Net-centric ACT-R-Based Cognitive Architecture with DEVS Unified Process

Saurabh Mittal
L-3 Communications, Air Force Research Laboratory,
Mesa, AZ 85212 USA
Saurabh.Mittal@L-3com.com

Scott A. Douglass
Air Force Research Laboratory,
Mesa, AZ 85212, USA
Scott.Douglass@mesa.afmc.af.mil

Keywords ACT-R, DUNIP, DSL, SOA, DEVSML, DoDAF

Abstract

Air Force Research Lab (AFRL) research efforts employing cognitive and behavioral modeling are growing in scope and complexity as they work to integrate models into larger distributed systems as cognitive agents, synthetic teammates or human operator surrogates. Efforts to transition cognitive modeling from the laboratory to operational environments are stymied by the isolated nature of current cognitive modeling software tools that are not readily extensible, interoperable or scalable e.g. ACT-R. In this paper, we describe an attempt to build a component-based architecture using the Discrete Event Systems Unified Process (DUNIP) on a distributed net-centric platform that eliminates these impediments. We show how the ACT-R architecture is extensible and can serve as a component in larger net-centric systems of systems frameworks such as Department of Defense Architecture Framework. We will also address the issue of platform independent modeling and how Domain Specific Languages (DSLs) can be integrated within DUNIP. We then demonstrate how developing the architecture and related software infrastructure in DUNIP gives it net-centric capabilities that support large-scale integration.

1. INTRODUCTION

AFRL research efforts employing cognitive and behavioral modeling are growing in scope and complexity. These research efforts are developing complex high-fidelity synthetic entities capable of functioning as teammates and instructors in complex distributed training environments. In today's context these distributed frameworks have taken the shape of net-centric frameworks wherein heterogeneous components communicate each other through defined World Wide Web Consortium (W3C) standards such as XML schema, Web Service Description Language (WSDL), Simple Object Access Protocol (SOAP), HTTP etc. A net-centric system is a distributed system that provided interoperability at the semantic level as opposed to syntactic level in a distributed system. A Service Oriented Architecture (SOA) is one such net-centric framework that incorporates these standards to provide the needed interoperability.

These efforts to transition cognitive modeling from the laboratory to operations settings are pushing currently available cognitive modeling languages and tools to their limits. AFRL scientists wanting to build large-scale models of human cognitive activity are therefore facing core challenges associated with: (1) increasing the scale of their models; and (2) integrating them into simulations and synthetic task environments that are integrated with larger distributed system of systems frameworks such as Department of Defense Architecture Framework (DoDAF) [1]. An AFRL Large-Scale Cognitive Modeling (LSCM[1]) initiative is currently researching and developing a solution to these challenges based on high-level languages for describing cognitive models and simulation frameworks supporting these languages based on the Discrete Event System Specification (DEVS) modeling and simulation formalism [2] on a SOA platform. This paper describes the basic nature of this solution.

The paper starts with a description of The Adaptive Character of Thought-Rational (ACT-R), a unified theory of human cognition and a cognitive modeling and simulation framework [3] in Section 2. ACT-R is an example of a cognitive modeling framework being pushed to its limits by large-scale modeling. Section 3 describes the DEVS Unified Process that provides a foundation for making ACT-R component-based. It provides an overview of DEVS implemented on Service Oriented Architecture (SOA) execution platform and extends the existing DEVS Modeling Language (DEVSML) stack to incorporate Domain Specific Languages (DSLs) that are platform independent and are made executable using various model to DEVS transformations. Section 4 next describes how ACT-R has been decomposed and formalized using the DEVS formalism. Some of the scale and integration benefits of the DEVS formalization of ACT-R components are discussed in this section. Section 5 next contrasts this new architecture with ACT-R's current implementation and presents a range of benefits obtained by the new architecture. Finally, the paper will describe how future work will integrate DEVS/ACT-R into a net-centric infrastructure on SOA. The transition of DEVS/ACT-R to a SOA will further extend AFRL's abilities to scale and integrate large models of cognitive activity in system of systems.

[1] **This research is being funded through AFOSR grant # 10RH05COR**

2. OVERVIEW OF ACT-R

Many cognitive scientists consider cognitive activity to be a product of an essentially permanent open system that interacts with the environment [4]. This perspective has motivated such cognitive scientists to study cognitive architecture, the invariant structural and behavioral system properties underlying cognitive activity that remain constant across time and situation. The Adaptive Character of Thought-Rational (ACT-R) is a theory of human cognition in the form of a cognitive architecture and a cognitive modeling and simulation framework [3]. A key characteristic of the ACT-R cognitive architecture is that it distinguishes between declarative and procedural knowledge.

Declarative knowledge represents factual information that can be retrieved and acted upon. The name of the newest Associate Justice of the United States Supreme Court is an example of a unit of declarative knowledge. Units of declarative knowledge are known as chunks in ACT-R. Declarative knowledge chunks are stored in a long-term associative memory.

Procedural knowledge represents steps of central cognitive processing. Instances of procedural knowledge are known as productions in ACT-R.. Productions represent associations between context-constraints and actions. For example, a production might represent the contextual pre-conditions and subsequent actions required to shift visual attention to a novel aspect of visual context. Productions are stored in a flat procedural memory.

Module	Buffer(s)	Role in Cognition
Audio	aural-location, aural	Localizing and identifying sounds in the environment
Declarative	retrieval	Storing and retrieving information in an associative memory
Goal	goal	Tracking progress towards current goals and intentions
Imaginal	imaginal, imaginal-action	Maintaining internal representations of problems & situations
Motor	manual	Controlling the hands
Procedural	production	Initiating and coordinating the behavior of all other modules
Speech	vocal	Producing speech
Vision	visual-location, visual	Identifying objects in the visual field

Table 1. The modules and buffers making up ACT-R's architectural core.

The base architecture of ACT-R consists of a set of independent modules that each processes a different kind of declarative knowledge. Table 1 lists the central modules in ACT-R. Transient declarative knowledge is stored in module buffers. Only the module maintaining a buffer or the procedural module can modify the contents of that buffer. Cognitive activity arises from interactions between a central production system and these modules. This central cognitive process operates through a series of recognize-decide-act cycles during which contextually appropriate productions are executed. If multiple productions match context, they form a conflict set. Under these circumstances, production utilities based on a type of reinforcement learning are used to determine which production fires.

