Behavioral Models for Systems Architecture and Workflow Analysis

Mikhail Auguston, Kristin Giammarco
Computer Science Department, Systems Engineering Department
Naval Postgraduate School
Monterey, California, USA

"Every system has an architecture, whether or not it is documented and understood."

ROZANSKI, N., WOODS, E., 2012,
Software Systems Architecture, 2nd Edition, Addison-Wesley

Technical Rationale

- A system architecture description belongs to a high level of abstraction, ignoring many of the implementation details, such as algorithms and data structures
- The architecture plays a role as the bridge between requirements and implementation of a system
- Errors in early system design are the most expensive to fix when detected later in the development lifecycle
- Modeling is an approach to the design and verification of system architecture

Technical Rationale

- One of the major concerns in architecture design is the question of the behavior of the system
- An architecture specification should be supportive for the refinement process
- Composition operations focus on the interactions between the parts of the system
- An architecture of a system is considered in the context of the environment in which it operates, including business processes
- The architect needs a number of different views of the architecture for the various uses and users

What is Monterey Phoenix?

http://wiki.nps.edu/display/MP

MP is a framework for software system architecture and related workflow modeling with the focus on behavior of software system and its environment

Behavior is defined as a set of events (event trace) with two basic relations: precedence and inclusion

- The MP trace generator produces all possible scenarios of system behavior up to a scope limit.
- MP model separates component behaviors and component interactions.

The Innovations

- An **executable** system architecture model - Monterey Phoenix scenario generator can produce event traces with several hundred or small thousands of events

- An event trace **visualization framework** that enables human analysts to focus on the behavior of the system and provides **multiple views** for different stakeholders

- Mechanisms to run **queries** on the automatically generated event traces, and a language for event trace analysis (**assertion checking**)

The main MP innovations in BPM

- Traditional business process modeling frameworks (BPEL, BPMN, UML, IDEF) are constrained by the *"single flowchart"* paradigm

- MP separates component behaviors from the component interaction, and thus provides a multidimensional picture of concurrent behaviors, with overlapping threads of process phases and participating actors

Basic concepts for behavior modeling

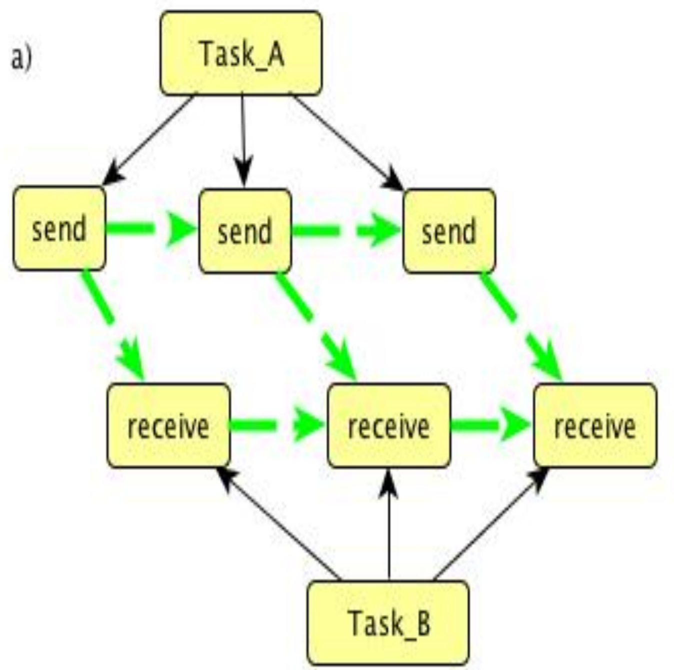
Event - any detectable action in system's or environment's behavior

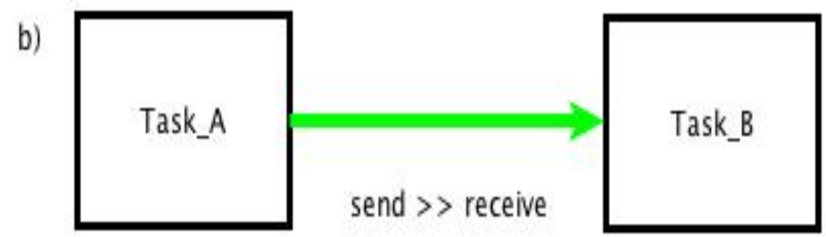
Event trace - set of events with two basic partial ordering relations, **precedence** (PRECEDES) and **inclusion** (IN)

