Behavioral Modeling of System Architectures with Monterey Phoenix

Kristin Giammarco, Ph.D.
Associate Professor, Department of Systems Engineering

Monica Farah-Stapleton
Ph.D. Candidate, Department of Computer Science

Mikhail Auguston, Ph.D.
Associate Professor, Department of Computer Science

Monterey, California
Emergent behavior is:
you create a system,
you think you know how it’s going to behave,
you think you know how it’s going to control,
you think you’ve written down the equations or analyzed it,
and it configures itself in a way that you did not anticipate,
and exposes a behavior, a phenomenon, a result, a performance,
that you did not see coming at you.

And it’s because the control is all over the place.
The interaction of a large number of simple things is very hard to predict.

The complexity is not in the individual things, it’s in the way in which they’re interconnected.

- Leonard Kleinrock, Distinguished Professor of Computer Science, UCLA

http://dsc.discovery.com/tv-shows/curiosity/topics/l-kleinrock-what-is-emergent-behavior.htm

February 2014
Agenda

• Monterey Phoenix, a new strategy that separates the specification of system interaction from system behavior
• Limitations in current strategies lacking separation of concerns
• Using MP to generate views, use cases, and to conduct architecture verification and validation
• Using MP to expose undesired emergent behavior in an architecture model in advance of system testing or operation
• Summary, Way Ahead, Discussion

The key advancement for SoS modeling: a decoupling of system interaction and system behavior.
What Does Separation of System Interaction from System Behavior Mean?

From...

**Sequencing external interactions in the same model as internal interactions**

To...

**Specifying behavior of each system separately from interactions among those systems, and compute possible sequences from there**

February 2014
Why Separate System Interaction from System Behavior?

A set or arrangement of systems that results when independent and task-oriented systems are integrated into a larger systems construct, that delivers unique capabilities and functions in support of missions that cannot be achieved by individual systems alone.

1. User provides a general identification.
2. System requests unique identification.
3. User provides a unique identification.
4. If the credentials are valid, the System authorizes the User to access the services; otherwise the System notifies the User that credentials are invalid and the user may re-attempt access up to two more times.
5. The User or the System ends the session.

**Issue with this approach:** No variability in behavior is modeled in the external system (User).
1. User provides a general identification.
2. System requests unique identification.
3. User provides a unique identification.
4. If the credentials are valid, the System authorizes the User to access the services; otherwise the System notifies the User that credentials are invalid and the user may re-attempt access up to two more times.
5. The User or the System ends the session.

**Issue #1:** Selection of behavior (for creds valid, creds invalid) is not automatically coordinated in the notation itself.

**Issue #2:** Multi-actor activity diagrams do not scale well when attempting to represent a broad range of each actor’s behavior.
Approach to formal software and system architecture specification based on behavior models

A view on the architecture as a high level description of possible system behaviors, emphasizing the behavior of subsystems and interactions between subsystems

The emphasis on specifying the interaction between the system and its environment

The behavior composition operations support architecture reuse and refinement towards design and implementation models

Executable architecture models provide for system architecture testing and verification with tools
Monterey Phoenix Basic Concepts

- **Event**: Any detectable action in a system’s or environment’s behavior
- **Event Trace**: Set of events with two basic partial ordering relations, precedence (PRECEDES) and inclusion (IN)
- **Event Grammar**: Specifies the structure of possible event traces
- **Schema**:
  - Represents instances of behavior (event traces)
  - Contains collection of events called roots (components and connectors)
  - Composition operations specify interactions between root behaviors
  - Additional constraints on behaviors
- **Behavior Model of a System**: Specified as a set of all possible event traces using a schema
- **Executable Architecture Model**: Provides rapid prototype of system under consideration
- **Automated tools**: MP supports automated tools for system architecture validation and verification (MP-Alloy, MP C++ Implementation, Eagle 6)
Model Behavior of Environment: User Behavior

01  ROOT User:
02    (*  request_access
03      (* creds_invalid request_access *)
04      ( creds_valid (run_services | abandon_access_request) |
05          creds_invalid (attempt_exhausted | abandon_access_request) )
06    end_User_session *)

07  request_access:  provide_general_ID  provide_unique_ID;
ROOT System:

(* request_unique_ID
  [ creds_invalid request_unique_ID
    [ creds_invalid request_unique_ID
      [ creds_invalid attempt_exhausted
        invalid_creds_notice cancel_access_request ] ] ]
  [ (creds_valid ( authorize_access run_services |
      long_wait_for_User cancel_access_request ) |
    creds_invalid long_wait_for_User cancel_access_request ) ]
end_System_session *)
Model Component Interactions Abstractly and Separately

