ANALYSIS OF MIXED CRITICAL AVIONICS AND SYSTEMS

Chris Walter
cwalter@wwtechgroup.com
WW Technology Group
Ellicott City, MD

Brian LaValley
blavalley@wwtechgroup.com
WW Technology Group
Westerly, RI

ABSTRACT

A goal of next generation avionics systems is the integration of avionics and mission applications on shared resources to reduce their size/weight/power requirements and enable increased range, additional payload capacity, and reduced logistics footprint and costs. The integration complexities can increase costs especially if mixed-criticality applications (e.g., safety-critical, mission-critical, non-critical) are hosted on the same resource since every application, regardless of criticality, must be certified (tested) to the level required by the highest criticality application hosted on the resource. This adds significant cost to fielding complex systems, e.g. helicopter avionics and Unmanned Autonomous Systems (UAS) that must integrate mixed-critical capabilities while mindful of size, weight and power concerns (SWaP). In this environment, the ability to provide mixed-critical attributes, such as security, real-time fault tolerance and performance is critical for predictable system response.

INTRODUCTION

A goal of next generation avionics systems is the further integration of avionics and mission applications on shared resources. Doing so enables significant reductions in size/weight/power requirements, logistics footprint and costs while permitting increased range and additional payload capacity. The integration complexities can introduce additional cost elements related to certification when mixed-criticality applications (e.g., safety-critical, mission-critical, non-critical) are hosted on the same resource. Every application, regardless of criticality, must be certified (tested) to the level required by the highest criticality application hosted on the resource or in the system partition in which it is co-located. This adds significant cost to fielding complex systems, e.g. helicopter avionics and Unmanned Autonomous Systems (UAS) that must integrate mixed-critical capabilities while being mindful of size, weight and power concerns (SWaP).

In this environment, the ability to specify and analyze mixed-critical attributes, such as security, real-time fault tolerance and performance is essential for predictable system response. The independence between the systems that may have been previously ensured through separate hardware and dedicated communications buses needs to be established in the new configuration. Complexities are introduced as applications now share critical hardware and software resources where subtle interrelationships may exist that are difficult for system designers to discover. Various strategies have been developed using the concept of partitioning to address this challenge. When applications are mapped to a single hardware computing platform, the independence of processing and memory operations across applications needs to be provided through the partitioning mechanism. With a platform that correctly provides the underlying partitioning functions (such as ARINC 653), the distributed system can be implemented on a single shared resource. The ability to partition applications has been traditionally performed based on a single aspect such as safety. Our work expands the concept to a multi-dimensional approach that considers aspects of performance, safety, security and dependability.

Figure 1 illustrates an example for transitioning from a federated to an integrated system with a number of applications of different criticality levels that share a common platform. This arrangement brings significant benefits in terms of resource utilization and reduction in the space weight and power required to field an equivalent level of functionality. With these benefits there are also significant challenges that need to be met in terms of handling increased levels of coupling and complexity while verifying that required level of independence between functions is still met.

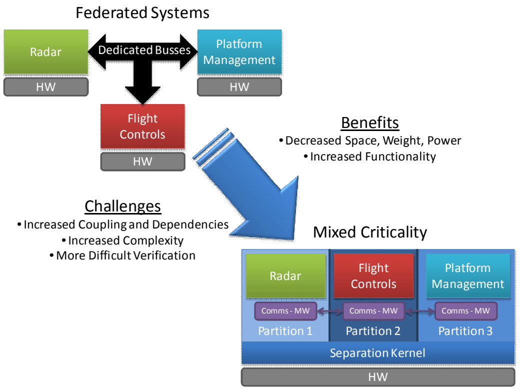


Figure 1 - Mixed Criticality Systems

Presented at the AHS 69th Annual Forum, Phoenix, Arizona, May 21–23, 2013. Copyright © 2013 by the American Helicopter Society International, Inc. All rights reserved.

The separation kernel and associated communications middleware provides the fundamental mechanisms to support mixed criticality by establishing regions that isolate behavior and interactions. These can be logical or physical in nature, where the former are simply referred to as containment regions while the latter are partitions. When designing distributed systems, it is more appropriate to use the concept of containment regions since they may cross a variety of hardware boundaries and also be logical in nature (e.g. peer processes). In this environment, the ability to provide real-time fault tolerance and performance features are also required for control systems and the independence between the systems that was previously provided through separate hardware and dedicated communications buses must be established when the applications must share critical hardware and software resources.

The enforcement of separation between applications of different criticalities is a multi-dimensional problem. First, the separation and functional and physical dependencies at varying criticality levels for a single systems aspect (such as safety) must be addressed. Next, the criteria must also be evaluated across the multiple system aspects of interest and a means of achieving the overall combination of objectives. Each architecture component may be considered at different criticality levels across aspects (for example a component is safety crucial but not security critical) and the criticality level in each of the aspects must be considered when evaluating partitioning and physical and functional dependencies.

CONCEPTUAL BASIS

Fundamental to our approach for mixed critical system design is a capability to express criticality in a structured manner for all aspects. This capability is supported by a model based environment for rapid specification and evaluation and allows sets of criticality attributes to be created that can be assigned to architecture components. Notionally, a criticality attribute set is a tuple with an entry for each of the system aspects under consideration thus permitting as much rigor to the descriptions are necessary. This provides a powerful method for describing mixed levels of criticality across many system aspects. Architectural abstraction (hierarchy) and relations (component bindings) are leveraged to flow criticality specifications through the architecture to create a criticality map of the system.

Each of the criticality aspects considered under this effort (performance, dependability, safety and security) is defined by a set of meta-data tags that can be used to establish the levels of criticality within each of these aspects. The meta-data tags contain attributes that characterize the criticality within the aspect and are suitable for comparing criticality between architecture components that have been assigned tags. These tags form the basis of aspect criticality groupings based on the tag values and the cross criticality dependency analysis we have proposed.

When working with the tags, we strive to provide a flexible definition that can be used to represent tags that are useful under varying development processes and system certification regimes. The criticality tags are designed for rapid adoption by programs by providing tags that use the criticality definitions from standard certification processes. We also provide extensible tag definitions so that they can be customized to the needs of specific systems under development or new processes and standards as they are developed

Our objective is to develop methods to specify a flexible and extensible set of foundational criticality tags. Tags enable stakeholders to identify criticality values for dependability, safety, security and performance and assign the tags to system components. Furthermore, the tags can be flowed down architectural hierarchies from higher levels of the system architecture down to the deployment level comprised of the computing platform. As WW Technology Group's EDICT tool [1] [2] [3] provides a robust tool environment for developing the tag framework, it provides the necessary environment and features to model, edit, and apply tags to system models.