Event grammar - specifies the structure of possible event traces

A simple pipe/filter architecture pattern

```
SCHEMA   simple_message_flow
ROOT Task_A:     (* send *);
ROOT Task_B:     (* receive *);
COORDINATE     $x: send       FROM Task_A,
               $y: receive  FROM Task_B
               DO        ADD $x  PRECEDES  $y;  OD;
```





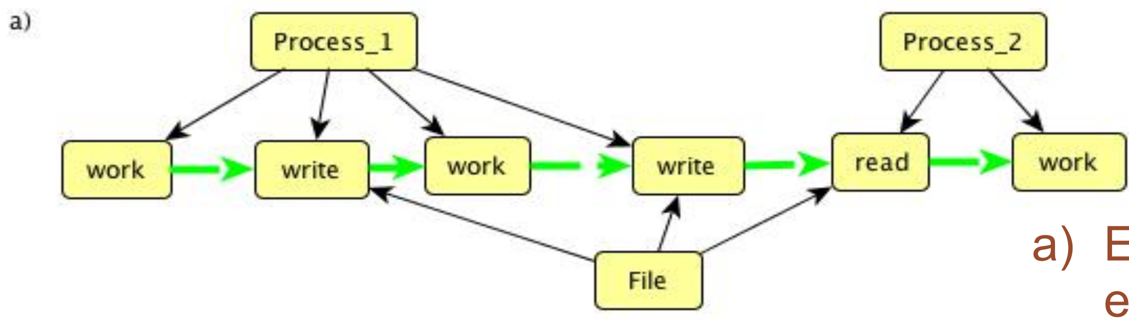
a) Example of a composed event trace

b) An architecture view for the schema

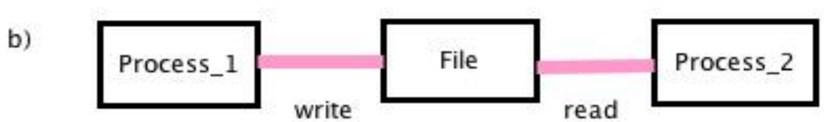
Data items as behaviors

Data items are represented by actions that may be performed on that data

```
SCHEMA          Data_flow
ROOT Process_1:       (*  work   write  *);
ROOT Process_2:       (* ( read | work ) *);
ROOT File:            (+ write +)  (* read *);
Process_1, File       SHARE ALL   write;
Process_2, File       SHARE ALL   read;
```



a) Example of a composed event trace



b) An architecture view

Architecture Verification & Validation

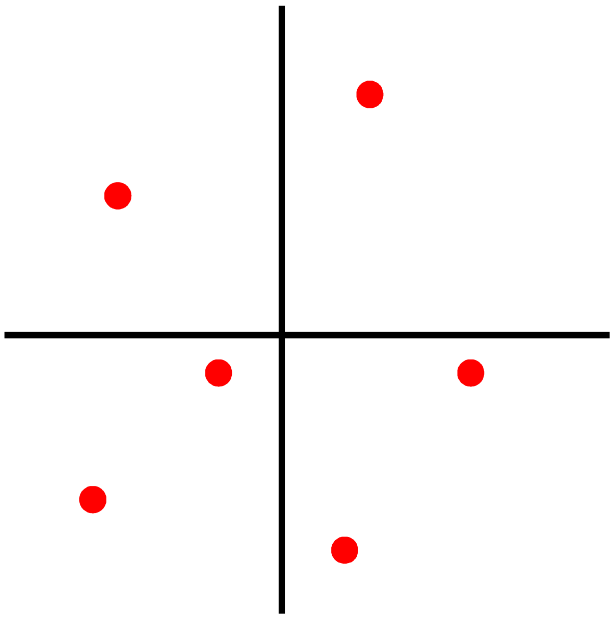
Advantages of Monterey Phoenix approach compared with the common simulation tools are as follows:

- Means to write **assertions** about the system behavior and tools to verify those assertions.
- **Exhaustive search** through all possible scenarios (up to the scope limit).
  - The **Small Scope Hypothesis**: most flaws in models could be demonstrated on small counterexamples
- Integration of the architecture models with **environment models** for verifying system's behavior on typical scenarios (Use Cases).
- **Event attributes**, like timing, can be used for non-functional requirements (like performance estimates) V/V and queries (like critical path estimates in PERT diagrams).
- Assigning **probabilities** to certain events makes it possible to obtain statistical estimates for system behaviors.

