Putting Teeth into Open Architectures: Infrastructure for Reducing the Need for Retesting

Valdis Berzins
Professor – Department of Computer Science (NPS)
E-mail: berzins@nps.edu, Phone: 831-656-2610

Context

- The Navy is moving towards an Open Architecture (OA) paradigm
  - Joint interoperable systems that adapt and are built using open interfaces, open design principles, and open architectures
- FORCEnet – the Navy's network centric concept of operations
  - The viability, affordability and sustenance of FORCEnet necessitates an architecture that is fully compliance with OA technology
  - The development of OA within FORCEnet will result in a superior, adaptive, "plug and fight" capability for the modern war-fighter
- Expected long term benefits from Navy OA
  - Business benefits:
    - Flexible acquisition strategies and contracts that enable the Navy to reuse software, easily upgrade systems, and share data throughout the enterprise
  - Technical benefits:
    - Layered and modular open architectures that facilitate portability, maintainability, interoperability, upgrade-ability and long-term supportability

Problem Statement

- Our preliminary investigations indicate that current methods for achieving dependability in Open Architectures are insufficient
  - Navy is currently able to deliver open architecture-based systems
    - However, known methods for achieving dependability with OA are expensive and not clearly understood
  - According to Navy and other experience, traditional approaches to testing are usually unsuitable in open environments
    - They are too expensive, take too long and lack agility to react to changes during acquisition
    - Have to be repeated after every change
- Typical testing assumptions are not valid for Open Architectures
  - Conventional methods for testing require that the environment of a typical system is fixed and known in detail to the quality assurance team at test and evaluation time
- Conventional testing is strongly context dependent
  - The effectiveness of testing is very sensitive to the expected operating environment, which is unknown for reusable subsystems
  - The majority of failures in software systems are due to requirements and specification errors, and commonly show up after a subsystem has been moved to a different environment

Objectives

- Reduce testing cost
  - Reduce the need for re-testing
  - Eventually eliminate integration test after every reconfiguration

- Make testing more effective by augmenting it with other quality assurance methods
  - Develop conceptually new and different testing methods to achieve dependability in Navy OA systems in presence of reuse, reconfiguration, changes and unpredictable environments

- Enable Persistent Open Architectures
  - The architecture should not have to change or be retested every time the system configuration changes
  - All architecture changes should be compatible extensions
    - Avoid retesting previously existing parts

Challenges for DoD Testing Approaches

- Navy systems are subject to frequent changes
  - E.g., Many Navy systems seek to provide **migrating services** and **reconfiguration of service oriented architectures** (SOA)
  - Architectural changes impact Key Performance Parameters (KPP), availability and other system requirements

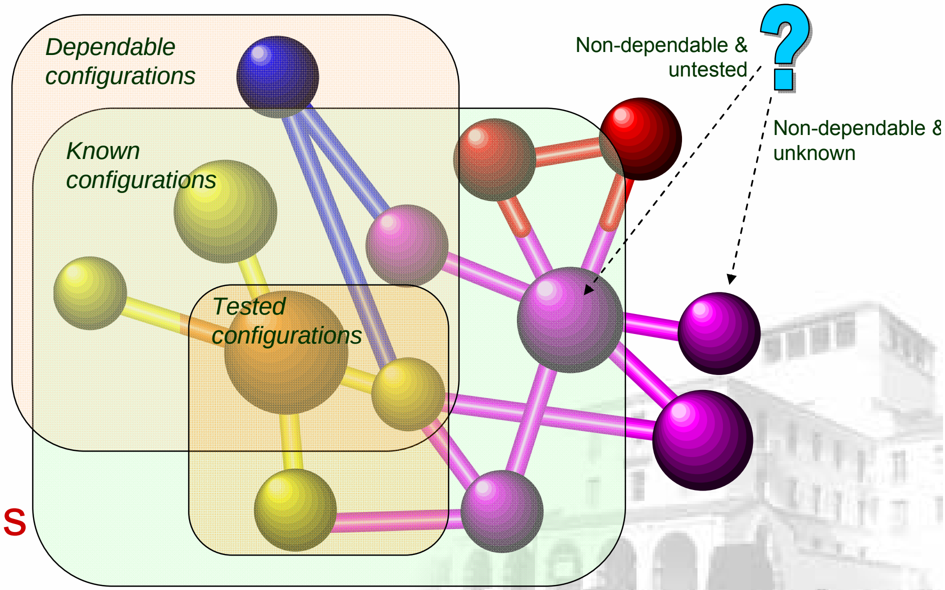
- Scenario-based testing is commonly used
  - Dependent on a particular system configuration and environment
  - Does not currently deal with system modularity
  - When the system configuration or environment changes, the designed test cases, scenarios and operational profiles also need to be changed.

