Eric Verhulst
Open License Society
ALTREONIC

An Interacting Entities
Modeling Methodology
For Robust Systems Design

VALID 2010,
Nice 24.08.2010
An Interacting Entities Modelling Methodology for Robust Systems Design

OpenCookbook is a web-based requirements and specifications capturing tool supporting a coherent and unified system development methodology based on the Interacting Entities paradigm
History

• Original R&D project of Open License Society:
  – Metamodel for systems engineering
    • “systems grammar”
  – OpenSpecs implemented as web portal

• EVOLVE ITEA project
  – Evolutionary Validation, Verification and Certification

• ASIL: Flanders Drive project on developing a common safety engineering methodology
  – Why are engineering and safety standards so heuristic?

• Currently commercialised and further productised by Altreonic under OpenCookBook
  – part of Concurrent Systems Composer development framework
Unified Systems/Software engineering

**OpenCookBook ©**
Formalised requirements & specifications capturing
Project repository

**OpenVE ©**
Formalized modelling
Simulation
Code generation

**OpenComRTOS ©**
Formally developed
Runtime support for concurrency and communication

**OpenTracer ©**
Visual Event Tracer

**SIL 3/4 Controller ©**
Control & processing platform natively supporting distributed concurrency & communication
Why FORMAL (ISED)?

Cost structures compared

- **Base cost**
- **Cost of change**

### Traditional bottom-up engineering
- **Run-away cost risk**

### Formalised iterative engineering

**First time right**
- = Less residual errors
- = Higher reliability
- = Less costs

**Incremental changes gives Requirements on process and architecture**

**Testing will only demonstrate absence of certain errors.**

**Formal verification can prove absence of any errors.**
OpenCookBook design goals

• Universality:
  – modelling any type of system, i.e. physical, software, hardware etc. (possibly with heterogeneous parts)

• Scalability:
  – support the development from small to very large and complex systems

• Extensibility:
  – possibility to change and to modify the meta-model (based on system grammar structure of database)
Support for Systems Engineering Process Activities

- Domain can be diverse:
  - technical engineering, organisation, engineering or business process, …
  - Engineering process will always combine engineering activities with process flow

- Requirements and specifications capturing
- Defining models and methodologies
- Defining architecture of a system in terms of interacting entities
- Defining workplan as set of work packages containing development, verification, test, and validation tasks
OpenCookbook Principles

- Using natural language for requirements and specifications capturing and architecture definitions
- Separation of concerns, concepts hierarchically decomposed and structured
- Unified repository (database) based on the Systems Grammar
- Using unified workflow for whole system engineering process
General System Definition Process

System Description

- Specifications
- Normal Cases
- Test Cases
- Fault Cases
- Issues

Conceptual (intentional) level

System Definition = Specs

Architectural (extensional) level

Stakeholders activity

Requirement

System

Models + Methods

Evolve
Relationships between conceptual and architectural levels of a system under development (1)
Relationships between conceptual and architectural levels of a system under development (2)

Conceptual Level

- Simplicity of driving
- Device generating movement
- Anti-stealing safety
- Soft and easy movement
- Safety at collisions

Architectural Level

- Control system
- Engine
- Locks
- Belts and pillows
- Chassis

Architectural Decomposition
OpenCookBook conceptual schema = project’s state space

- **Project**
  - System description
  - System definition
  - Work Plan

- **Models**
  - Conceptual
  - Architectural
  - Implementation
  - Formal
  - Simulation

- **Methodology**
  - Analysis
  - Development
  - Implementation
  - Testing
  - User Specified

- **Task**
  - Development
  - Verification
  - Validation
  - Test

- **Work Package**

- **Specifications**
  - Requirements
    - Normal Case
    - Test Case
    - Fault Case

- **Entity**
  - Subsystem
  - Interaction
  - Function
  - Interface

- **Method**
  - Procedure
  - Tool
  - Role

- **Issues**

**Open License Society**

meta-meta-level definitions: generic & abstract

Meta-level: domain specific
The state transitions during system definition

∀((Requirement.Status = Approved))

∀(Requirement.Status = Not applicable))

→ Specification.Status = Approved
Requirements for evolutionary/incremental verification/validation/certification

