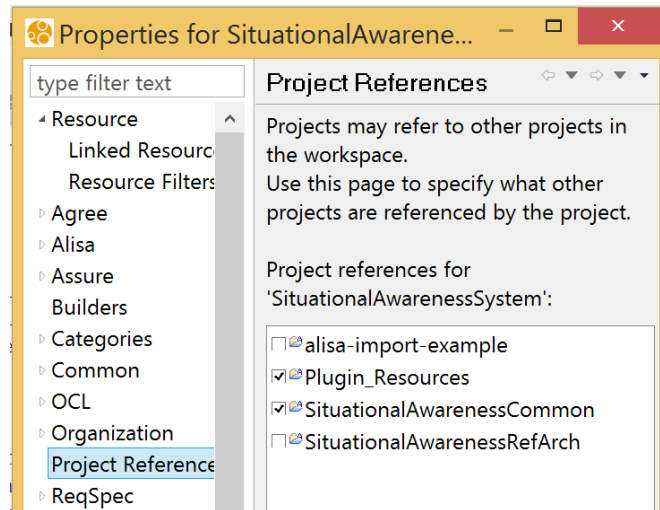


Figure 2 Dialog to set Project References

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

Figure 4 shows a set of requirement specifications for a situational awareness system called the ASSA system. These requirements originally came from a requirement document and have been migrated into a ReqSpec annotation to an AADL model.

The interface specification of the ASSA system itself is represented by an AADL system type and graphically shown in Figure 3.

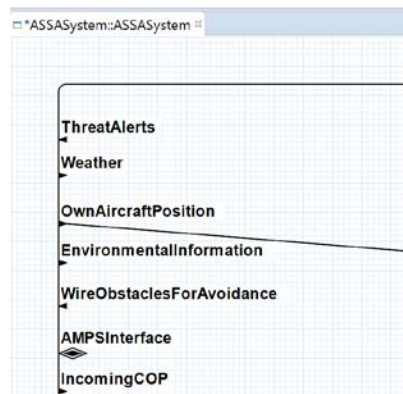


Figure 3 Graphical Presentation of the ASSA System Interface (System Type)

```

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 MISStakehol
  ]


  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
  ]
]

```

Problems ▢ Properties 🔍 AADL Property Values 🔍 Search		
errors, 3 warnings, 0 others		
description	Resource	Path
ReqSpec Problem (3 items)		
System requirement should have stakeholder goal or requirement reference	ASSASystem.reqspec	/Si

Figure 4 Requirement Specification for the ASSA System

The top-level requirement specification identifies the classifier of the ASSA system. The reference is qualified by the package name containing the classifier. References are hyperlinked to the target of a reference. When the user holds down the <control> key while pausing the cursor over the reference it is shown as hyperlink 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 ASSA system called AMPSInterface to reflect the fact that it is a requirement on the interaction between the ASSA system and AMPS. The *See goals* element identifies several stakeholder goals that reflect the need for an interaction between 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. Since the declaration does not include a references to a stakeholder goal or a system requirement it refines, a warning is issued to the user (see bottom of Figure 4). This traceability and other consistency checks are automatically performed on ReqSpec files. These checks are either performed incrementally as the user edits or when the user saves the file.

Other traceability and consistency checks include inability to trace to a stakeholder, and ensuring that every component and interface feature has at least one requirement associated with it.

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

```

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 sameas property JMRMIS::ObservationRadius
    compute measuredDistance
    value predicate measuredDistance >= DesiredObservationRadius
    see document goal MISStakeholderRequirements.SR_87
  ]
]

```

Figure 5 Requirement Predicate on Values

Figure 5 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. The *as* clause, when uncommented, specifies that the value is to be correctly reflected in the *JMRMIS::ObservationRadius* property associated with *TerrainSphere*.

The stakeholder requirement for this system requirement can be found as a goal in an imported requirement document.

## 4.2 An Example System in ReqSpec

We are using ReqSpec in three ways for the ASSA system.

First, we have imported the content of the MIS stakeholder requirement document and of the MIS system requirement specification document into the OSATE environment (*MISStakeholderRequirements.alisa* and *MIS-SSS.alisa*). In this case the requirements are initially not associated with an AADL model. Once imported users can create an AADL model and manually associated 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 that identifies document sections that span multiple system components.

