Project Number 318772

D2.3 General principles for mapping AADL models to GSN

Version 1.1
11 March 2014
Final Release

Public Distribution

University of York

Project Partners: **Fondazione Bruno Kessler**, **fortiss**, **Frequentis**, **LynuxWorks**, **The Open Group**, **RWTH Aachen University**, **TTTech**, **Université Joseph Fourier**, **University of York**

Project Partner Contact Information

Fondazione Bruno Kessler Alessandro Ciamatti Via Sommarive 18 38123 Trento, Italy Tel: +39 0461 314320 Fax: +39 0461 314591 E-mail: cimatti@fbk.eu	**fortiss** Harald Ruess Guerickestrasse 25 80805 Munich, Germany Tel: +49 89 36035 22 0 Fax: +49 89 36035 22 50 E-mail: ruess@fortiss.org
Frequentis Wolfgang Kampichler Innovationsstrasse 1 1100 Vienna, Austria Tel: +43 664 60 850 2775 Fax: +43 1 811 50 77 2775 E-mail: wolfgang.kampichler@frequentis.com	**LynuxWorks** Yuri Bakalov Rue Pierre Curie 38 78210 Saint-Cyr-l'Ecole, France Tel: +33 1 30 85 06 00 Fax: +33 1 30 85 06 06 E-mail: ybakalov@lnxw.com
RWTH Aachen University Joost-Pieter Katoen Ahornstrasse 55 D-52074 Aachen, Germany Tel: +49 241 8021200 Fax: +49 241 8022217 E-mail: katoen@cs.rwth-aachen.de	**The Open Group** Scott Hansen Avenue du Parc de Woluwe 56 1160 Brussels, Belgium Tel: +32 2 675 1136 Fax: +32 2 894 5845 E-mail: s.hansen@opengroup.org
TTTech Wilfried Steiner Schonbrunner Strasse 7 1040 Vienna, Austria Tel: +43 1 5853434 983 Fax: +43 1 585 65 38 5090 E-mail: wilfried.steiner@tttech.com	**Université Joseph Fourier** Saddek Bensalem Avenue de Vignate 2 38610 Gieres, France Tel: +33 4 56 52 03 71 Fax: +33 4 56 03 44 E-mail: saddek.bensalem@imag.fr
University of York Tim Kelly Deramore Lane York YO10 5GH, United Kingdom Tel: +44 1904 325477 Fax: +44 7976 889 545 E-mail: tim.kelly@cs.york.ac.uk	

Contents

Document Control

Version	Status	Date
0.1	Document outline	25 July 2013
0.2	Intitial Draft	27 August 2013
1.0	First Version	31 August 2013
1.1	Post-review version	11 March 2014

1 Introduction

This deliverable establishes the principles of how to map an AADL architectural model to a modular GSN argument structure. This includes general assurance argument patterns for the MILS-AADL subset. In addition, we establish the semantic mapping between AADL and modular GSN concepts. Finally, we show how the links between AADL and modular GSN can be notationally represented.

This is preliminary work to support the overall task aims as follows:

- Development of strategies for mapping AADL architectures to modular assurance case architectures.
- Establishing links with AADL interfaces
- Establishing the relationships with AADL error modelling and the analysis it supports
- Developing the necessary backing arguments to support the translation approach of WP2.2, in order to provide a overall compelling assurance case.

Establishing these principles and patterns will help build compelling, compositional, assurance cases using MILS-AADL modelling and further support the tool development process within other work packages by more formally capturing data and process relationships.

2 Background

This section provides the principles of mapping of models expressed in the MILS-AADL dialect (as specified in project deliverable [4]) to an assurance case expressed using the Goal Structuring Notation (GSN). The general principle of this work is that the AADL model will capture essential properties of components within the D-MILS, as well as providing data that can be used as input for various analyses and verification activities[1]. The assurance case will contain information demonstrating that a D-MILS system is fit for purpose, and comprises of an argument explaining why the system is acceptably dependable, backup by evidence. The argument is expressed in GSN, and the evidence is in the form of various development artefacts.

As there are general patterns and principles for assurance (such as demonstrating certain timing properties), we can extract specific system data from the AADL models into our assurance case patterns - thus partly automating the assurance process, and providing bespoke system specific data.

2.1 Introduction to GSN

The assurance case for a system presents an argument that a system will be acceptably dependable (e.g. safe/secure) to operate based in the expected operating environment and context. We use a very broad definition of a system, which encompasses the physical (e.g. computer hardware, or heating system), procedural (e.g. railway operations) and software (e.g. a decryption algorithm). Assurance cases can be used throughout the life cycle of a system, from early design, to commissioning, to in-service maintenance, and finally decommissioning.

The assurance case is built from two critical elements:

- Supporting Evidence - Results of observing, analysing, testing, simulating and estimating the properties of the system that provide the fundamental information from which safety/security/dependability can be inferred. Other, less procedural, evidence can include tools used to gather data, contextual data (e.g. where the system will be operating), system components themselves and so on.
- High Level Argument - Explanation of how the available evidence can be reasonably interpreted as indicating acceptable safety/security/dependability - usually by demonstrating compliance with standards, compliance with requirements, competence of staff carrying out the design, and sufficient mitigation and avoidance of hazards.

Each of these elements is required in order to put together a coherent case. If argument is presented without evidence, then the argument is unfounded as there is no basis in fact to support any of the claims present. If evidence is presented without argument, then there is no explanation to the reader to clarify the importance of the information.

The Goal Structuring Notation (GSN) [7, 6], as formalised in [1], is used for arguing about the dependability of systems. GSN provides a method for documenting the argument, and demonstrating

[1] This deliverable does not directly deal with the issue of whether the model is an accurate representation of an actual system or component, however evidence of this will be required

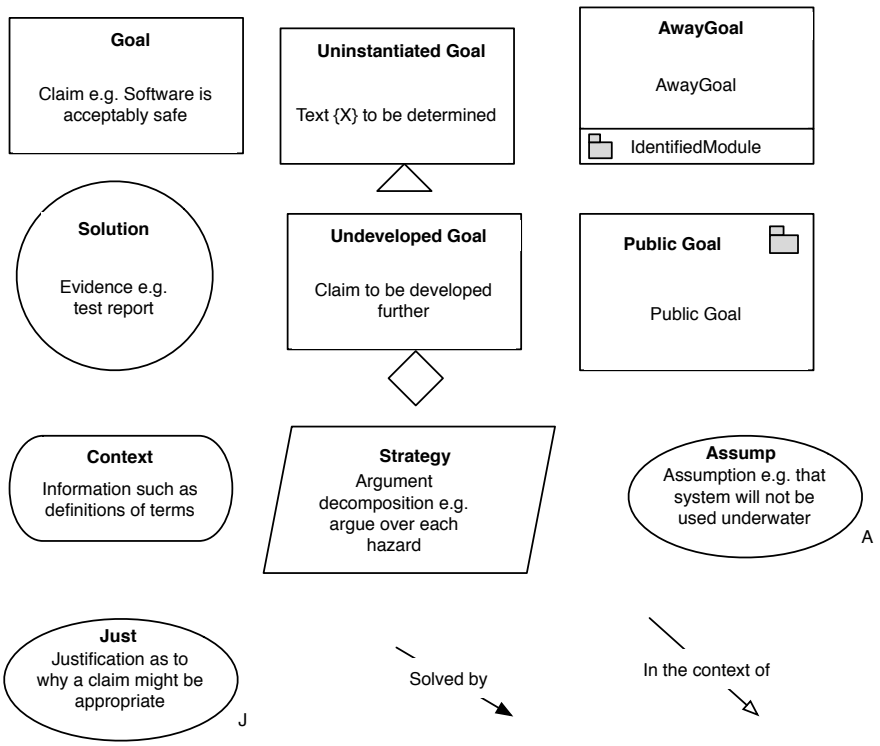

Figure 1: GSN Symbols

what has been thought of in the design of the system. It is composed of a set of symbols which are shown in Figure 1.

