Net-centric Cognitive Architecture
Using DEVS Unified Process

Saurabh Mittal, PhD
November 10, 2010
Outline

- DEVS Standard
- Model and Simulator Interoperability
- DEVS Unified Process
- Synthesis of Cognitive Agents
- Net-centric DEVS/ACT-R
- Cognitive System as a SoS component
- Summary
DEVS Standard

- Allows different modeling platforms and model portability, leading to model repositories
- Allows independent development of varied simulation engines
Model & Simulator Interoperability
DEVS/SOA Operation

1. DEVSML SERVICES?
2. LIST OF RESOURCES
3. CONFIG SIMULATION
4. DISTR SIMULATE

INTERNET

CLIENT APPLICATION + ROOT COUPLED MODEL (XML) + MY MODELS

Assign IP addresses to Models

Components | IP Address Assigned
--- | ---
JCASNum1 | 159.135.226.246.08080
UAV | 159.135.226.246.08080
CAOCServer | 159.135.226.246.08080
CAOC | Select IP
JTAC | Select IP
AWACS | Select IP
DEVS Unified Process

- System Requirements as the starting point
- DEVS Hierarchical System specifications

<table>
<thead>
<tr>
<th>Level</th>
<th>Name</th>
<th>What we specify at this level</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>Coupled Systems</td>
<td>System built up by several component systems which are coupled together</td>
</tr>
<tr>
<td>3</td>
<td>I/O System</td>
<td>System with state and state transitions to generate the behavior</td>
</tr>
<tr>
<td>2</td>
<td>I/O Function</td>
<td>Collection of input/output pairs constituting the allowed behavior partitioned according to the initial state the system is in when the input is applied</td>
</tr>
<tr>
<td>1</td>
<td>I/O Behavior</td>
<td>Collection of input/output pairs constituting the allowed behavior of the system from an external Black Box view</td>
</tr>
<tr>
<td>0</td>
<td>I/O Frame</td>
<td>Input and output variables and ports together with allowed values</td>
</tr>
</tbody>
</table>
Extracting Components from ACT-R

DEVS MODEL components

Intentional Module (not identified)
Declarative Module (Temporal/Hippocampus)

Goal Buffer (DLPFC)
Retrieval Buffer (VLPFC)

Productions (Basal Ganglia)
Matching (Striatum)
Selection (Pallidum)
Execution (Thalamus)

Visual Buffer (Parietal)
Manual Buffer (Motor)

Visual Module (Occipital/etc)
Manual Module (Motor/Cerebellum)

net-centric devices / hardware
Formalizing DEVS/ACT-R

**Production** $P = (D, B_p, Z, C, A)$

where,

- $D$ is usual atomic DEVS $D = (X, S, Y, δ_{int}, δ_{ext}, δ_{con}, λ, τ_a)$
- $B_p$ is set of Buffer proxies
- $Z$ is a set of bindings between buffer-slots and local variables
- $C$ is set of Conditions
- $A$ is set of Actions

$P$ will be in DSL / XML, with DEVS $D$ in PSM as an ‘ACT-R primitive’

Some Other ACT-R primitives:

- **Slot** $SL = (Key, Value)$
- **ChunkType** $CT = <SL>$
- **Chunk** $CH = (CT_{type}, <SL>)$
- **SlotCompare** $SLC = (SL, BOOLequals)$
- **Condition** $C = (Pname, Bname, BOOLclearBuf, <SLC>)$
- **Action** $A = (Pname, Bname, CT, BOOLclearBuf, <SL>)$

- **Buffer** $B = (D, CH, DOUBLE_{processingTime})$
- **Module Declarative Mem** $MDM = (D, <CH>)$
- **Selector** $S = (D)$
- **ActrModel** $M = (C, <CT>, <P>)$

where $C$ is coupled $DEVS C = (X, Y, M, EIC, EOC, IC)$

Legend:

- Atomic, Coupled
- $<m> = $ Set of $m$
- $(m) = $ single value
Casting in formal DEVS
Casting in formal DEVS

Conflict Resolution among matching productions based on Utility function

Selects the Winning Production

- Retrieves the matching Chunk based on Activity Equation
Log Trace Matches ACT-R

Log Trace:

- Set Goal
- Conflict Resolution
- Select Winner
- Fire Production

ACT-R Trace:

- time
- Set Goal
- PROCEDURAL
- GOAL
- 0.000
- CONFLICT-RESOLUTION

- PRODUCTION-SELECTED START
- BUFFER-READ-ACTION GOAL
- 0.000
- PRODUCTION-FIRED START

- PRODUCTION-SELECTED BEGIN
- BUFFER-READ-ACTION GOAL
- 0.050
- BUFFER-READ-ACTION REtrieval

DEVS/ACT-R Trace:

- [UNIT 1 Semantic] BEGINING...
- [UNIT 1 Semantic] defining Declarative Memory chunks...
- [UNIT 1 Semantic] MATCHED
- [PrInitRetrieve] REQUEST RETRIEVAL
- [PrInitRetrieve] SET BUFFER: [chunk:p14,\{\{object.bird\}\} {\{attribute.category\} {value.bird}}]
- [PrChainCategory] REQUEST RETRIEVAL
- [PrChainCategory] SET BUFFER: [chunk:p14,\{\{object.bird\}\} {\{attribute.category\} {value.bird}}]
- [PrInitRetrieve] REQUEST RETRIEVAL
- [PrInitRetrieve] SET BUFFER: [chunk:p14,\{\{object.bird\}\} {\{attribute.category\} {value.bird}}]
Cognitive Agents

- **Motivation:**
  - Reuse existing ACT-R models
  - Production networks are flat and not scalable but we need the aggregate behavior of model
  - Production is a rule-action pair. So, is the entire ACT-R model ~ A cognitive capacity?
    - that executes an ‘action’ sequence based on certain ‘conditions’

- Can we compress an ACT-R production network to a single atomic and not loose behavior?

- Solution provided by DEVS closure under coupling that allows us to treat a coupled model an atomic one.

- OR should we start with baseline cognitive capacities and build the Agent’s behavior repertoire?
Cognitive Agent Synthesis

- Agent observes
  - Goal Updates
  - Retrieval Requests
  - Retrieval Outcomes
  - Winner Productions

- Relevant Input Stream

Events are used to extract:
1. Time-ordered Goals
2. Time-ordered Retrievals
3. Time-ordered Expectations
Cognitive Agent Synthesis

- Same behavior and no need of production network
- Develop multi-agent network with a coordinator/manager to manage the behavior repertoire

Capable of being injected in real existing system using DEVS variable structure capability that allows structural change in the system
Recalling DEVS Unified Process
DEVS/Net-centric ACT-R Architecture

Net-centric Infrastructure (Internet, GIG/SOA, NCES, etc.)

DEVS Runtime environment (Net-centric Virtual Machine)

NACT-R Middleware (adapters and transformers)

NACT-R COMPONENT LAYER

Buffer

Module
DEVS/Net-centric ACT-R Architecture

NACT-R WORKBENCH EDITOR

Production
Agents/Units
EXP. Frames

NACT-R Repository

XML-Based PIM

NACT-R RUNTIME ENVIRONMENT (VIRTUAL MACHINE)

Buffer
NACT-R Middleware (adapters and transformers)
DEVS Runtime environment (Net-centric Virtual Machine)

Net-centric Infrastructure (Internet, GIG/SOA, NCES, etc.)
DEVS/Net-centric ACT-R Architecture

NACT-R - R WORKBENCH EDITOR

DEVS/Net-centric ACT-R Architecture

Net-centric Infrastructure (Internet, GIG/SOA, NCES, etc.)

DEVS Runtime environment (Net-centric Virtual Machine)

NACT-R Middleware (adapters and transformers)

XML-Based PIM

NACT-R Repository

NACT-R Registry Lookup

Controller

Buffer

NACT-R Runtime Environment (VIRTUAL MACHINE)

NACT-R COMPONENT LAYER

Modul

Module

Module

Module

Buffer

Visualizer

NACT-R VISUALIZER

MODULE

EXP.

Frames

NACT-R RUNTIME ENVIRONMENT

(VIRTUAL MACHINE)
Cognitive System as a Component

ACT-R can be component based using DEVS but is still not reusable due to the resolution at the Production level. We need capacities!