Overall processing activity in ACT-R consists of a mixture of parallel and serial processing in and across the modules. Parallel activity can occur within each of the modules. For example, retrieval requests processed by the declarative module are based on a parallel search through long-term memory for chunks matching constraints expressed in retrieval requests. Parallel processing can also occur across the modules. For example, the motor module can manipulate the hands while the vision module identifies an object in the visual field.

Architectural constraints impose important limits on parallelism in ACT-R. A constraint that buffers can only hold one chunk of knowledge at a time leads to a serial bottleneck in each of the modules. For example, since the retrieval buffer of the declarative module can only hold one chunk at a time, an ACT-R model can only request/retrieve one declarative fact at a time. Additionally, a constraint that only one production can fire at a time in the procedural module leads to a serial bottleneck in central cognition.

Common Name	Equation
Activation	$A_i = B_i + \sum_{j \in C} W_j S_{ji}$
Base-Level Learning	$B_i = \ln\left(\sum_{k=1}^{n} t_k^{-d}\right)$
Attention Weighting	$W_j = {}^{W}/_{n}$
Associative Strength	$S_{ji} = \ln(prob(i\|j)/prob(i))$
Retrieval Time	$Time = Fe^{-A_i}$
Retrieval Probability	$Prob = 1/(1 + e^{-(A_i - t)/s})$
Utility	$U_i(n) = U_i(n-1) + \propto [R_i(n) - U_i(n-1)]$
Conflict Resolution	$P_i = \frac{e^{U_i/s}}{\sum_j e^{U_j/s}}$

Table 2. Equations describing chunk activation and production utility in ACT-R.

One of the most empirically verified aspects of the ACT-R cognitive architecture is its declarative module. Human behavior is as flexible as it is because we know lots of facts about objects, situations and effective actions. This vast repository of knowledge allows us to, with seemingly little effort, use what we know to craft contextually appropriate and effective actions in diverse circumstances. We know a lot and we can quickly cull through all that we know and retrieve and apply the right knowledge given our circumstances. The heart of ACT-R's memory system is a

set of equations describing how sub-symbolic calculations based on the frequency and recency of chunk utilization and context priming effects underlie these critical properties of human memory.

Collectively, the equations in Table 2 precisely explain how changes in activation and utility over time allow knowledge and behavior to fit the information structure of the environment (see [3][5] for detailed descriptions).

The behavioral semantics of ACT-R are based on well implemented discrete event simulation system. This discrete event system realizes the recognize-decide-act cycle producing cognitive activity in ACT-R by managing information sharing though buffers and coordinating module activities. Unfortunately, the technically isolated nature of ACT-R's discrete event simulator isolates cognitive models from the methods, libraries and tools utilized by the larger systems engineering community. This isolation leads to significant interoperability challenges when a modeler wants to integrate a non-trivial cognitive model into a larger simulation or synthetic task environment.

3. DEVS UNIFIED PROCESS

The DEVS Unified Process [6] is an overarching process employing the DEVS framework in a net-centric domain that links requirements with the model under study. It is based on elements such as DEVS formalism, the emerging DEVS standard for M&S interoperability, platform independent modeling, transparent simulators in a net-centric domain and model continuity principles. The following sub-sections provide a brief overview.

3.1. DEVS Formalism

Discrete Event System Specification (DEVS) [2] is a formalism which provides a means of specifying the components of a system in a discrete event simulation. The DEVS formalism consists of the model, the simulator and the experimental frame as shown in Figure 1. The Model component represents an abstraction of the source system using the modeling relation. The simulator component executes the model in a computational environment and interfaces with the model using the simulation relation or the DEVS simulation protocol in the present case. The Experimental Frame facilitates the study of the source system by integrating design and analysis requirements into specific frames that support analyses of various situations the source system is subjected to.

While historically models have been closely linked to the platform (such as Java, C, C++) in which the simulator was written, recent developments in platform independent modeling and transparent simulators [7] have allowed the development of both the models and simulators in disparate platform. Current efforts are focusing on a standardization process [8-10] wherein the simulation relation can be standardized for further interoperability

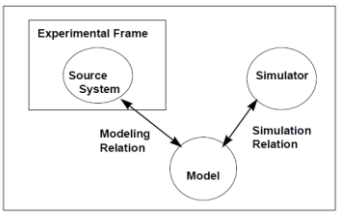

Figure 1. DEVS Framework elements

In DEVS formalism, one must specify Basic Models and how these models are connected together. These basic models are called Atomic Models (Figure 2) and larger models which are obtained by connecting these atomic blocks are called Coupled Models (Figure 2). Each of these atomic models has inports (to receive external events), outports (to send events), a set of state variables, an internal transition function (to specify state transitions with timeouts), an external transition function (to specify state transitions on receiving external event), a confluent transition function (to specify in explicit terms whether to execute internal transition and/or external transition on the event of receiving external input when making internal transition) and a time advance function. The models specification uses or discards the message in the event to compute, deliver an output message on the outport, and make a state transition.

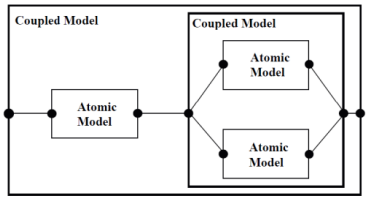

Figure 2. Atomic and Coupled models

A DEVS-coupled model designates how atomic models are coupled together and how they interact with each other to form a complex model. The coupled model can be employed as a component in a larger coupled model and can construct complex models in a hierarchical way. The specification provides components and coupling information. A Java based implementation (DEVSJAVA [11]) can be used to implement these atomic or coupled models.

3.2. DEVS/SOA

The DEVS/SOA framework [12] is analogous to other DEVS distributed simulation frameworks like DEVS/HLA, DEVS/RMI and DEVS/CORBA [13-17]. The distinguishing mark of DEVS/SOA is that it uses SOA as the network communication platform and XML as the middleware and

thus acts as a basis of interoperability using XML [18]. Furthermore, it uses web-services as the underlying technology to implement the DEVS simulation protocol. The complete setup requires one or more servers that are capable of running DEVS Simulation Service, as shown in Figure 3. The capability to run the simulation service is provided by the server side design of DEVS Simulation protocol supported by the latest DEVSJAVA Version 3.1 [11].