The symbology of GSN is used to show how *Goals* (to be achieved) are broken down into sub-goals, and eventually supported by evidence in the form of *Solutions* whilst making clear the *strategies* that have been adopted outlining the process that has taken place. Whilst noting this it is important to record the rationale that has been used for the approach in *Assumptions and Justifications* and recording the *Context* in which the system operates. We are using the modular extensions to GSN, which provide us with the ability to develop sections of argumentation independently, and re-use them for multiple systems. These sections or *Modules*, may have dependencies and guarantees. These are represented by *away goals*, where a given module relies on external information to satisfy the claims within, and *public goals* which are guaranteed pieces of information offered to other modules. Thus one argument module has an interface which needs to be integrated with other modules.

Argument patterns can be used to represent re-usable trains of logic or justifications, e.g. for similar types of component or for similar types of analyses applied to different systems. These can capture commonly known strengths and weaknesses in reasoning, e.g. pessimism in worst case execution time analysis. In this case we use claim text which requires partial instantiation for a given system. The convention used in this deliverable is to show uninstantiated text using braces e.g. "{component name} is sufficiently assured".

2.2 Types of mapping

In general there are three broad categories of information to be extracted from the models: architectural structure (components, sub-systems, threads), specific data items (e.g. names, constants, ports), and composite data (e.g. error analyses, port connections for policy architectures).

- **Structure** The assurance argument will be structured as a series of "modules". Each component in the D-MILS model must have at least one assurance case module associated with it. Some modules may contain data from more than one AADL component, and some modules may exist which do not take data directly from a component model (e.g. for compiled code, although it is assumed that the software interface would be represented within the model).
- **Data Items** These will be used for instantiating text within specific claims, justifications, contexts, and all GSN symbols. However, it is noted that using data for solutions depends on the nature of an assurance case. Solutions within an assurance case for any commissioned or deployed D-MILS should refer to an actual piece of verifiable evidence. Whilst properties may be stated within the model, there should be some proof that they are accurate in reality. On the other hand, if the assurance case is being used during development to help predict or assess potential designs or maintenance of the D-MILS then it is acceptable to use solutions/data that are not yet fully verified.
- **Composite data** This typically refers to results of various analyses that were extracted from the AADL-MILS models. This may be used for both instantiation of text and for solutions. It is noted that some composite data will be extracted from multiple components in the model at once (e.g. representing a policy architecture or error propagation between components), therefore may be used in more complex assurance modules within the argument. This is the most challenging area for research.

Examples are represented in figures 2 and 3. In figure 2 we show the high-level structural mappings from MILS-AADL model, for components, infrastructure elements and composite data. In figure 3 we show the lower level of detail, where the set of goals is intended to represent the internal structure of an assurance case module (right hand side) for the component represented in the MILS-AADL model (left hand side). The name of the thread *input* is used multiple times in the goal structure, and should be consistent within the module for "{component name}". Further data items of note include the values for timeouts - which may or may not be acceptable for the braking system in question. The rest of the argument text will need to explore this. Finally, composite data such as timing analyses and error condition analyses are linked to solutions at the bottom of the assurance argument.

2.3 Assurance Case Content

Having established the principles of mapping from MILS-AADL to the assurance case, we need to consider the content of that assurance case, in order to identify which elements of the model are required, which analyses we may make use of, and whether any new analyses are required.

D-MILS has a number of different networked processors (nodes), and running each of those is a separation kernel [5, 2] , that ensures all other software elements (middleware e.g. file systems and

