System Architecture Virtual Integration: A Case Study

P. Feiler[1], L. Wrage[1], J. Hansson[1,2]

1: Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA 15213, USA
2: Chalmers University of Technology, SE-412 96 Gothenburg, Sweden

Abstract: Aerospace industry is experiencing exponential growth in the size and complexity of onboard software. It is also seeing a significant increase in errors and rework of that software. All of those factors contribute to greater cost; the current development process is reaching the limit of affordability of building safe aircraft. An international consortium of aerospace companies with government participation has initiated the System Architecture Virtual Integration (SAVI) program, whose goal is to achieve an affordable solution through a paradigm shift of—integrate then build. A key concept of this paradigm shift is an architecture-centric approach to analysis of virtually integrated system models with respect to multiple operational quality attributes such as performance, safety, and reliability. By doing so early and throughout the life cycle at different levels of fidelity, system-level faults are discovered earlier in the life cycle—reducing risk, cost, and development time. The first phase of this program demonstrated the feasibility of this new development process through a proof of concept demonstration and a return on investment analysis, which are the topics of this paper.

Keywords: virtual integration, embedded software system, safety-critical, validation & verification

1. Introduction

Aerospace industry is experiencing exponential growth in size, complexity, errors, rework and cost of their onboard software. The current development process is reaching the limit of affordability of building safe aircraft. The size in terms of source lines of code (SLOC) has doubled every four years. The cost 27M SLOC of software is estimated to exceed $10B (see Figure 1).

The Aerospace Vehicle Systems Institute (AVSI) has launched an international, industry-wide multi-phase program called System Architecture Virtual Integration (SAVI), whose members are acknowledged in Section 7. The objective of SAVI is to reduce cost/cycle-time and risk by using early and repeated "virtual" integration and analysis.

SAVI builds on model-based engineering as practiced by the system and software engineering communities, where models of different aspects of a system are developed and analyzed (see Figure 2). However, industrial practice has shown that such independently developed models tend to result in multiple versions of the "truth." SAVI improves current practice by ensuring that analytical models are consistent with each other and the evolving architecture throughout the lifecycle. To achieve these goals, the SAVI paradigm necessitates

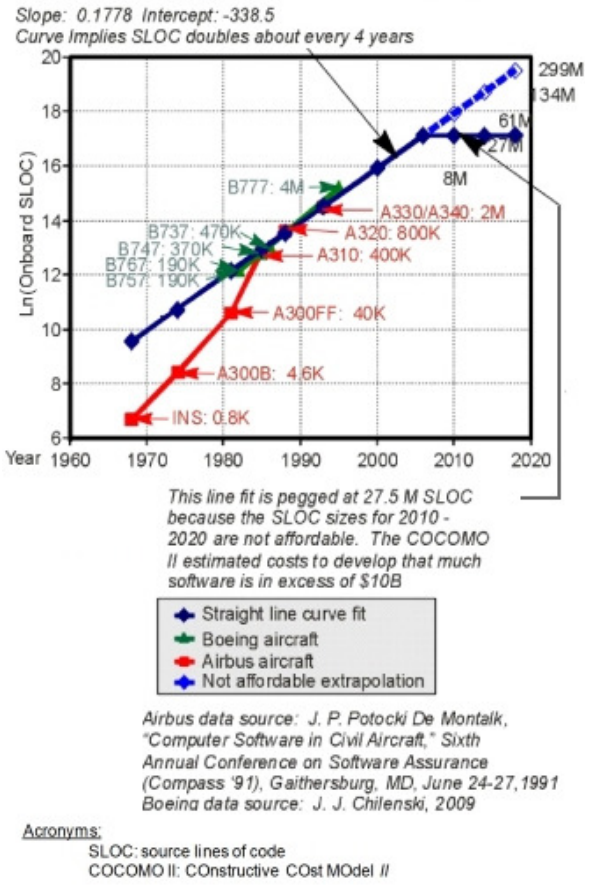

Figure 1: Estimated Onboard SLOC Growth and associated Cost

- an architecture-centric multi-aspect reference model approach as the single source of "truth",
- a component-based framework in support of model-based and proof-based engineering,
- a model bus for consistent model interchange between repositories and tools, and
- an architecture-centric acquisition and development process throughout the system life cycle that is supported by industrial standards and tool infrastructure.

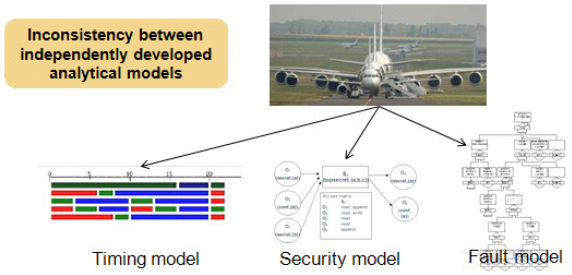

Figure 2: Multiple Truths due to Inconsistent Models

2. Key Concepts of SAVI

SAVI is driven by three key concepts: a single source architecture reference model, a model repository and model bus, and a model-based development process that encompasses both system engineering and embedded software system engineering.

2.1 Single Source Architecture Reference Model

A key concept of virtual integration is the use of an annotated architecture model with well-defined semantics as the single source for architecture analysis, as illustrated in Figure 3. The annotations provide relevant analysis-specific information (e.g., fault rates or security properties) that allow auto-generation of an analytical models.

