Quantifying Resilience in Component-Based Software Architecture Models

Daniel Balasubramanian
Abhishek Dubey, Nag Mahadevan, Tihamer Levendovszky, Will Otte, Gabor Karsai

Supported by USAF/AFRL under Cooperative Agreement # FA8750-13-2-0050
Overview

• Focus: how can component-based, software-intensive systems be made more resilient?
  – *Through model-based techniques for design-time architecture specification and analysis accompanied by corresponding run-time support.*
What is Resilience?

• Webster:
  – *Capable of withstanding shock without permanent deformation or rupture*
  – *Tending to recover from or adjust easily to misfortune or change*

• Technical:
  – *The persistence of the avoidance of failures that are unacceptably frequent or severe, when facing changes.* [Laprie, ‘04]
  – *A resilient system is trusted and effective out of the box in a wide range of contexts, and easily adapted to many others through reconfiguration or replacement.* [R. Neches, OSD]

• Intuitive:
  – The ability to “bounce-back” after something changes

• We focus on model-based development for Resilient Software Systems
  – Design-time techniques + run-time support = resilience
Context for Resilience

- Resilient Software Systems are needed in many domains: desktop applications, web-based systems, collaborative systems, service-oriented systems, etc.
- Focus area: DREMS
  - Distributed: applications are executed on a distributed platform with dynamically changing topology
  - Real-time: applications have to satisfy real-time requirements
  - Embedded: applications may interact with the physical world
  - Managed: applications are managed external by an authority
  - Systems
- Examples:
  - On-board software for vehicles with networked processors
  - Swarm of UAVs executing wide-area surveillance missions
  - Distributed C2 systems with real-time requirements
  - Fractionated spacecraft (with wireless links) that provides a ‘platform as a service’
Modeling Overview

- Development is centered around *models*

**Design-time**

1. Resilience algorithm calculates metrics for specific architecture.

2. Generate corresponding run-time pieces.

**Run-time**

3. Middleware provides run-time support.

- Domain-Specific Model
  - Resilience calculator
  - Resilience metrics

- Resilience Manager
  - App Component
  - App-specific code
  - Middleware
  - Operating System
  - Hardware

Software generator
Domain-Specific Modeling Language

• A DSML is used to define the resilient software architecture
• Why a DSML?
  – Existing modeling languages do not cover the entire development process and are not integrated with comprehensive generators
  – Lack of support for resilience in existing modeling languages (SysML, AADL)
• What does a model enable?
  – Specification of software architecture
  – Code generation (app code, glue code, deployment scripts)
  – System integration (integrating multiple applications, deployments)
  – Analysis (resilience, scheduling)
• What does a model contain?
  – Software (communicating components)
  – Hardware (the physical nodes, resources)
  – Deployment of software onto hardware
  – Resilience description
2 Resilience Scenarios

Consider a system with three nodes running an image processing application. The application has 3 components. Comp3 requires a specialized camera piece of hardware.
Consider a system with three nodes running an image processing application. The application has 3 components. Comp3 requires a specialized camera piece of hardware.

Scenario 1: the camera on node 1 fails. 
-> **Redeploy** Comp3 on Node 2
Consider a system with three nodes running an image processing application. The application has 3 components. Comp3 requires a specialized camera piece of hardware.

Scenario 2: Comp1 on Node 1 fails.
-> Use Comp4 on Node 3 (it provides the same functionality)
Capturing Resilience

• The examples above provide resilience in two ways:
  1. Redeploy a component onto another node
  2. Use an alternate implementation of the same functionality provided by another component

• Method 1 requires a way to specify resource requirements
• Method 2 requires a way to specify functionality
Models Specify...