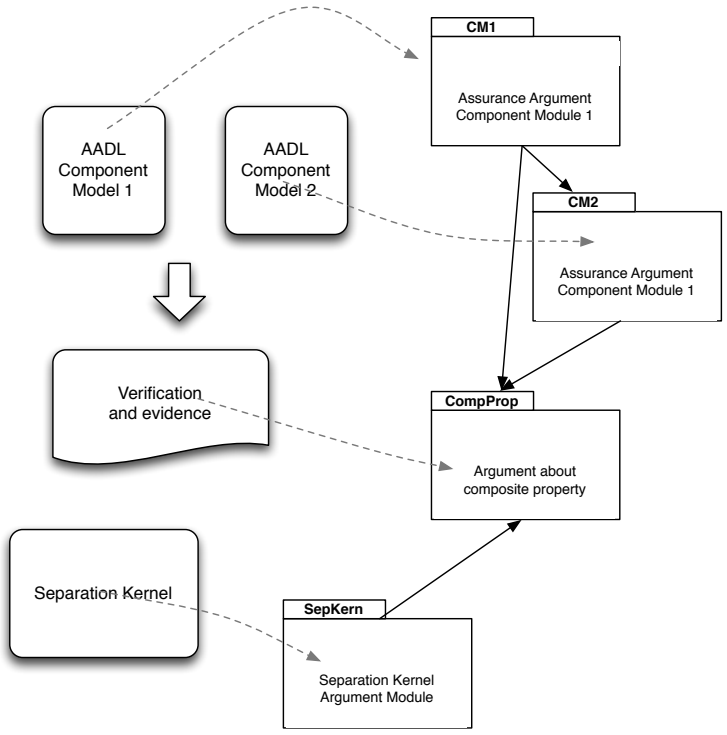

Figure 2: Structural Elements to Assurance Case Modules

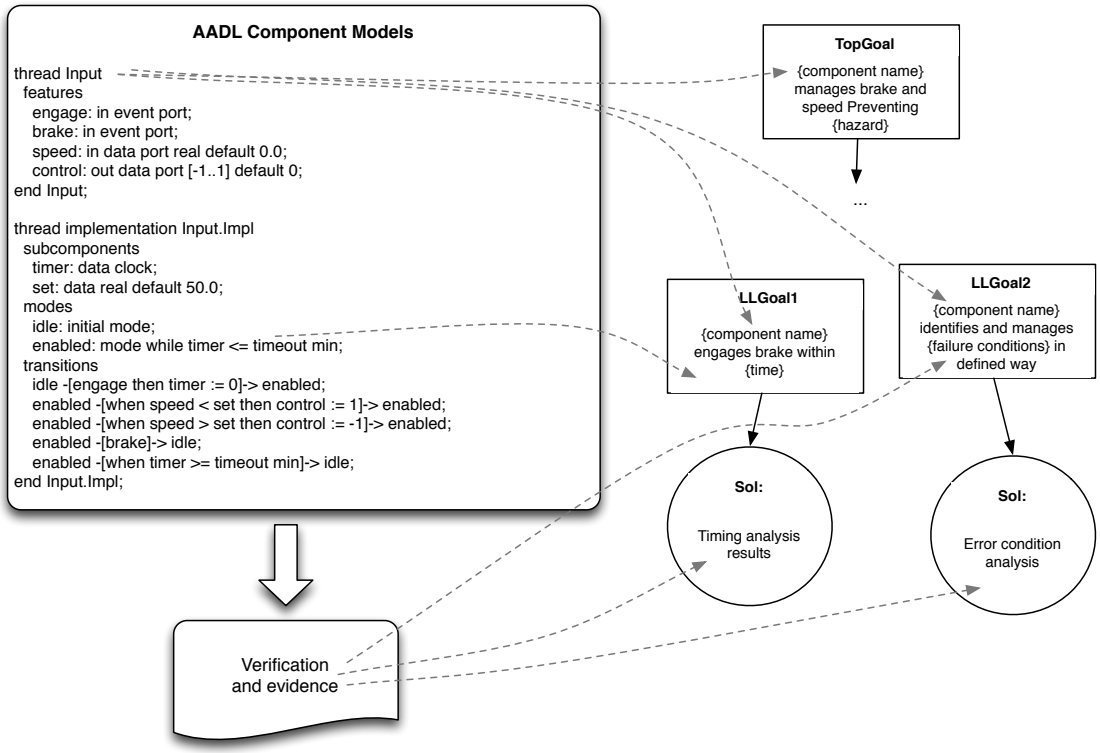

Figure 3: How elements of an AADL model may link to assurance argument module elements

applications) are partitioned from one another and cannot interfere which one another's resource usage.

Policy architectures are used to describe individual, permitted interactions in an application, which is created out of a number of different pieces of software. An Abstract Policy Architecture (APA) describes this logically with no reference to hardware. The Concrete Policy Architecture (CPA) describes the interactions in an actual, physical, potentially distributed, system.

Rather than build an individual monolithic assurance case for each D-MILS application, we wish to take advantage of fundamental principles of MILS, and build an assurance case module for each application and piece of middleware, re-using assurance data about the separation kernel, and platform elements. We will take advantage of the separation of concerns in order to modularise the assurance. The relationship between these elements is shown figure 4.

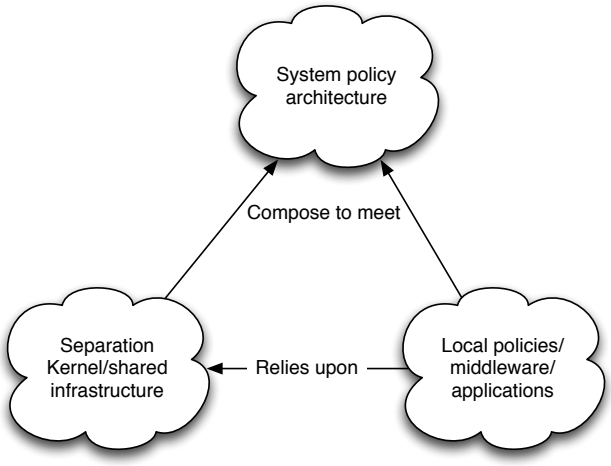

Figure 4: Elements of D-MILS assurance case

We note that the three strands have varying degrees of re-usability.

- Kernel and underlying platform assurance should be off-the-shelf and relatively fixed. Assumption: if the kernel is altered then fundamental properties are unlikely to change or will be within defined bounds
- Middleware assurance case data may be more or less fixed, depending on the nature of the middleware. It may be re-usable across multiple application arguments, if re-used. Template arguments (and associated required evidence) are suggested for each type of application: SLS, MSLS, MLS.
- Specific policy assurance case data, which are used for individual applications. These are likely to be constructed on a case by case basis, however, we can produce generalised patterns and considerations for their construction. These are underpinned by guarantees about platform and implementation enforcing both required separation and required interactions/configuration.

This deliverable concentrates on the assurance of middleware and applications, as well as for policy architectures.

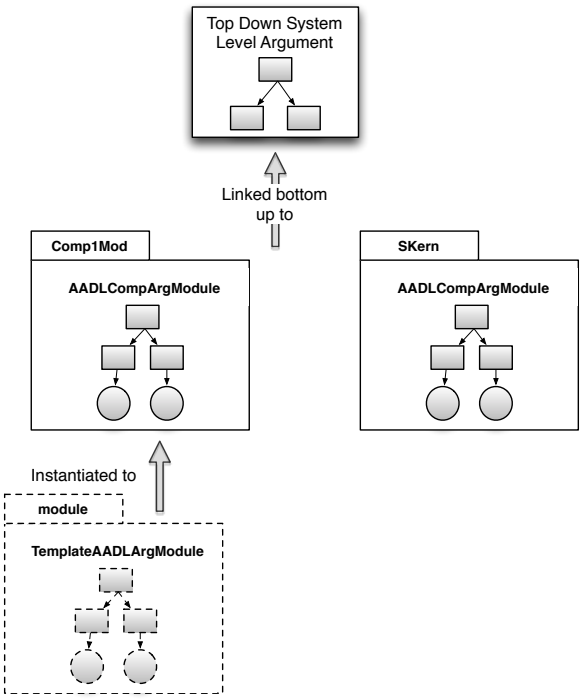

Figure 5: How individual modules are mapped to a top level argument

The assurance case will be composed broadly as shown in figure 5. Individual templates for parts of an assurance argument will be produced. These will be partially instantiated using data from AADL models. The partial instantiations will be revised and reviewed manually - manual review and completion is required due to the nature of the claims which may be expressed in natural language and contain qualitative and relative terminology. Note that within WP4 of D-MILS we are researching using constrained natural language in order to make this process more rigourous. Each completed module (which includes other modules, such as for the separation kernel - again these are being examined in WP4) will be mapped to a set of top level goals, capturing individual requirements and policy architecture for a specific system.

As noted, we need an understanding of the content of each of these items, based on the broad principles of separations of concern. There are a number of alternative strategies to consider, and trade offs to be made, in how the assurance case is constructed. A basic outline for a top level assurance argument for any D-MILS application, as shown in figure 6. The top level goal for this argument is that the *D-MILS is "sufficiently assured"*. We need an understanding of what "sufficiently assured" means, which will alter in every context. Broadly speaking, though, it means that all relevant threats are mitigated or removed. Again, this has a series of qualitative terms, which may be hard to measure objectively. However, each threat can be assessed in turn for a specific system, clarifying the details.