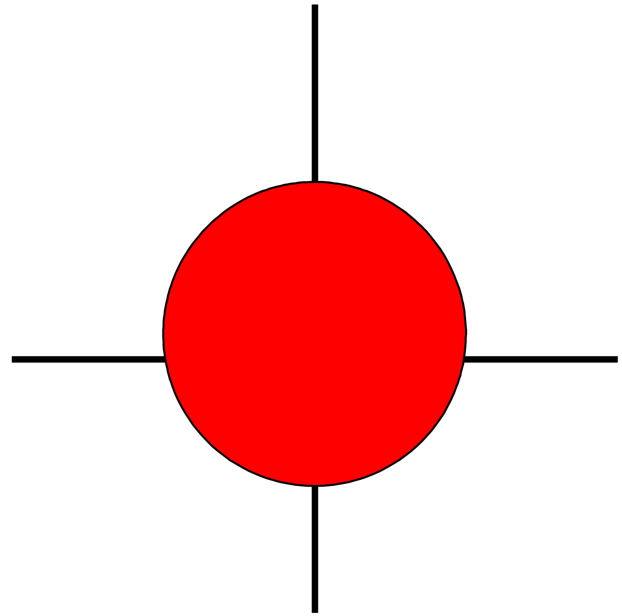
Architecture verification & validation

- It is much easier for different stakeholders to understand and verify stand-alone scenarios (Use Cases) neither the complete formal description of the system
- Scenario inspection in MP can be automated by assertion checking tools
- Interactions of subsystems and environment can be used for detecting emerging behaviors of System of Systems
- Different views can be automatically extracted and visualized for different stakeholder needs

Model verification within limited scope



Testing:
A few cases of arbitrary size



Scope-complete:
All cases within a small bound

Implementation

On-line MP editor/trace generator and a set of pre-loaded examples are available at

http://firebird.nps.edu

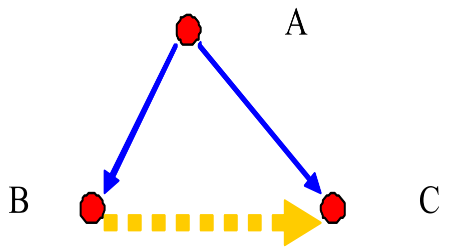
MP wiki with Crash Course and reading materials (publicly available part):
https://wiki.nps.edu/display/MP/Monterey+Phoenix+Home

MP model checking tool was implemented at the National University of Singapore by Dr. Jin Song Dong's team

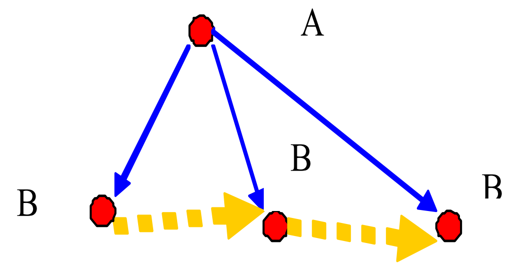
Backup slides

Event grammar

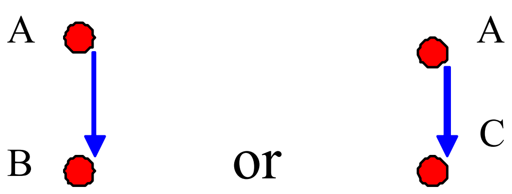
The rule A:: B C; specifies the event trace



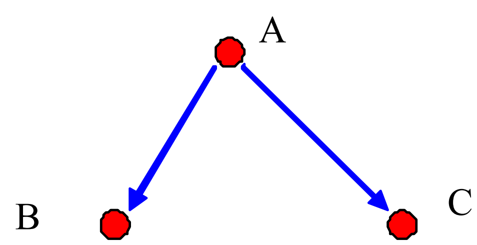
A:: (* B *); means an ordered sequence of zero or more events of the type B.