Once a DEVS model package is developed, the next step is simulation as illustrated in Figure 3. The DEVS/SOA client (Figure 3) takes the DEVS models package and through the dedicated servers hosting DEVS simulation services, it performs the following operations:

1. Using the client application locate DEVS simulation servers
2. Select the Simulation resources
3. Compose your root coupled model
4. Perform Simulation on SOA
    a. Upload the models to specific IP locations i.e. partitioning
    b. Run-time compile at respective sites
    c. Simulate the coupled-model
5. Receive the simulation output at clients end

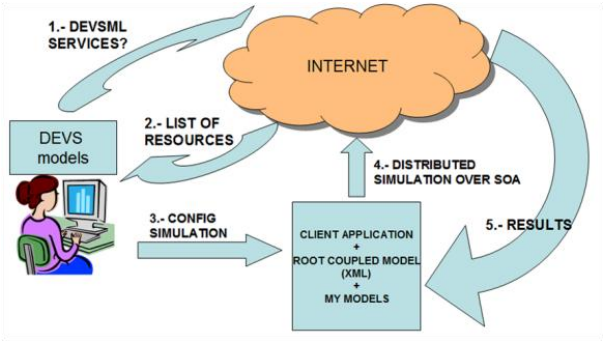

Figure 3. DEVS/SOA Execution flow

3.3. Model and Simulator Interoperability using DEVSML

The earlier version of DEVSML stack [7] developed models in Java and in platform independent DEVS Modeling language that used XML as a means for transformation. The model semantics were bound together by XML. The latest version of the DEVSML, the language, is based on EBNF grammar and is supported by DEVS middleware API. The middleware is based on DEVS M&S Standards compliant (under evaluation) API and interfaces with a net-centric DEVS simulation platform such as a service oriented architecture (SOA) that offers platform transparency. With the maturation of technologies like Xtext [20] and Xpand [21] we have now extended the concept of XML-based DEVSML to a much broader scope wherein Domain Specific Languages (DSL) can continue to be expressed in all their richness in a platform independent manner that is devoid of any DEVS and programming language constructs (Figure 4). The key idea being domain specialists need not delve in the DEVS world to reap the benefits of DEVS framework.

The DEVSML 2.0 stack in Figure 4 adds three transformations at the top layer:

1. Model-to-Model (M2M)
2. Model-to-DEVSML (M2DEVSML)
3. Model-to-DEVS (M2DEVS)

The end-user as indicated in Figure 4 will develop models in their own DSL and the DEVS expert will help develop the M2M and M2DEVSML transformation to give a DEVS backend to the DSL models [19]. The M2DEVSML transformation will give us the DEVS models in a platform independent manner that is now open for collaborative development per DEVS Unified Process (Section 3.4). While M2DEVSML transformation delivers an intermediate DEVS DSL (the DEVSML DSL), the M2DEVS transformation directly anchors any DSL to platform specific DEVS. There are many DEVS DSLs that implement a subset of rigorous DEVS formalism. One example of DEVS DSL is XML-based Finite Deterministic DEVS (XFFDEVS) [21]. DEVSSpecML [22] built on BNF grammar is another example of DEVS DSL.

Below the DEVS Modeling Language layer, there is a DEVS middleware that translates the semantics into syntactic operations and hand it over to the DEVS/SOA layer for simulation purposes. This stack also allows certain DSL at the end user layer who can directly communicate with DEVS middleware through an API. Recent developments provide evidence that a hybrid simulation of DEVS and non-DEVS models can be executed using DEVS/SOA.

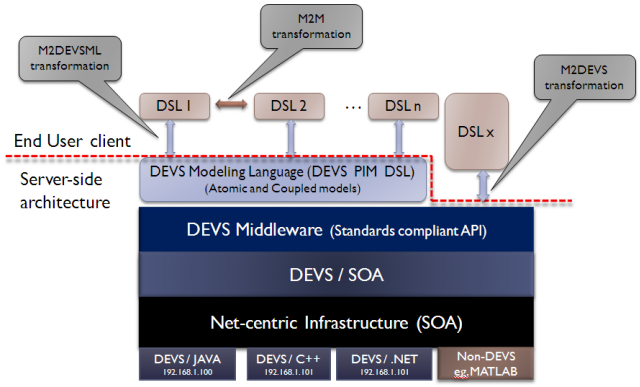

Figure 4. DEVSML stack employing M2M and M2DEVS transformations for Model and simulator transparency

3.4. Complete process

This section describes the bifurcated Model-Continuity process [22] and how various elements like automated DEVS model generation, automated test-model generation (and net-centric simulation over SOA are put together in the process, resulting in DEVS Unified Process (DUNIP) [6][23]. The design of simulation-test framework occurs in parallel with the simulation-model of the system under design. The DUNIP process consists of the following elements:

1. Automated DEVS Model Generation from various requirement specification formats
2. Collaborative model development using DEVS Modeling Language (DEVSML)
3. Automated Generation of Test-suite from DEVS simulation model
4. Net-centric execution of model as well as test-suite over SOA

Considerable amount of effort has been spent in analyzing various forms of requirement specifications, viz, state-based, Natural Language based, UML-based, Rule-based, BPMN/BPEL-based and DoDAF-based, and the automated processes which each one should employ to deliver DEVS hierarchical models and DEVS state machines [6][24]. Simulation execution today is more than just model execution on a single machine. With Grid applications and collaborative computing the norm in industry as well as in scientific community, a net-centric platform using XML as middleware results in an infrastructure that supports distributed collaboration and model reuse. The infrastructure provides for a platform-free specification language DEVS Modeling Language (DEVSML) [7] and its net-centric execution using Service-Oriented Architecture called DEVS/SOA [12,26]. Both the DEVSML and DEVS/SOA provide novel approaches to integrate, collaborate and remotely execute models on SOA. This infrastructure supports automated procedures for test-case generation leading to test models.

Using XML as the system specifications in rule-based format, a tool known as Automated Test Case Generator (ATC-Gen) was developed which facilitated the automated development of test models [22][25]. DUNIP (Figure 5) can be summarized as the sequence of the following steps:

1. Develop the requirement specifications in one of the chosen formats such as BPMN, DoDAF, Natural Language Processing (NLP) based, UML-based, DSL or simply DEVS-based for those who understand the DEVS formalism.
2. Using the DEVS-based automated model generation process as per the M2M transformation or M2DEVS transformation as outlined in Section 3.3, generate the DEVS atomic and coupled models from the requirement specifications
3. The generated models which are Platform Independent Models (PIMs) in XML/DSL can participate in collaborative development using DEVSML middleware.
4. From step 2, either the coupled model can be simulated using DEVS/SOA or a test-suite can be generated based on the DEVS models.
5. The simulation can be executed on an isolated machine or in distributed manner (using SOA middleware if the focus is net-centric execution). The simulation can be executed in real-time, virtual-time, logical-time or wall-clock time [2].
6. The test-suite generated from DEVS models can be executed in the same manner as laid out in Step 5.