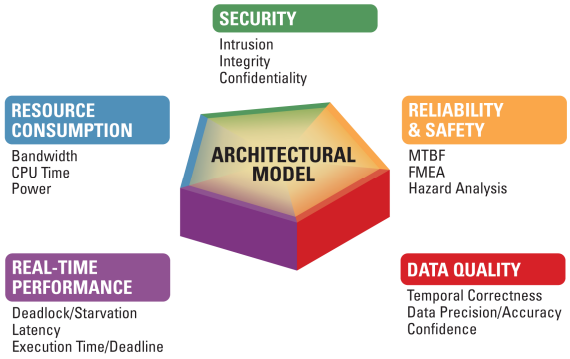

Figure 3: Single Source Annotated Architecture Model

Any changes to the architecture throughout the life cycle are reflected in this model and automatically propagate to all dimensions of analysis. For example, substitution of a faster processor to accommodate a high workload is reflected in schedulability analysis, impacts end-to-end response time, and requires re-evaluation power consumption against capacity as well as its impact on the mass of the system.

2.2 Model Repository and Model Bus

A second key concept is the use of a model repository and model bus, illustrated in Figure 4. The model repository contains the annotated architecture reference model, as well as detailed models that are refinements of architecture components. For example, details of physical system components can be modeled with Modelica, control system components with Simulink, and discrete application behavior with UML statecharts. This architecture reference model may reside in a single model repository or it may be distributed across different model repositories, e.g., those of different suppliers and the system integrator.

The model bus is a data interchange mechanism a mechanism that operates with a standardized model representation. It supports interchange between model repositories and translation into – as much as possible standardized - representations acceptable to different analysis and generation tools. For example, it supports the interchange of annotated architecture models in a standardized XML format. Similarly, transformation specifications provide the translation from an annotated architecture model to specific analytical model formats, such as those for timing models, fault trees, or security models.

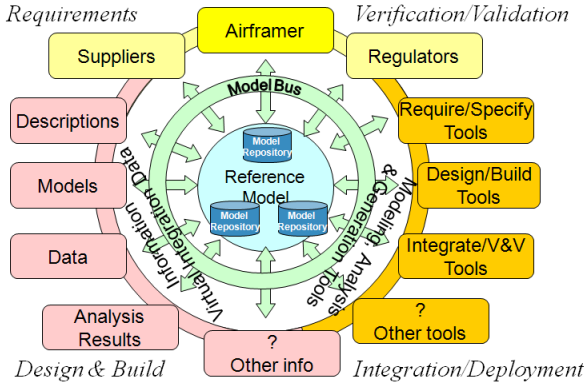

Figure 4: Model Repository and Model Bus

2.3 Development Process

The SAVI development process must support collaboration between

- system engineers—whose primary focus is the architecture of the physical system
- embedded software system engineers—whose focus is the interaction between the physical system architecture, the computer platform architecture, and the task and communication architecture of the embedded application
- other engineers—whose focus is the architecture and detailed design of physical components, computer hardware components, and software components.

Thus, the system is expressed in a combination of modeling notations whose models are mapped into a single underlying reference model in the model repository with minimal redundancy of information

that must be maintained manually. Figure 5 illustrates the relationships between such models.

To support the development process, the repository also includes requirements, test data, and analysis results. Standardized interchange formats such as the Requirements Interchange Format (RIF) and AP233 for system engineering artifacts are emerging. Therefore, SAVI will define the data structures needed in the model repository for information storage and analysis, and data transformations needed for data interchange and to leverage ongoing efforts in standard organizations.

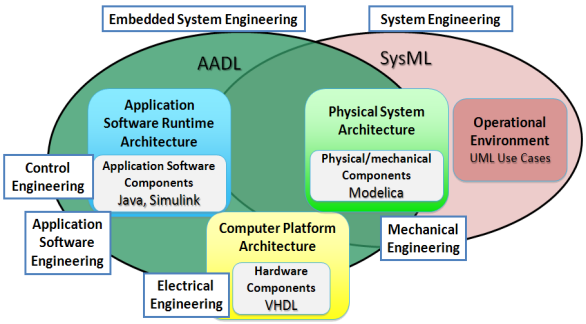

Figure 5: Collaborative Engineering

Standardized interchange representations for these analytical models facilitate tools chains (integration of multiple tools) to enable new and more effective integration checks and analyses by leveraging existing tools and minimizing transformations into tool-specific and proprietary representations. Examples of such emerging interchange formats include ISO/IEC 15909 for Petri nets to interface with Petri net analysis tools [1], and FIACRE for state-based behavior specification to interface with different model checking tools [2].

SAVI virtual integration activities augment traditional design reviews by a model-based approach. Subsystem requirements are recorded in an initial system model during request for proposals and made available to suppliers. Suppliers respond to the proposals by including subsystem models with interface specifications that include non-functional properties. This allows the system integrator

- to validate supplier model compatibility, interface consistency, and initial resource allocations during proposal evaluation,
- to assure compatibility of functionality with interface specifications including non-functional properties during preliminary design integration, and
- to verify system-level non-functional properties such as performance or safety during critical design integration.

Early and continuous virtual model integration based on standardized representations insures that

- errors are detected as early as possible with minimal leakage to later phases,
- models with well-defined semantics facilitate auto-analysis and generation to identify and eliminate inconsistencies,
- automated compatibility analyses at the architecture level scale easily,
- industrial investment in tools is leveraged through well-defined interchange formats.

