A Process Modeling Framework for Formal Validation of Panama Canal System Operations

Mark Austin and John Johnson,
Department of Civil Engineering and ISR,
University of Maryland, College Park, MD 20742.
Outline

• Motivation
• Early Validation of Systems
• Automated Composition of Process Models
• LTSA
• Lockset Process Model
• Model Checking
• Conclusions/Future Work
Motivation

The Panama Canal has been operational since 1914.
Left: Congestion in the Miraflores lock and lake.

Right: Limitations of present-day canal capacity…
Motivation

The Panama Canal is currently undergoing a US $5.25 billion renovation.

Cross section of water saving basins and a laterally filled lock chamber
Many present-day systems are limited in their situation awareness and ability to look ahead and predict events.

<table>
<thead>
<tr>
<th>Industrial-Age Systems</th>
<th>Information-Age Systems</th>
</tr>
</thead>
<tbody>
<tr>
<td>Small, simple, Linear</td>
<td>Large, complex, nonlinear</td>
</tr>
<tr>
<td>Systems of Components</td>
<td>System of systems</td>
</tr>
<tr>
<td>Dominated by hardware</td>
<td>Combinations of hardware, software and communications</td>
</tr>
</tbody>
</table>

- **Decision Making**
- **Knowledge**
- **Information**
- **Data**

Understanding Patterns

Understanding Relations
Early Validation of Systems

Traditional Approach to Airport Design and Test.

Increasing cost to fix errors

- Requirements
  - Preliminary Design
  - Detailed Design
  - Implementation
  - Testing

Phase where design decisions are made.
Phase where design errors are found.

Build little design logic models.
Analyze them thoroughly for potential violation of requirements.
Don’t move forward until design (or parts of design) are provably correct.

Reduced reliance on testing here … to minor issues.

Early detection of errors and "system operation" that is "correct-by-construction"...

Presented to the INCOSE 2008 Symposium
Composition of Models

Environment

Continuous

User-Supplied

Sensors
Constraints
Commands

Inputs, Constraints
System Response

Model of Environment

Reactive System

Control
Components
Subsystems

Discrete

Automated Composition

Distributed System Model

Presented to the INCOSE 2008 Symposium
What is LTSA?

The labeled transition system analyzer (LTSA) is…

verification tool for concurrent systems. It mechanically checks that the specification of a concurrent system satisfies the properties required of its behavior.

It is particularly suitable for high-level modeling and verification of systems dominated by processes that have concurrent behaviors, including interaction with other processes.

LTSA supports specification animation to facilitate interactive exploration of system behavior.
Working with LTSA

LTSA Editor used to write FSP models and check them for deadlocks.

LTSA Draw: Draws the state diagram of the model and animates with the LTSA animator. As we change the state of any object in the LTSA animator, the corresponding state in the diagram also changes.

LTSA Animator: Simulates the behaviour of the model.
Model Formulation

From requirements to architectures:

- Requirements
  - Goals of the system...
  - Scenarios / Use case models....
  - Properties of interest ....

  - Identify main events, actions, and interactions ....
  - Identify and define main processes ...
  - Identify and define properties of interest ...
  - Structure processes into an architecture

- Model
  - Check traces of interest ...
  - Check properties of interest ....
A Simple Example

Two friends talk over coffee ....

```plaintext
// Create a person who: (1) talks and drinks coffee, or
// (2) just waits and then drinks coffee ....

PERSON = ( talk -> drink -> PERSON |
             wait -> drink -> PERSON ).

// Jack and Diane meet ....

||JACK_AND_DIANE_MEET = ( jack:PERSON || diane:PERSON ).

// To learn, conversation needs to be two way ....

TWO_WAY = ( jack.talk -> diane.talk -> TWO_WAY ).

// Conversation should be polite ....

||JACK_AND_DIANE_LEARN = ( JACK_AND_DIANE_MEET || TWO_WAY ) { |
                            jack.talk/diane.wait, diane.talk/jack.wait }.

// End:
```
Jack and Diane Talk!!