The results from Step 5 and Step 6 can be compared for verification and validation process using the Experimental Frames that are designed from the requirements in Step 1.

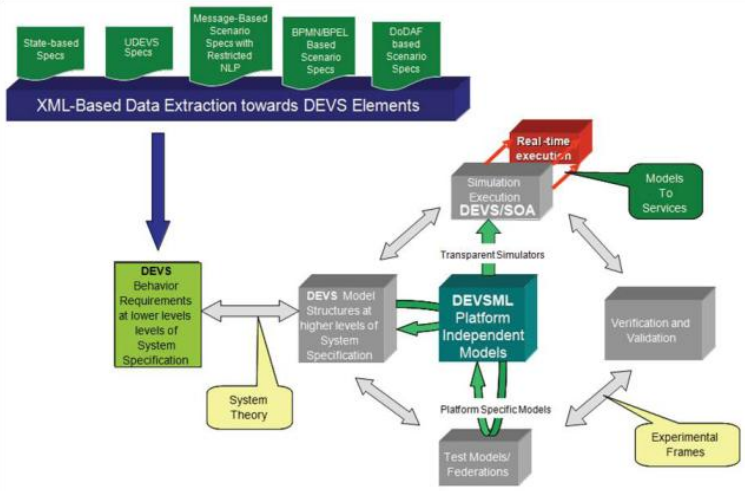

Figure 5. DEVS Unified Process

This section has described the overarching DEVS Unified Process that will serve as the enabling framework for implementing a cognitive architecture. The subsequent sections will focus on the cognitive architecture aspects and how they are formalized into DEVS.

4. FORMALIZATION OF ACT-R COMPONENTS

We shall now see how a process-oriented architecture is componentized and ultimately formalized in the discrete event specifications. The first step is to identify a clear separation of concerns between various elements of ACT-R architecture. Once such separation is made, the next step is to identify the behavior of interacting components such that the coupled behavior is identical to the original process-oriented system. In a process oriented system, the elements communicate each other by way of function calls and arguments. As a result, we have to identify such method calls and exchanged messages towards an 'interface' in component-based methodology. After extraction of such interfaces, the behavior of each of these elements is described using a state machine or a DEVS atomic to be precise. Finally, coupled model of ACT-R architecture is created by specifying couplings per interfaces and hierarchical construction. The following subsections provide more in-depth analysis.

4.1. Separation of concerns in ACT-R architecture

As laid out in Section 2, ACT-R architecture has the following system level functions:

I. Productions evaluate the encoded conditions represented in buffer data and execute actions which impact buffer data
II. Of the set of productions that match context conditions, a single winner is selected (based on the Utility equation) and allowed to execute its actions.
III. Buffers are interfaced with corresponding modules and are the only means by which *productions* can access modules.
IV. Declarative memory is queried via retrieval requests. The Activity equation is used to select a single chunk from the set meeting the constraints of the request.
V. The productions evaluate their condition every simulation cycle to learn about changed buffer data.
VI. Each Module interfaces with a corresponding Buffer that interfaces with productions.
VII. Goal buffer interacts with productions to keep the current goal into focus as updated by last winning production

Consequently, these system level functions are now assigned to the components so that behavior can be accounted for. From Table 3, it is clear that the *component* Production replicates the base behavior of ACT-R productions as other components of the architecture and in addition performs the function of evaluating the condition whenever any buffer is updated. There is also a conceptualization of a new component, called Selector that performs the job of selecting a winning production based on Utility equation.

Function	Component	Feature Existing/Modified/New
I	Production	Existing
II	Selector	New
III	Buffer	Existing
IV	Declarative Memory Module	Existing
V	Production	Modified
VI	Buffer-module relationship	Existing
VIII	Goal Buffer	Existing

Table 3. Function to component mapping

4.2. Casting in DEVS Formalism

After understanding the separation of concerns, various architectural elements of the ACT-R theory were mapped to the corresponding primitives in the DEVS domain as shown in Table 4.

Component/ Primitive	Formal Description
Production	P = (D, Bp, Z, C, A) where, -D is usual atomic DEVS $D=(X,S,Y,d_{int}, d_{ext}, d_{con}, l, ta)$ -Bp 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
Slot	SL = (Key, Value)
ChunkType	CT = <SL> [set of slots]
Chunk	CH = (CT, <SL>)
Buffer	$B = (D, CH, Double_{processingTime})$
SlotCompare	$SLC = (SL, Bool_{equals})$
Condition	$C = (P_{name}, B_{name}, Bool_{clearBuf}, <SLC>)$
Action	$A = (P_{name}, B_{name}, CT, Bool_{clearBuf}, <SL>)$
Declarative Memory	MDM = (D, <CH>)
Selector	S = (D)
Module	MO = (D)
ModuleSystem	MS = (B, MO)
Actr Model	M = (C, <CT>, <P>, <B>) where, -C is usual DEVS coupled C = (X, Y, M, EIC, EOC, IC)

Table 4. Formal description of DEVS/ACT-R primitives

Figure 6 shows the DEVS/ACT-R System Entity Structure (SES) [2] diagram. It shows which entities are formalized as DEVS components while others are supporting data structures. The DEVS/ACT-R System at the top is an entity that is made of Components, Models and DEVS formalism. The entity Components is made of many such Component, which can be Chunks, ChunkTypes, Selector, Buffers, Productions and Module Systems entities. A ChunkType is

made of Slots and each Slot is made of key value pair. A Chunk is made of a Chunk Type. A Buffer is made of Chunk and can be a Vision, Goal, Imaginal or Retrieval buffer. A Buffer is also an atomic, like the Selector. A Module System component can be a coupled and is made of a Module and a Buffer. A Module can be specialized into a Vision module, Manual module or a Declarative Memory module. These interfaces of these modules can very well be formalized so as to make them visible using a WSDL in a net-centric deployment. A Production is an atomic component and is made up of Bindings, Conditions and Actions. A DEVS/ACT-R System Model is a coupled entity and is made of ChunkTypes, Productions, and Buffers. And finally, DEVS is made of Model and Simulator as distinct entities. Model can be a coupled or atomic where coupled is made of Couplings and Components. Simulator can be implemented on a local machine or a net-centric platform such as DEVS/SOA.

