AutoDEVS: A Methodology for Automating M&S Software Development and Testing of Interoperable Systems

by

Manuel C. Salas
Bernard Zeigler

Abstract

Despite the increasing use of software system development methodologies that follow a highly structured and systematic process, there remain critical limitations that need more advanced treatment. One such limitation is the failure to explicitly incorporate modeling and simulation (M&S) methodology which becomes essential as the scale of systems under development exceeds the range supported by analytical tools. Other limitations include lack of automation to reduce routine work and incoherence among the abstractions developed at various stages in development. Originally introduced as formalism for discrete event modeling and simulation, the DEVS (Discrete Event System Specification) methodology has become an engine for advances within the wider area of information technology. In this paper, we present a new DEVS-based tool called "AutoDEVS" that automates systems development and exploits "model continuity" to maintain coherence through the development process and to support test artifact continuity and traceability through phases of system development. AutoDEVS creates models and systems that can be mapped to DEVS/SOA, a DEVS implementation compliant with a Service Oriented Architecture infrastructure such as adopted by the DoD's Global Information Grid initiative. After describing the AutoDEVS environment, we show how it provides a rapid means of integrated development and testing of defense command and control systems for interoperability.

1. Introduction

The development of complex software-intensive systems requires systematic methodologies to ensure that all requirements goals and objectives are met. Many organizations make use of the System Development Life Cycle (SDLC) [Sdl08], Joint Application Development (JAD) [Jad08] and Rapid Application Development (RAD) [Rad08] methodologies to assist in developing systems. However, these methodologies tend to miss-handle evolving requirements during the development of the system, calling for support of feedback and iteration that is integral to the methodology and tools. Of particular interest in this paper, are two main limitations:

1. None of the current methodologies integrate Modeling and Simulation (M&S) techniques to develop systems. Modeling and Simulation helps to better understand and optimize performance and/or reliability of systems and verify the correctness of designs. Simulation allows for the development of “virtual environments” to observe and better understand the system under development. These virtual environments also permit speeding up the development process by validating the system early on the development cycle as possible, before the actual hardware is ready, thus reducing risks and costs encountered during the testing phase.
2. These methodologies do not overcome the incoherence problem caused among the different phases of the development process. For instance, the lack of clarity in requirements could impact the elaboration of architectures and test plans for the system that could be detected not until later in the development process causing major design changes and costing the project both time and money.

1.1 Motivation

Several goals motivated the development of "AutoDEVS", a tool that brings new methodologies based on Discrete Event System Specification (DEVS) modular and hierarchical formalism:

- improve systems development to reduce human effort, time constraints, and production costs between different design stages,
- overcome the "incoherence problem" between different stages of the development process,
- help developers spend most of their time on the parts of the system that really matter and reduce as much overhead as possible on repetitive activities,
- introduce automation in the development of systems to increase productivity and produce high-quality, structured solutions to complex problems,
- organize a family of alternative models from which a candidate model can be selected, generated, and evaluated, and
- meet the needs of the entire team, from requirements capture to deploying high-performance, real-time distributed executing models.

Overall, the main contribution of this article is the demonstration of a practical methodology to automate the development and testing of complex, distributed, real-time systems. This methodology is based on DEVS and SES formalisms and overcomes the “incoherence problem” between different stages of the development process by connecting the outputs of one stage to the inputs of the next stage. It allows the users to go from requirements to automated models and simulation of a real-time distributed software system and make use of these models to auto-generate test models and make the system ready to run and interact with the real-world. In particular, we discuss application of the methodology, as supported by the AutoDEVS tool set, to complex distributed command and control systems for which interoperability is a key attribute.

The remaining discussion is organized as follows: Section 2 discusses the state of the art in software development. Section 3 provides a brief review of the most advanced tools that are based on DEVS formalisms and methodologies for the development of systems. Section 4 introduces AutoDEVS as the next generation tool to automate the development and testing of systems, going from natural language to a real time executing system; exploiting the “model continuity” concept through the stages of a development process. Finally, section 5 provides conclusions for AutoDEVS and the importance of M&S in design and evaluation of systems of systems for data and computation interoperability.