Extending ACT-R:

1. Extract aggregate behavior from ACT-R models in cognitive agents using DEVS hierarchy of system specifications
2. Develop a repository of such cognitive capacities that can be coupled together towards a Behavior Repertoire
3. Observers can be used to evaluate ‘equivalence’ between two similar ACT-R models
4. Include Top level Goals, Resources and Situatedness per ‘System’ requirements or Mission Thread requirements
5. Compose agent’s ‘capabilities’ insitu with higher level goals, constraints and environment specified/managed using Experimental Frames
7. Keep it modular i.e. promote interfaces of constituent components.
Agent as a ‘System’

Cognitive System

~requirements

Agent | Goals | Environment | Behavior | Resources | Constraints

Capacities

Capacity

Internal | External
Summary

- DEVS is component based framework with PIM and transparent simulators in net-centric domain
- DEVS Unified Process and SES allows multi-formalism M&S
- ACT-R can be made extended with well-defined component interfaces
- DEVS hierarchy of System specifications allows development of components by observing at lower levels of specification
- Cognitive Agents or capacities can be created from existing repository of ACT-R models
- Cognitive agents can now reap the benefits of integration with larger System of Systems or ‘human operator’
- DEVS agent models become live entities that can interface with live web services or hardware (*the model-continuity principle*).
- DEVS is a production system with near zero transition cost from Model to deployable Software.
Questions & Comments
DEVS MSVC Paradigm

- Model-Simulator-View-Controller
## DEVS Hierarchy of System Specification

<table>
<thead>
<tr>
<th>Level</th>
<th>Name</th>
<th>What we specify at this level</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>Coupled Systems</td>
<td>System built up by several component systems which are coupled together</td>
</tr>
<tr>
<td>3</td>
<td>I/O System</td>
<td>System with state and state transitions to generate the behavior</td>
</tr>
<tr>
<td>2</td>
<td>I/O Function</td>
<td>Collection of input/output pairs constituting the allowed behavior partitioned according to the initial state the system is in when the input is applied</td>
</tr>
<tr>
<td>1</td>
<td>I/O Behavior</td>
<td>Collection of input/output pairs constituting the allowed behavior of the system from an external Black Box view</td>
</tr>
<tr>
<td>0</td>
<td>I/O Frame</td>
<td>Input and output variables and ports together with allowed values</td>
</tr>
</tbody>
</table>
DEVS/NACT-R Unit 2

- Visual Buffer and Visual-Location coupled together
  - Act of ‘looking’ (move-attention + encoding)
  - Reducing the number of productions
Unit 2: Demo 1

Attend Letter

```java
@Override
public void loadConditions() {
    Condition c = new Condition("evalGoal", getName(), CONST.GOAL, "read-letters");
    addSlotCondition(c, "state", "start", true);
}
```

```java
@Override
public void loadActions() {
    Action a = new Action("setGoal", getName()).CONST.GOAL;
    addUpdateSlotAction(a, "state", "attend");
    Action a2 = new Action("setVisualLocation", getName()).CONST.VISUAL.true().getChunkType("visual-location");
    addUpdateSlotAction(a2, "attended", "nil");
}
```

Focus Letter

```java
@Override
gatherBufferVariables() {
    addBinding("value", gatherBufferSlot(CONST.VISUAL, "value"));
}
```

```java
public void loadConditions() {
    Condition c = new Condition("evalGoal", getName(), CONST.GOAL, "read-letters");
    addSlotCondition(c, "state", "attend", true);
    Condition c2 = new Condition("evalVisual", getName(), CONST.VISUAL, "Visual");
    addCondition(c2);
}
```

```java
public void loadActions() {
    Action a = new Action("setGoal", getName()).CONST.GOAL;
    addUpdateSlotAction(a, "state", "respond");
    Action a2 = new Action("setImaginal", getName()).CONST.IMAGINAL.true().getChunkType("array");
    addUpdateSlotAction(a2, "letter", getBinding("value"));
}
```

Action Requesting Visual System for a visual location (move attention)

Checking condition that Visual system has completed the ‘encoding’ process and is filled with ‘Object’

Action to put in ‘Imaginal’
Real Agent Behavior

- Set of Time ordered Goals
- Set of Time ordered Retrievals
- Set of Time ordered Expectations

1. Setting a new goal is linked to requesting the next retrieval
2. Once the expected retrieval is success, a new goal is set
Production primitive
Log generated by RealAgent
DEVS/Net-centric ACT-R Architecture

WORKBENCH

NACT-R Repository

Production

Agents/Units

EXP. Frames

XML-Based PIM

Controller

NACT-R Registry Lookup

End-user Client

Visualizer

Server Side OR Virtual Machine

NACT-R COMPONENT LAYER

Buffer

Module

NACT-R Middleware (adapters and transformers)

DEVS Runtime environment (Net-centric Virtual Machine)

Net-centric Infrastructure (Internet, GIG/SOA, NCES, etc.)