• Hardware
  – Nodes (with security labels)
  – Physical resources (e.g., camera)
  – Computation resource limits (e.g., memory)
  – Network links

• Software
  – Components and their interfaces
  – Required resources and % amounts
  – Security labels

• Applications
  – How components are connected
  – The “critical” components
  – Deployment (including constraints)
Modeling resilience

• Model three things:
  – *Functionality* the system provides
  – *How* the system can provide that functionality
  – Deployment *constraints*

• Functionality:
  – Hierarchically decompose functionality into basic functions

• How functionality is provided:
  – Map functionality hierarchically to applications, component assemblies or components

• Deployment constraints:
  – Restrict how software is deployed onto computing nodes and networks (e.g., application requires a camera)
Resilience Example

- Consider an example with three satellite nodes
- **Satellite 1 contains:**
  - High-resolution (HR) camera
  - Low-res (LR) camera
  - GPU
  - Ground link
- **Satellite 2 contains:**
  - HR camera
  - GPU
  - Ground link
- **Satellite 3 contains**
  - LR camera
  - Ground link
- Each satellite has an instance of two different applications...
2 Applications

• Cluster Flight Application
  ▪ 4 components

Provides access to/from ground.

Provides access to sat bus hardware.

Runs control loop.
Commands thruster through bus interface.

Updates orbit based on ground commands.
2 Applications

- **Wide-area imaging application**
  - Uses cameras on different nodes to create a combined image
  - Each satellite runs an image grabber component (HR or LR)
  - Only one instance of ImageProcessor runs at any time, but it can be redeployed

Combines and processes individual images.
Requires a GPU on node.

Captures images.
Requires either a HR or LR camera on node.
Functional Requirements

• Capture the functional breakdown required for the mission
  – Cluster flight
  – Wide area imaging
• All functions map to application/component instances
• Failure of one component/hardware resource/network link is used to compute whether the mission function is unavailable.
• Thereafter an alternative configuration (if available) can be chosen to recover the functionality.
Specifying Functionality I

• Define functionality (hierarchically)
  – Specifies what must be present on system
Specifying Functionality - II

• For each functionality, specify how it is provided

  The classification functionality is provided by the ImageProcessor Component. **Exactly** 1 instance should be running.

  ![Diagram of classification functionality]

  The image capture functionality is provided by the ImageGrabber component. **At least** 1 of these components must be running.

  ![Diagram of image capture functionality]
Deployment Constraints

- One instance of CFA runs on each node
- An application instance requires the orbit manager and satellite bus interface to be on the same node
Specifying Resource Requirements

• The ImageGrabber components need an LR or HR camera

• The ImageProcessor components need a GPU
Operational requirements

- All components/Application have operational requirements
  - **CFA Application**
    - SameNode(OrbitManager,SatelliteBus)
  - **OrbitManager**
    - SameNode(CallSatThrusterCtrl)
    - SameNode(GetStateVector)
  - **TrajectoryPlanner**
    - Atleast(1, (SatCommand_Subscriber,ReceiveSatCommand))
  - **ImageGrabber**
    - ImageGrabber _1: Atleast(1,(HR_1,LR_1))
    - ImageGrabber _2: Atleast(1,(HR_2))
    - ImageGrabber _3: Atleast(1,(LR_3))
  - **ImageProcessor**
    - ImageProcessor _1: Atleast(1,GPU_1)
    - ImageProcessor _2: Atleast(1,GPU_2)
    - Atmost(1,(ImageProcessor _1, ImageProcessor _2, ImageProcessor _3))
Calculating resiliency metric

• Question: how to measure the resiliency?

• Two ways to quantify:
  – **Worst case**: Minimum number of failures that make the mission infeasible
  – **Best case**: Maximum number of failures that the system can sustain while the mission remains feasible

• We translate the requirements and specifications into an SMT problem which calculates the metrics
Resilience Metric

• Metric = [2,23]
  – Assumption: all 6 functions are required
• Complete failure of Sat2
  – ImageProcessor on Sat2 is out, another ImageProcessor on Sat1 or Sat3 should be activated.
• Failure of GPU on Sat1
  – GPU is required by the Image Processor
  – Therefore, a reconfiguration is required which activates image processor on Sat3
• Failure of Ground Link on Sat 1
  – No reconfiguration is required. The ground command is disseminated by either Sat2 or Sat3 via pub sub ports
Future Work

• Given a configuration and a failure, what is the “optimal” reconfiguration?
  – Consider increasing horizon
  – Integrate empirical reliability measures
Questions?