A:: (B | C); denotes alternative



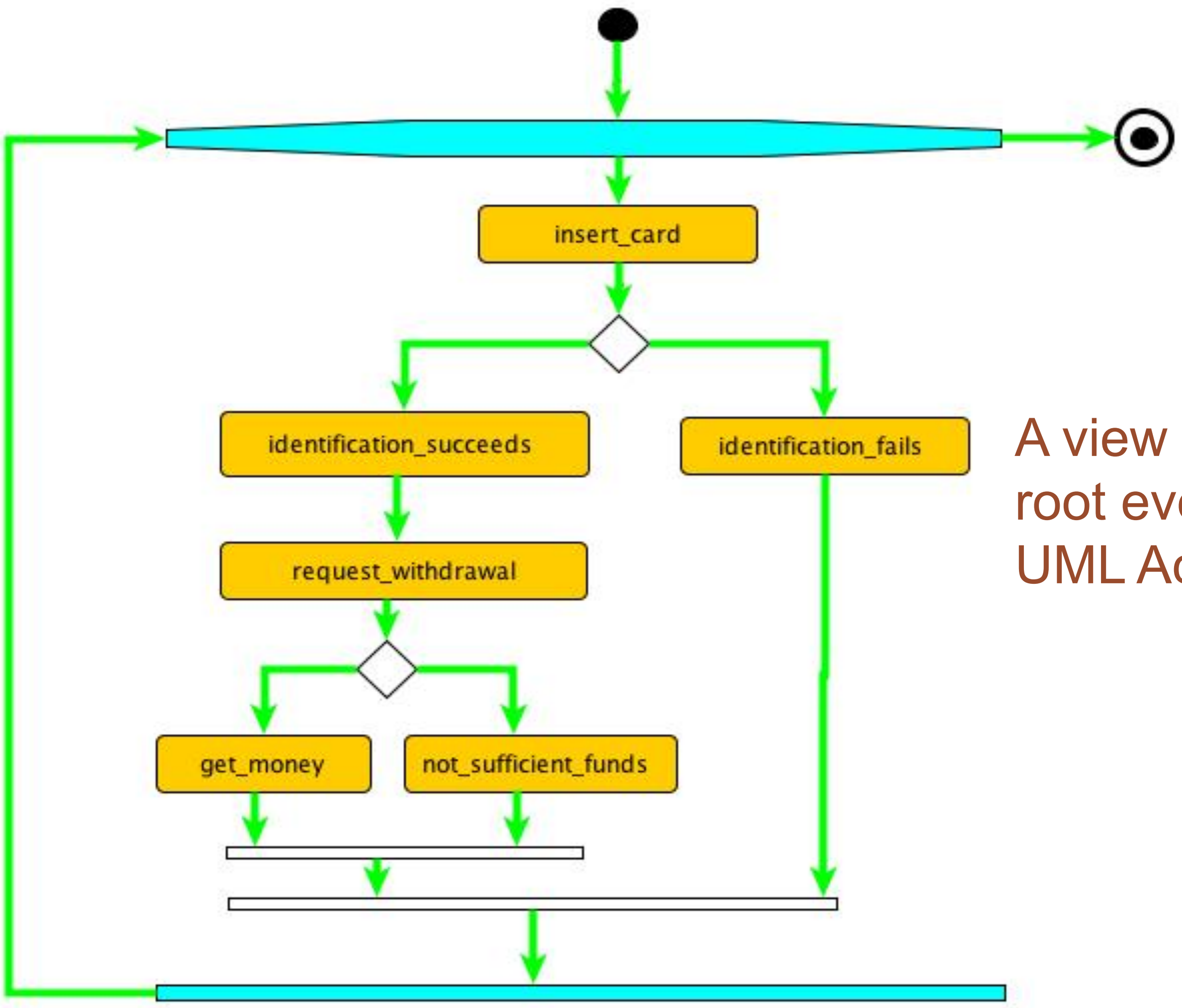
A:: { B, C }; denotes a set of events B and C without an ordering relation between them