2. Software Development State of the Art

Traditionally, many organizations make use of the Systems Development Life Cycle (SDLC) methodology to assist in developing systems, as it ensures that all functional and non-functional requirement goals and objectives are met. SDLC provides a set of models or methodologies such as waterfall, v-shaped, incremental, spiral, build and fix, and synchronize and stabilize. The waterfall model defines a sequence of stages in which the output of each stage becomes the input for the next [Wri08]. This model allows phases to be processed and completed one at a time but it is very rigid and changes to requirements can potentially have a negative impact on the system. The v-shaped model is similar to the waterfall model with the difference that testing procedures are developed before starting the implementation phase. The V-shaped model has more probability for success than the waterfall model due to the development of test plans early on during the life cycle process. On the other hand, no early prototypes of the system are produced [Lew08]. The incremental model consists of dividing the waterfall model into smaller, more easily managed iterations. The incremental model allows having a base working version of the system, where flaws are detected quickly and early in the process. On the other hand, system architecture problems may arise due to changes in requirements in later iterations [Lew08]. The spiral model is most often used in large, complex and expensive systems. This model is similar to the incremental model, with more emphases on risk analysis. The spiral model includes high amount of risk analysis that allows evaluating different alternatives to mitigate a specific problem and then choose the best fit for the system in development. On the other hand, the project success is highly dependent on the risk analysis phase. In the build and fix model a system is built with minimal requirements and specifications, no design or testing is executed. The build and fix model is usually effective in very small projects where the complexity is very limited. However, maintenance could become an issue since no documentation is produced and when this model is applied to highly-complex systems it can result in a low quality, delayed and costly system [Lew08]. In the synchronize and stabilize model, teams work concurrently on individual application modules, executing frequent code synchronization between the teams, and allowing to stabilize the system regularly throughout the development process. Since the synchronize and stabilize model allows changes at any point throughout the process, it is inherently more flexible, and allows responsiveness to changing business requirements.

There are alternatives to the SDLC methodologies such as Joint Applications Design (JAD) and Rapid Application Development (RAD). The JAD technique allows end users to participate in the requirements development process to better understand their needs and set up the desired system. When the size of the group and the system is too large, the JAD technique could become expensive and cumbersome; however it reduces the ambiguity produced during the system requirements elaboration phase [Woo95]. The RAD model involves iterative development and construction of prototypes that allow emulating and provide the ability to quickly build working systems to test their usefulness. This model provides the ability to rapidly change system design as demanded by users. Following the rapid prototyping model can lead to a succession of prototypes that never culminate in a satisfactory production application [Mar91]. More speed and lower cost may lead to lower overall quality. Potential for inconsistent designs within and across systems and the difficulty with model reuse for future systems are some of the most common weaknesses of the RAD model [Cms08].

Rational Rose RealTime is a complete lifecycle Unified Modeling Language (UML) development environment intended for modeling and component construction of enterprise-level software applications that are highly event driven, concurrent and distributed. Rose RealTime unifies the project team by providing an extensive set of tool integrations to meet the needs of the entire team, from requirements capture through high-performance code generation and debugging for real-time operating system targets [Acc00]. However, Rose RealTime is still suffering from several problems. First, the tool focuses on design and it is not fully suited for requirements modeling. Secondly, Rose RealTime does not support concurrency in Statechart Diagrams, which is a disadvantage during requirements and analysis phases. Finally, the tool doesn’t support activity diagrams at all [Ant01], and important processes such as the verification of properties (safety, utility, liveness), the simulation of the system, and the generation of test cases.

Matlab is a high-level technical computing language and interactive environment for algorithm development, data visualization, data analysis, and numeric computation. Using the Matlab tool, developers can solve technical computing problems faster than with traditional programming languages, such as C, C++, and Fortran. Matlab is currently being used by a wide range of applications, including signal and image processing, communications, control design, test and measurement, financial modeling and analysis, and computational biology. In addition, this tool provides a number of features for documenting and sharing developers work. Matlab can be integrated with other languages and applications. Simulink, is integrated with Matlab and it is an environment for multidomain simulation and Model-Based Design for dynamic and embedded systems. It provides an interactive graphical environment and a customizable set of block libraries that let you design, simulate, implement, and test a variety of timevarying systems, including communications, controls, signal processing, video processing, and image processing. The Matlab environment for Model-Based Design (MBD) allows engineers to mathematically model the behavior of the physical system, design the software and model its behavior, and then simulate the entire system model to accurately predict and optimize performance. The system model becomes a specification from which you can automatically generate real-time software for testing, prototyping, and embedded implementation, thus avoiding manual effort and reducing the potential for errors. [Mat08]. The major drawbacks of this environment are its size and relative complexity; it takes some time to become familiar with its language and become familiar with several of the main routines needed for basic simulations. Equations must be handled in certain form and sequence, requiring the user to understand the assumptions underlying the tool, in addition to the system's phenomena being analyzed. Furthermore, good comprehension of numerical analysis, linear algebra and linear systems is required [Can97].