18 COORDINATE (* $x$: provide_general_ID *) FROM User,
19 (* $y$: request_unique_ID *) FROM System
20 ADD $x$ PRECEDES $y$;
21 COORDINATE (* $x$: request_unique_ID *) FROM System,
22 (* $y$: provide_unique_ID *) FROM User
23 ADD $x$ PRECEDES $y$;
24 User, System SHARE ALL creds_valid, creds_invalid, attempt_exhausted, run_services;

Branches shared by components:
creds_valid shared by User, System
creds_invalid shared by User, System

Functions shared by components:
Attempt Exhusted shared by User, System
Run Services shared by User, System
## 30 Unique Use Cases from the one Authentication Model in MP

### February 2014

#### Exhaustive Scenario Generation • Random Scenario Generation

### Code Editor

```plaintext
// January 11, 2014
// Authentication scenario architecture model, first version
// From Kristin Cimmaro, Mikhail Auguston. Well, You didn’t Say not to! A Formal Systems Engineering Approach to Teaching an Unively Architecture Good Behavior.

ROOT User: (* request_access
(* <0-3> creds_invalid request_access *)
  ( creds_valid (m_services | abandon_access_request)) )
end_user_session
)
request_access: provide_general_ID provide_unique_ID;

ROOT System: (* request_unique_ID
[ creds_invalid request_unique_ID ]
[ creds_invalid request_unique_ID ]

Edit Code
```
Use Case 1: User gets access after one unsuccessful attempt.
Use Case 2: User abandons access request after two unsuccessful attempts.
1. Behavior description for each system + abstract interaction specification

2. Scenario (use case) generation

3. Scenario visualization, inspection, and assertion checking

4. Modify the design to exclude unwanted scenarios

e.g., “Are there any instances where a user can gain access to the system after three attempts?”
## Stakeholder Examples

<table>
<thead>
<tr>
<th>Stakeholder</th>
<th>Typical Questions or Groups of Questions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Customer</td>
<td>Are user, technical, cost, and management expectations being met?</td>
</tr>
<tr>
<td>Users</td>
<td>Does this system do what was expected? Does it fulfill prioritized requirements?</td>
</tr>
<tr>
<td>Engineers / Designers</td>
<td>What implementation option(s) should be considered to meet performance expectations? What are environment interactions and constraints for each option?</td>
</tr>
<tr>
<td>Testers</td>
<td>What are optimal instrumentation points? What statistics should be gathered? What is the correct level of abstraction?</td>
</tr>
<tr>
<td>Cost Analysts</td>
<td>What is the cost of the system from requirements elicitation thru software evolution?</td>
</tr>
</tbody>
</table>

### System’s Architecture

**User Requirements**

**High Level Design**

February 2014
Use Monterey Phoenix to Reason About System and Environment Behaviors, and Inform Resourcing Decisions

- Resourcing decisions must consider overall project costs
  - Include facilities, processes, staffing, tools, products, and development
  - For many projects, development effort represents the biggest component of cost
- Spectrum of estimation strategies from Excel through software parametric models
  - MP can provide a framework to support many existing resource estimation methodologies
  - Function Point analysis
    - Software size & cost estimates using well-established methodologies
  - Software Non-functional Assessment Process (SNAP) analysis
    - Nonfunctional requirement verification using events with attributes such as duration, delay
Function Point Terminology

- Describe interactions of a user, system and its environment.
- External Inputs (EI): Input data that is entering a system.
- External Outputs (EO) and External Inquires (EQ): Data that is leaving the system.
- Internal Logical Files (ILF): Data that is processed and stored within the system.
- External Interface Files (EIF): Data that is maintained outside the system but is necessary to satisfy a particular process requirement.

Function Point Analysis Practice

- Count Data and Transactional Function Types.
- Determine Unadjusted FP count and the Value Adjustment Factor.
- Calculate final Adjusted FP Count.

February 2014
Software Non-functional Assessment Process (SNAP)

- **Software Non-functional Assessment Process (SNAP)**
- **Apply SNAP sizing process**
  - Identify application boundary.
  - Associate non-functional requirements with relevant categories and their sub-categories.
  - Size each sub-category according to type and complexity using standardized set of basic criteria (NOTE: Size is sum of sizes sub-categories).
  - Total sizes to give measure of non-functional size of the software application.
Relating MP to FP and SNAP

- Use MP to unambiguously describe boundaries and interactions of the system, user, and environment
- Apply FPA practice
- Apply SNAP sizing process
- Assess applicability of FPA and SNAP to inform project and program life cycle cost estimates
How Do We Implement?

• An MP prototype has been implemented as a compiler generating an Alloy relational logic model from the MP schema and then running the Alloy Analyzer to obtain event traces and to perform assertion checks.