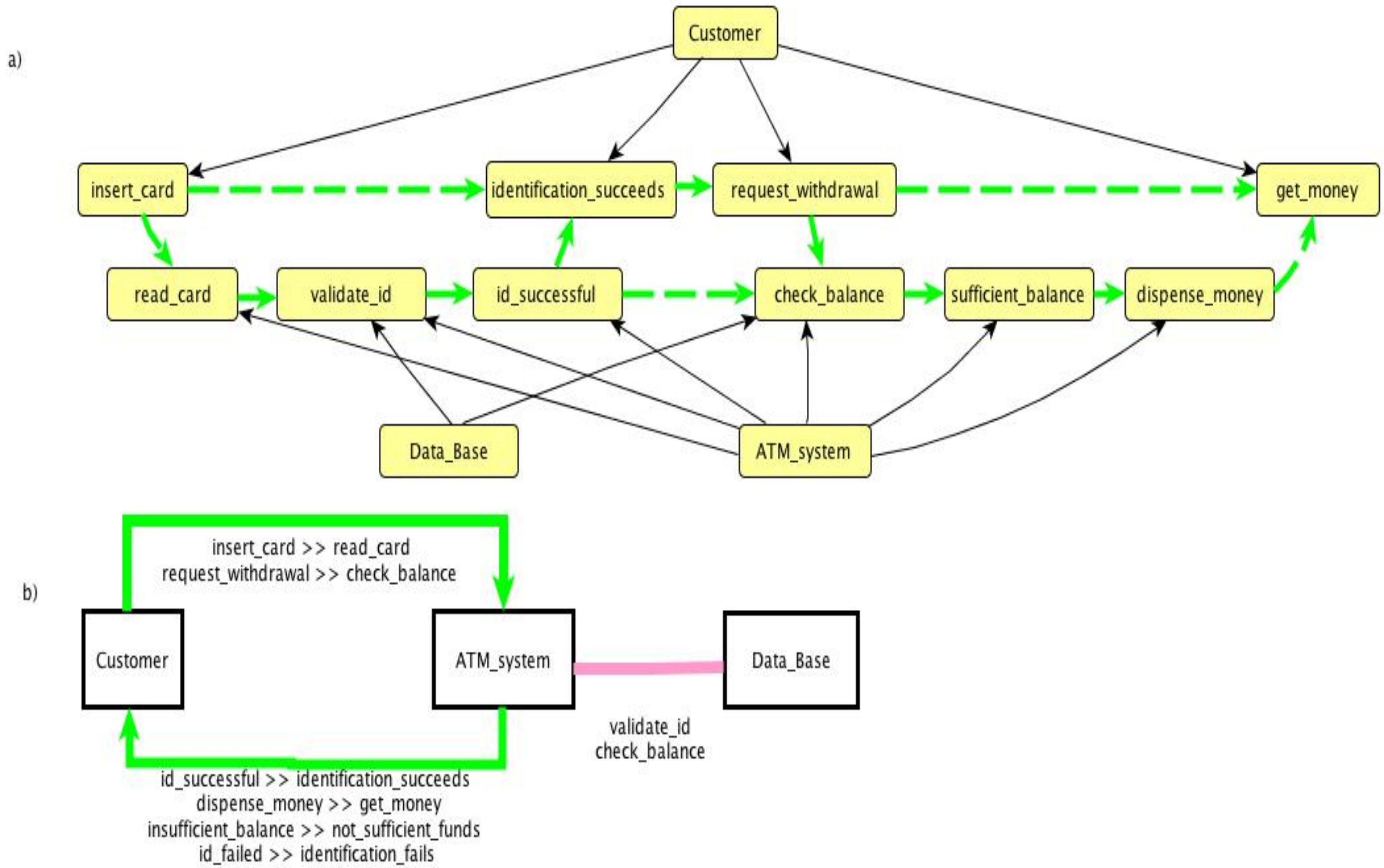
Integrating environment's behavior

```
SCHEMA            ATM_withdrawal
ROOT Customer:    (*  insert_card
                  ( (  identification_succeeds  request_withdrawal  ( get_money | not_sufficient_funds ) )  |
                       identification_fails   )     *);
ROOT ATM_system:  (*  read_card   validate_id
                         ( id_successful          check_balance
                                 (  (sufficient_balance    dispense_money) |
                                    unsufficient_balance )                     |
                           id_failed    )   *);
ROOT Data_Base:   (* ( validate_id | check_balance ) *);

Data_Base,  ATM_system SHARE ALL validate_id, check_balance ;

COORDINATE        $x: insert_card           FROM Customer,
                  $y: read_card             FROM ATM_system     DO ADD $x PRECEDES $y; OD;
COORDINATE        $x: request_withdrawal  FROM Customer,
                  $y: check_balance         FROM ATM_system  DO ADD $x PRECEDES $y; OD;
COORDINATE        $x: identification_succeeds FROM Customer,
                  $y: id_successful            FROM ATM_system   DO ADD $y PRECEDES $x; OD;
COORDINATE        $x: get_money             FROM Customer,
                  $y: dispense_money        FROM ATM_system    DO ADD $y PRECEDES $x; OD;
COORDINATE        $x: not_sufficient_funds FROM Customer,
                  $y: unsufficient_balance FROM ATM_system   DO ADD $y PRECEDES $x; OD;
COORDINATE        $x: identification_fails    FROM Customer,
                  $y: id_failed                FROM ATM_system      DO ADD $y PRECEDES $x;OD;
```

Architecture view on the component behavior



A view on the Customer root event behavior as UML Activity Diagram



a) An example of event trace (Use Case) for the ATM_withdrawal schema
b) An architecture view for the ATM_withdrawal schema