MODELING STEPS

A key element in our strategy for Mixed Critical Architecture systems is to understand how each aspect is intended to work, first individually and then integrated with other aspects. The individual aspect problem is therefore broken into steps that require:

a) Specification of aspect regions and the components that reside in the region.

b) Evaluation of criticality levels within an aspect and identification of conflicts within the aspect (conflicts could be dependencies that violate criticality precedence hierarchy or separation violations that compromise criticality).

c) Removal of criticality conflicts within the aspect through several methods:

   1. *Aspect Re-Characterization* –
      Redefine the criticality tags within an aspect to remove conflicts.

   2. *Aspect Criticality Separation* –
      Re-allocation of functionality to separate conflicting aspect criticalities into partitionable areas.

   3. *Aspect Criticality Boundary Enforcement* – provide architectural services that enforce the separation identified by the aspect criticality boundaries.

4. *Aspect Criticality Dependency Enforcement –* provide architectural services that enforce the desired dependency flow between criticalities within an aspect and provide the protections required to ensure cooperation without compromise of the aspect regions.

d) Repeat steps b/c until all conflicts have been resolved for the aspect.

The underlying theory we will use is based on the concept of an *aspect region*. The use of boundaries as an underlying concept has been proposed for safety, security and dependability. The nature of these boundaries requires a degree of rigor in order to support artifact generation for certification.

Of particular interest for a mixed-critical system is:

- the separation and/or cooperation of these aspect regions for a given system implementation to ensure violations do not exist across any boundaries where separation is necessary.

- the ability to cooperatively share resources and overhead functions when allowable in order to maximize efficiency of the system architecture.

ASPECT SPECIFICATION

The system aspect boundaries are comprised from the union of sets of individual aspect regions. Each aspect region consists of groupings of system functions or components that have a common aspect criticality tag. Aspect regions are comprehensive in architectural scope, that is, every component in system architecture must belong to at least one region.

Once the regions are defined through the assignment of criticality tags to system functions and components, the process is further detailed by aspect specific specifications or with system level analysis for dependability, safety or security. The aspect criticality tags follow the form and format described in the following sections and are used to define the aspect regions by associating the tags with components in the architecture model.

The criticality tags for an aspect can be assigned at various levels of abstraction in the architecture model. A common approach is to assign the criticality tags at a logical or functional level of the architecture and then use the hierarchical decomposition of the architecture model to enable the flow down of the tags to lower level system components. This approach matches well with common development practices where major system functions or subsystems are designated with criticality levels based on stakeholder concerns. Safety criticality is often assigned to system functions based on a preliminary hazard analysis that generates a set of system hazards and the conditions that can lead to hazard occurrence. This analysis often assigns criticalities to the hazard, which in turn can be the basis for safety criticality tags.

In addition to the hierarchical flow down of criticality tags to establish aspect regions, the binding of software components to the platform architecture is used to define the scope of aspect regions. The allocation of system functions (in terms of processes and threads) to the execution platform (processors, partitions, networks, etc.) has a major impact on the definition of aspect regions and the identification of aspect region conflicts. In our approach the aspect criticality tags follow the software components as they are bound to the platform. This technique provides extensive flexibility in aspect region specification because as the bindings are modified the aspect region definitions are automatically modified as well.

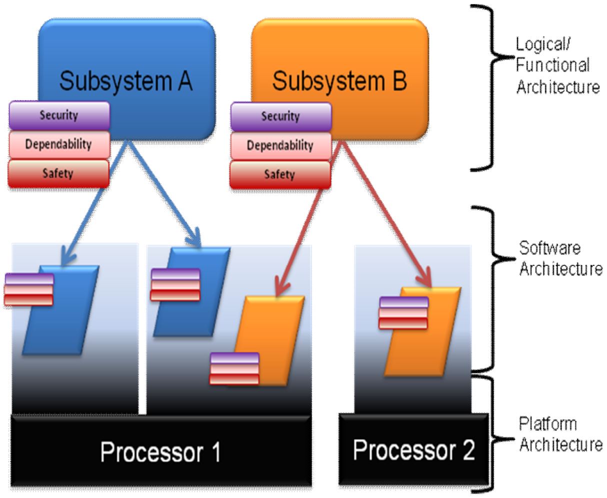


Figure 2 - Aspect Criticality Tag Assignment

The concept of aspect criticality tag flow down and binding is illustrated in Figure 2. In this example, Subsystem A and Subsystem B are assigned a tuple of criticality tags and the tags are flowed down to the set of threads that implement the subsystem functions in the software architecture. These threads are in turn bound to a set of partitions and processors in the platform architecture. The binding of the software components to the platform architecture sets the stage for the evaluation of the aspect regions.

Dependability Meta-Data Tags

The dependability aspect tag encompasses a specification for the level of reliability and availability required of the system component along with a general description of a dependability level. The tag definition will provide the ability to expand the definitions to other dependability level sets beyond the two initial sets specified here. Figure 3 depicts the structure of the tag design for dependability.

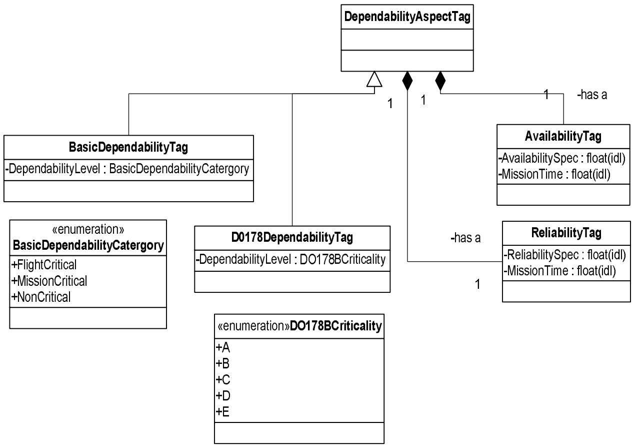

DependabilityAspectTag
1
1
1
-has a
BasicDependabilityTag
-DependabilityLevel : BasicDependabilityCatergory
AvailabilityTag
-AvailabilitySpec : float(idl)
-MissionTime : float(idl)
«enumeration»
BasicDependabilityCatergory
+FlightCritical
+MissionCritical
+NonCritical
D0178DependabilityTag
-DependabilityLevel : DO178BCriticality
-has a
ReliabilityTag
-ReliabilitySpec : float(idl)
-MissionTime : float(idl)
1
«enumeration»DO178BCriticality
+A
+B
+C
+D
+E

Figure 3 - Dependability Criticality Tags

Each dependability tag contains a dependability level specification, a reliability tag and an availability tag. The reliability and availability tags are composed of a numerical specification and a mission time. An availability and reliability tag will be specified for each of the dependability levels that are contained within the dependability tag. This provides flexibility for system analysts to provide customized reliability and availability specifications for each level of dependability.