3. Proof-of-Concept Phase

To establish cost-effective management and limit risks of the SAVI program, the Proof-Of-Concept (POC) project has been executed as the first of multiple phases with the following goals:

- Document the main differences between a conventional acquisition process and the projected SAVI acquisition process and identify potential benefits of the SAVI acquisition process.
- Evaluate the feasibility and scalability of the multi-aspect model repository and model bus concepts central to the SAVI project.
- Assess the cost, risk, and benefits of the SAVI approach through a return on investment (ROI) study and development of a SAVI development roadmap.

3.1 Proof of Concept Demonstration Requirements

The SAVI POC team established a prioritized set of requirements, which are summarized in Table 1. Note that emphasis was placed on validation early in the development process.

Table 1: Prioritized POC Requirements

#	Requirement	Category
1	Establish Model Bus infrastructure	Process
2	Establish Model Repository Infrastructure	Process
3	Inform return on investment (ROI) estimates through POC performance & results	Process
4	Analyses be conducted across the system	Analysis
5	Two or more analyses must be conducted	Analysis
6	Analyses be conducted at multiple levels of abstraction	Analysis
7	Analyses must validate system model consistency at multiple levels of abstraction	Analysis
8	Analyses must be conducted at the highest system level abstraction	Analysis
9	Model infrastructure must contain multiple model representations	Model
10	Model infrastructure must contain multiple communicating components	Model

3.2 The POC Aircraft Model

An aircraft system is modeled at three tiers:

Tier 1 the aircraft from a system engineering perspective

Tier 2 the aircraft IMA system as an embedded software system

Tier 3 and elements of the IMA at the subsystem/LRU level

A set of analyses is defined for each of those tiers, propagating and validating requirements and constraints across model levels and across multiple operational quality dimensions.

Architecture models were developed in AADL because it specifically supports modeling safety-critical, software-reliant systems and is extensible to support analysis of operational quality attributes along multiple dimensions [3].

Figure 6 shows a drawing of the aircraft system provided to the POC team. It shows major physical subsystems, some providing aircraft capability, such as navigation or landing gear, and others providing physical resources to the subsystems, such as the electrical power, hydraulics, and fuel.

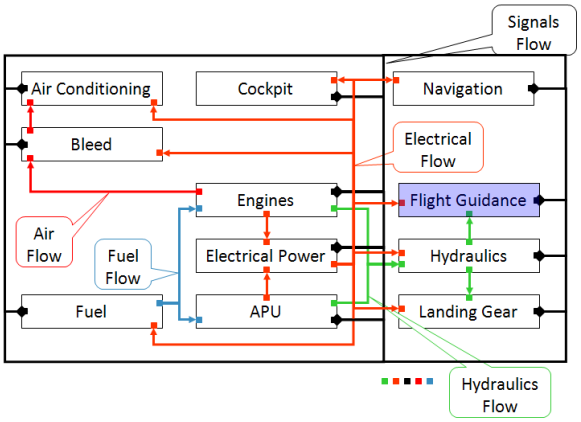

Figure 6: Aircraft System Drawing

Figure 7 shows a portion of the corresponding Tier 1 AADL model. In this model, we have represented the physical subsystems as AADL systems that can later be refined and the physical resources as AADL buses. Each aircraft subsystem is represented by a separate AADL system type, and each type of resource by a bus type. Bus access connections represent the physical connection between subsystems and their resources. The model elements include properties about physical characteristics (e.g., mass) of the subsystem. In addition, each bus type has a resource capacity property, and the bus access features (connection points) have resource supply or budget properties, e.g., the engine contributing electrical power and the cockpit drawing electrical power.

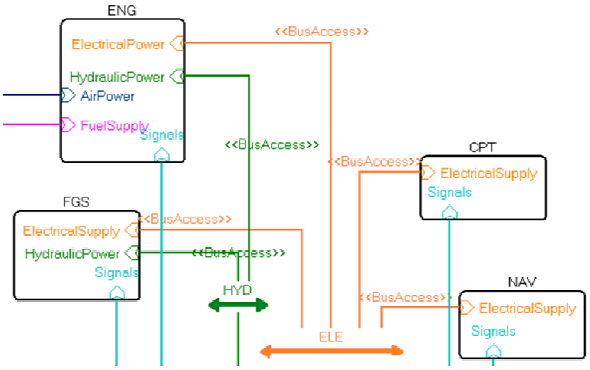

Figure 7: AADL Model of Aircraft System

We have elaborated the flight guidance system (FGS) into a Tier 2 model representing the distributed computer platform (physical view) and the embedded application subsystems (logical view) of the IMA subsystem. This elaboration is not a separate model, but a refinement of the FGS system component using the AADL *extends* mechanism. Because of this refinement, we can now specify a Tier 1 variant and a Tier 2 variant of the aircraft model and instantiate both for analysis from a single source.