- A shift from scenario based testing to architecture based testing is needed

Complexity of testing OA

- An architecture is related to a family of systems, while a design is traditionally associated with a single instance of a system
- Assembly of plug compatible components leads to many system configurations
  - Slots in an open architecture can be filled by different subsystems
    - The number of choices for each slot multiplied together lead to an astronomical number of possible configurations for Navy systems
  - Can include new components that did not exist when the architecture was designed

- Unbounded number of configurations
  - An unpredictable number of new subsystems can be created in the future
  - It will be impossible to test all configurations
  - A majority of the configurations will not be tested at all



Solution Approach

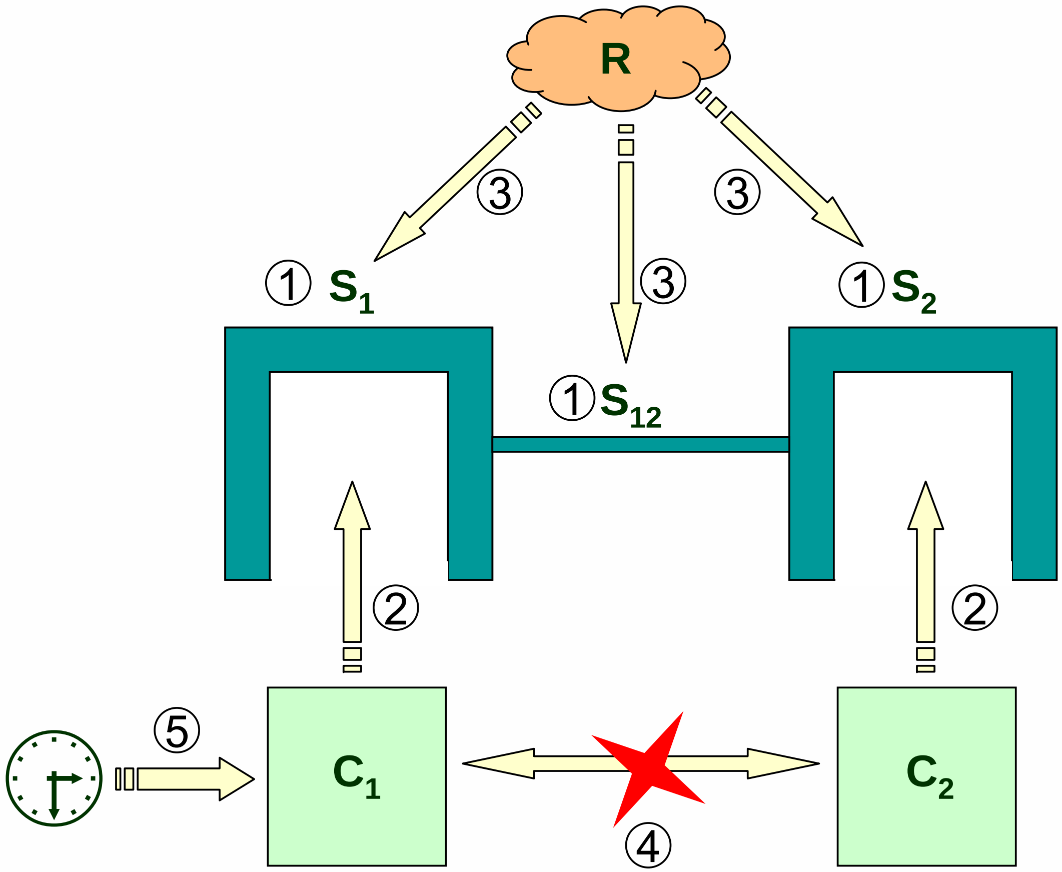
- Refine the open architecture concept to support system development and testing with interchangeable software parts