• A prototype trace generator converts MP schemas into a C++ code, compiles and runs it. Generation speed reaches $10^4$ events/sec, the search space up to $10^{15}$ traces.

• On-line demo is available at
  • http://modeling.eagle6.com

• MP model checking tool implemented at the National University of Singapore by Dr. Jin Song Dong group.

• Current research efforts at Naval Postgraduate School looking at how to integrate the MP approach into the systems engineering vernacular.
MP Research Areas (Next Steps)

• Composition and reuse of MP models (e.g., for improved system of systems modeling)
• Architecture views (extracting different diagrams from the MP model, such as UML/SySML activity diagrams, functional flow diagrams, sequence diagrams, context diagrams, fit-for-purpose (custom) diagrams)
• Next implementation of the event trace generator and assertion checker
• Improved trace visualization
• Behavior patternning
• System performance estimates based on event timing attributes: duration, latency, critical paths
• Statistical system behavior simulation; system safety estimates based on interaction with environment models
• Methodology for using MP in a Systems Engineering problem space (e.g., requirements engineering, risk analysis, reliability/maintainability/availability analysis, architecture and design, test and evaluation, cost estimation)

• Use case scenarios for a specific system
  – Representative sets of use cases (event traces) in a domain of interest (see above for examples)
  – Use case generation methods (efficient strategies for organizing MP models)
  – MP models incorporating criteria from different stakeholders to inspect resulting use cases for contradictions
  – Application for implementation testing, maintenance, and/or documentation
  – Assertion checking for use cases leveraging Small Scope Hypothesis (testing an MP model for the presence or absence of some suspected property).
How Does Monterey Phoenix Address Issues With Traditional Architectures?

- Captures behaviors and interactions between system and environment
- Captures design decisions early, assess and modify without incurring the costs of incorrect implementations
- Enables Verification and Validation of system, focused on behaviors, interactions, and automated tools for early verification
- Enhances and extends DoDAF, UML, and SysML frameworks and notations
- Executable architecture model leveraging “lightweight” formal methods to unambiguously describe behaviors
- Provides uniform way to extract use cases from single architecture model composed of component behavior algorithms and abstract interaction specification

Why Do We Care?

- Inform quantifiable cost estimates across enterprise and lifecycle of system
- Specifically address concerns of multiple stakeholders
- Allow automatic generation of behavior examples (Use Cases) for early system architecture analysis, testing, verification, and validation

February 2014
Questions Please

kmgiamma@nps.edu
monica.farah-stapleton@dha.mil
maugusto@nps.edu

Monterey Phoenix and Related Work:  http://faculty.nps.edu/maugusto

Simplicity does not precede complexity, but follows it.

-Alan J. Perlis, Yale University

Perlis, Alan J., “Epigrams in Programming.” SIGPLAN Notices 17(9), September 1982
More epigrams at http://www.cs.yale.edu/quotes.html

February 2014


Monterey Phoenix was presented at CAS on November 15, 2013. The full paper can be accessed from:
Monterey Phoenix was an invited presentation topic at the Southern Jersey Professional Societies meeting on January 15, 2014.

Well, you didn’t say not to! A formal systems engineering approach to teaching an unruly architecture good behavior
Kristin Giammarco, PhD, Naval Postgraduate School

This presentation introduces a new formal modeling approach known as Monterey Phoenix (MP), which has features that enable prediction of emergent reactive system and system of systems (SoS) behaviors that result from interactions among subsystems and among the system and its environment. The approach emphasizes specification of component behavior and component interaction as separate concerns at the architectural level, consistent with well-accepted definitions of SoS. Implementing this separation of concerns in modeled architectures substantially increases the number of behaviors appearing in simulations for better prediction of design flaws and other latent behaviors. MP provides a new capability for automatically verifying system behaviors early in the lifecycle, when design flaws are most easily and inexpensively corrected. MP extends existing frameworks and allows multiple visualizations for different stakeholders, and has potential for application in multiple domains.

Wednesday Jan 15, 2014
Call in your reservations by noon on Tuesday, Jan 14th.

Mays Landing Golf and Country Club, Frasier Room
1855 Cates Rd. Mays Landing, NJ (641-4411)
Check their website for directions: mayslandinggolf.com
Monterey Phoenix was presented to the INCOSE SoS WG on January 17, 2014.
Modeling the System in the Context of Its Environment

Modeling the System and Its Environment in the Context of One Another

User

System

User/System Interaction Specification

Branches shared by components:
- creds valid shared by User and System
- creds invalid shared by User and System

Functions shared by components:
- Attempt Exhausted shared by User and System
- Run Services shared by User and System

February 2014