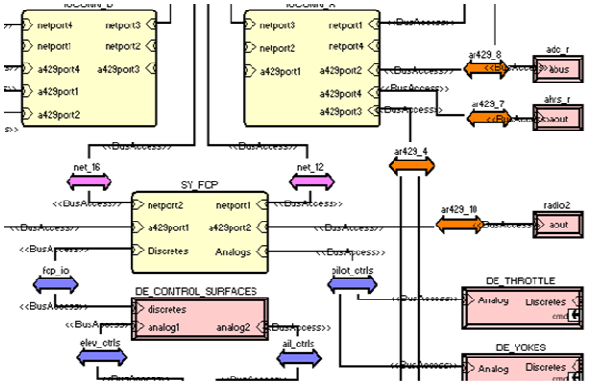

Figure 8: IMA Computer Platform

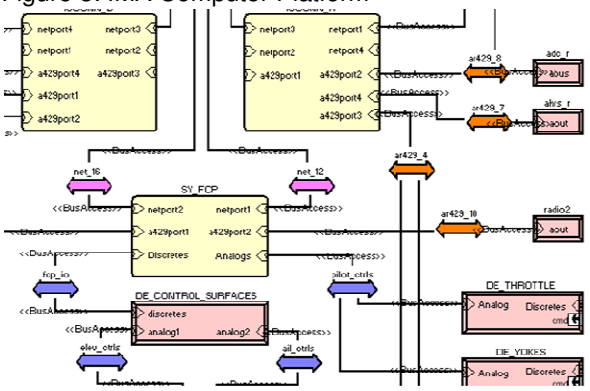

Figure 8 shows a portion of the physical view—that is, devices to represent sensors and actuators to the physical system, buses to represent networks such as ARINC429, and systems to represent processing units and communication units. The symmetry

reflects the dual redundant nature of the IMA platform.

Figure 9 shows a portion of the logical view as a collection of embedded application subsystems. We have used AADL port groups and connections to model interaction between subsystems. Port groups represent a collection of individual port connections, which suppliers later elaborate through port group types.

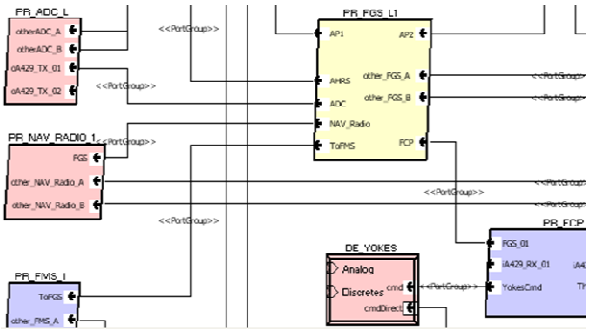

Figure 9: IMA Embedded Application Subsystems

We have also included two end-to-end flows in order to analyze the stick–to-surface response time when operating in direct mode (with a maximum allowed latency of 150 milliseconds) and with flight guidance and autopilot involved (with maximum allowed latency of 25 milliseconds).

IMA computer resources (MIPS for processors, memory size for RAM and ROM, and bandwidth for networks), in addition to weight and electrical power are specified as properties of processor and bus components. Similarly, computer resource budgets are assigned to the application subsystems and end-to-end latency requirements to the flows.

In our demonstration, seven of the IMA subsystems are contracted out to suppliers. The suppliers first refine the subsystem specification with interface details such as

- detailed communication properties,
- properties of communicated data, and
- mapping into protocols such as ARINC429.

Then they elaborate their subsystem into a Tier 3 model in terms of application tasks and communication between them. The tasks (as AADL threads) have periods, deadlines, and worst-case execution times. For sampled processing, the connections indicate whether mid-frame or phase-delayed communication is desired to minimize latency jitter.

At various times during this development process, the airframer virtually integrates the model and performs Tier 1, Tier 2, and Tier 3-level analysis. In that context, the airframer evolves the IMA model and aircraft model using AADL refinement mechanisms (*extends*) to specify configurations that include subsystems in their Tier 2 or Tier 3 elaboration and software to hardware deployment bindings.

4. Proof of Concept Scenarios

The requirements for the POC demonstration include several development use scenarios:

- Aircraft system modeling
- Modeling of the IMA system as embedded software system
- Subcontracting support between airframer and suppliers
- Subsystem development by supplier
- Virtual integration testing by airframer

4.1 Tier 1 Aircraft System Modeling Scenario

The first scenario illustrates analysis early in the life cycle based on the Tier 1 model, whose results are revalidated throughout the life cycle as the fidelity of the model increases.

Analysis Reporting

Since AADL is a strongly typed language, valid models guarantee a certain level of system architecture consistency without specialized analysis. For example, the landing gear system type has been specified to require access to the power system and the hydraulic system. The AADL semantic checker ensures that the correct bus is connected to the bus access feature and that all features that require a connection are connected.

Physical characteristics relevant to system engineers, such as mass and electrical power, are represented by AADL properties.

Information regarding the mass of a system is typically kept in a spreadsheet that must be manually updated and analyzed from time to time. Instead, we associate mass information with the AADL model and drive the analysis from the model. By doing this, we can analyze the mass of the aircraft for the Tier 1 model and then revisit the analysis with more details about the mass at the Tier 2 level. The analysis examines the net weight, gross weight, and weight limit of the physical system components and connections. In the case of the Tier 1 model, we can run an analysis that adds up the gross values of the Tier 1 elements and compares the total against the limit for the aircraft.

We have implemented the analysis as a model bus transformation that translates the relevant data into the CSV (comma separated values) file format for import into a spreadsheet.