The next step in the argument has been to separate the APA from the CPA (as noted in the strategy box), on the basis that the APA effectively captures an abstract specification for the system, which reflects assurance requirements. Obviously, a simple policy architecture (of circles and arrows) will

not contain reasoning for its structure, and will only (currently) address security issues. The policy architecture is the binding theme throughout this argument, and feeds down to the next couple of layers, where a division is made between the foundational elements of the D-MILS system, and the specific components within the system that enforce local policies as necessary. The important goal for this deliverable is *LTOTop*, which will be subdivided for each different component, or combinations of components, and their contributions to the CPA. It is below this point that each of the individual modules can be attached, bottom up, to the top level argument.

It should be noted that this argument (figure 6), only addresses a small set of assurance properties, as found in the policy architecture. There may be other properties of concern, such as the safety requirements within the case studies. When further developing the argument we will include requirements (e.g. from threat analyses or hazard logs). In this situation, the top level argument will have multiple strands, which group properties into related sets. However, it is anticipated that each of the component level modules will contain a complete set of assurance data which could link to any of the grouped properties. In fact, each assurance module will contain a set of data which is effectively "context neutral", only becoming related to a specific system requirement in that context. The assurance modules will merely state the properties which are offered, and the confidence we have in the evidence relation to those properties. E.g. test data, and the number of conditions covered, or failure analyses and the competency of the staff performing the analyses. These properties may be of relevance for multiple dependability aspects, e.g. the time taken to decrypt a secure message may have an affect on a response time to a safety failure.

The more complex linking between is demonstrated in figure 7[2] The next section of this report addresses the data to be considered for an assurance module.

[2]In situations where it is difficult to directly link bottom up claims to top down claims, further arguments are added (sometimes described as safety case contracts). This issue is not directly addressed in this deliverable, but forms later research in the project.

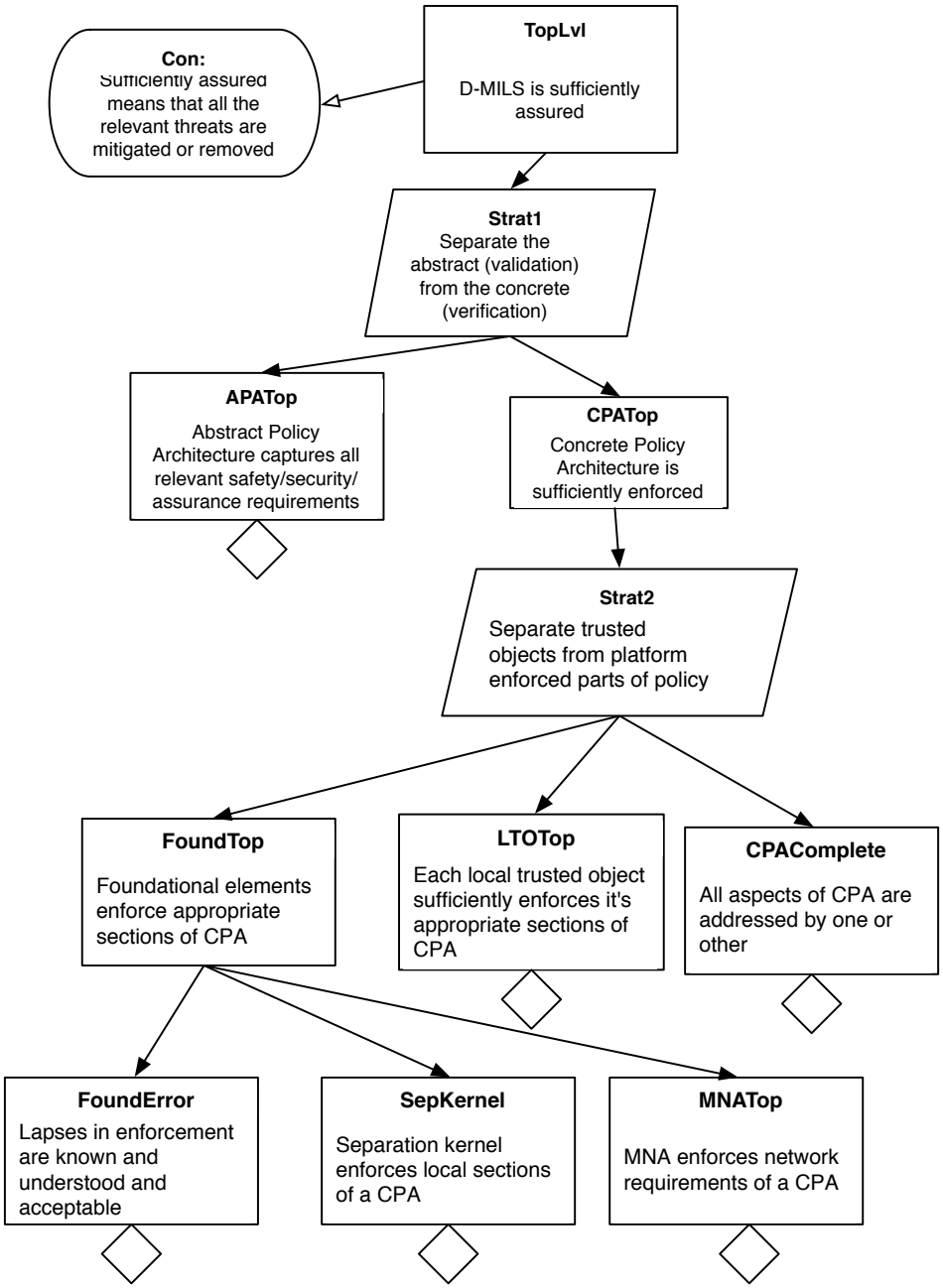

Figure 6: Typical Top Level Assurance Argument for D-MILS

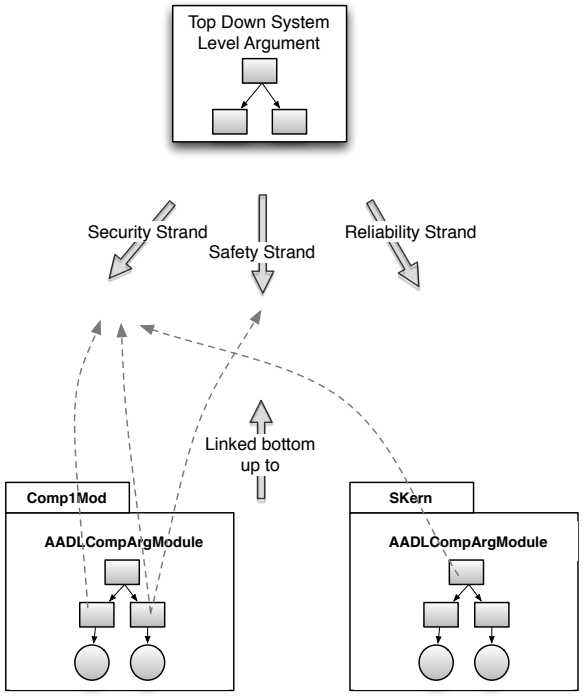

Figure 7: How assurance modules link to multiple strands in the top level argument

3 AADL Argument Template Example

This section contains examples showing how argument patterns for MILS-AADL subset can be used to demonstrate compliance with a CPA, as well as validating and verifying some of the data within that CPA. Two separate patterns are described: one CPA pattern which should be instantiated for a specific system, and one MILS-AADL model pattern, which can be instantiated for multiple items in the model, and then re-used for multiple assurance cases. The patterns are described using a standard format [1].