The component Production is a central piece of the architecture, and it is worth looking at its design in further detail. In Figure 7, we see a DSL that formalizes the information needed to specify a production i.e. a set of *bindings*, a set of *conditions* and a set of *actions*. This DSL closely matches the original ACT-R production specification which can be viewed at [3].

Figure 8 shows the behavior represented in DEVS state machine. The solid lines show external event transitions that occur on the advent of event (prefix with ? and shown in blue). The dotted lines show internal transitions. The generated output at specific states is shown in green (prefix with !). The DSL was developed using Xtext [20] that builds a fully functional Eclipse textual editor with syntax checking and code completion. Underlying that editor is an Extended Bachus Naur Form (EBNF) grammar. Using the code generation tool Xpand [21], the grammar is then transformed to a platform specific model (PSM) in Java. The PSM implementation translates the Bp, Z, C, A in DSL and inherits the DEVS atomic Production that is specified as per Figure 8. Finally, such productions are coupled together to have a running DEVS model (Figure 9). The execution of the model, example and comparison of the log traces were demonstrated at [27].

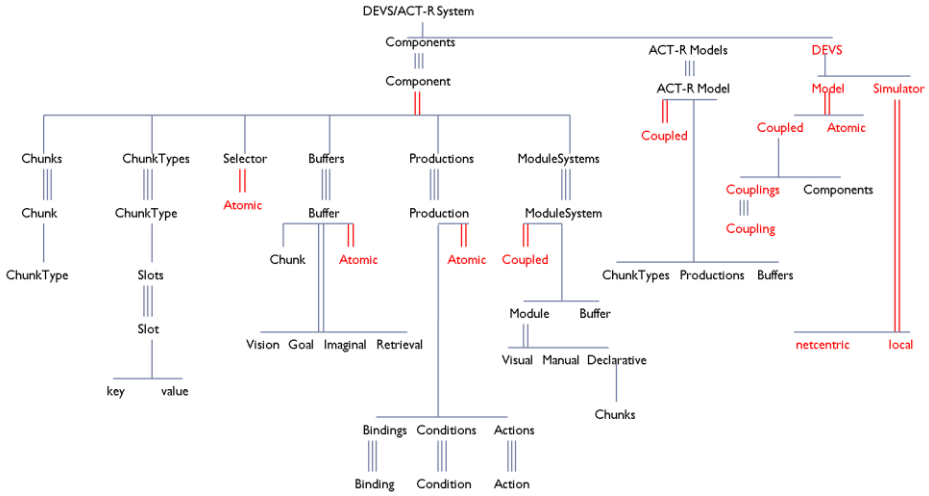

Figure 6. System Entity Structure for DEVS/ACT-R

```
production PrIncrement{
    binding =goal countfrom {
            count to num1
        }
    binding  =retrieval countorder{
        second to num2
    }

    condition =goal>ISA countfrom{
        end != num1
    }
    condition =retrieval>ISA countorder{
        first == num1
    }
    action =goal>{
        count = num2
    }
    action +retrieval>ISA countorder{
        first = num2
    }
}
```

Figure 7. DSL for Production

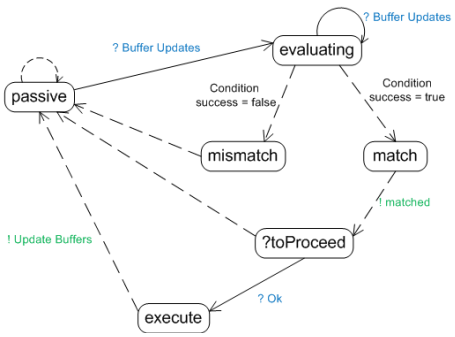

Figure 8. DEVS Behavior for Production

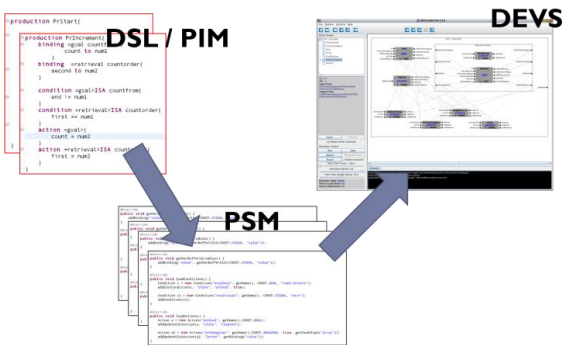

Figure 9. From DSL to DEVS Execution

5. DISCUSSION

We have shown how a procedural system like ACT-R can be componentized using DEVS component-based modeling and simulation framework. In this section we will contrast the developed DEVS/ACT-R framework with the original one and evaluate how this new framework provides additional benefits.

5.1. Contrasting the original ACT-R simulator with DEVS/ACT-R Simulator

The original ACT-R architecture is distributed as an integrated piece of software that runs in Common Lisp. The end user is required to program in Lisp and consequently, must overcome two learning curves before becoming proficient in ACT-R. In addition, there are many aspects where the newly developed DEVS/ACT-R shows advantages over the existing ACT-R Version 6.0 software. Table 5 lists some of the comparisons. While it is easy to see that the component-based architecture of DEVS/ACT-R has inherent benefits such as extensibility, scalability, plugin framework, etc., the major benefit of this effort is the development of simulator in Discrete Event formalism as opposed to a programmatic event scheduler that looks at a queue to execute the next event. By working towards the identification of communication interfaces of these modules, we came about a position in which we can send message explicitly to the intended recipient. This had a major effect on the simulator efficiency. We know that productions interface with Buffers which in turn interface with Modules. Productions are responsive to the updates in the buffers. In the original ACT-R architecture, all the productions in the model evaluate their conditions whenever any buffer update occurs. Clearly, not all the productions need to observe all the buffers. For example, a production that is just listening to Visual buffer and is not interested about what happens in the Retrieval buffer need not evaluate its condition. With DEVS/ACT-R all such unnecessary processing is avoided as each production is now explicitly coupled to the desired buffer. Hence, whenever a buffer update is made as a result of action by a *production* or through processing by the corresponding module, the intended *productions* are sent the buffer updates. On receiving these selective updates, the intended productions evaluate their conditions and the process repeats over. There is just the needed processing in discrete event simulator based on DEVS.