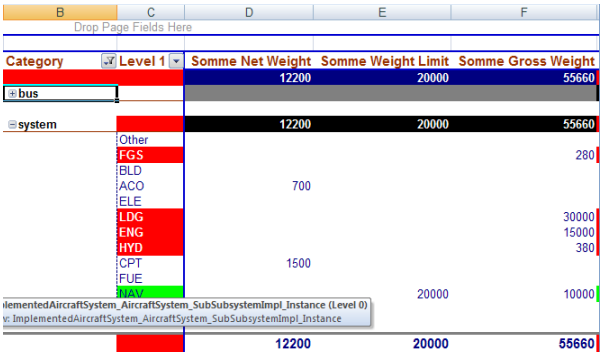

Figure 10: Spreadsheet Based Mass Analysis

The electrical power information is recorded in the AADL model as *PowerCapacity*, *PowerSupply*, and *PowerBudget* properties. The analysis, in this case implemented as an OSATE plug-in, compares the power supplied to the power system (an instance of bus type *PowerSystem*) from both the engine and the auxiliary power unit against the power system capacity as well as the power budgets of components drawing power from the power system. This analysis can be revisited when the Tier 2 model of the IMA is available to look at the power distribution of the IMA power subsystem to the computer hardware components and compare the demand against the power budget assigned in the Tier 1 model.

Use Scenario

In our use scenario, the analysis reports that

1. The power supplied by the engine and the auxiliary unit exceeds the capacity of the power system.
2. The power budgets also exceed the capacity but are less than the power supply.

As a remedy, we choose a higher capacity variant of the power system (from the component specification library, an AADL package) and rerun the electrical power analysis. We must also rerun the mass analysis from the same model to ensure that the change in the power system has not exceeded any weight limits.

The AADL model of the system drives various high-level quantitative system analyses. From the same model, we can perform analyses for hydraulic pressure, fuel flow, and airflow once the component specifications are annotated with the relevant properties. The models can be made more realistic by adding by refining the AADL component specifications or by associating detailed physical models using a specialized notation, e.g., Modelica, with a component.

4.2 Tier 2 Embedded Software System Modeling Scenario

The Tier 2 model refines the IMA part of the system into a networked computing platform and an interacting set of application subsystems and blackbox subsystems, which will get subcontracted to suppliers. However, before doing so, the airframer will analyze the Tier 2 model to revalidate the mass and electrical power results, by taking into account the more detailed architecture specification, and to validate properties specific to the elaborated IMA subsystem (i.e., computer resource usage and end-to-end flow response time). The analysis shows that the IMA power subsystem draws less power from the main power system and that the power consumption by the computer platform is at 60% of the locally available power. Therefore, we could consider reversing the earlier main power system upgrade.

Analysis Reporting

The computer resource analysis comes in two variants:

1. Budget totals against capacity totals
2. Budgets of deployed application components against the target resource once deployment decisions have been recorded

This computer resource analysis was demonstrated with an OSATE plug-in and can also be supported through a spreadsheet interface similar to the mass analysis.

Use Scenario

In our use scenario, the analysis reports that the MIPS budget totals exceed the total capacity of all processors. It also indicates that only a subset of the application components has a MIPS budget. The memory budget totals reflect 70% of the components with budgets and are well within memory capacity.

In our use scenario, we reduce the budgets and expect suppliers to meet them with their Tier 3 models. Such a decision can be justified based on historical data, if available.

At this point in the life cycle, or at a later stage, the system architect may make a first attempt at an allocation of major application subsystems to hardware. A variant of the computer resource analysis will consider the deployment in its results.

Sampling jitter and changes in latency due to implementation decisions regarding the runtime architecture can affect the stability of control systems [4]. The end-to-end latency analysis [5] at the Tier 2 level takes into account

- processing latency in the stick and surface (represented by a latency property on the flow specification in the respective AADL device type)
- communication latency associated with the connections involved in the flow

- processing and sampling latency of IMA subsystems involved in one of the two end-to-end flows

The latency analysis [6] calculates the minimum worst-case latency for the two flows (a lower bound that can only increase as the model is refined) and reports that in direct mode the response time requirement is met (121ms versus a 150ms requirement), while the IMA-based response time is almost twice the original requirement (46 ms versus the required 25 ms). The main contributor to the increased latency value is the sampling latency of the partition. We could reduce the latency by not sampling (moving to a data-driven architecture), double the partition execution rate (doubling our processor resource requirements), or renegotiate the response time requirement as an inherent property of the chosen runtime architecture. In our scenario, we pursue the latter option.

4.3 Airframer-Supplier Subcontracting Scenario

In support of subcontracting, we have organized the AADL model into a number of separate AADL packages that are version controlled through a model repository. For the demonstration, the AVSI organization hosted this model repository on a Subversion server with POC demo team members playing the airframer and the supplier roles located at two sites in the U. S. and two sites in Europe.

Repository Organization

The repository has different access-controllable public and internal areas for the airframer and the subcontractors, as shown in Figure 11. The standardized AADL XMI representation [7] enables inter-repository model interchange.

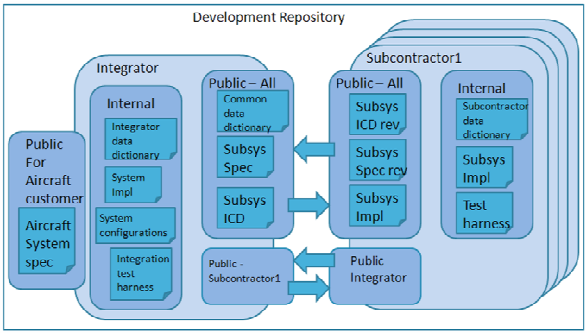

Figure 11: Distributed Model Repository

Use Scenario

In the scenario, as part of a request for proposals (RFP), the airframer makes available the AADL specification of the desired subsystems with a possibly partial interface specification (system types) including resource budget properties as well as expected latency requirement on flow specifications through the subsystem, and an interface control document in the form of port group type specifications. We have done so for seven subsystems to be contracted out.