Second, we have created *stakeholder goals* sets and a *system requirements* sets that are associated with different systems in the architecture. We have created separate files 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 6 shows an example of a *stakeholder goals* set specified for ASSASensor. The keyword *stakeholder goals* introduces a name for a set of goals associated with the component *ASSASensor*. Each *goal* specification has a unique name within the goal set. In our example it includes a *title*, *description*, *stakeholder* reference, and a list of references into 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 document requirement: MISStakeholderRequirements.SR_13,
  ]
]

```



```

    MISStakeholderRequirements.SR_69, MISStakeholderRequirements.SR_15;
  ]
]

```

Figure 6 Goal set for ASSA Sensors

Third, we are illustrating requirement specifications that take advantage of variables to parameterize the requirement and that specify that a property in the AADL model should have the same value as the variable or a particular value. This ensures that a verification activity operating on the model utilizes the correct values in performing the verification. In Figure 7, we have two example scenarios, one using the constant variable with the *sameas property* clause, the other using a *value predicate*.

- The value of the requirement is defined by a constant variable, in our example as *DesiredObservationRadius*. The variable declaration includes the *sameas property* clause, which ensures that the specified property for the *TerrainSphere* port has the same value. The intent is that the tool automatically maintains the property value to be consistent with the constant variable value. (This feature is not yet supported in the tool prototype.)
- The user has developed the model with a property value independently, in our second example requirement *JMRMIS::Sensortype*. In this case, we do not specify a constant variable but only a *value predicate* to assure that the *SensorKind* property has the value *Passive*. We chose not to use the constant variable since its value is not used anywhere else.