Presented to the INCOSE 2008 Symposium
Framework for Model Development

(Scheduler)
East-Bound Monitor
West-Bound Monitor

(Passageway Control)
East-Bound Passageway Control
West-Bound Passageway Control

(Ship Passage)
East-Bound Passage
West-Bound Passage

(Canal System)
Gates
Pumps

(Traffic Demand)
State of Traffic Demand

Request passage
Ordered Ship Requests
Operational Constraints

Initiate Passage
Observations
Action Sequences
Observations
Observations

Constitute
Join queue
Arrival

East-Bound Queue
West-Bound Queue

East-Bound Ships
West-Bound Ships
Lockset Process Model

Lock System Architecture

Component-level Processes:
- Ship 1
- Ship 2
- low:Gate
- low:Pump
- middle:Gate
- high:Pump
- high:Gate

[1..NoShips], (east:SHIPS)
- east:TransitDemand
- east:ShipControl
- east:PassagewayControl

Lockset-level Processes:
- Scheduler

GATESYSTEM and PUMPSYSTEM processes
Component Processes

Basic Processes In Canal Model

Presented to the INCOSE 2008 Symposium
Scheduler Design

东-方向交通

西侧交通

锁系统

调度器

调度器过程模型

当前交通方向

高水位

低水位

向东

向西
Scheduler Design

SCHEDULER = SCHEDULER[0][0][East][Low],
SCHEDULER[we:0..NoShips][ww:0..NoShips][td:TrafficDirection][wl:WaterLevel] = {

  // Register requests to transit lock in east- and west-bound directions.

    when ( we <= NoShips ) [S].east.request -> SCHEDULER[we+1][ ww][td][wl]
  | when ( ww <= NoShips ) [S].west.request -> SCHEDULER[ we][ww+1][td][wl]

  // East-bound assignments to ascend the lock system.

    | when ( we >= 1 && td == East && wl == Low ) [i:S].east.acquire ->
      ascend    -> [i].east.depart -> SCHEDULER[we-1][ ww][West][wl]
    | when ( ww == 0 && we >= 1 && wl == Low ) [i:S].east.acquire ->
      ascend    -> [i].east.depart -> SCHEDULER[we-1][ ww][East][wl]
    | when ( we == 1 && td == East && wl == High ) [i:S].east.acquire ->
      resetlow  -> [i].east.depart -> SCHEDULER[we-1][ ww][West][Low]
    | when ( ww == 0 && we >= 1 && wl == High ) [i:S].east.acquire ->
      resetlow  -> [i].east.depart -> SCHEDULER[we-1][ ww][East][Low]

  // West-bound assignments to descend the lock system.

Lockset Behavior
We would like to design systems that have properties that are guaranteed to be satisfied.

<table>
<thead>
<tr>
<th>Design Properties</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Safety</strong></td>
<td>A safety property asserts that nothing bad happens.</td>
</tr>
<tr>
<td><strong>Liveliness</strong></td>
<td>A liveliness property asserts some good “eventually” happens.</td>
</tr>
<tr>
<td><strong>Progress</strong></td>
<td>A progress property asserts that it is always the case that eventually an “action” will be executed.</td>
</tr>
</tbody>
</table>
In practice the model checking procedure has two steps: (1) unfold the finite state machines into trees, and (2) exhaustively search the tree to see if the property specification is violated.

Example path: ABACCCCB
Model Checking

Subsystem–level Process Hierarchies

Plan View of Networked Processes

Presented to the INCOSE 2008 Symposium
Sequence of Simplified Process Hierarchies

Problem 1. Validate Behavior of Subsystem C

A

B

Property Automata for Subsystem C

Problem 2. Validate Behavior of Subsystem D

E

F

Property Automata for Subsystem D
This is a work in progress -- so what’s next?

- Models for System-Level Operations (i.e., the Full Canal)
- Sensors, Non-Deterministic Models of Travel Demand
- Use of abstraction to simplify complexity of validation computations

How to systematically simplify the validation of system-level concerns?

```
Validation of System-Level Concerns

\[ C^* \quad \text{abstraction} \quad D^* \quad \text{abstraction} \]
```

Presented to the INCOSE 2008 Symposium