3.1 Concrete Policy Architecture Argument Pattern

This section describes a template argument structure for the CPA (figure 8), developing the goal *CPATop* found in figure 6.

3.1.1 Intent

The intent of this argument is to provide a basic pattern to demonstrate how a D-MILS assurance argument, using the Concrete Policy Architecture as a binding theme, can be supported by re-usable assurance argument modules, some of which are instantiated from MILS-AADL models (as found in 3.2). The ultimate purpose of the assurance argument is to help demonstrate that a given D-MILS system is fit for purpose. This will form part of a larger assurance case, however it is anticipated that development of, validation of, and conformance to, a CPA will always be part of any D-MILS design and deployment process. Therefore, we have chosen it as the first example demonstrating the mapping principles.

3.1.2 Structure

The structure of the argument is shown in figure 8.

3.1.3 Participants

The participants described are twofold: first we describe text items that need to be instantiated, but will be common to multiple elements in the argument pattern (table 1). Second, we describe each of the symbolic elements, and their content, in turn (table 2).

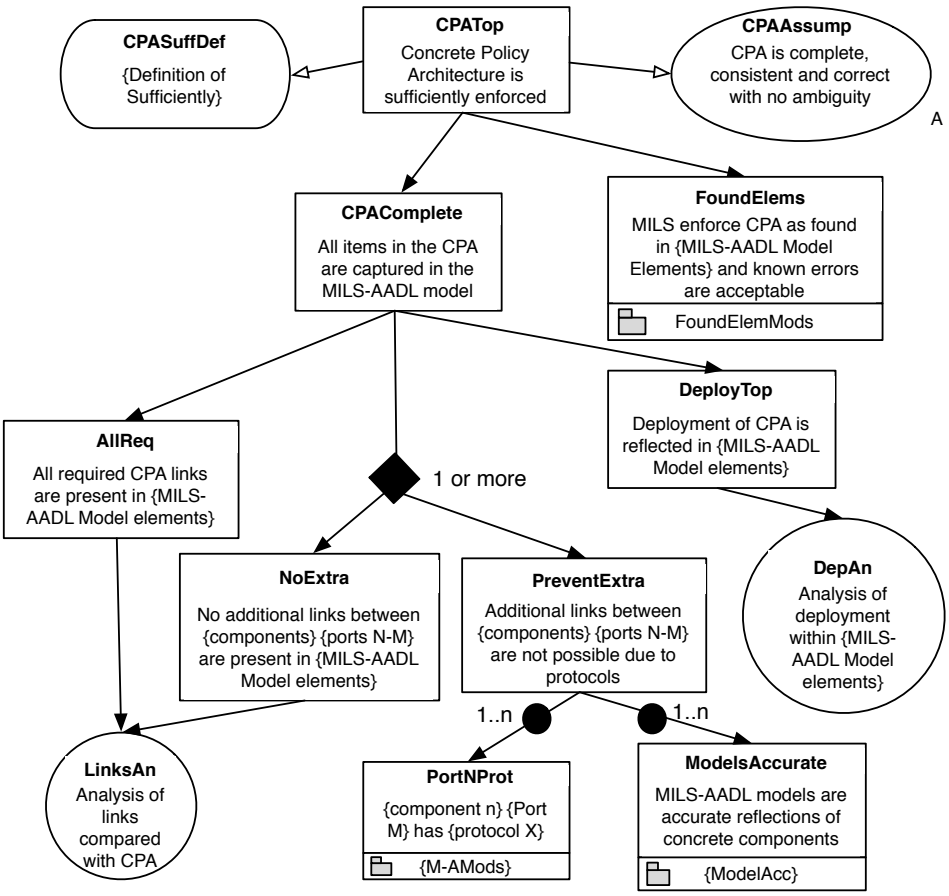

Figure 8: Decomposition of CPA goal across MILS-AADL model

Table 1: Participant text in figure 8

Text to be instantiated	Description
{MILS-AADL Model Elements}	This should be instantiated as a list of MILS-AADL model elements, all of which are contained within the policy architecture. This will include software elements, with deployment information. Any item which appears as a circle within the policy architecture must be included.
{ports N-M}, {port N}	Lists of ports reflecting the arrows in a policy architecture diagram. Every port in the policy architecture must be covered in the assurance case. However, part of the demonstration of the final D-MILS system is both that every intended link is there, and that no unintended links are there. For the latter we may rely on knowledge that some types of connection simply cannot occur (due to the protocols used) as well as analysing the models for extra links which may have somehow appeared.
{protocol X}	This should be instantiated for each relevant security protocol which might prevent an accidental passing of data between architectural elements
{M-AMods}	Instantiated for each individual MILS-AADL module used to support this pattern
{ModelAcc}	Instantiated with one or more assurance modules demonstrating that each part of the MILS-AADL model is an accurate reflection of the related final concrete component.
{Definition of Sufficiently}	This will need instantiation on a system by system/ case by case basis. For some lower security systems, it may be the case that the evidence need not be strong, as some minor lapses may be permitted. For higher security systems, very compelling evidence (possibly from diverse sources) may be required.

Table 2: Participant Elements in figure 8

Argument Element	Description
CPATop	This is the top level goal in this pattern, but has been taken from figure 6. The main aim of this branch of the assurance case is to demonstrate that the CPA is "sufficiently" enforced within the final D-MILS system. This overall aim, has been broken down into specific technical claims which can be verified and shown to be true/false.
Continued	

Argument Element	Description
CPAAssump	The assumption has been made that the CPA is complete, and consistent and correct with respect to the overall aims of a particular D-MILS system. This assumption could also be linked to a separate assurance argument module to that effect. Note that we have made the assumption that either the APA has been defined as an input to the MILS-AADL model, and/or, there is a set of security/dependability requirements for this system that have been reflected in the policy architecture. There is an issue of circularity in the argument if the PA is only defined in the model, without any rationale. However, analysis of the MILS-AADL model itself can be used to partially support this assumption, if it can be shown to have some or all of these properties (see the claim AllReq).
CPASuffDef	This context defines what is meant by "sufficient". It may be that the degree of rigour required to protect some data/-connections is less for some than others. This information must be initialised by hand.
CPAComplete	This claim is that all the required items from the CPA are reflected in the MILS-AADL model. Assuming that the Foundational Elements will then enforce all the connections from the model, then supports our claim that the CPA is reflected in the final system.
FoundElems	This claim is an away goal reference, to another module which includes guarantees that the foundational elements of the D-MILS will enforce the CPA exactly as it is described in the AADL model. If we can verify this claim it allows us to use the MILS-AADL model elements as an input for analyses of the consistency of the CPA as found in the AADL model.
AllReq	This claim is that all the CPA links are present in the MILS-AADL Model elements (this is partly a completeness claim). This requires there to be something to compare the MILS-AADL PA model to, or manual inspection.
DeployTop	This claim refers to the need to reflect the deployment of the PA in the final system, i.e. the concrete parts of it. We need to consider whether the deployment, as described, reflects all the links, in a manner which is practical (e.g. timing requirements could not be met due to components being located too far away from one another on the network.