One other major gain from the component based architecture is separation of concerns in components like Selector (executes Utility equation) and the Declarative Memory module (executes Activity equation). As there is no way for parallelism in the underlying simulator in original ACT-R, let alone separation of concerns, the performance takes a hit when the number of productions is huge. The *conditions* inside each production have no complex mathematics but only key-value comparisons which require very less resources. The bulk of computation is happening in these two components, which now in DEVS/ACT-R can be placed on high number crunching machines with may be parallel architectures. Further, an altogether different Declarative Memory module can be plugged in as a replacement or for evaluation purposes.

S.No.	Aspect	ACT-R	DEVS/ACT-R
1	Modular	Yes	Yes
2.	Component-based	Yes, but in Common Lisp only.	Yes. Platform independent
3.	Language independent	No. Only works with Lisp.	Yes. Model is platform independent and can be DSL semantically anchored in DEVSML or DEVS.
4.	Simulator efficiency	Very limited. The event scheduling loop is the only place where optimizations can be made.	Yes. Discrete Event simulator that advances time based on events explicitly communicating to imminent components.
5	Module Plugin framework	Yes. Additional modules can be developed in Lisp.	Yes. Capable of adding/removing modules based on defined interfaces
6	Communication between modules	No explicit message passing. Method-call is the chosen mode of action	Yes. Component updates are communicated precisely to the intended components by way of explicit coupling
7.	Scalability	No. The underlying event scheduler is the bottleneck.	Yes. The simulator has been proven in distributed environment. Models are separated from simulators.
8.	Extensibility	No. It is cumbersome to make it as a component in larger System of System.	Yes. With DEVS structure in the underlying architecture, it can very well serve as a black-box in larger System of systems
9.	Net-centric	No. There are no standardized interfaces	Yes. All the message passing is in XML

Table 5. Comparison of ACT-R and DEVS/ACT-R capabilities

5.2. Verification and Validation of existing ACT-R models

In general M&S, verification attempts to establish the correctness of morphism between the simulator and the model. Due to a lack of verification support tools, cognitive modelers, rarely if ever, determine the correctness of the mapping between their model and the underlying software systems responsible for simulation.

In general M&S, validity is a property of the relationships between a model, a system being studied, and an experimental frame [2]. Replicative validity relates model to system at the I/O behavior level. Cognitive modelers assess the replicative validity of their models by comparing their I/O behavior to human performance data.

They currently lack tools that automate this process. Predictive validity combines replicative validity and an ability to correspond to unseen system behavior. Cognitive modelers sometimes demonstrate a compelling "modeling relation" using model/data comparisons that demonstrate predictive validity across multiple tasks. Structural validity relates model and system at the state transition and coupled levels—a structurally valid model replicates/predicts system behavior AND mimics the "state-by-state and component-by-component" mechanisms underlying the system. While cognitive modelers would welcome the opportunity to explore and describe the structural validity of their models, they currently lack the methods and tools to do so.

As the DEVS/ACT-R framework matures, it will be possible for cognitive modelers using it to unify parts of their modeling life-cycle with the life-cycle seen in general M&S. Cognitive modelers will benefit considerably from the verification and validation capabilities available in the DEVS Unified Process after such unification.

Summary

We have contrasted DEVS/ACT-R with the standard ACT-R and have delineated major benefits of the component-based architecture. We also have shown how such DEVS based component architecture is inherently subjected to hierarchy of system specification [2] leading to observations at various levels. Such capability to monitor at a specific level of resolution is instrumental in the development of verification and validation frameworks which we intend to pursue further in the near future. Next section will make it a component in larger System of Systems.

6. NET-CENTRIC ACT-R AND SYSTEM OF SYSTEMS

Organizations such as AFRL, promote Technology Readiness Levels (TRLs) as a means of evaluating the readiness of technologies to be incorporated in a weapon or Military System [28]. However, we often fail to account for the critical human element [29]. Therefore, additional methodologies are needed that would capture this human element as the integral part of systems engineering and technology transitions. The level of technology that US provide to its armed forces is unparalleled. However, the technology is as good as its usage by the user in the real world and human operator is a critical piece in this puzzle. While the technology is ready, the failure to make the human ready brings a huge gap in what the 'system' is supposed to do and what actually happens in real world.

While the Modeling and Simulation community can help simulate these systems to a good degree of fidelity, the absence of human operator model that is cognitively plausible presents with results that are difficult to map in the real world. The present work with componentizing ACT-R towards a systems component is a work in this direction wherein a cognitively plausible agent can represent a human operator at the required time scales. To address this critical component in larger DoD frameworks such as Department of Defense Architecture Framework (DoDAF)[1] or Ministry of Defense Architecture Framework (MoDAF, UK) [30], a Human View is proposed that addresses this critical need [31]. These architecture frameworks produce common Systems Engineering (SE) approaches to development, presentation, and integration of current and future system of systems. Newer architectures like DoDAF V2.0 address Net-centric, System of Systems and System/Services concepts.

Human View is to enable effective Human System Integration (HSI) processes within the design of these complex, large-scale, socio-technical systems. The North Atlantic Treaty Organization (NATO) Human View Handbook [32] facilitate design decisions by identifying relevant elements emphasizing the explicit need of merging seamlessly and efficiently with sound systems engineering practice. It establishes a logical and systematic framework for HSI studies and makes explicit human, crew, and team socio-behavioral processes as integral to total systems performance. Although HSI is a fundamental component of a total systems approach, the successful integration of HSI into systems engineering and acquisition life cycles continues to be a challenge [33].

In our earlier work, we have already shown how DoDAF-DEVS mapping can help develop an executable architecture with formal rigor and methodology [34-35]. We have also shown how any DEVS component can be made net-centric or any web service description (WSDL) can be made a DEVS component [36] and interoperate with an existing DEVS net-centric system as fully deployed software [37]. Making ACT-R now DEVS enabled allows us to take the ACT-R theoretical framework to larger system of system in which a network node can host the entire DEVS/ACT-R system. Each instance of the proposed DEVS/ACT-R system becomes a cognitively plausible agent and can be coupled together towards formation of

crew, teams etc. in a hierarchical manner with shared knowledge structures and environment. The component based framework allows consolidation of components at different levels of hierarchy. The next subsection presents the overall net-centric architecture of DEVS/ACT-R.