- A Dependable Open Architecture should include:
  - Not only components and connections but also constraints expressing the most important dependability properties
  - Links to requirements, capabilities and standards
  - Variable parameters – KPP’s / features
  - Components and connectors should be swappable within compatibility groups defined by testable dependability properties

- Apply testing at the architectural level, not only at the system implementation level

Solution Approach

- The proposed method is globally decomposed into four major steps:



1. Formulate dependability contracts
2. Test Components vs. Standards
3. Verify Architecture vs. Requirements & Standards
4. Ensure non-interference among components
5. Monitor environment assumptions

R Requirements

S_1 Standard for Component 1

S_2 Standard for Component 2

S_{12} Standard for connection between components 1 and 2

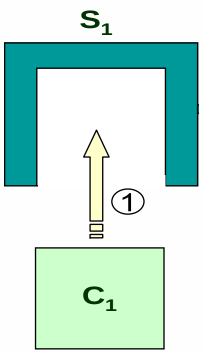
C_1 Component 1

C_2 Component 2

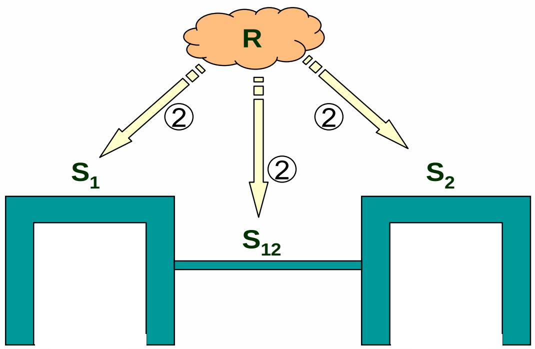
Solution Approach

- **Step 1**: Identification of dependability contracts
  - System wide guarantees and assumptions
    - Dependability properties that must hold in all configurations at the system level
    - Primarily technical constraints rather than legal documents
      - Intended to be checkable/testable via software, also at reconfiguration or runtime
    - Improved methods for requirements determination, analysis, representation and allocation might be required
  - Component requirements
    - Component-level dependability contracts for the subsystems and connectors of the architecture
    - Constraints apply to the architectural connection patterns and subsystem slots

- **Step 2**: Testing components vs. standards
  - Test each subsystem and connector against its dependability contract
  - Automated process to enable sufficient large sets of test cases for statistically significant conclusions about desirable dependability levels
  - Cost is proportional to the number of components, not number of combinations
  - Must be done once for each version of each atomic component
  - Well-known methods and techniques available

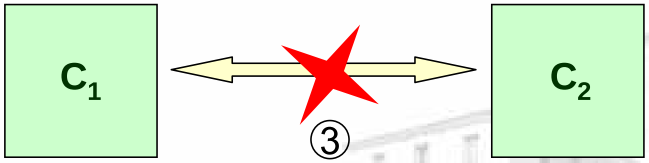


Solution Approach



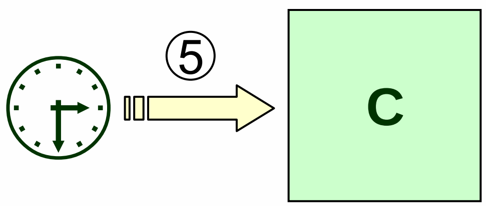
- **Step 3**: Verify architecture vs. requirements and standards
  - Check the system-wide dependability properties in all possible configurations vs. the structure of the architecture and the dependability contracts for subsystems and connectors
  - One-time process that uses symbolic analysis techniques