3. DEVS Framework for M&S-Driven Software Development

Discrete Event System Specification (DEVS) is a unified framework for developing real-time software systems in which logical analysis, performance evaluation, and implementation can be performed, all based on the DEVS formalism [Sar01]. DEVS framework is based on systems engineering principles and is used for modeling and simulation in many application domains. This framework supports a number of important features such as component based hierarchical simulation model development, scalability, reusability and distributed simulation. The framework was later extended to support parallel model specification and execution, hierarchical model development and modularity, and object orientation in 2003, namely Communicating DEVS for logical analysis and Real-time DEVS for implementation. DEVS framework has been implemented in different programming languages such as DEVSJAVA (java), ADEVS (C++) and DEVS# (C#) and over various middleware such as DEVS/CORBA, DEVS/HLA and DEVS/SOA [HuX04].

Real-Time DEVS (RTDEVS) formalism extends the classic DEVS formalism in atomic DEVS models. The RTDEVS formalism for coupled models remains the same as the original. In the classic DEVS formalism, simulation time advances only when a simulator calls the time advance function ta of the associated model. The time advance function ta in the RTDEVS formalism behaves the same as that in the classic DEVS formalism except that here the time is an integer, while in classic DEVS, time is a real number. The time calculated by the time advance function also synchronized with the wall clock time. This is because a simulation clock in RTDEVS is no longer a virtual clock but a real-time clock. An activity is an operation that takes a certain amount of time to complete the assigned task [Hon97]. This was adopted by [Hon97] to represent some time-consuming operations such as waiting for a message, processing a job, and so forth.

3.1 DEVS & DEVSJAVA

DEVSJAVA implements the DEVS formalism on java. It provides a set of libraries on top of the java native libraries that allows the building of modular and hierarchical model compositions based on the closure-under-coupling paradigm to create a system, see Figure 3.1.

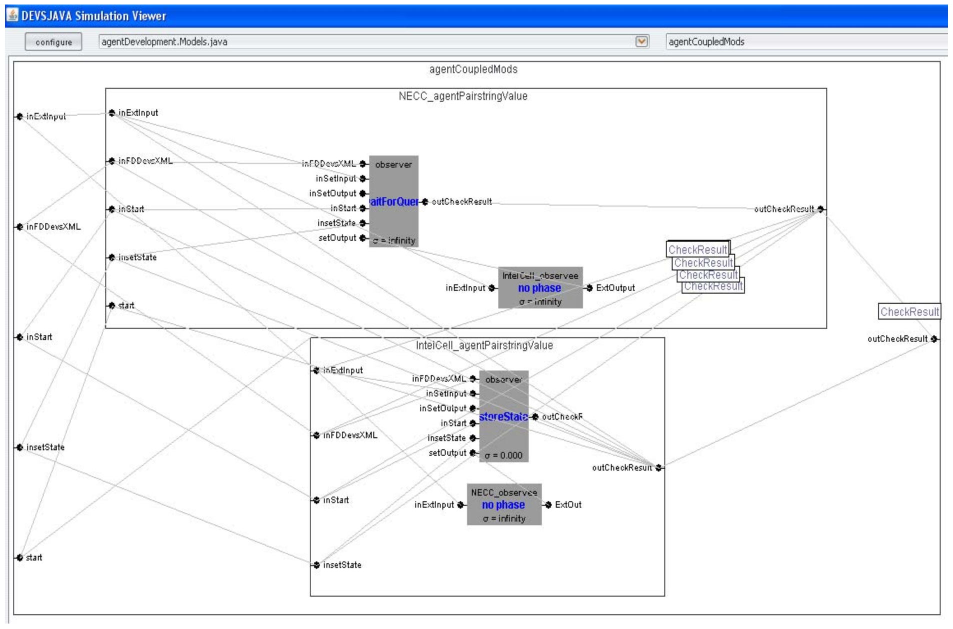


Figure 3.1 Example system developed in DEVSJAVA.