The suppliers respond, in this scenario, with a completed subsystem specification including details about the exchanged data and its mapping into the ARINC429 protocol. The airframer verifies overall consistency by virtually integrating the suppliers' AADL subsystem specifications as a variant of the Tier 2 model and performing functional integrity checks. Figure 12 shows reported inconsistencies that are traceable to the model.

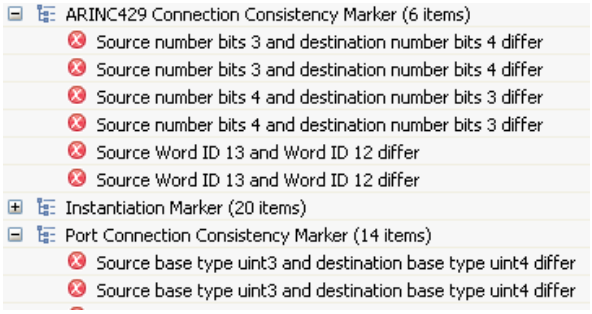

Figure 12: Functional Integrity Checking

4.4 Tier 3 Supplier Subsystem Development Scenario

The suppliers refine their subsystem AADL models to model their architecture and reflect implementation decisions. This example included three suppliers expanding their subsystems. For the air data computer, we have included UML diagrams, Ada code, a test harness and automatic build scripts, as shown in Figure 13.

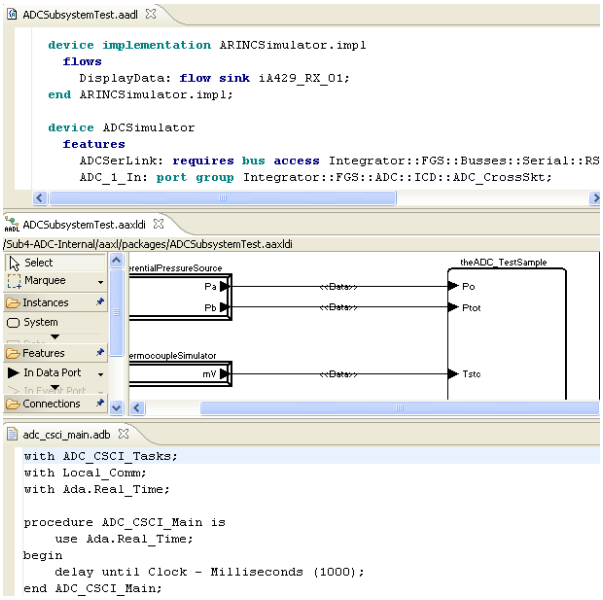

Figure 13: Supplier Subsystem Model & Analysis

The supplier allocates threads of the application task model to computer hardware and performs

scheduling analysis. The scheduling protocol property of the processor determines which scheduling analysis algorithm is used. Analysis results indicate that the subsystem is schedulable on its internal hardware with 45% utilization.

Scheduling analysis uses the worst-case execution time of threads. This figure may be an estimate early in the development that is scaled for processors of different speed. Once the source code exists and has been benchmarked on different processors the benchmark figures replace the estimates in the model resulting in higher fidelity results.

4.5 Virtual System Integration Testing Scenario

At various stages of the development, each supplier delivers an AADL model of their subsystem architecture to the airframer. These updated models identify properties of the subsystems pertinent to integration, but do not necessarily include detailed design descriptions. The airframer virtually integrates them into an aircraft model refined down to Tier 3, while the supplier independently validates that the model properties of the architectural components reflect the detailed design.

Analysis Reporting

The models include properties which can be used for traceability to requirements. The airframer queries the model repository to determine which subcontracted subsystems are involved in satisfying certain requirements. This allows the airframer to focus on virtually integrating and analyzing the system with respect to requirements of greatest concern.

Use Scenario

The airframer revisits the mass and power analysis to include the Tier 3 details in the results. Similarly, computer resource analysis aggregates periods and execution times of the subsystem task models, compares them against the assigned budgets and rolls them up for a comparison against capacity. The analysis results show that many subsystems stay within the reduced budgets from the Tier 2 analysis.

As part of the scenario, the airframer revisits the end-to-end latency analysis, now taking into account any latency or latency jitter contributed by the subsystem task models that exceed the expected latency as recorded in the flow specification of the subsystems in the Tier 2 model. When task models are first delivered by the supplier, the airframer reruns the end-to-end latency analysis, which now takes the task model into account in its latency calculation. The airframer discovers that the minimum worst-case end-to-end latency for the IMA mode has increased considerably to 185ms on a synchronous hardware platform and to 196ms for processors operating on independent clocks. Examination of the detailed analysis report reveals a low-rate thread in one of the subsystems contributing a sampling latency of 100ms. Without virtual integration, such a problem would only be discovered during system integration test.