```

system requirements PassiveSensorReqs for ASSASensors::PassiveTerrainSensor
[
  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
    value predicate measuredDistance >= DesiredObservationRadius
    see document goal MISStakeholderRequirements.SR_27
  ]
  requirement Req4 : "Passive sensor"
  [
    description "Radiates no energy"
    value predicate JMRMIS::sensorkind == Passive
    see document goal MISStakeholderRequirements.SR_27
  ]
]

```

Figure 7 Example of Requirement Specification Aligned with AADL Model

---

## 5 ReqSpec and ReqIF

The ReqSpec plugin for OSATE includes a bridge to import/export requirements designed with the ReqIF format. ReqIF is a standardized interchange format for exchanging requirements developed under tools such as DOORS. This format consists of a specification of the data representation to be exchanged and the data content according to this representation. Users of DOORS have two levels of flexibility. First, they can tailor the representation of requirements within DOORS. Second, they specify how this data interchanged via ReqIF. In other words, the ReqIF files being exchanged may differ from project to project. DOORS uses the convention of pre-fixing every generated data representation name with “ReqIF”. If users import ReqIF from other sources, other naming conventions may be used.

This document details the mapping rules and how to import or export requirements from/to OSATE/ReqSpec to/from ReqIF.

### 5.1 Implementation Considerations and Requirements

The bridge is built on top of ReqSpec and the Eclipse Requirements Modeling Framework (RMF) [www.eclipse.org/rmf]. RMF is an Eclipse-based implementation of the OMG ReqIF standard and includes an EMF meta-model for the ReqIF interchange format. To be able to use the ALISA/ReqIF bridge, you need to have the RMF plugins installed in your Eclipse environment.

As both ReqSpec and ReqIF have their meta-models defined with EMF, the bridge tool translates between the two representations by taking advantage of these meta-models.

### 5.2 Importing ReqIF Requirements to ReqSpec

To import a ReqIF document, you have to open it within Eclipse using the RMF tool. You do so by double clicking on the file with the extension *reqif*. If the RMF tool does not start up as shown in Figure 8, please make sure you have installed RMF from its update site found at [www.eclipse.org/rmf](http://www.eclipse.org/rmf).

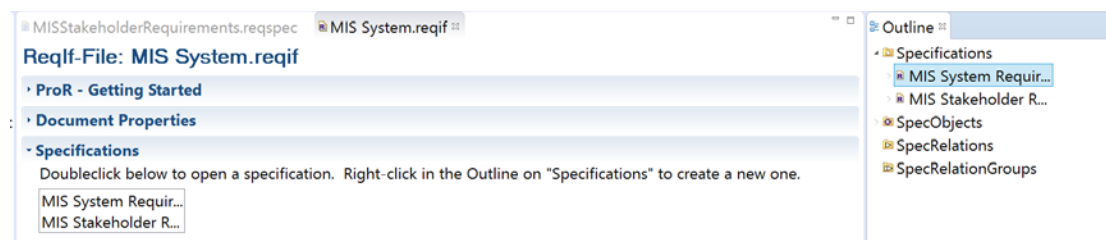


Figure 8 ReqIF file Opened in RMF Tool

The first step is to select the ReqIF specification hierarchy to export in the outline view, as shown in Figure 8 on the right. In our example the ReqIF file contains both a set of stakeholder requirements and a set of system requirements. So you will have to select each in turn and invoke the respective import command.

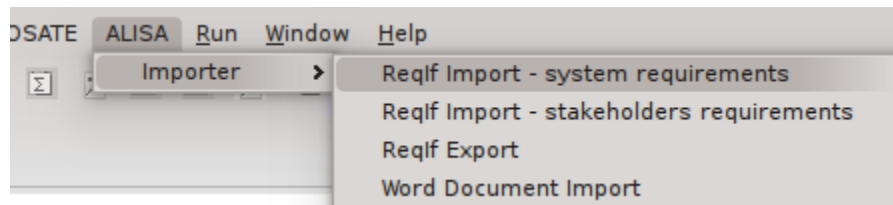


Figure 9 Selecting the ReqIF Import Command

Once you selected the specification to import, select the *ReqIF import* menu in the ALISA menu as shown in Figure 9. Note that two import options are available:

- *System Requirements*: imports ReqIF requirements as system requirements. This option will then map the ReqIF requirements into system requirement objects in ReqSpec.
- *Stakeholder Requirements*: imports ReqIF requirements as stakeholder requirements. This option will map the ReqIF requirements into ReqSpec goal objects.

Once the import is completed, the tool has created a new ReqSpec file with the same name as the ReqIF file. In the case of system requirements the word *system* is appended, and in the case of stakeholder requirements the word *stakeholder* is appended.

### 5.3 Mapping Rules

Both ReqSpec and ReqIF support representation of collections of requirements in a hierarchical structure. The hierarchical structure is maintained by interpreting the following equivalent elements.

ReqSpec Concept	ReqIF Concept
ReqDocument	Specification object with multiple children SpecHierarchy objects
DocumentSection	SpecHierarchy object with a ChapterName or multiples children SpecHierarchy objects
Requirement/Goal	SpecHierarchy object without children or a ChapterName

*Specification* objects have attributes, whose representation is defined by the data representation specification of ReqIF. *SpecHierarchy* objects have associated *SpecObject* objects that carry the actual requirement information as attributes.

The following attributes are currently supported for *ReqDocument* and are retrieved from a *Specification* object:

- ReqIF.Name: used as *Title* in *ReqDocument*. If this attribute is absent a default name is used.
- ReqIF.Description: used as *Description* in *ReqDocument*. If this attribute is absent no description will be set.

- ReqIF.Prefix: used as prefix to generate *Name* values for Requirement or Goal objects from the corresponding ID attribute (by default *ReqIF.ForeignID*). If this attribute is absent the user can specify an appropriate prefix in the *Preferences* dialog (see Section 5.3.1).

The following attributes are currently supported for *DocumentSection* and are retrieved from the *SpecObject* of a *SpecHierarchy* object. The ReqIF attribute names shown are the default and can be changed to a different attribute name.

- The name of DocumentSection is constructed as underscore (“\_”) separated section number prefixed with *Section*.
- ReqIF.ChapterName: Used as *Title* of the document section.
- ReqIF.Text: Used as *Description* in the document section. This attribute is optional.

The following attributes are currently supported for *Requirement* or *Goal*. *Requirement* is used for system requirements and *Goal* is used for stakeholder requirements. The ReqIF attribute names shown are the default and can be changed to a different attribute name.

- ReqIF.ForeignID: an integer value used to construct a unique name with a specified prefix (see above). If the attribute is absent a unique ID count is generated.
- ReqIF.Name: Used as *Title* of the requirement or goal. This attribute is optional.
- ReqIF.Text: Used as *Description* in the requirement or goal. This attribute is optional.
- Rationale: Used as *Rationale* for the requirement or goal. This attribute is optional.