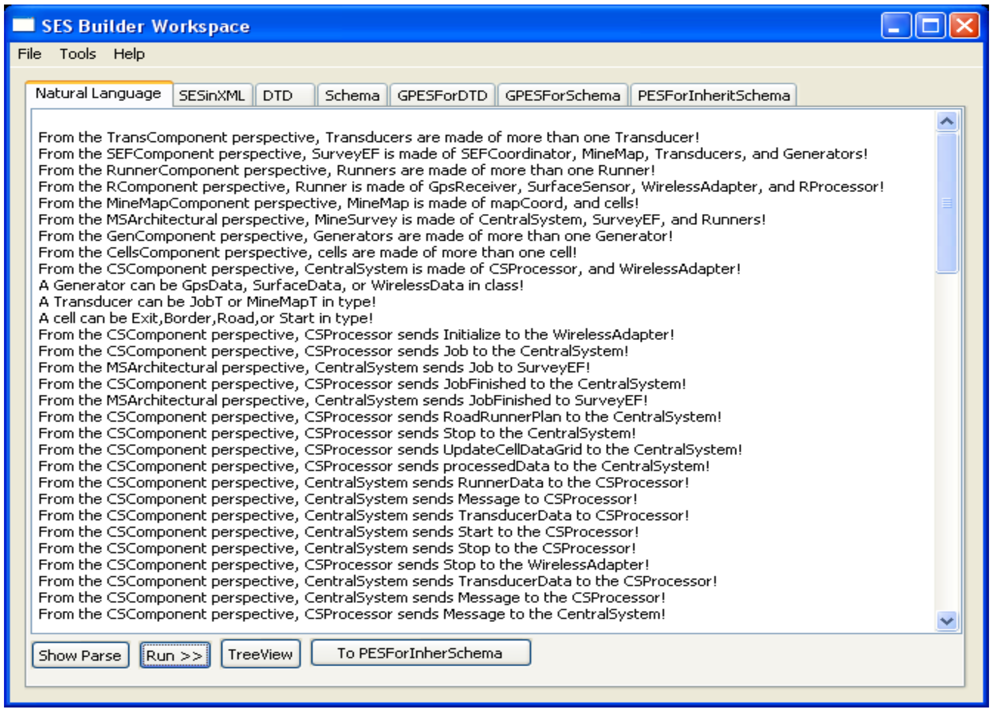
A system is described as a set of input/output events and internal states along with behavior functions regarding the event consumption/production and internal state transitions. Generally models are considered as either atomic models or coupled models. The atomic model can be illustrated as a black box having a set of inputs (X) and a set of outputs (Y). The atomic model includes a description of the interface as well as the data flow between itself and other DEVS models. The atomic model also specifies a set if internal states (S) with some operation functions such as external transition function (δ_{ext}), internal transition function (δ_{int}), output function λ and time advance function ta to describe the dynamic behavior of the model. δ_{ext} carries the input and changes the system states; δ_{int} changes internal variables from the previous state to the next when no events have occurred since the last transition; λ generates an output event to outside models in the current state; ta determines the time of the next internal event after generating an output event. Basic models may be connected in the DEVS formalism to form a coupled model. A coupled model is the major class which embodies the hierarchical model composition constructs of the DEVS formalism [Zei00]. A coupled model is made up of component models, and the coupling relations which establish the desired communication links. A coupled model specified how to couple (connect) several component models together to form a new model. Therefore two essential activities involved in coupled models are specifying its component models and defining the couplings which create the desired communication paths

[Zei00].

3.2 SES & SESBuilder

The System Entity Structure (SES) [Zei07] is a high level ontology framework targeted to modeling, simulation, systems design and engineering. Its expressive power, both in strength and limitation, derive from that domain of discourse. An SES is a formal structure governed by a small number of axioms that provide clarity and rigor to its models. The structure supports hierarchical and modular compositions allowing large complex structures to be built in stepwise fashion from smaller, simpler ones. Tools have been developed to transform SESs back and forth to XML allowing many operations to be specified in either SES directly or in its XML guise. The axioms and functionally based semantics of the SES promote pragmatic design and are easily understandable by data modelers. Together with the availability of appropriate tool support, this makes development of XML Schema transparent to the modeler. Finally, SES structures are compact relative to equivalent Schema and automatically generate associated executable simulation models [Mit07].