- **Step 4**: Ensure non-interference among components
  - Check components for non-interference
    - Ensure components working correctly in isolation will continue to do so when they are connected
  - Computer-aided process
  - Some known methods and techniques



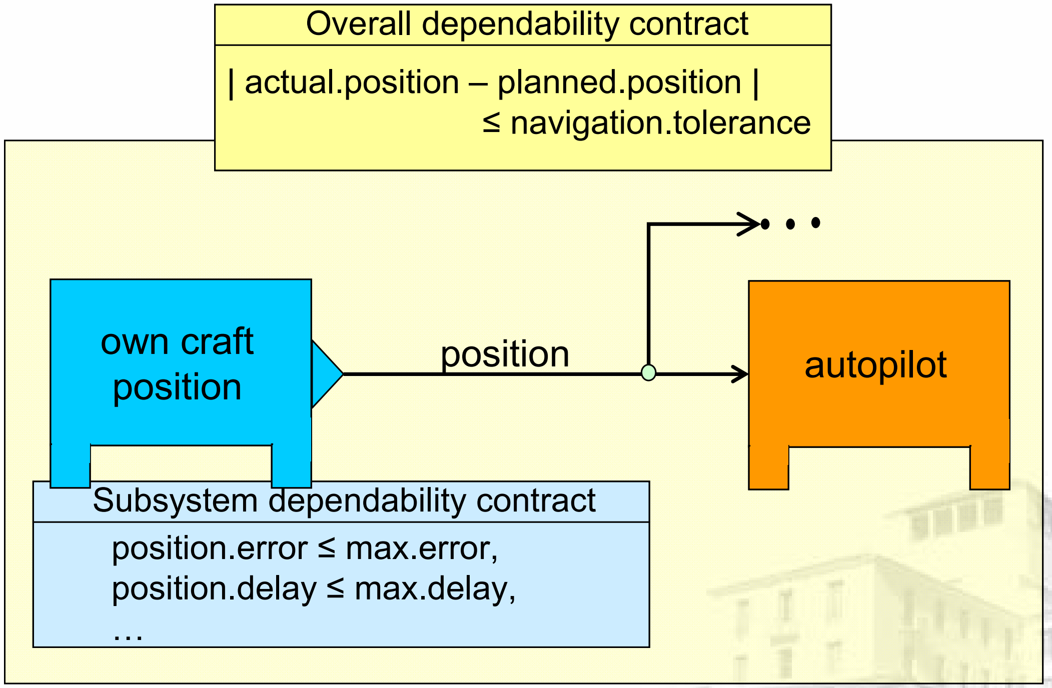
Solution Approach

- **Step 5**: Monitor Environment Assumptions
  - Formulate assumptions about the environment as constraints attached to the architecture and components
  - Check constraints after reconfiguration, e.g., resource limits, schedulability, etc.
  - Operating environment assumptions checked by runtime monitoring, e.g., Built-In-Test(BIT) technology used in DoD systems
    - E.g., Patriot Missile was not supposed to operate for more than 8 hours continuously



Example: craft position control subsystem

- Architecture
  - Two component slots
    - Software driver for a position sensor (can be filled with a variety of sensors, such as GPS, inertial, VOR/DME, etc.,)
    - Control software module for an autopilot (can be filled with different control algorithms)
  - One connector
    - Carries information about the current craft position
- Objective
  - Keep the platform on course
- Dependability contracts
  - Tolerances for the sensor accuracy and the allowable time delay for transmitting the position
  - To be fulfilled by any acceptable subsystem configuration