The baseline tag definition provides two types of dependability tags: Basic and DO-178B. Each dependability tag type contains a list of criticality levels to define criticality of dependability within the type. For instance the Basic Dependability Tag uses 3 levels (1) Flight Critical, (2) Mission Critical and (3) Non Critical while the DO-178B tag set uses the five criticality levels (A – E) defined by DO-178B [4].

Safety Meta-Data Tags

The set of meta-data tags for the safety aspect, shown in Figure 4, provides three types of criticality specifications. These types of safety tags provide a basic safety critical designation along with options for military standard 882 development or commercial flight systems under DO-178B.

The basic safety tag provides the most rudimentary definition of safety criticality, safety critical or not critical. This tag is useful for systems that are not bound to a specific development process and have low levels of complexity. This tag can also be useful early in design processes where the safety criticality of architecture components has been established but detailed hazard assessments have not yet been completed.

The 882 Safety Tag provides a set of criticality values that correspond with the software control categories used in the standard to define the level of control the function provides. This rating of I through IV is combined with a severity metric that rates the impact of the improper operation of the function on system hazards. The levels are described in the left hand portion of Figure 5. The DO-178B safety tag uses the same set of criticality levels that were used in the dependability tags. These levels are defined in the right hand portion of Figure 5.

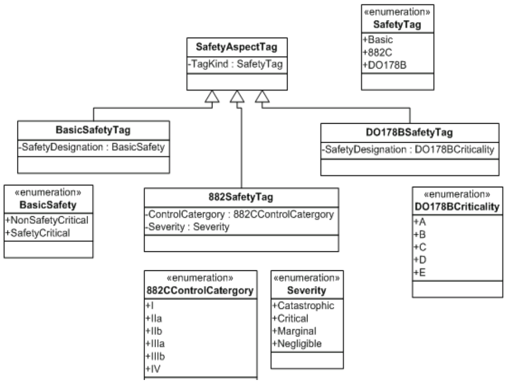
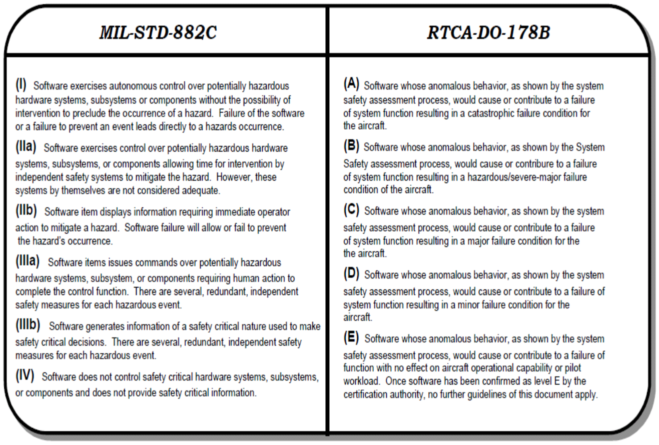


Figure 4 - Safety Criticality Tags

Security Meta-Data Tags

The security tags are capable of defining the security criticality of an architecture element using attributes from information assurance certification processes. These processes provide the background required to rigorously define the security attributes and also provide a well-accepted methodology for designating security levels.

Figure 6 depicts the tag design for the Security Aspect Tag. All of the supported security tags contain a classification designation. This designation represents the highest level of information classification that the component processes. The base classification level is augmented by tag types that are designed to support standardized information assurance certification processes. The initial set of tags will support JAFAN 6/3 [6] , DCID 6/3 [8] and DoD 8500.2 [7].



MIL-STD-882C	*RTCA-DO-178B*
(I) Software exercises autonomous control over potentially hazardous hardware systems, subsystems or components without the possibility of intervention to preclude the occurrence of a hazard. Failure of the software or a failure to prevent an event leads directly to a hazards occurrence. **(IIa)** Software exercises control over potentially hazardous hardware systems, subsystems, or components allowing time for intervention by independent safety systems to mitigate the hazard. However, these systems by themselves are not considered adequate. **(IIb)** Software item displays information requiring immediate operator action to mitigate a hazard. Software failure will allow or fail to prevent the hazard's occurrence. **(IIIa)** Software items issues commands over potentially hazardous hardware systems, subsystem, or components requiring human action to complete the control function. There are several, redundant, independent safety measures for each hazardous event. **(IIIb)** Software generates information of a safety critical nature used to make safety critical decisions. There are several, redundant, independent safety measures for each hazardous event. **(IV)** Software does not control safety critical hardware systems, subsystems, or components and does not provide safety critical information.	**(A)** Software whose anomalous behavior, as shown by the system safety assessment process, would cause or contribute to a failure of system function resulting in a catastrophic failure condition for the aircraft. **(B)** Software whose anomalous behavior, as shown by the System Safety assessment process, would cause or contribure to a failure of system function resulting in a hazardous/severe-major failure condition of the aircraft. **(C)** Software whose anomalous behavior, as shown by the system safety assessment process, would cause or contribute to a failure of system function resulting in a major failure condition for the the aircraft. **(D)** Software whose anomalous behavior, as shown by the system safety assessment process, would cause or contribute to a failure of system function resulting in a minor failure condition for the aircraft. **(E)** Software whose anomalous behavior, as shown by the system safety assessment process, would cause or contribute to a failure of function with no effect on aircraft operational capability or pilot workload. Once software has been confirmed as level E by the certification authority, no further guidelines of this document apply.

Figure 5 - MIL882C and DO-178B Safety Criticality [5]

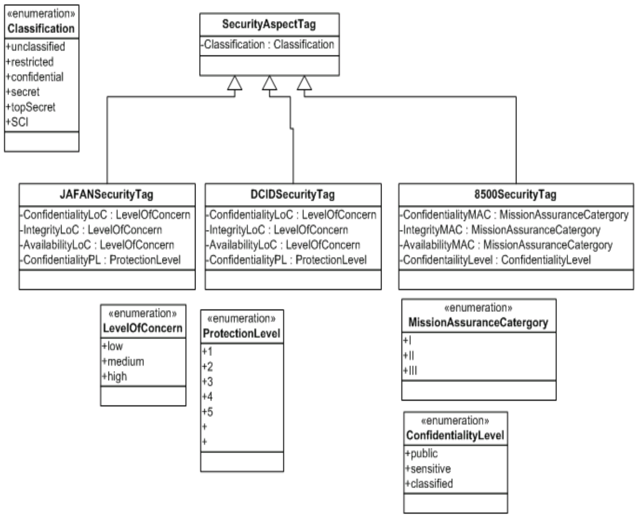
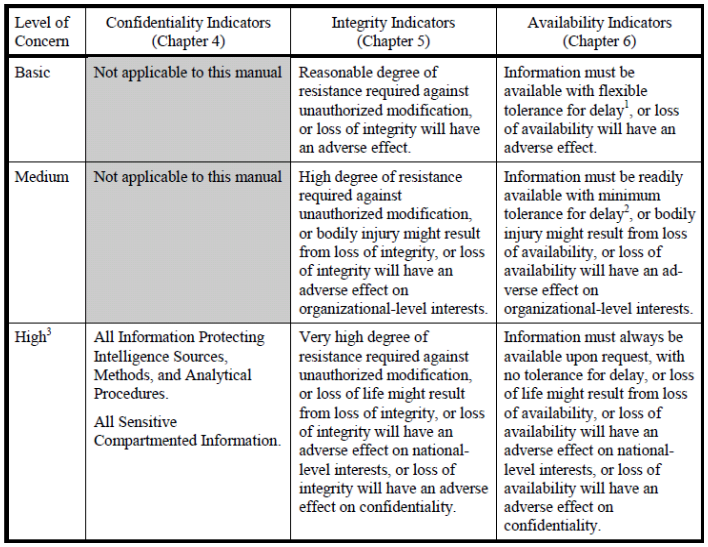