SESBuilder is a powerful stand alone application which supports operations on System Entity Structures (SES), see Figure 3.2. SESBuilder employs the XML to create Pruned Entity Structures (PES) that represents the logically possible set of world state descriptions consistent with the SES. At the implementation level, an SES is represented by a schema or DTD whose instance documents represent possible prunings. The SESBuilder supports convenient specification of SESs, pruning to create PESs, and transformation to XML representations, all through natural language and graphical interfaces [Sae08, Zei07]. Thus, it can be used to do serious data engineering with a powerful multi-tab graphic user interface. It supports local file system to load and save intermediate and final results for different purposes. SESBuilder is built on Java Standard Widget Toolkit (SWT) and could be deployed on either Windows or Unix/Linux based operating system [Rts08].



```
From the TransComponent perspective, Transducers are made of more than one Transducer!
From the SEFComponent perspective, SurveyEF is made of SEFCoordinator, MineMap, Transducers, and Generators!
From the RunnerComponent perspective, Runners are made of more than one Runner!
From the RComponent perspective, Runner is made of GpsReceiver, SurfaceSensor, WirelessAdapter, and RProcessor!
From the MineMapComponent perspective, MineMap is made of mapCoord, and cells!
From the MSArchitectural perspective, MineSurvey is made of CentralSystem, SurveyEF, and Runners!
From the GenComponent perspective, Generators are made of more than one Generator!
From the CellsComponent perspective, cells are made of more than one cell!
From the CSComponent perspective, CentralSystem is made of CSProcessor, and WirelessAdapter!
A Generator can be GpsData, SurfaceData, or WirelessData in class!
A Transducer can be JobT or MineMapT in type!
A cell can be Exit,Border,Road,or Start in type!
From the CSComponent perspective, CSProcessor sends Initialize to the WirelessAdapter!
From the CSComponent perspective, CSProcessor sends Job to the CentralSystem!
From the MSArchitectural perspective, CentralSystem sends Job to SurveyEF!
From the CSComponent perspective, CSProcessor sends JobFinished to the CentralSystem!
From the MSArchitectural perspective, CentralSystem sends JobFinished to SurveyEF!
From the CSComponent perspective, CSProcessor sends RoadRunnerPlan to the CentralSystem!
From the CSComponent perspective, CSProcessor sends Stop to the CentralSystem!
From the CSComponent perspective, CSProcessor sends UpdateCellDataGrid to the CentralSystem!
From the CSComponent perspective, CSProcessor sends processedData to the CentralSystem!
From the CSComponent perspective, CentralSystem sends RunnerData to the CSProcessor!
From the CSComponent perspective, CentralSystem sends Message to CSProcessor!
From the CSComponent perspective, CentralSystem sends TransducerData to CSProcessor!
From the CSComponent perspective, CentralSystem sends Start to the CSProcessor!
From the CSComponent perspective, CentralSystem sends Stop to the CSProcessor!
From the CSComponent perspective, CSProcessor sends Stop to the WirelessAdapter!
From the CSComponent perspective, CentralSystem sends TransducerData to the CSProcessor!
From the CSComponent perspective, CentralSystem sends Message to the CSProcessor!
From the CSComponent perspective, CSProcessor sends Message to the CentralSystem!
```