Continued

Argument Element	Description
NoExtra and PreventExtra	When instantiating this pattern the assurance case developer has a potential choice to make between these two claims - one or both should be used. NoExtra refers to the analysis of the CPA, indicating that there are no extra links included (for example, hanging links). This requires there to be something to compare the MILS-AADL PA model to, or manual inspection. PreventExtra is an alternative means to demonstrate that there are no additional links, e.g. that certain protocols are in place which simply do not allow the links to occur.
PortNProt	This is an away goal reference, which may be instantiated one or more times depending on the number of ports, to the MILS-AADL component assurance module. It would be a particular instantiation of the Properties assurance section of figure 9, *PublicPropX* goal.
ModelsAccurate	This is an additional away goal reference to a different piece of assurance, that the MILS-AADL models are an accurate reflection of the implemented components. This is required for any completed system, but if the assurance argument is being used during development, so the potential layouts are being considered, then it may not be required.
DepAn	This is a reference to supporting evidence, namely an analysis of the CPA as embedded in the MILS-AADL models. As noted above, it is expected that this analysis will determine completeness, and consistency of the CPA. In addition, correctness may be determined with respect to a set of requirements.
LinksAn	This is a reference to supporting evidence, namely an analysis of the deployment aspects of a MILS-AADL model.

3.1.4 Applicability

This assurance argument pattern will be required for all D-MILS systems, as it is assumed that each will have a concrete policy architecture which needs examining.

3.1.5 Consequences

Having instantiated the claims within this pattern, with details from the MILS-AADL model, it is anticipated that a number of elements will still require development, namely *CPASuffDef*, and the

away goals becoming references to actual public goals in other assurance argument modules. The analyses described in *DepAn* and *LinksAn*, should also be produced.

3.1.6 Implementation

Instantiations of the text are described in table 1. However, we note the need for further research into how to extract composite data, namely the lists of relevant MILS-AADL model elements and ports (see discussion in 2.2). Part of the next phase of work is to examine how to do this in an automated fashion, thus improving the efficiency of instantiations.

3.1.7 Possible Pitfalls

The main concern when instantiating this pattern, is about circularity. it is not clear whether the CPA has another representation, other than that described in the models, or whether this part of the assurance case should be entirely about the verification of the CPA with respect to the APA and its internal state. In other words, do examine the correctness of the deployment by considering only if it is feasible in terms timing etc., or do we consider it with respect to another description.

The second concern, is whether the away goals can be supported. Related to this, there will be some known error modes, which could be exploited, breaking some security requirements. These are not discussed, other than briefly in the claim FoundElems. This will be explored further.

3.1.8 Related Patterns

This relies on instantiations of the Generic MILS-AADL component pattern, for each appropriate element, as well as developments of assurance arguments about the foundational elements of MILS (to be developed within WP4), as well as assurance arguments that the MILS-AADL models are accurate reflections of the concrete components. The latter is only required for completed systems.

3.2 Generic Pattern for a MILS-AADL component assurance argument module

3.2.1 Intent

This assurance pattern is intended to be instantiated for any MILS-AADL component model (be that a component, thread, sub-system or system) as required. It is written in a very generic and flexible way, but can be tailored for varying types of component, and properties. It is intended to be used to support many other different assurance case modules, not just of the type shown in figure 8, but all across the assurance case. The public goals may be used to support multiple claims in these structures (as discussed in section 2.3 and shown in figure 7). The instantiated pattern should be used for multiple assurance cases.

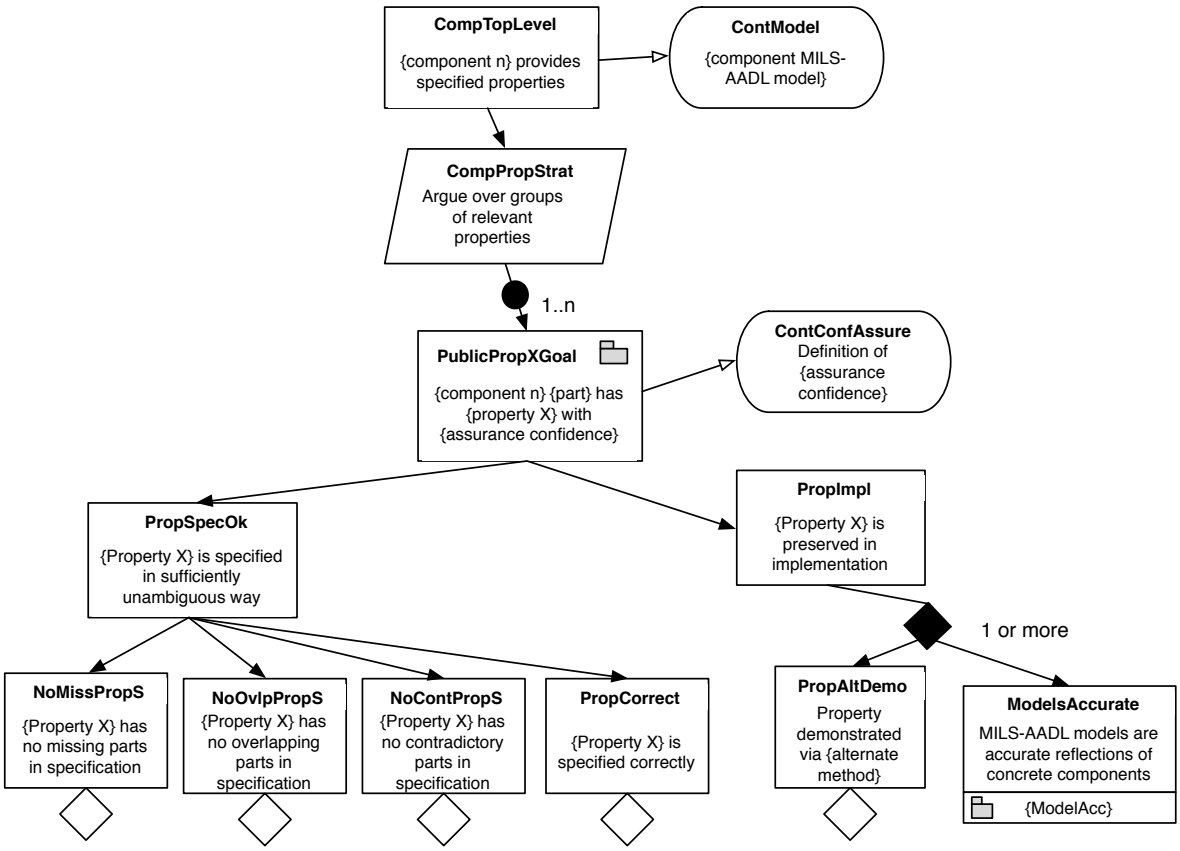

Figure 9: Generic Pattern for a MILS-AADL component argument

3.2.2 Structure

The structure of the argument is shown in figure 9.