«enumeration»
Classification
+unclassified
+restricted
+confidential
+secret
+topSecret
+SCI
SecurityAspectTag
-Classification : Classification
JAFANSecurityTag
-ConfidentialityLoC : LevelOfConcern
-IntegrityLoC : LevelOfConcern
-AvailabilityLoC : LevelOfConcern
-ConfidentialityPL : ProtectionLevel
DCIDSecurityTag
-ConfidentialityLoC : LevelOfConcern
-IntegrityLoC : LevelOfConcern
-AvailabilityLoC : LevelOfConcern
-ConfidentialityPL : ProtectionLevel
8500SecurityTag
-ConfidentialityMAC : MissionAssuranceCatergory
-IntegrityMAC : MissionAssuranceCatergory
-AvailabilityMAC : MissionAssuranceCatergory
-ConfidentailityLevel : ConfidentialityLevel
«enumeration»
LevelOfConcern
+low
+medium
+high
«enumeration»
ProtectionLevel
+1
+2
+3
+4
+5
+
+
«enumeration»
MissionAssuranceCatergory
+I
+II
+III
«enumeration»
ConfidentialityLevel
+public
+sensitive
+classified

Figure 6 - Security Criticality Tags

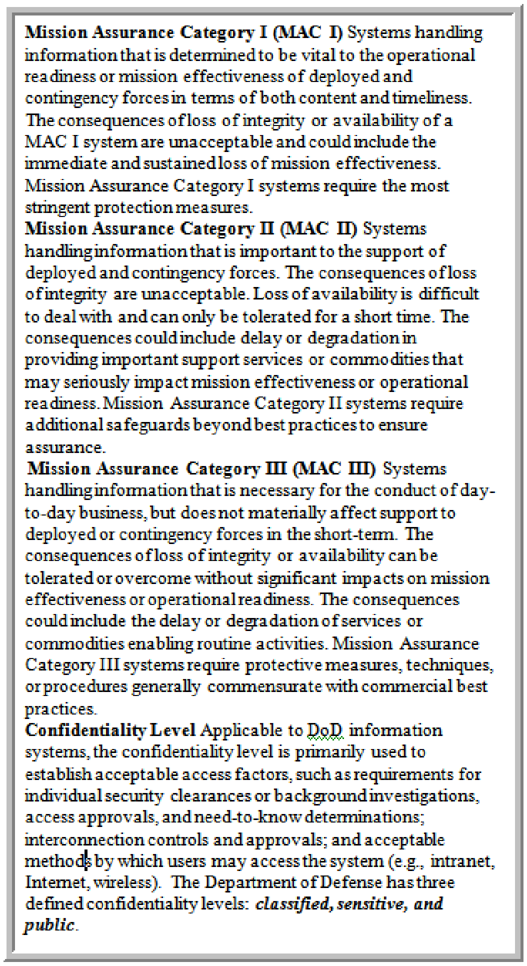
JAFAN 6/3 and DCID 6/3 both use the same set of attributes for security criticality: levels of concern for Confidentiality, Integrity and Availability (CIA) and a Protection Level. The definitions for the levels of concern for CIA are shown in Figure 7. The protection levels 1 -5 are defined based on the mix of classification levels and user access that is in the system.

Another IA certification standard that applies to many DoD systems is 8500.2. This process rates the security criticality of systems with a set of Mission Assurance Categories for each of the Confidentiality, Integrity and Availability concerns. These categories are supported by a Confidentiality level that is used to address access control factors. These attributes are defined in Figure 8. We have defined a security tag that uses these attributes to describe criticality in 8500.2 terminologies.



Level of Concern	Confidentiality Indicators (Chapter 4)	Integrity Indicators (Chapter 5)	Availability Indicators (Chapter 6)
Basic	Not applicable to this manual	Reasonable degree of resistance required against unauthorized modification, or loss of integrity will have an adverse effect.	Information must be available with flexible tolerance for delay[1], or loss of availability will have an adverse effect.
Medium	Not applicable to this manual	High degree of resistance required against unauthorized modification, or bodily injury might result from loss of integrity, or loss of integrity will have an adverse effect on organizational-level interests.	Information must be readily available with minimum tolerance for delay[2], or bodily injury might result from loss of availability, or loss of availability will have an ad-verse effect on organizational-level interests.
High[3]	All Information Protecting Intelligence Sources, Methods, and Analytical Procedures. All Sensitive Compartmented Information.	Very high degree of resistance required against unauthorized modification, or loss of life might result from loss of integrity, or loss of integrity will have an adverse effect on national-level interests, or loss of integrity will have an adverse effect on confidentiality.	Information must always be available upon request, with no tolerance for delay, or loss of life might result from loss of availability, or loss of availability will have an adverse effect on national-level interests, or loss of availability will have an adverse effect on confidentiality.

Figure 7- JAFAN LoC Definition



Mission Assurance Category I (MAC I) Systems handling information that is determined to be vital to the operational readiness or mission effectiveness of deployed and contingency forces in terms of both content and timeliness. The consequences of loss of integrity or availability of a MAC I system are unacceptable and could include the immediate and sustained loss of mission effectiveness. Mission Assurance Category I systems require the most stringent protection measures.

Mission Assurance Category II (MAC II) Systems handling information that is important to the support of deployed and contingency forces. The consequences of loss of integrity are unacceptable. Loss of availability is difficult to deal with and can only be tolerated for a short time. The consequences could include delay or degradation in providing important support services or commodities that may seriously impact mission effectiveness or operational readiness. Mission Assurance Category II systems require additional safeguards beyond best practices to ensure assurance.

Mission Assurance Category III (MAC III) Systems handling information that is necessary for the conduct of day-to-day business, but does not materially affect support to deployed or contingency forces in the short-term. The consequences of loss of integrity or availability can be tolerated or overcome without significant impacts on mission effectiveness or operational readiness. The consequences could include the delay or degradation of services or commodities enabling routine activities. Mission Assurance Category III systems require protective measures, techniques, or procedures generally commensurate with commercial best practices.