- Product/system development process builds several dependent "state-spaces"
  - Top level is "mission" (top-requirement) for requirements/specifications view
  - Top level is system under development in its environment for architectural view
  - Validation/certification is top level for workplan view

- Consequences:
  - Orthogonality requirement to reduce dependencies and localise state-spaces
  - Strict version management
  - Traceability
Systems grammar = information model
OpenCookBook developed as a multi-user web portal
OpenCookbook functionality

- System definition through the web
- Possibility of work in local mode on PC
- Organisation of discussion on system requirements, specifications, architecture and work plan
- Queries to project database
- Intuitive interface and easy navigation, using WYSIWYG web-based editors
OpenCookbook functionality

• Generation of project documentation (in html)
• Generation of Task Juggler reports
• Import/export project database
• Implementation of mapping between project levels by hyperlinks.
Dependency tree

- From checkpoint to release, dependency tree can be displayed and navigated

=> first step towards “delta-management” for incremental verification/validation/certification
Precedence tree

- From release or validation task to requirement, precedence tree can be displayed and navigated
- => first step towards “delta-management” for incremental verification/validation/certification
Export to Task Juggler

- For all tasks in WPs, task project management parameters are exported to Project Management tool (Task Juggler)
Gant chart, generated from Task entries
Integration with real-time embedded frameworks

- **Integration with OpenVE RT-modelling environment**
  - Software entities => OpenComRTOS tasks and SW functions
  - Interactions => OpenComRTOS hubs and comm protocols
  - RT attributes (e.g. UML Marte, SMART, …)

- **Attributes and state transition conditions** for project scheduling and management

- **Attributes and transition conditions to support certification processes**

- **Metamodel supporting organisation specific flows**
Integration with OpenVE using metamodel (xml)
Technical info

• OpenSpecs
  – Based on Drupal 5.21 Content Management System
  – Web server (tested with Apache v. 2.2)
  – PHP (tested with 5.3)
  – MySQL (tested with 5.0)
  – scalability and maintainability issues

• OpenCookBook v1
  – See OpenSpecs

• OpenCookBook v2
  – Wt: compiled web portal in C++
  – Enhanced metamodel
Conclusion

• Systems engineering process can be formalised using common metamodel

• Challenges
  – Integration of different domains
    • Process, architectural, certification
    • System Engineering processes (“standards”) are heuristic standards
  – Human interface design: must be intuitive
  – Formal(ised) analysis of requirements

• Progress through formalisation
  – Reduction of design space give reliability
Conclusion

• More info:

www.altreonic.com

Eric.Verhulst@altreonic.com

OpenCookBook1 freely downloadable
Panel

• In search for hardware that executes specifications efficiently

• Correlate:
  – In search for software that executes requirements efficiently
Panel

- Project is “walking the tree” in project’s statespace
  - Requirements -> specifications -> model -> implementation in SW and HW
  - Final model is implementation (model)
  - The larger the statespace the more error prone, more difficult to verify and validate
  - Less is also less for power and cost!
Panel

• Changing / increasing requirements
  – Before: only “normal” case: easy (sic)
  – Then: also “test case” (intrusive)
  – Now also: “fault case” => safety & security!
    • Decomposition in entities and interactions
      – (concurrency and communication)
    • Error trapping
    • Fault containment
    • Fault recovery
    • Resource metering (time, memory, bandwidth, power)
    • => additional complexity and system behaviour!
Panel

• But:
  – We program mostly with sequential programming languages as abstraction layers on top of sequential von Neuman CPUs
  – Software doesn’t execute hardware!
  – Software must be efficient in translating requirements in specifications

• Hence:
  – Hardware must be efficient to execute specifications!
Challenges in Testing and Validating Complex Systems

Keith Stobie
Microsoft

Validating approximately:
Decision Systems & Loose Consistency
Decision Systems

• systems created via Machine learning
  – Rule based
  – Neural Networks
  – Decision Trees

• How to test approximations?
  Heuristic Oracles for what is clearly wrong.
Large Scale Distributed Systems

• Asynchronous, loosely coupled
• NoSQL architectures provide weak consistency guarantees such as eventual consistency
• database terminology, BASE (Basically Available, Soft state, Eventually consistent)
• PAXOS (consistency) - Liveness(C;L)
  – If value C has been proposed, then eventually learner L will learn some value (if sufficient processors remain non-faulty).
BASE

- Weak consistency
  - stale data OK
- Availability first
- Best effort
- Approximate answers OK
- Aggressive (optimistic)
- Simpler!, Faster, Easier evolution

Testing Eventual Consistency?