3.2.3 Participants

The participants described are twofold: first we describe text items that need to be instantiated, but will be common to multiple elements in the argument pattern (table 3). Second, we describe each of the symbolic elements, and their content, in turn (table 4).

Table 3: Participant text in figure 9

Text to be instantiated	Description
{component n}	The name of the particular component in the MILS-AADL model being referred to. This is deliberately vague, in that it may refer to a sub-system, component or system, depending on the level of detail required by the assurance case developer. However, the item referred to must be consistent across all elements in the pattern when instantiated.
{part}	This refers to a particular part of the component model, any of which may have an associated property. E.g. a port or a variable, or the whole component. Again, this is flexible.
{property X}	A given property (or possibly a set of properties). This should be consistent across all elements below every instantiation of {PublicPropXGoal}.
{alternate method}	If there is a particular method which can demonstrate the accuracy of a property as defined in the MILS-AADL model (other than by comparison with the concrete artefact) then it can be stated here. For example, a formal proof.
{assurance confidence}	Some kind of measure of the confidence we have in a given property, this may be qualitative or quantitative (or a mixture of both).

Table 4: Participant Elements in figure 9

Argument Element	Description
CompTopLevel	This is the overall claim being made in this assurance case module. Note that this only offers assurance that specified properties are actually provided by the component, in other words we do not discuss whether any of these properties meet system requirements of any kind. This is due to the need to assess two things on a case by case basis: 1. Whether the property provided helps meet the requirements (also part of the general design process) 2. Whether there is sufficient evidence to show that the property is provided. For example, the level of sufficiency may be determined by a domain standard, or, by requirements of the system designers to manage particular threats or hazards. The definition of a component is that discussed as Structure in section 2.2.
ContModel	This piece of contextual information is the actual model of the component in question.
CompPropStrat	The strategy used for this argument is to partition properties as sets or singular items, as appropriate. The appropriate grouping may depend on the overall system claims we are likely to have to support, e.g. groups of timing properties, error propagations, or individual protocols. Further research is required into these groupings for the next phase of work. Ideally, once defined groups have been identified, we can extract information about them from the models automatically.
PublicPropXGoal	This goal (and all subsequent claims) should be instantiated for each identified property or set of properties. The level of confidence is determined for each, as it may depend on the techniques used to support a claim. For example, the level of testing coverage, the confidence in the values input into a probabilistic analysis, or the competence of the person performing the analysis. This goal is publicly available, and may be used to support away goals, e.g. of the type *Port-NProt* in figure 8. In other words, these properties may be either data items, or composite data (as discussed in section 2.2).
Continued	

Argument Element	Description
ContConfAssure	This context should define the level of confidence we have in the related claim.
PropSpecOk	This claim is an umbrella goal for an examination of the property specification, ensuring that its definition is sufficiently unambiguous. If the specification is, for example, incomplete, then we may think a component is suitable to be used when in fact there is some undocumented side effect. Further, if the specification is not complete we may have less confidence in any evidence used to demonstrate its veracity.
NoMissPropS	This claim states that there should be no missing data in the specification. An alternative to this goal would be that there is no relevant data missing in the specification.
NoOvlpPropS	This claim states that there should be no overlapping elements in the specification (e.g. parts of the model overwritten by others).
NoContPropS	This claim states that there should be no contradiction in the specification (it is assumed that MILS-AADL is sufficiently well defined that automated analyses may be run to support this claim for many types of property).
PropCorrect	This claim states the property is defined correctly. This might be in terms of the syntax/semantics as well as a specific value.
PropImpl	This claim is that the property being discussed is in fact implemented or realised in the final component.
PropAltDemo and ModelsAccurate	There may be a choice of methods to demonstrate the the property is implemented. It may be possible via comparison with the final component (e.g. source code inspection) or via any number of different analyses (such as those from WP3 or software testing etc.). Alternatively (or in conjunction) we may use an away goal reference to the *ModelsAccurate* module(s).

3.2.4 Applicability

This pattern should be applied for all MILS-AADL model components which need to be assured for a given system.

3.2.5 Consequences

The initial instantiation of this pattern should complete all elements, apart from *ContConfAssure*, *PublicPropXGoal*, and the choice of demonstration (*PropAltDemo* and *ModelsAccurate*), ideally via

automated mappings from the AADL Model. The various claims about the property specification will need expansion and development, but, depending on the property type, further patterns for these will be developed.

3.2.6 Implementation

Instantiations of the text are described in table 3. However, we note the need for further research into how to extract the various bits of data (see discussion in 2.2) as follows.

- **Structure** This defines the area of the model that is address by this module instantiation.
- **Data Items** These will apply for certain singular properties.
- **Composite Data** This will apply for certain sets of properties.

Part of the next phase of work is to examine how to do this in an automated fashion, thus improving the efficiency of instantiations.

3.2.7 Possible Pitfalls

There are a number of possible pitfalls in instantiating this pattern, mainly relating to how the public goal may be used. As it may be used to support a large number of other away goals in assurance case modules, the exact breakdown and groupings of properties may be difficult to predict. If the properties are described for each individual item in turn the argument will become largely unreadable, and difficult to manage. However, if they are in too large groups it may be difficult to match them to individual away goals (though, as noted earlier, assurance argument contracts may be one method to alleviate this problem). In addition, the analysis used to support verification that the property specified is actually realised, may well cover multiple properties at once (e.g. the error annex). If a single property changes, and an analysis re-run, then a large number of goal strands (and public goals) may be affected at once. This makes maintenance of the module particularly challenging.

3.2.8 Related Patterns

This will be used to support multiple different argument patterns, within the whole D-MILS assurance case.

3.3 Summary

This section has introduced two assurance argument patterns, which use data mapped from the MILS-AADL assurance module. The first is based on demonstrating a given concrete policy architecture is met, and should be instantiated for a specific system. The second demonstrates properties for a specific section of a MILS-AADL model, and can be instantiated to be re-used to support multiple different assurance cases for multiple systems.

4 Feature mappings from AADL to Assurance Case

The previous section noted the need to develop property groupings, partly based on the types of assurance needed for the overall assurance case. In deliverable 1.3 [3] a number of specific requirements for assurance were identified. This section of the deliverable describes those various features/properties, considers which MILS-AADL elements are related to them, and comments on how to develop these further to assist in automated data extraction. Table 5 summarises this.

Table 5: MILS-AADL features for Assurance

Broad Category of Assurance	MILS-AADL Model Data	Comments
Modular Design	Component Type Component implementations Component categories Node System	Partly this is related to the structure of the argument, and partly this is naming items that claims refer to
Policy Architecture (Abstract)	Data ports Data flows Modes Component categories	The MILS-AADL representation of the APA may not be required for the assurance case, other than as a partial representation of the CPA.
Policy Architecture (Concrete)	Data ports Data flows Modes Component categories Physical Bindings	The CPA will include deployment information in addition to permitted links and connections of the APA.
Security Annotations and Security Properties	Data ports Data flows Key Declarations Key Identifiers	The security annotations will limit and describe whether some ports may or may not be linked (see the template in figure 8
Space Isolation	Component implementations Physical Bindings	Related to Policy Architecture
Time Isolation	Component implementations Modes Physical Bindings	This aspect of the assurance case may be used for general dependability and assurance that tasks will complete in the time required.