Confidentiality Level Applicable to DoD information systems, the confidentiality level is primarily used to establish acceptable access factors, such as requirements for individual security clearances or background investigations, access approvals, and need-to-know determinations; interconnection controls and approvals; and acceptable methods by which users may access the system (e.g., intranet, Internet, wireless). The Department of Defense has three defined confidentiality levels: ***classified, sensitive, and public***.

Figure 8 - 8500.2 MAC Levels

TOOL SUPPORT

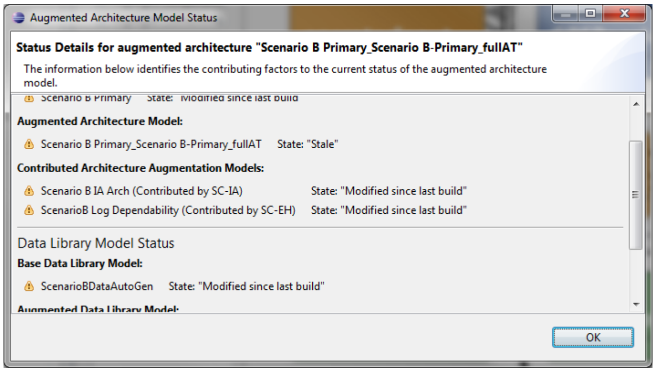
The EDICT Tool Suite by WW Technology Group [1] [2] [3] is a model based engineering platform that provides a means for establishing coherent views of organization and behavior by integrating architectural and analytical models of systems and their constituent components/services. The EDICT tool provides a robust tool environment for developing the tag framework and possesses the features needed to model, edit, and apply tags of all types to system models. The creation of common and reusable tagging components within EDICT enables faster and more flexible development of the various individual tag types.

The EDICT tools provide two primary features to define architectures: the Logical Architecture Editor and the System Architecture Editor. The Logical Architecture Editor allows users to define an abstract functional organization of a system and capture it in a model. The Logical Architecture Model describes the design from a functional decomposition stand-point that captures initial structural decisions as to how the system's functions and services will be broken out and related to each other. The System Architecture Model provides a description of the physical components in a system and the connections through which they interact. The System Architecture model describes the computing platform and software elements that implement the internals of the functional areas identified in the Logical Architecture Model. It is often useful for design completeness and various types of aspect analysis to capture the relationship between these two representations of the design.

Tagging Logical Architectures

Tool support provides for the creation a Logical Architecture Aspect Model that associates individual aspect tags with components (or groups of components) within a logical architecture model. This association also acts as an augmentation model on the architecture model; it appends the base logical architecture model with tagging information, so a single unified augmented model can hold tagging information for all tag types. The aspect model also ties into the general EDICT framework for augmenting models, and leverages all the existing augmentation features. For example, it appears in common displays for viewing augmentation models as shown in Figure 9.

User interface features allow for creating, viewing, and editing these aspect models with wizards that enable users to add a new aspect model to their current design effort. The wizard allows users to select a particular set of tags to associate with the current logical architecture model, and guides them through the process of updating their current design effort and design option with the new model. Once created, these aspect models can be viewed and edited (see Figure 10).


Augmented Architecture Model Status

Status Details for augmented architecture "Scenario B Primary_Scenario B-Primary_fullAT"

The information below identifies the contributing factors to the current status of the augmented architecture model.

Scenario B Primary State: "Modified since last build"

Augmented Architecture Model:

Scenario B Primary_Scenario B-Primary_fullAT State: "Stale"

Contributed Architecture Augmentation Models:

Scenario B IA Arch (Contributed by SC-IA) State: "Modified since last build"

ScenarioB Log Dependability (Contributed by SC-EH) State: "Modified since last build"

Data Library Model Status

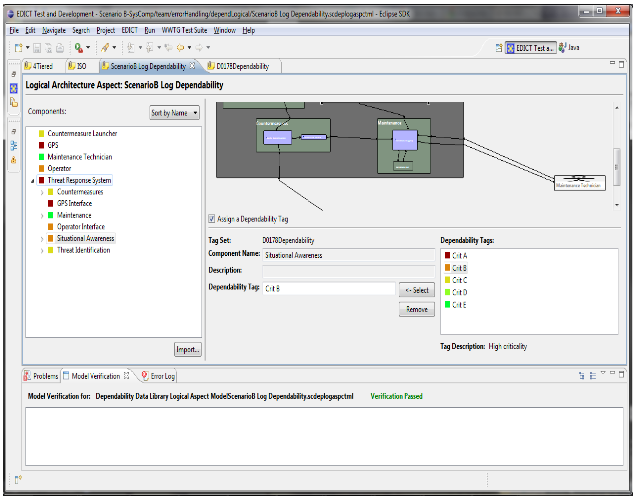
Base Data Library Model:

ScenarioBDataAutoGen State: "Modified since last build"

Augmented Data Library Model:

Figure 9: Logical Tag Augmentation status display

As can be seen in this screenshot, the editor is composed of a number of controls. The top left portion of the editor displays a list of all the components in the current logical architecture model. This list can be sorted by name, or by assigned tag type. This list is a hierarchical tree view, with child/parent components shown in a nested fashion. Each component is accompanied with an icon representing their tag assignment. Each tag has an associated colored icon along a red-green color spectrum, representing the criticality of that tag (Green = least critical, Red = most critical). For components with no children, their associated


EDICT Test and Development - Scenario B-SysComp/team/errorHandling/dependLogical/ScenarioB Log Dependability.scdeplogaspctml - Eclipse SDK

File Edit Navigate Search Project EDICT Run WWTG Test Suite Window Help
EDICT Test a...
Java
4Tiered
ISO
ScenarioB Log Dependability
D0178Dependability

Logical Architecture Aspect: ScenarioB Log Dependability

Components: Sort by Name

- 🟨 Countermeasure Launcher
- 🟥 GPS
- 🟩 Maintenance Technician
- 🟧 Operator
- 🟥 Threat Response System
  - 🟨 Countermeasures
  - 🟥 GPS Interface
  - 🟩 Maintenance
  - 🟧 Operator Interface
  - 🟧 Situational Awareness
  - 🟨 Threat Identification

☑ Assign a Dependability Tag

Tag Set: D0178Dependability

Component Name: Situational Awareness

Description:

Dependability Tag: Crit B

Dependability Tags:

- 🟥 Crit A
- 🟧 Crit B
- 🟨 Crit C
- 🟩 Crit D
- 🟩 Crit E

Tag Description: High criticality

Model Verification for: Dependability Data Library Logical Aspect ModelScenarioB Log Dependability.scdeplogaspctml Verification Passed

Figure 10: Logical Architecture Tag Aspect Editor

tag icon is simply that of the tag they have been assigned. For components with sub-components, the displayed icon represents the most critical dependability tag that any of their child components has been assigned.

