Normalized Algorithm Object Messaging Interface (NAOMI)

Mac Radigan

Contents

1 Some Advice From Richard Hamming

...I was solving one problem after another after another; a fair number were successful and there were a few failures. I went home one Friday after finishing a problem, and curiously enough I wasn't happy; I was depressed. I could see life being a long sequence of one problem after another after another. After quite a while of thinking I decided, "No, I should be in the mass production of a variable product. I should be concerned with all of next year's problems, not just the one in front of my face." By changing the question I still got the same kind of results or better, but I changed things and did important work. I attacked the major problem - How do I conquer machines and do all of next year's problems when I don't know what they are going to be? How do I prepare for it? How do I do this one so I'll be on top of it? How do I obey Newton's rule? He said, "If I have seen further than others, it is because I've stood on the shoulders of giants." These days we stand on each other's feet!

You should do your job in such a fashion that others can build on top of it, so they will indeed say, "Yes, I've stood on so and so's shoulders and I saw further." The essence of science is cumulative. By changing a problem slightly you can often do great work rather than merely good work. Instead of attacking isolated problems, I made the resolution that I would never again solve an isolated problem except as characteristic of a class.

Now if you are much of a mathematician you know that the effort to generalize often means that the solution is simple. Often by stopping and saying, "This is the problem he wants but this is characteristic of so and so. Yes, I can attack the whole class with a far superior method than the particular one because I was earlier embedded in needless detail." The business of abstraction frequently makes things simple... *– Richard Hamming*

2 Introduction

The Normalized Algorithm Object Messaging Interface (NAOMI) is a design specification for simplifying Service-Oriented Architecture (SOA) development.

The NAOMI architecture utilizes the following design practices:

- Software Component Interfaces
- Lean POJO
- OSGi (JSR-291)
- μ-Services

- RESTful Web Services
- Single Source Of Truth (SSOT)
- Atomic Operations (JSR-166)
- Functional Programming
- Identity, State, and Values (Rich Hickey)
- Continuous Integration (Karaf dev:watch with Nexus)
- Maven Integration and Provisioning (SpringSource DM)
- Lifecycle Management (Airies Blueprint)
- Unit Testing
- Dependency Injection Patterns
- Functional Mock-up Patterns

3 Overview of NAOMI

The NAOMI specification provides the following for Rapid Application Development (RAD) in a SOA:

- Normalized Algorithm Object Messaging Interface (NAOMI)
  - parent container for all subprojects (below)
  - flexible design can be integrated into a variety of existing architectures:
    - OSGi service
    - Java library
    - Standalone executable JAR
    - Script interpreter
  - SSH console administration and built-in logging facility (OSGi only)
  - Lifecycle management (built-in, but improves with OSGi)
  - Continuous Development (built-in, but improves with OSGi)
  - Provisioning (built-in, but improves with OSGi)
- Resource-Oriented Algorithm Repository (ROAR)

  - RESTful webservice interface to NAOMI
  - single repository for interface definitions
  - updated dynamically from Maven without service interruption
  - user-friendly XHTML interface (by XSLT from XML)
  - all reporting and transactions controllable through HTTP methods (URLs) using any web client or browser (by RESTful nature)
  - provides view on software configuration and operation

- Workflow Unified as a single Matrix for Processing Unlimited Services (WUMPUS)
  - workflow engine based on basic network graph theory
  - all configuration stored as a single, 2-D matrix
  - network graph constructed automatically from software component interfaces
  - workflow execution with concurrent atomic execution
  - each workflow is also a 2-D matrix (submatrix)
  - cannonical form provides a unique workflow hashcode ("normalized")
  - RESTful interface provided by ROAR
  - tools provided for both static and dynamic analysis
  - access to internal matrix provided in m-file and HDF5 format

- Now Yet ANother Control Algorithm Tool (NYANCAT)
  - library of analysis tools for reporting and logging
  - RESTful interface provided by ROAR
  - provides converters from WUMPUS matrix to a variety of formats (HTML, JPG, XML, XSL, XSD, HDF5, m-file, ...)
  - allows for runtime and log analytics of processing

- Robust Universal Test Harness (RUTH)
  - test facility for unit, thread, and regression tests
  - RESTful interface provided by ROAR
  - automatic execution of uploaded (scriptable) test plans

- automatic simulation available in the absence of functional services
- all actions and report accessible via URL (RESTful)
- user-friendly XHTML interface
- reporting and analytics provided by NYCANCAT

4 NAOMI Architecture

4.1 Architectural Diagram

5 Normalized Algorithm Components

5.1 Resource-Oriented Algorithm Repository (ROAR)

Resource-Oriented Algorithm Repository (ROAR) is an extensible, RESTful interface to the NAOMI system.

Updates may be installed from Maven with provisioning, and can be dynamically from Maven without service interruption (using Karaf dev:watch).

A variety of views on the both the configuration and running system are available from any web browser or HTTP client.

5.2 Workflow Unified as a single Matrix for Processing Unlimited Services (WUMPUS)

Workflow Unified as a single Matrix for Processing Unlimited Services (WUMPUS) provides a simple workflow engine based on basic graph theory from any undergraduate computer science curriculum. The internal configuration may be represented by a single matrix, in particular, an oriented incident matrix with degree. From this matrix, derivations may be made that satisfy all other graph theory needs.

By writing this configuration in cannonical form (normalized), the unique software configuration may be written as a single number (hashcode). This number uniquely represents the system, and may be independently verified during static analysis.

The system configuration is specified by function interfaces representing each service in the SOA (functors). The functor implementations themselves are calls to the individual services. This function signature looks like a C-style function prototype that we are familiar with. Parameters are further annotated additional metadata needed for the particular system. This list of functors is the only information that is needed to construct the workflow-defining matrix.

In the absence of absence of functional services, a built-in simulator or custom simulator may be used (see RUTH).

Figure 1: WUMPUS on day good day

5.3 Now Yet ANother Control Algorithm Tool (NYANCAT)

Now Yet ANother Control Algorithm Tool (NYANCAT) is a software tool for accessing WUMPUS configuration and processing, as well as RUTH test plans and processing logs.

The NYCANCAT library contains a rich set of converters for converting this data into a variety of formats:

- XHTML (with MathJax)
- Markdown
- ASCII text
- HDF5
- m-file
- GIF, JPEG, PNG, TIF
- XML
- XSL
- XML Schema

These results are accessible via commandline, as well as through the RESTful ROAR service.

Figure 2: NYANCAT on any given day

5.4 Robust Universal Test Harness (RUTH)

Robust Universal Test Harness (RUTH) is a facility for unit, thread, and regression testing. It allows for these tests to be scripted and run automatically.

The tests are recorded using RUTH's logging facility, and analytics provide detailed a detailed summary and rich graphical reporting of the historical results.

6 Summary