Dynamic Evolution of .NET Systems

Jason Hallstrom, Mariana Barca, William Leal, Anish Arora

The Ohio State University
The Dynamic Reconfiguration Premise

- Systems evolve over time.
  - System mission changes.
  - Improved implementations are developed.
  - Defects are discovered.
- Evolutionary paths cannot be predicted.
- Long-running systems must be evolved dynamically.
  - Difficult/expensive/unsafe to shut system down.
  - Make changes while system is running.
DRSS
Dynamic Reconfiguration SubSystem

- Support for cross-cutting concerns
  - Example: add encryption/decryption to all component communication

- Scalable
  - Example: add encryption/decryption to many components, or to just one.

- Incremental evolution
  - Example: add encryption/decryption to a component, later add visualization.

- Accessible to practitioners
  - Not require major change in development methodology

www.cis ohio-state.edu/siefast/msr-cont-self-maint/
Basic Interception

Original Linkage

A → Method Invocations → B

Add Interceptor

A → Access Points → B

Access points (e.g., method signatures)
Access point invocation (e.g., method invocation)
Environment-supplied invocation binding

Functional unit interfaces (e.g., component interfaces)
Functional unit implementations (e.g., component implementations)
Send/Receive Interceptors
Interceptor Chains
How to use DRSS

1. Develop system for DRSS
2. Write interceptor
3. Compose interceptor with system
4. System continues, with interceptor
Example

Add message loss interceptor

Add visualization interceptor

Add tolerance interceptor

Note

(1) Original system *not* changed.

(2) Original system *not* designed for testing, visualization or tolerance.
Arrow Protocol

Distributed Queuing
The Demo
Using the Arrow Protocol

- Dynamically add visualization and fault injectors
  - Start Arrow
  - Add visualization interceptors
  - Add message loss interceptors
  - Cause message loss, observe intolerance

- Dynamically add tolerance to Arrow
  - Start Arrow, add message loss, visualization interceptors
  - Add tolerance interceptors
  - Cause message loss, observe tolerance

- Demonstrate DVI toolkit
Demo Recap

- The Arrow protocol was *not* written with interception in mind.
- It was *not* written to enable testing, visualization or tolerance
- Fault injection, visualization, tolerance were all added after the fact.

- Summary: The system was evolved dynamically to provide capabilities not originally planned.
DRSS Summary

- DRSS supports cross cutting concerns
  - Example: visualization does not depend on system modularization.

- DRSS is scalable
  - Example: call tracking interceptor added to two components.

- DRSS supports incremental evolution
  - Example: added visualization, then fault injection.

- DRSS is easy to use
  - Does not require major change in approach to programming.
Tools

- **DVI** – Dynamic Visualization Infrastructure
  - Generic visualization to assist system development.

- **CAT** – Custom Adaptation Toolkit
  - Converts a .NET dll into a DRSS-compatible version.