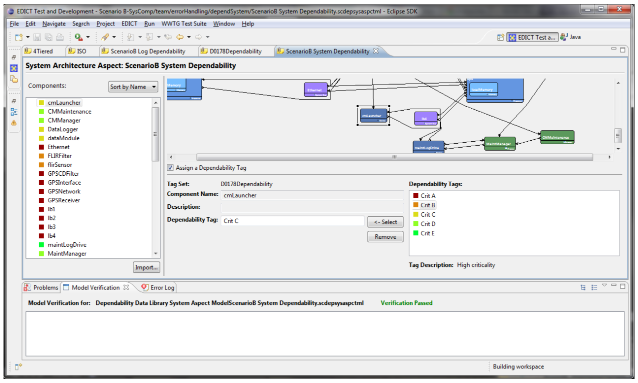
The top-right portion of the display shows a visual representation of the logical architecture model. The currently selected component in the tree view will be highlighted in the visualization. The visualization can be manipulated as would be expected of any EDICT architecture visualization: it can be zoomed, scrolled, etc. The bottom-right portion of the visualization shows a listing of all the tags in the associated tag set, and a description of the currently selected tag. There are controls for assigning and removing a tag to the currently selected component. The currently selected component's name and description are also shown.

A checkbox is also shown in the middle of the display. This checkbox can be unchecked to specify that a component, or set of components, should be explicitly left untagged. This indicates in the architecture model that certain components are deliberately untagged, and not to be considered to have any aspect assignment. When this checkbox is unchecked, the entire lower-right hand portion of the display will be disabled. This reinforces that no tags are to be assigned, and prevents the user from performing an inconsistent tag assignment operation.

Shown at the very bottom of this screenshot is the model verification view. This floating view will display any known problems that might arise with the aspect model

Tagging System Architectures

A separate, but visually similar set of features is provided for associating tags with system architectures. The editor (Figure 11) provides a corresponding feature set as provided for logical architectures. The architecture model is shown in a hierarchical tree form, with a visualization of


EDICT Test and Development - Scenario B-SysComp/team/errorHandling/dependSystem/ScenarioB System Dependability.scdepsysaspctml - Eclipse SDK
File Edit Navigate Search Project EDICT Run WWTG Test Suite Window Help
EDICT Test a...
Java
4Tiered
ISO
ScenarioB Log Dependability
D0178Dependability
ScenarioB System Dependability
Sort by Name
Ethernet
localMemory
cmLauncher
lb4
maintLogDrive
MaintManager
CMMaintenance
Import...
<- Select
Remove
Problems
Model Verification
Error Log
Building workspace

System Architecture Aspect: ScenarioB System Dependability

Components:

- 🟨 cmLauncher
- 🟩 CMMaintenance
- 🟩 CMManager
- 🟨 DataLogger
- 🟨 dataModule
- 🟥 Ethernet
- 🟧 FLIRFilter
- 🟧 flirSensor
- 🟥 GPSCDFilter
- 🟥 GPSInterface
- 🟥 GPSNetwork
- 🟥 GPSReceiver
- 🟥 lb1
- 🟥 lb2
- 🟥 lb3
- 🟥 lb4
- 🟩 maintLogDrive
- 🟩 MaintManager

☑ Assign a Dependability Tag

Tag Set: D0178Dependability
Component Name: cmLauncher
Description:
Dependability Tag: Crit C

Dependability Tags:

- 🟥 Crit A
- 🟧 Crit B
- 🟨 Crit C
- 🟩 Crit D
- 🟩 Crit E

Tag Description: High criticality

Model Verification for: Dependability Data Library System Aspect ModelScenarioB System Dependability.scdepsysaspctml Verification Passed

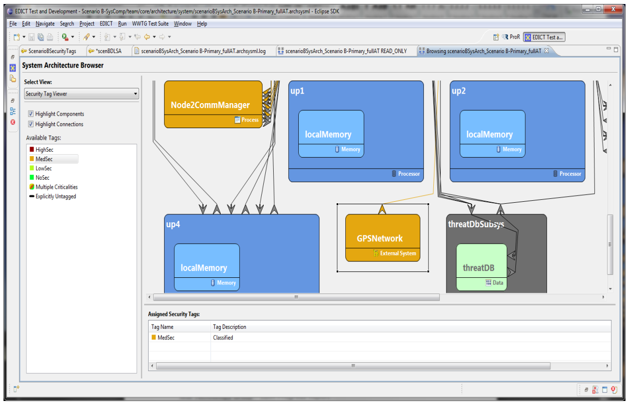
Figure 11: System Architecture Tag Aspect Editor

the entire system architecture. Components are listed with color-coded icons, and the same basic tag and component data is displayed. Tags can be assigned to components or groups of components, and removed/reassigned with same controls. A model verifier is provided to update the Model Verification View with information about the displayed aspect model.

Viewing the System Architecture

A set of visualizations for viewing aspects provides a central place for users to explore and visualize various aspects of a system architecture model. The viewer shown in Figure 12 allows the user to visually explore the model from a tag-based perspective. Controls along the top left allow users to highlight tagged components and/or tagged connections. When the components checkbox is selected, this view will show aspect tags applied to system components. When the connections checkbox is selected, this view will show connections that carry data that has been tagged via the data library aspect model for tags.

This example, illustrating security tags, shows a list of tags from the security tag set and an option for showing components with multiple criticalities. When a given tag is selected, the visualization in the right-hand side of the display will update to highlight components and/or connections that have been augmented with the selected security tag. If the “Multiple Criticalities” entry is selected, then components with more than one security tag will be highlighted. This feature is useful for identifying components that must support multiple levels of criticality. The final option is for explicitly untagged items. When that option is selected, all items that have been marked as explicitly having no security values will be highlighted.