The airframer allocates the application tasks from the different supplier task models to the various processors and performs scheduling analysis. For a given deployment configuration of a three-processor system, the analysis reports that all deadlines are met with processor utilizations of 54%, 55%, and 75%. The airframer can validate the analysis results by applying a different scheduling analysis tool. In both cases, a model transformation is performed to generate a timing model in the representation acceptable to the analysis tools from the AADL model. A resource allocation tool [8] provides an option to explore alternative deployment configurations, showing that the system would be schedulable with 97% utilization on two processors, and suggesting a three processor allocation that better balances the task load, and supporting a quick what-if analysis of a four processor system to reduce the average processor utilization to a target of 50%.

Finally, the airframer performs network bandwidth analysis on the aircraft model with Tier 3 detail and a specific deployment configuration. This analysis identifies all application task connections that are routed over a particular network and determines the data volume from the size of the data communicated through ports and their transfer rate. This data volume is then compared against the bandwidth budget assigned at the Tier 2 level and against the capacity of each the network.

5. Return on Investment Analysis

In this section we summarize the result of an ROI analysis [9] that focuses on cost avoidance due to early detection of design defects through virtual integration. Rework cost is dominantly driven by the cost of managing defects injected in the requirements and design phases but which are detected late in the system development, most often in the integration and test phases, where the cost to repair the defect is one to two orders of magnitude higher. Virtual integration reduces and prevents this down-stream flow.

Current development processes allow 70% of faults to be introduced early in the life cycle, while 80% of them are not caught until integration test or later with a repair cost of 16x or higher. Figure 14 shows percentages for fault introduction, discovery, and cost factor of repair [10, 11, 12]. If we can use the SAVI approach of architecture-centric virtual integration and analysis to discover a reasonable percentage of system-level faults earlier in the process, we can expect cost savings larger than the additional investment in modeling and analysis.

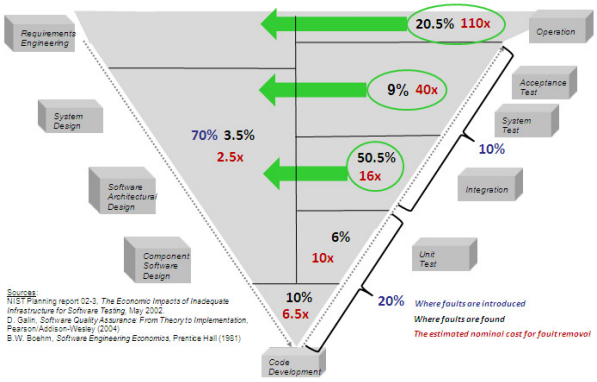

Figure 14: Benefits of Early Fault Discovery

We estimate the size and complexity of an avionics software system based on data from previously built aircraft. We use two scenarios that reflect the current aircraft software (27 and 30 MSLOC), and a synthetic software system (60 MSLOC), which would not be affordable using current methods. Cost was estimated using the COCOMO II model [13].

Based on these "as-is" costs, we compute cost reduction for the "to-be" scenario, i.e., after introduction of the SAVI methodology. To estimate avoided rework cost, we calculate an estimate for the cost to manage defects in the various phases. This is based on published empirical data from case studies and on observations providing data on the number of defects that would be introduced in a system of a defined size and complexity, and the nominal cost of removing defects. Table 2 shows the resulting cost to remove a defect of a given type relative to the total cost of defect removal. For example, requirements defects account for 79% of rework cost and 62% of rework cost occur during integration.

Table 2: Relative Defect Removal Cost

	Phase in which defect is removed					
Defect Type	**Require-ments**	**Design**	**Code**	**Test**	**Integration**	***Sum***
Require-ments	0.03%	0.21%	1.87%	28.11%	48.73%	*78.97%*
Design		0.04%	0.37%	5.79%	9.75%	*15.96%*
Code			0.19%	1.28%	3.10%	*4.57%*
Test				0.17%	0.26%	*0.43%*
Integra-tion					0.09%	*0.09%*
Sum	*0.03%*	*0.26%*	*2.44%*	*35.36%*	*61.92%*	*100.00%*

Increasing defect detection and removal efficiency would positively influence rework cost. The cost avoided is computed as

$$CostAvoidance = DevelopmentCost * \%Rework * \%RequirementsErrorsRework * \%RemovalEfficiency$$

Two scenarios of rework cost were considered: 30% and 50% of the total development cost. Similarly, we considered two scenarios for defect removal efficiency due to SAVI: 33% and 66% as applied to requirement defects. This is based on using Hayes' fault taxonomy [42] and estimating removal efficiency for the per fault category.

Finally, ROI and NPV were calculated in light of an estimated cost of $86M to implement SAVI.

In summary, the ROI analysis showed the following: The nominal cost reduction for a 27MSLOC system is $2,391M (out of an estimated $9,176M) occurring at 70% reuse, 50% rework, and with a removal efficiency of 66%. The arithmetic and logarithmic ROIs are 26.8 and 3.33, with an NPV of $1,076M.

Studying the most conservative of the scenarios, the smallest cost avoidance is $717M, occurring at 70% reuse, 30% rework cost, with a removal efficiency of 33% for a 27MSLOC system. The arithmetic and logarithmic ROIs are 7.3 and 2.12, with an NPV of $263M.

6. Conclusion

In this paper, we have focused on reporting the experiences derived from AVSI's SAVI project, utilizing an architectural reference model to achieve virtual integration. Virtual integration demonstrated early and repeated quantitative analysis at various levels of fidelity, validation of architecture consistency across subcontracted subsystem interfaces and independent protocol mappings, and discovery of intricate operational system-level faults due to design problems in the runtime architecture. The resulting early discovery of system-level faults reduces risk, lowers system life cycle costs, and improves quality.