6.1. Netcentric ACT-R (NACT-R)

Figure 9 shows the architecture for Net-centric DEVS/ACT-R. For better usability and acceptance by the end user, the formal DEVS/ACT-R is structured in a client-server paradigm. The server side rests on net-centric infrastructure such as SOA or it may be a virtual machine (eg. Java Virtual Machine) that runs locally on client's machine for experimental use. On top of it is the DEVS run-time environment that encapsulates the DEVS middleware and DEVS/SOA layers in DEVSML Stack (Section 3.3). Next is the NACT-R middleware layer that interfaces with with various ACT-R components. The client side of the NACT-R architecture is the end-user that is provided with a Workbench that can be used to develop Productions, Agents and Experimental Frames. The workbench allows accessing the NACT-R repository which may be local or on network. It also provides a Registry that makes available the theoretical components of ACT-R architecture which the end-user can reference in his models. There is a Visualizer to view various facets of the simulation model and finally, the Controller to perform the simulation or dynamically control the running simulation [35].

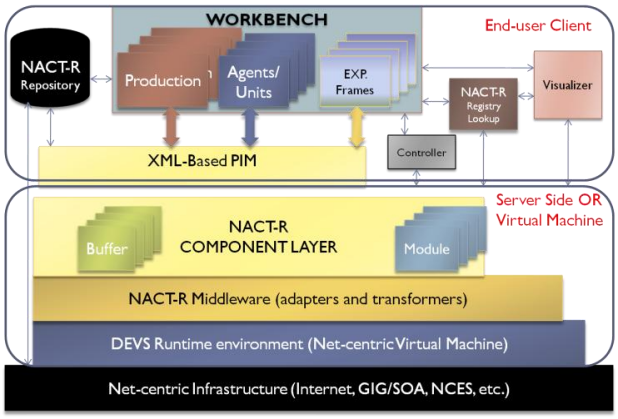

Figure 9. Net-centric DEVS/ACT-R

While the vision of this work is towards the development of human operator in larger system of systems, the NACT-R in its present architecture is currently confined to initial studies and analysis of the entire approach. In our future work we will augment the architecture with more components as we move towards the SoS integration.

7. CONCLUSIONS AND FUTURE WORK

This paper begins with the premise that efforts to develop large-scale cognitive models and integrate them into software-intensive distributed synthetic task environments are pushing cognitive modeling frameworks to their limits. To give the reader a sense for the current state-of-the-art in cognitive modeling, the ACT-R cognitive architecture and simulation framework were described. This paper then described how ACT-R has been decomposed and re-implemented in the DEVS formalism in order to extend its limits. This process of formalizing ACT-R in DEVS followed the DEVS Unified Process and led to DEVS/ACT-R. DEVS/ACT-R represents a full circle resulting in the development of new ACT-R implementation that transforms models specified in platform independent DSLs to a platform specific execution framework using DEVS. We also extended the earlier DEVSML stack with DSL's and suggested M2M, M2DEVSML and M2DEVS transformations as the preferred way to achieve model interoperability and larger integration of modeling framework with an underlying DEVS distributed simulation net-centric infrastructure. We illustrated this concept by developing a DSL for ACT-R and executing it on DEVS platform. In addition, some of the immediate scale and integration benefits of making ACT-R component-based in DEVS were discussed.

Current efforts to make DEVS/ACT-R net-centric will allow cognitive modelers to evaluate and field their models through Service Oriented Architectures (SOA) and other net-centric infrastructures. This paper lays the foundation and suggests how future work will amplify the benefits of componentizing ACT-R in DEVS. By unifying ACT-R modeling practices into the DEVS Unified Process, future versions of DEVS/ACT-R will facilitate the verification and validation of ACT-R models within the DEVS hierarchy of system specification. Refinements and extensions to net-centric DEVS/ACT-R will enable cognitive scientists to "black-box" models of cognitive activity into larger system of systems. Such capabilities will help AFRL addresses the Human Factor Integration aspect in framework like DoDAF and MoDAF.

The work described in this paper illustrates how methods and processes common in general M&S can be exploited by other fields—in this case cognitive modeling. The immediate contribution of this work is DEVS/ACT-R, an architecture that is allowing AFRL to begin integrating cognitive models into net-centric infrastructures such as a Service Oriented Architecture (SOA). The transition of DEVS/ACT-R to a SOA has the potential to literally revolutionize how AFRL develops and fields large-scale cognitive models.

References

[1] Department of Defense. (2007, April 23). *DoD Architecture Framework Version 1.5, Volume I: Definitions and Guidelines.*

Retrieved September 1, 2009, from Acquisition Community Connection: http://www.defenselink.mil/cionii/docs/DoDAF_Volume_I.pdf

[2] Zeigler, BP, Kim, TG and Praehofer, H, "Theory of Modeling and Simulation" New York, NY, Academic Press, 2000

[3] Anderson, J. R., Bothell, D., Byrne, M. D., Douglass, S. A., Lebiere, C., & Qin, Y. (2004). An integrated theory of the mind. *Psychological Review , 111* (4), 1036-1060.

[4] Wilson, M. (2002). Six views of embodied cognition. *Psychonomic Bulletin & Review , 9* (4), 625-636.

[5] Anderson, J. R. (2007). *How can the mind exist in the physical universe?* Oxford: Oxford University Press.

[6] Mittal, S, "DEVS Unied Process for Integrated Development and Testing of Service Oriented Architectures", Ph. D. Dissertation, University of Arizona, 2007 accessible at http://acims.arizona.edu/PUBLICATIONS/PDF/Thesis_Mittal.pdf

[7] Mittal, S, Martin, JLR, Zeigler, BP, "DEVSML: Automating DEVS Simulation over SOA using Transparent Simulators", DEVS Syposium, 2007

[8] Wainer, G, Al-Zoubi, K, Dalle, O, Hill, D, Mittal, S, Martin, JLR, Sarjoughian, H, Touraille, L, Traore, M, Zeigler, BP, "DEVS Standardization: Ideas, Trends and Future", chapter in "Discrete Event Modeling and Simulation: Theory and Applications", 2010, CRC Press.