EDICT Test and Development - Scenario B-SysComp/team/core/architecture/system/scenarioBSysArch_Scenario B-Primary_fullAT.archsysml - Eclipse SDK
File Edit Navigate Search Project EDICT Run WWTG Test Suite Window Help
ProR
EDICT Test a...
ScenarioBSecurityTags
*scenBDLSA
scenarioBSysArch_Scenario B-Primary_fullAT.archsysml.log
scenarioBSysArch_Scenario B-Primary_fullAT READ_ONLY
Browsing scenarioBSysArch_Scenario B-Primary_fullAT
Node2CommManager
Process
up1
localMemory
Memory
Processor
up2
localMemory
Memory
Processor
up4
localMemory
Memory
GPSNetwork
External System
threatDbSubsys
threatDB
Data

System Architecture Browser

Select View: Security Tag Viewer

☑ Highlight Components
☑ Highlight Connections

Available Tags:

- 🟥 HighSec
- 🟧 MedSec
- 🟩 LowSec
- 🟩 NoSec
- ● Multiple Criticalities
- ━ Explicitly Untagged

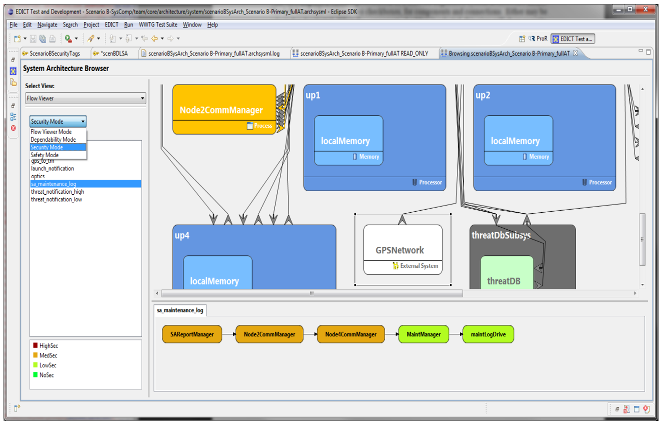
Assigned Security Tags:

Tag Name	Tag Description
🟧 MedSec	Classified

Figure 12: System Architecture Tag Viewer

The visualization can be scrolled, zoomed, selected, etc. as with a typical EDICT architecture view. When a particular component is highlighted, the bottom portion of the display will show a list of all the security tags that apply to the selected component. This enables a user to see all the security tag values that have been applied to a component, if more than one has been applied.

Other tag viewing features within the system architecture browser have also been updated to make use of security tags. One such feature is the flow viewer within the system architecture browser (Figure 13).


EDICT Test and Development - Scenario B-SysComp/team/core/architecture/system/scenarioBSysArch_Scenario B-Primary_fullAT.archsysml - Eclipse SDK
File Edit Navigate Search Project EDICT Run WWTG Test Suite Window Help
ProR
EDICT Test a...
ScenarioBSecuritytags
*scenBDLSA
scenarioBSysArch_Scenario B-Primary_fullAT.archsysml.log
scenarioBSysArch_Scenario B-Primary_fullAT READ_ONLY
Browsing scenarioBSysArch_Scenario B-Primary_fullAT
Flow Viewer Mode
Dependability Mode
Security Mode
Safety Mode
Node2CommManager
Process
up1
localMemory
Memory
Processor
up2
localMemory
Memory
Processor
up4
localMemory
GPSNetwork
External System
threatDbSubsys
threatDB
SAReportManager
Node2CommManager
Node4CommManager
MaintManager
maintLogDrive

System Architecture Browser

Select View: Flow Viewer

Security Mode

gps_to_tm
launch_notification
optics
sa_maintenance_log
threat_notification_high
threat_notification_low

- 🟥 HighSec
- 🟧 MedSec
- 🟩 LowSec
- 🟩 NoSec

sa_maintenance_log

Figure 13: Flow Viewer Security Mode

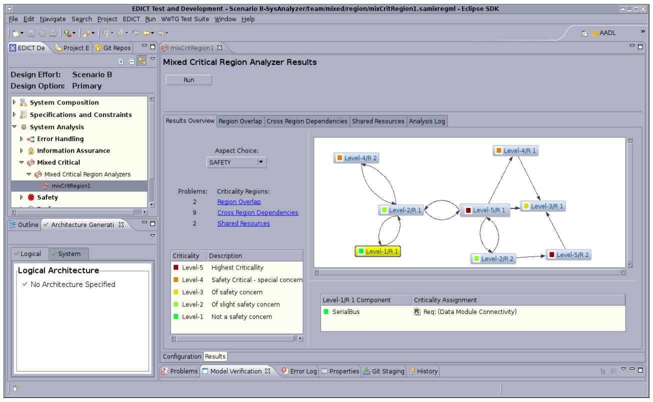
This viewer allows users to explore data flows between components within the system architecture. The left-hand side portion of the view shows a list of all the end-to-end flows within the current system architecture. To the right is a visualization of the current system architecture. When a flow is selected, the visualization updates to highlight the components and connections that compose the selected flow. Additionally, a simplified representation of the selected flow is displayed along the bottom-right portion of the display.

When the security tag view option is selected, the bottom of the display also shows a list of the security tags that make up the security tag set applied to the current augmented architecture. Additionally, the bottom-right visual will be color-coded with the security tags that have been applied to the components and connections that make up the selected flow. These visualizations make it easy to see how the criticality level varies along the entire path of the flow.

A listing appears below of the aspect tags contained in the current model. When a given tag is selected, a visualization on the right-hand side of the display highlights components and connections associated with the given tag. The highlighting color will be the same shade as the icon shown next to the current selected tag. Users can manipulate the visualization (scroll, zoom, etc.) to navigate the system model and get a sense for how dependability tags have been applied.

Viewing Mixed Criticality Regions

The creation of the Mixed Criticality Region Model is performed dynamically in an analyzer. After running the analyzer, an overview of the region model is displayed depicting the regions in a directed graph. The overview page is shown in Figure 14 with an example Mixed Criticality Region Model displayed. The center portion of the page shows the region model with each region and all of the region dependencies. This display uses several visual cues to help the user understand the model. The regions display the color associated with the criticality level of the region. The display also uses size to indicate general relations of the region to the underlying architecture. The size of each region is based on the number of components that have been allocated to the region (more component = bigger region) and the number of architectural interfaces that cross region boundaries (thicker line = more interfaces).



EDICT Test and Development - Scenario B-SysAnalyzer/team/mixed/region/mixCritRegion1.samixregml - Eclipse SDK

Design Effort: Scenario B
Design Option: Primary

Logical Architecture
✓ No Architecture Specified

Mixed Critical Region Analyzer Results

Aspect Choice: SAFETY

Problems:	Criticality Regions:
2	Region Overlap
9	Cross Region Dependencies
2	Shared Resources

Criticality	Description
🟥 Level-5	Highest Criticallity
🟧 Level-4	Safety Critical - special concern
🟨 Level-3	Of safety concern
🟩 Level-2	Of slight safety concern
🟩 Level-1	Not a safety concern

Level-1/R 1 Component	Criticality Assignment
🟩 SerialBus	Req: (Data Module Connectivity)

Figure 14: Table Based Display for Region Viewer

A selection box allows the designer to pick the specific aspect they are interested in viewing. By selecting a specific aspect, all of the analyzer pages will filter their display to show only the details belonging to that aspect. Beneath the aspect selection box, totals for problems found in the aspect region model are displayed. These problems point out areas where there is potential aspect overlap, cross region dependencies and areas where shared resources have criticality conflicts. Beneath the totals, the Tag Set associated with the selected aspect is displayed listing all of the criticality levels to help guide the user when exploring the aspect region model.