### 5.3.1 Tailoring the Import Process

We have provided a mechanism for the user to specify which ReqIF attributes get mapped into the ReqSpec representation. The name of each data representation, which is defined in the data types section of the ReqIF document, must be used in the mapping specification. To see the data types of your ReqIF document, you open the data type definition while in the RMF tool. To do so, open the requirements document with RMF and click on the icon shown in Figure 10.



Figure 10 Open the data types definition of the ReqIF document

Once you open the data types definition, you can see the names of the data types in the Spec Object section of SpecTypes as shown in Figure 11. The example is from a ReqIF files exported from DOORS.

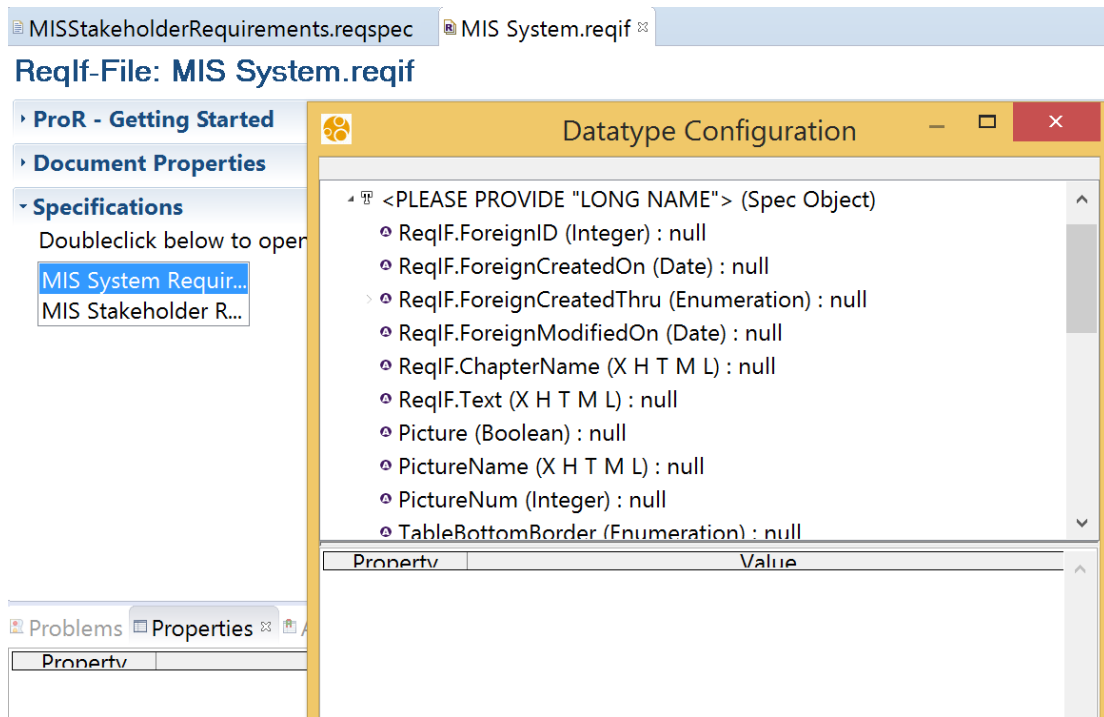


Figure 11 View of Data Types in a ReqIF file

In order to adapt the bridge to each ReqIF specification model, the bridge has a preference mechanism for the user to specify how to associate an an ReqIF attribute with a ReqSpec attribute.

This preference mechanism is available within the Eclipse preferences, as shown below. For each property on ReqSpec object, the user can specify the corresponding ReqIF attribute. The attribute is designated by its long name. If you do not specify an identifier attribute, the bridge will then try to build identifiers automatically. If you specify a mapping between ReqIF and ReqSpec, you have to make sure that identifiers are unique in the document.

Finally, you add a prefix to the identifier in the preferences. It can help to distinguish stakeholders and system requirements.

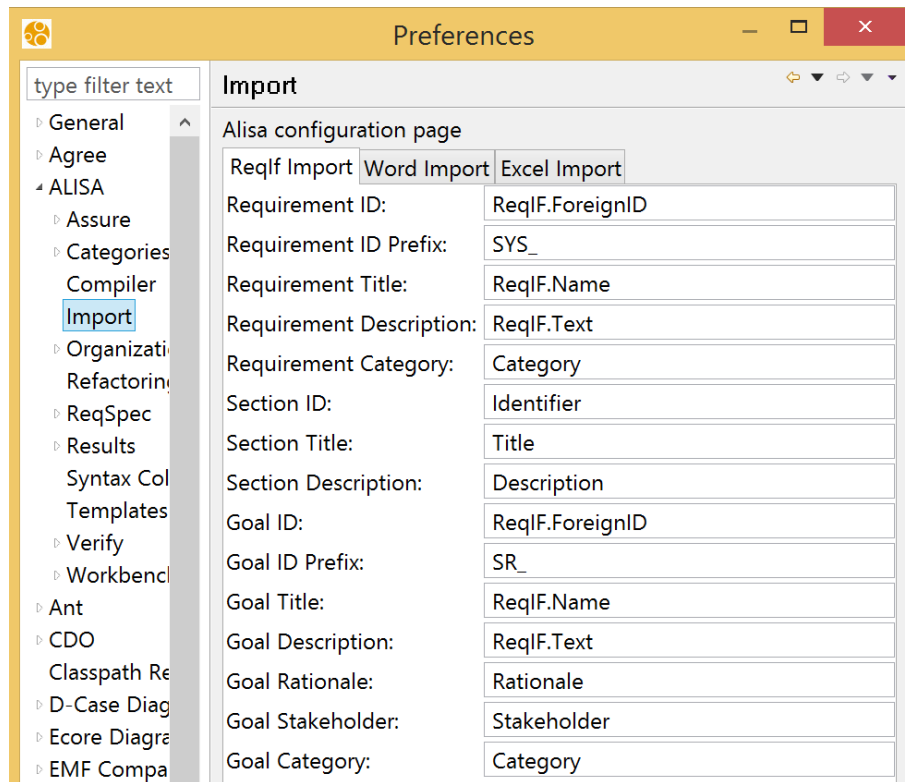
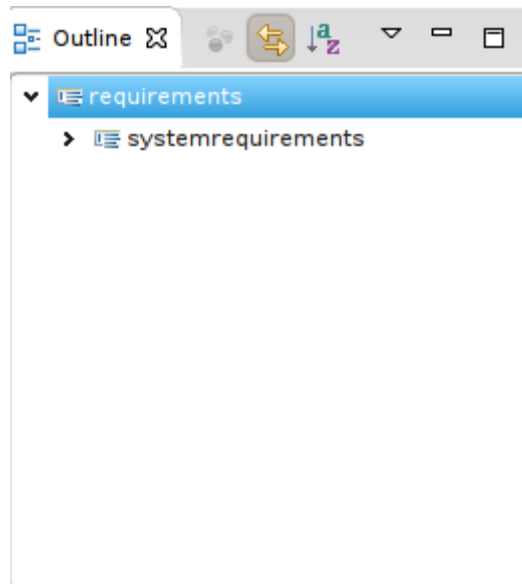


Figure 12 Mapping Between ReqSpec and ReqIF

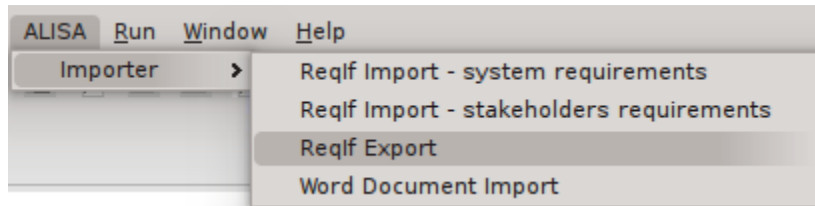
## 5.4 Exporting ReqSpec requirements to ReqIF

To export the ReqSpec requirement into ReqIF, you first need to select the ReqSpec document in the outline view as shown below.



Selecting the ReqSpec document to export in the Outline view

Once the main ReqSpec object has been selected, you invoke the tool by selecting *ReqIF Export* in the ALISA tool menu as shown below.



Invoking the Export tool

Once the export is completed, a new file called *reqspec-export.reqif* appears in the project that contains the imported ReqSpec file.

---

## 6 Summary and Conclusion

This document introduces a textual notation to specify stakeholder and system requirements and associate them with AADL models. This supports an architecture-led requirement specification process. This report has described this textual notation and its use on an example. The report has also described a capability for importing and exporting existing requirement documents through the ReqIF interchange format.



---

## 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](http://www.faa.gov/aircraft/air_cert/design_approvals/air_software/media/AR-08-32.pdf).

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

[JMR-SR2 2014] Peter Feiler, Requirement and Architecture Specification of the JMR MIS System. Special Report CMU/SEI-2014-SR-JMR-2, Software Engineering Institute. 2014.

[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=&ar-number=6051622&isnumber=6051621>