Continued

Broad Category of Assurance	MILS-AADL Model Data	Comments
Task Scheduling	Component implementations Event Ports Modes Mode Transitions	This aspect of the assurance case may be used for general dependability and assurance that tasks will complete in the time and order required.
Error Injection	Error model types Error model implementations Error states Error events	Errors may be used for both safety and security, to demonstrate the effect on potential hazards and threats.
Dependencies	Modes Physical bindings Component implementations Node System	Such as ordering, indirect dependencies (e.g. diverse locations to avoid common cause failure).
Safety requirements	Any	Any safety requirements that are not related to the Error modelling, so part of the core functional behaviour.

5 Assurance Case Evaluation Criteria

This section provides some criteria for evaluation of the assurance case. There are three aspects to this which are now discussed in turn. Many of the quantitative criteria are formalised in D2.4.

- Generic, objective, quantitative, criteria for every assurance case.
- Qualitative criteria about content of a specific assurance case
- Qualitative and quantitative criteria about the efficiency of the production a specific assurance case

5.1 Metrics and Sanity Checks

Some simple metrics and sanity checks can be performed for an assurance case, particularly one which is partly automatically instantiated from a model such as MILS-AADL. These allow the assurance case developer to ensure that all the appropriate elements are there and have been discussed. However, they cannot measure the *quality* of the argument or evidence.

- Check that all nodes in the final argument are instantiated and developed. If they are not, we can highlight them and provide a justification if necessary (for example, a placeholder for functionality which will be integrated yet).
- Look for criticality of certain nodes in terms of the argument structure. For example, how many claims are supported by a single solution. If many claims are supported by a single piece of evidence, then changing that evidence will have a high impact on the argument. Highlighting this issue will allows us to consider whether the evidence can be decomposed further in order to improve the potential impact of a change.
- Ensuring all relevant elements of an input model, or input data, are included. For example, do all components in a policy architecture have an argument module. Another example, would be whether all hazards or threats have been mentioned in a claim somewhere in the argument.

5.2 Content Assessment

The purpose of the assurance case is to provide a compelling case to show that the D-MILS system is fit for purpose. Further, it may need to show that several applications running on the same set of nodes are fit for purpose and sufficiently protected from one another. As such, it needs to demonstrate the the configured D-MILS system is sufficiently dependable, safe and secure for the related and identified reliability, hazards and threats. For example, will a particular data asset be secure from a particular kind of un-authorised access and will a particular hazard be mitigated by detection. Another aspect, may be demonstrating that there is no undesired emergent behaviour.

Assessment of these aspects will need to be determined on a bespoke case by case basis. Whilst a process and checklist for review can be defined, the content of the different assurance cases will differ so much that the results may be difficult to compare. Allowing the case study partners to provide independent feedback on the usefulness of the assurance case is recommended, along with a checklist of potential content themes to examine.

5.3 Efficiency of the approach

In addition to providing a compelling assurance case, ideally we want to construct an assurance case in an efficient and maintainable way. A non-modular, manually constructed assurance case would be compelling, but without improvements in the production methods there is limited benefit to the approach described here. There are several aspects to this: cost of producing the assurance case, time taken to produce the assurance case, re-usability of argument parts, maintainability of the argument (e.g. when a component is changed).

The cost benefits would be a demonstration that the model based approach to building an assurance case is less costly than current methods. Similarly, by performing some instantiation automatically, there should be an improvement in the time take to construct the argument. At present, there is no assurance case for D-MILS systems, manual or otherwise. Instead, the concept is being introduced for this generation of systems in order to enhance the design process, and demonstrate to end users that the system is fit for purpose (e.g. sufficiently secure). As such, there is no current baseline against which we can show improvements in cost and time. However, we can, at least compare loosely with the build of a manual assurance case in other domains (e.g. safety domain). In addition, once an initial baseline has been developed, we can show improvements in time take to incorporate changes, e.g. re-generate an argument following a component alteration or replacement.

The re-usability aspect can be demonstrated and assessed for the D-MILS supporting platform arguments by showing they can be re-used for both case studies. For example, does the assurance argument for the separation kernel fit and support both the Fortiss and Frequentis application assurance cases, without alteration. The re-usability of the patterns can be assessed by instantiating them for many different components and seeing how robust they are to variations, e.g. in component type, scale and error types.

The maintainability aspect, can be judged by generating some typical change scenarios, and assessing the impact that has on the argument. For example, how many claims in a module (and related modules) are affected by the introduction of a new error or software requirement, or the alteration of same. Another test might be a different PA configuration of the same components. Finally, a change in the supporting environment (e.g. separation kernel or networking) should be considered.

6 Conclusions

This deliverable has established and described the basic principles for mapping from MILS-AADL models to a GSN assurance case. Three levels of mapping were identified: structure, data items and composite data. The general principles of how this would apply to the symbolic elements of a GSN assurance case were described. Then, a top level system argument pattern was described, for the purposes of identifying the type of content that is expected to be within a D-MILS assurance case.

Some example argument patterns were described in section 3, showing how the three levels of mapping would be used, and further identifying areas for research, particularly for automation of instantiating text within the patterns.

The following areas were identified for the next stage of research within this task for WP2 of the D-MILS project:

- Examining the use of assurance argument contracts, in order to help match away goals to public goals.
- Development of automated methods to extract composite data from AADL-MILS, including model elements, appropriate levels of abstraction (component, sub-system etc.), ports and associated properties.
- Further development of proper groupings, again for composite data extraction, as identified in the initial project requirements and listed in section 4.
- Further patterns for assurance of the specification of specific property types, and supporting evidence analyses

References

[1] GSN community standard. Technical report, Origin Consulting (York) Limited, 2011.

[2] Jim Alves-Foss, W. Scott Harrison, Paul Oman, and Carol Taylor. The MILS architecture for high-assurance embedded systems. *International Journal of Embedded Systems*, 2(3/4):239–247, 2006.

[3] Requirements for distributed MILS technology. Technical Report D1.3, Version 1.2, D-MILS Project, August 2013.

[4] Specification of MILS-AADL. Technical Report D2.1, Version 1.3, D-MILS Project, March 2014.

[5] Rance DeLong. An Evaluation and Certification Scheme for MILS. In *Proceedings of the 4th Layered Assurance Workshop*, Austin, Texas, December 2010.

[6] Tim Kelly and Rob Weaver. The goal structuring notation–a safety argument notation. *Proc. DSN 2004 Workshop on Assurance Cases*, 2004.

[7] Tim P Kelly. Arguing safety-a systematic approach to managing safety cases. *UNIVERSITY OF YORK DEPARTMENT OF COMPUTER SCIENCE-PUBLICATIONS-YCST*, 1999.