The components which compose the region are displayed in a table beneath the graph when a region is selected. This supporting view allows the user to browse the regions and see the relationship the regions have with the underlying architecture. By selecting a cross region dependency (connection between the regions), the interfaces that connect components from the two regions spanning the dependency are displayed in the table beneath the graph allowing the user to see the data that is transferred across region boundaries.

The Overview page also contains a tally of the problems found during region model creation. These problems can be classified as one of the following three cases:

1. *Region Overlap* – A component is found in more than one region within an aspect. This is an indication that the component is assigned functions with multiple criticality levels.

2. *Cross Region Dependency* – A region of one criticality level depends on a region of a different criticality level. This is an indication that data is flowing from one level of criticality to another and highlights key points in the architecture where data protections may need to be applied.

3. *Shared Resources* – Components from separate regions are bound to (or supported by) the same hardware components in the architecture. This indicates a shared resource which supports processing multiple criticality level.

For each of these cases, a separate viewing page is devoted to illustrating the problem in order to help the designer locate the specific architecture model objects involved.

Run-time Platform Configuration

Once the aspect criticality analysis has been completed, the system architecture that has been derived is ready to be implemented. Automatic extraction of parameters can be performed for the configuration of the run-time platform. Configuration of mechanisms that control partitioning, process allocation and communications from that validated system architecture model enhances confidence that the deployed system matches the analyzed system and eases the verification burden. The architecture model contains many essential elements for the run-time platform configuration:

- Partitioning
- Task allocations
- Task schedules
- Communication paths

This information can be used to configure system infrastructure and ensure a strong correlation between the analyzed system and the deployed system.

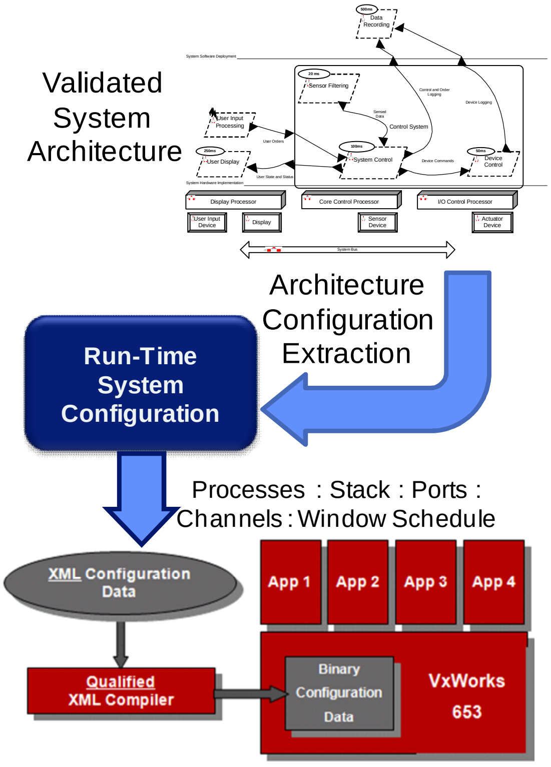


Figure 15 - Run-time Platform Configuration

Configuration information is extracted from architecture model and related property sets such as the ARINC 653 Annex under development for AADL 2 [10]. The extracted configuration data is formatted into an open XML based format for export to other tool sets through the EDICT Tools and the Eclipse tool platform. Figure 15 provides an overview of the configuration process from the validated architecture model though to the run-time component.

RTOS vendors are beginning to provide interfaces for XML based configuration of run-time platform. An example of this is the configuration approach and tools for Wind River's 653 RTOS [10]. These tools provide a path to use XML based configuration data to control the operating system features and services and their run-time configuration. This type of approach is not limited to RTOSs but can also be used for middleware components and communication services. Our approach is also viable with run-time components that do not support XML based configuration. We can support backend translators for the EDICT tools that can address the needs of vendor specific formats.

CONCLUSION

Our paper presents an approach that addresses the challenge of designing a mixed-critical system that is suitable for real-time systems with security/safety/dependability concerns. We have developed as set of techniques that will enable mixed-critical systems to be modeled, analyzed and verified early in design cycles to reduce development costs and time through early defect detection and removal, enhance system robustness through the establishment of known system properties for dependability, safety and security and reduce verification costs by providing incremental model based verification throughout the development process.

In summary, our work provides methods and tool support for:

1. ***Multi-aspect Criticality Specification*** - Techniques for the flexible and extensible specification of criticality across multiple system aspects of dependability, security and safety. These techniques enable the creation of criticality tuples to describe component criticality in multiple aspects.

2. ***Aspect Criticality Regions*** – Aspect Criticality Regions provide a powerful abstraction for representing mixed criticality levels and enabling analysis of mixed critical architectures. The regions define areas where there are criticality conflicts due to shared resources or functional allocations and identify the key interfaces between different levels of criticality.

3. ***Architectural Analysis for Criticality Regions*** – With defined regions of criticality a mixed critical architecture can be analyzed to ensure that the partitioning, information flow and protection services are deployed in the architecture to enforce the region attributes. This innovation enables early validation and trade-off analysis for mixed critical systems.

4. ***Run-time Platform Configuration from Architecture Models*** – Once architecture models are validated through aspect region analysis the configuration of the critical computing platform components (operating systems, communication services and middleware components)is extracted and used to set the characteristics of the partitioning and information flow in the deployed system. This enables a strong coupling between the validated architecture model and the deployed computing platform.

REFERENCES

[1] EDICT, http://wwtechnology.com/EDICT/

[2] LaValley, B. and Walter, C., "Information Assurance Certification with EDICT-IA", Layered Assurance Workshop affiliated with 28th Annual Computer Security Applications Conference (ACSAC). December 3-4, 2012.

[3] Walter, C. and LaValley, B., "The EDICT Tool Platform for Model Based Architecture Modeling and Analysis". In IEEE/AIAA 31st Digital Avionics Systems Conference, October, 2012.

[4] RTCA, “Software Considerations in Airborne Systems and Equipment Certification,” Radio Technical Commission for Aeronautics (RTCA), European Organization for Civil Aviation Electronics (EUROCAE), Standard Document no. DO-178B/ED-12B, December, 1992.

[5] Commitee, Joint Software System Safety. *Software System Safety Handbook.* 1999.

[6] Joint Air Force – Army – Navy JAFAN 6/3 Manual, October 15, 2004.

[7] Department of Defense Directive 8500.1, October 24, 2002.

[8] DCID 6/3 Protecting Sensitive Compartmented information within information systems. http://www.fas.org/irp/offdocs/DCID_6-3_20Manual.htm .

[9] MIL-STD-882D Standard Practice for System Safety. February 10 2000.

[10] The Architecture Analysis and Design Language (AADL) web site, http://www.aadl.info/

[11] Kinnan, L. “Use of ARINC 653 in Safety Critical Flight Systems”, Flight Software Workshop, Presentation, 2007.