Acquisition Process Implications

- Requirements analysis needs to span the entire problem domain and system life, not just individual versions of the System of Systems
  – Same architecture must support all future versions
  – Planned control of variation via ranges for parameters/features
- Re-orient development processes toward Design-to-Tolerances
  – Currently oriented towards Design-to-Fit, Test-to-Fit
- The architecture as a whole needs authority / priority
  – Responsible organization
  – Global system standards authority
  – Manage accountability for subsystems
  – Empower via change control, acceptance testing, budget control

Acquisition Process Implications

- Architecture development / QA needs substantial time/resources/technology development
  - Must be included in plan from the start
  - More detailed/precise standards and analysis needed
- New QA technologies needed
  - Some known in labs but not used currently
  - Tailoring/improvement may be needed for practical use
  - Some areas need new methods to reach long term goals
  - Will need tech transfer and training

Conclusions

- New approach to quality assurance is better for achieving Dependable Open Architecture
  - Support rapid reconfiguration without compromising dependability while remaining economically viable
  - Applies to Test & Evaluation in Navy Open Architecture initiative

- Benefits of the proposed methodology:
  - Reduction of testing and limited scope for retesting after changes
  - Assurance of dependability
    - Assurance that all possible configurations derived from the architecture can satisfy the stated dependability requirements
    - Enables agile dependable reconfiguration and on-the-fly "plug and fight"

- Overall, the proposed methodology will enable achieving dependability in Navy OA systems in presence of reuse, reconfiguration, changes and unpredictable environments

Backup Slides

Related Work

- As far as we know, there is no similar approach proposed in the related literature
- Comparison with Navy's testing approaches
  - Guidelines for testing are scarce and generic, and mainly rely on scenario-based approaches
  - E.g., testing recommendations in OACE
    - Functional and performance testing vs. specified system requirements organized as test cases and scenarios
    - Concept of "virtual homogeneity" to facilitate testing by identifying compatibility groups of sub-systems performing similarly (We define these via dependability constraints and slot standards)
    - Concepts of "tree of subsystems" and "aggregations of components" with no (considerable) interaction between choices of configurations for applying test cases
    - Schedulability analysis for ensuring that any configuration is schedulable (A kind of non-interference check)

Related Work

- Comparison with component based testing
  - Can be used in our methodology for testing components vs. standards
    - Traditionally performed by a component's developer before release to assure quality (white-box testing approach)
    - Also used by system integrators to check that a component works correctly in a host system (black-box testing approach)
  - Certification strategies based on component testing
    - Combination of black-box testing, system-level fault injection and defense protection through wrapping (Voas)
  - Approaches to make component data visible for testing
    - Components are usually acquired as black-boxes without access to data necessary for (integration) testing
    - Reflective techniques can help access the required data (Salles)
  - Techniques based on formal methods
    - Model checking and theorem proving are traditional formal techniques used to test and verify components' correctness vs. specifications

Related Work

- Comparison with runtime software reconfiguration
  - Used in service-oriented architectures (SOA), air-traffic control systems, telephone switching systems, high-availability public information systems, etc.
  - Variety of technology for Dynamic Software Architectures
    - Reconfigurable ADLs (e.g., Dynamic Wright), programming languages (e.g., Lisp, Smalltalk, Haskel), dynamic linking libraries, dynamic object technology (e.g., CORBA), etc.
  - Techniques for developing reconfigurable systems
    - Graph transformation methods, hypergraphs, grammar oriented programming (GOP), grammar oriented object design (GOOD), etc.
  - Techniques for checking reconfigurable systems
    - Usually applied to static configurations (model checking, conformance testing, etc.)
    - Runtime monitoring techniques also used
  - Several steps of our approach can benefit from these techniques
    - E.g.: derivation of dependability contracts for reconfiguration, topology and connections; verification of the structure of the architecture, identification of sources of interference, etc.