[9] Wainer, G, Al-Zoubi, K, Dalle, O, Hill, D, Mittal, S, Martin, JLR, Sarjoughian, H, Touraille, L, Traore, M, Zeigler, BP, "Standardizing DEVS Model Representation", chapter in "Discrete Event Modeling and Simulation: Theory and Applications", 2010, CRC Press.

[10] Wainer, G, Al-Zoubi, K, Dalle, O, Hill, D, Mittal, S, Martin, JLR, Sarjoughian, H, Touraille, L, Traore, M, Zeigler, BP, "Standardizing DEVS Simulation Middleware", chapter in "Discrete Event Modeling and Simulation: Theory and Applications", 2010, CRC Press

[11] DEVSJAVA: http://www.acims.arizona.edu/SOFTWARE/devsjava_licensed/CBMSManuscript.zip

[12] Mittal, S, Martin, JLR, Zeigler, BP, "DEVS/SOA: A Cross-Platform Framework for Net-Centric Modeling and Simulation in DEVS Unied Process", SIMULATION: Transactions of SCS, Vol. 85, No. 7, pp. 19-450, 2009

[13] Fujimoto, RM, "Parallel and Distribution Simulation Systems", Wiley, 1999

[14] Seo, C, Park, S, Kim, B, Cheon, S, Zeigler, BP, "Implementation of Distributed High-performance DEVS Simulation Framework in the Grid Computing Environment", Advanced Simulation Technologies conference (ASTC), Arlington, VA, 2004

[15] Cheon, S, Seo, S, Park, S, Zeigler, BP, "Design and Implementation of Distributed DEVS Simulation in a Peer to Peer Networked System", Advanced Simulation Technologies Conference, Arlington, VA, 2004

[16] Kim, K, Kang, W, "CORBA-Based Multi-threaded Distributed Simulation of Hierarchical DEVS Models: Transforming Model Structure into a Non-hierarchical One", International Conference on Computational Science and Its Applications, Italy 2004

[17] Zhang, M, Zeigler, BP, Hammonds, P, "DEVS/RMI-An Auto-Adaptive and Recongurable Distributed Simulation Environment for Engineering Studies", ITEA Journal, July 2005

[18] Mittal, S, Zeigler, BP, Martin, JLR, "Implementation of Formal Standard for Interoperability in M&S/System of Systems Integration with DEVS/SOA", International Command and Control C2 Journal, Special Issue: Modeling and Simulation in Support of Network-Centric Approaches and Capabilities, Vol. 3, No. 1, 2009

[19] Douglass, S., Mittal, S., "Using Doman-Specific Languages to Improve the Scale and Integration of Cognitive Models", Behavior Representation in Modeling and Simulation, Utah, March 2011 (submitted)

[20] Xtext Language Development Framework accesible at: http://www.eclipse.org/Xtext/

[21] Xpand Model Transformation Framework accessible at: http://www.eclipse.org/modeling/m2t/?project=xpand

[22] Zeigler, BP, Fulton, D, Hammonds P, and Nutaro, J, "Framework for M&SBased System Development and Testing In a Net-Centric Environment", ITEA Journal of Test and Evaluation, Vol. 26, No. 3, pp. 21-34, 2005

[23] DUNIP: A Prototype Demonstration http://duniptechnologies.com/training/demos/dunip.avi

[24] Martin, JLR, Mittal, S, Zeigler, BP, Manuel, J, "From UML Statecharts to DEVS State Machines using XML", IEEE/ACM conference on Multi-paradigm Modeling and Simulation, Nashville, September 2007

[25] Mak, E, Mittal, S, Hwang, MH, "Automating Link-16 Testing using DEVS and XML", Journal of Defense Modeling and Simulation, Vol. 7, No. 1, pp.39-62, 201

[26] Mittal, S, Martin, JLR, Zeigler, BP, "DEVS-Based Web Services for Net-centric T&E", Summer Computer Simulation Conference, 2007

[27] Mittal, S, "Net-centric Cognitive Architecture using DEVS Unified Process", Persistence and Generative Modeling Workshop, Scottsdate, AZ, 2010

[28] Graettinger, CP, Garcia, S, Siviy, J, Schenk, RJ, Van Syckle, PJ "*Using the technology readiness levels scale to support technology management in the DoD's ATD/STO environments*." Army CECOM. (2002).

[29] 711 HPW/HPO. (2009, January). *Air Force Human Systems Integration handbook.* Retrieved September 1, 2009, from Wright Patterson Air Force Base: http://www.wpafb.af.mil/shared/media/document/AFD-090121-054.pdf

[30] Ministry of Defence HFI DTC. (2008, July 15). *The Human View handbook for MODAF.* Retrieved September 1, 2009, from Human Factors Integration Defence Technology Centre:http://www.hfidtc.com/MoDAF/HV%20Handbook%20First%20Issue.pdf

[31] Handley, HA, Smillie, RJ, Knapp, B, "Architecture frameworks and the human view". Retrieved September 1, 2009, from National Defense Industrial Association: http://www.ndia.org/Divisions/Divisions/SystemsEngineering/Documents/HSI%20Subcommittee/August%202009/HVbrief4NDIA-august2.pdf

[32] NATO RTO HFM-155 Human View Workshop. (n.d.). "*The NATO Human View handbook"*. Retrieved September 1, 2009, from National Defense Industrial Association: http://www.ndia.org/searchcenter/Pages/Results.aspx?k=human%20view

[33] Phillips, EL, "The Development and Initial Evaluation of the Human Readiness Level Framework", MS thesis, Naval Postgraduate School, Monterey, CA, 2010.

[34] Mittal, S, "Extending DoDAF to Allow DEVS-Based Modeling and Simulation", Special issue on DoDAF, Journal of Defense Modeling and Simulation JDMS, Vol. 3, No. 2, pp. 95-123, 2006

[35] Mittal, S, Mak, E, Nutaro, JJ, "DEVS-Based Dynamic Modeling & Simulation Reconguration using Enhanced DoDAF Design Process", special issue on DoDAF, Journal of Defense Modeling and Simulation, Vol. 3, No. 4, pp. 239-267, 2006

[36] Mittal, S, Martin, JLR, Zeigler, BP, " WSDL-Based DEVS Agent for Net-Centric Systems Engineering", International Workshop on Modeling and Applied Simulation, Italy, September 2008

[37] Mittal, S, "Agile Net-centric System using DEVS Unified Process", chapter for "Intelligence Based Systems Engineer", Ed. Andreas Tolk, Lakhmi Jain, Springer-Verlag 2011