The SAVI POC demonstrated how multi-tier modeling and analysis across levels, coverage of system engineering and embedded software system analysis, propagation of changes across multiple analysis dimensions, maintenance of multiple model representations in a model repository, auto-generation of analytical models via model bus, interfacing of multiple tools to perform the same analysis, as well as distributed team development via repository to support airframe manufacturer and supplier interaction are possible through the SAVI concepts.

In addition, ROI analysis has shown that SAVI can lead to significant cost savings by avoiding defects in early development phases (requirements and design), by reducing corresponding costly rework in the later phases.

Several areas were not fully explored and will be addressed by SAVI phase 2:

- support for multiple architecture modeling notations (SAE AADL & OMG SysML), and integration with mechanical system modeling (e.g., Modelica) and control system modeling (Simulink),
- validation focus on a system-level non-functional property such as safety, reliability, or security,
- end-to-end validation of systems from requirements to models and system implementation,
- scalability of the model repository and commercial tool support.

7. Acknowledgements

System Architecture Virtual Integration (SAVI) is an industry initiative by a number of aerospace companies and government organizations to improve the engineering practice for software-reliant aircraft systems under the umbrella of the Aerospace Vehicle Systems Institute (AVSI). The proof-of-concept phase of this initiative was carried out by representatives of the following member companies and organizations: AVSI, Boeing, Airbus, Lockheed Martin, Rockwell Collins, BAE Systems, GE Aviation, U.S. Army, and Federal Aviation Administration (FAA). The Software Engineering Institute (SEI) contributions were funded by SAVI members through AVSI. In particular the authors would like to acknowledge the contributions by the other members of the proof-of-concept (POC) demonstration project, which is the focus of this case study report, to the definition of a to-be process and the development of a return on investment (ROI) model. Keith Appleby (BAE Systems), John Glenski (Rockwell Collins), Jean-Jacques Toumazet (Airbus), Joe Shultz (GE Research), and Dave Redman (AVSI) actively participated in creating the aircraft model for the demonstration and the Microsoft Excel spreadsheet version of the mass analysis. We would also like to thank the other members of the full SAVI team who participated in shaping the requirements for the POC demonstration and use cases.

8. References

[1] ISO, “Software and System Engineering – High-level Petri nets – Part 2: Transfer Format“, ISO/IEC 15909-2:2009.

[2] B. Berthomieu, et al., “Fiacre: an Intermediate Language for Model Verification in the TOPCASED Environment”, Proceedings of 4th International Congress on Embedded Real-Time Systems (ERTS 2008).

[3] SAE International. “Architecture Analysis & Design Language (AADL): SAE International Standards document AS5506A, Nov 2004, Revised Jan 2009.”

[4] A. Cervin, K.-E. Årzén, and D. Henriksson, “Control Loop Timing Analysis Using TrueTime and Jitterbug”, Proceedings of the 2006 IEEE Conference on Computer Aided Control Systems Design (CACSD), pp. 1194–1199.

[5] P. Feiler and J. Hansson, “Impact of Runtime Architectures on Control System Stability”, Proceedings of 4th International Congress on Embedded Real-Time Systems (ERTS 2008).

[6] P. Feiler and J. Hansson, “Flow Latency Analysis with the Architecture Analysis & Design Language (AADL)”, Software Engineering Institute Technical Note, CMU/SEI-2007-TN-010, Dec 2007.

[7] SAE International, “Architecture Analysis & Design Language (AADL) Annex Volume 1: Annex C: AADL Meta model & XML Interchange Format Annex, SAE International Standards: AS5506/1”

[8] Dio Deniz, Peter Feiler, “On Resource Allocation in Architectural Models”, Proceedings of the 11th IEEE International Symposium on Object/service-oriented Real-time distributed Computing, May 2008.

[9] Jörgen Hansson, Steve Helton, “ROI Analysis of the System Architecture Virtual Integration Initiative”, Software Engineering Institute, Technical Report, SEI-2010-TR-001, 2010.

[10] RTI, “The Economic Impacts of Inadequate Infrastructure for Software Testing”, National Institute for Standards and Technology, Washington, DC, NIST Planning report 02-3, 2002.

[11] D. Galin, “Software Quality Assurance: From Theory to Implementation”, Boston: Pearson/Addison-Wesley, 2004.

[12] B.W. Boehm, “Software Engineering Economics”, Englewood Cliffs, NJ: Prentice Hall, 1981.

[13] “COCOMO II”, available from: http://sunset.usc.edu/csse/research/COCOMOII/cocomo_main.html [Accessed: March 20, 2010].

[14] J .H. Hayes, “Building a Requirement Fault Taxonomy: Experiences from a NASA Verification and Validation Research Project”, IEEE International Symposium on Software Reliability Engineering (ISSRE), Denver, CO, Nov 2003.

9. Glossary

AADL: Architecture Analysis & Design Language

ARINC: Aeronautical Radio Inc.

AVSI: Aerospace Vehicle Systems Institute

COCOMO: Constructive Cost Model

IMA: Integrated Modular Avionics

SAE: Society of Automotive Engineers

SAVI: System Architecture Virtual Integration

POC: Proof of Concept

ROI: Return on Investment