• Heuristic Oracles for answers that are
  – too approximate
  – too inconsistent for too long
Composition of services for an effective measurement process

Maurizio D’Arienzo
Dipartimento di Studi Europei e Mediterranei
Seconda Università degli Studi di Napoli
Monitoring of complex systems

• Testing and validation through measurement process is not a straightforward task
• The use of a specific mechanism (familiar tool) may be inaccurate under certain conditions
• Some contraints on precision:
  - Convergence time
  - Tool selection
  - Synchronisation
  - Intrusiviness
  - Interference
Composition of services

• Static, independent tools are outdated, dynamic, interoperating tools are sound.

• Design of novel monitoring systems:
  – Composition of different measurement techniques in a fair environment
  – full compliance and open interaction with existing tools
  – mutual exclusion of concurrent measurements
  – automatic tool selection and their configuration
Many Ways to Automate Development of Automated Tests

Vladimir Rubanov, Ph.D.
vrub@ispras.ru
Head of Operating Systems Department at the Institute for System Programming of the Russian Academy of Sciences (ISPRAS)
Director of Russian Linux Verification Center (linuxtesting.org)
[Usually] Test Execution Should Be Automated
How To Create Automated SW Tests?

- **Manual Development:**
  - Plain programming language
  - Test development frameworks

- **Automatic Generation:**
  - Based on “nothing”
  - Based on thorough models
Dimensions of Automated Test Suite Quality

1. “Wideness”: how many target functions/blocks are tested at all?
   – the scope of testing suite.

2. “Deepness”: how many (and how smart) various input combinations in various internal states for particular function/block are iterated?
   – the quality of test actions / sequences.

3. “Checking Thoroughness”: how well is the correctness of the SUT responses checked?
   – the quality of test oracles.
There Are Different Technologies Available That Help Automate Development of Automated Tests

- Sometimes you need to use a combination of the technologies.
- It is good when you can “adjust” the test suite quality by the mentioned dimensions independently.
- Need to take into account:
  - Resources / cost / time to develop tests
  - Importance of particular SUT functions/blocks
Example: Testing Linux For Conformance with Linux Standard Base (LSB) Specification

1. LSB defines requirements for more than 30,000 APIs in more than 50 system libraries.
2. It is impossible to create good automated tests for all of these in reasonable time/resources.
3. We have classified all APIs by 3 categories of importance.
4. A combination of 3 different test development technologies have been used for creating the necessary tests.
Example: LSB Conformance Test Suite (1)

Low Importance APIs:

1. Generate shallow tests fully automatically based on “nothing”.
   - High “wideness”, low “deepness”, low “check thoroughness”.
   - Very low cost per API.

2. Further improvements as resources appear:
   - Add additional info for more advanced test generation – “nothing” converts to “little hints” – increasing “deepness” & “check thoroughness”.
   - Manual test cases’ tweaks using “normal” test development framework.
Example: LSB Conformance Test Suite (2)

Medium Importance APIs:

1. Step 1: Generate test templates automatically.
2. Step 2: Manually develop “normal” unit tests based on the templates using a unit test development framework:
   - Moderate “deepness”
   - Moderate-high “check thoroughness”.
   - Moderate cost per API.
Example: LSB Conformance Test Suite (3)

High Importance APIs:

1. Create a model-driven test suite based on formal specifications of the APIs.
   • High “deepness” & high “check thoroughness”, which can be independently adjusted (including dynamically configured from some minimal to maximum for different test runs).
   • High cost per API.
ISPRAS Linux Verification Center

- Founded in 2005
- A division of ISPRAS
- Working closely with Linux Foundation (formerly FSG), Intel, Motorola, local companies.
- ~30 engineers