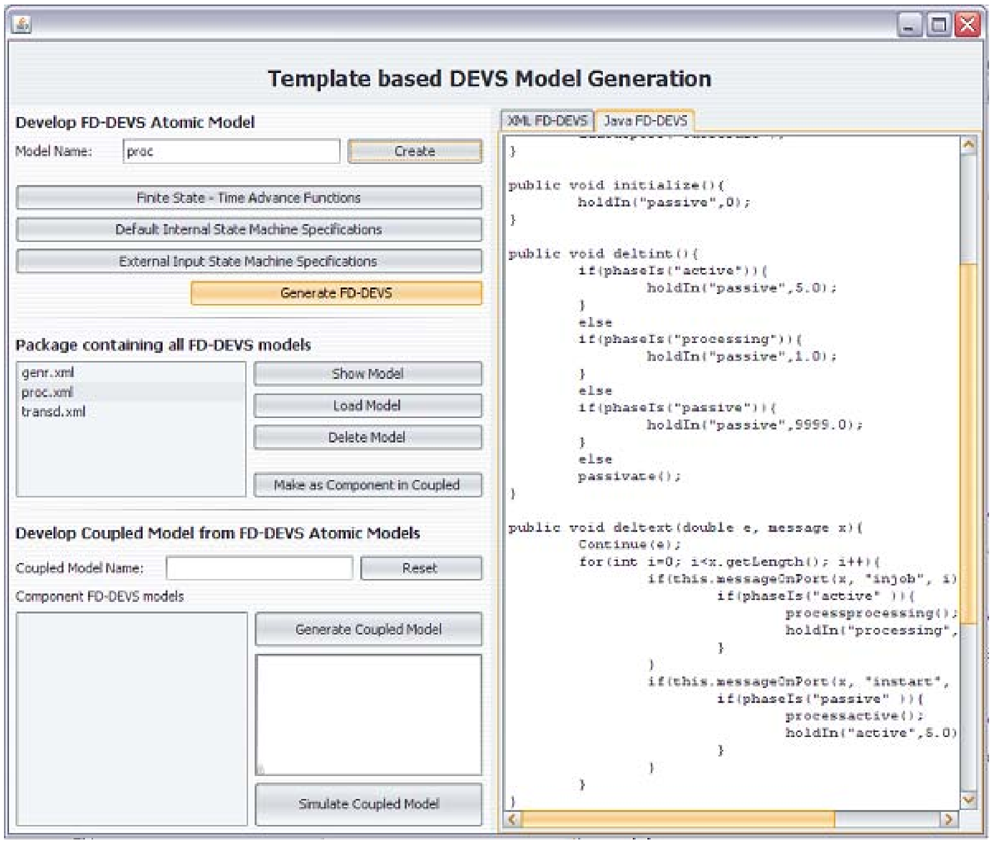
Figure 3.2 SESBuilder Natural Language View

Some of the highlights of the SESBuilder tool are [Ses08]:

- SES is an effective way of Knowledge Representation and data engineering in support of ontology development languages and environments based on SES Axioms.
- SES Builder provides a data engineering work space based in XML for syntax and validity checks.
- Modeling and Simulation-based Data Engineering is supported by SES & PES at ontology level implemented in XML Schemata and XML instances.
- With its Modeling & Simulation-based Data Engineering approach, the SESBuilder supports the framework for information exchange for Net-centric environment.

3.3 Finite Deterministic DEVS (FDDEVS)

Finite Deterministic DEVS (FDDEVS) formalism was first introduced to restrict the class of DEVS models so as to allow deeper analysis of verifiable properties for the restricted class [Hwa06]. The simplicity of the FDDEVS formulation was subsequently recognized to provide a useful abstraction for development of DEVS models using template-based design [Mit08].


Template based DEVS Model Generation
Develop FD-DEVS Atomic Model
Model Name:
proc
Create
Finite State - Time Advance Functions
Default Internal State Machine Specifications
External Input State Machine Specifications
Generate FD-DEVS
Package containing all FD-DEVS models
genr.xml
proc.xml
transd.xml
Show Model
Load Model
Delete Model
Make as Component in Coupled
Develop Coupled Model from FD-DEVS Atomic Models
Coupled Model Name:
Reset
Component FD-DEVS models
Generate Coupled Model
Simulate Coupled Model
XML FD-DEVS
Java FD-DEVS

```
}

public void initialize(){
        holdIn("passive",0);
}

public void deltint(){
        if(phaseIs("active")){
                holdIn("passive",5.0);
        }
        else
        if(phaseIs("processing")){
                holdIn("passive",1.0);
        }
        else
        if(phaseIs("passive")){
                holdIn("passive",9999.0);
        }
        else
        passivate();
}

public void deltext(double e, message x){
        Continue(e);
        for(int i=0; i<x.getLength(); i++){
                if(this.messageOnPort(x, "injob", i)
                        if(phaseIs("active" )){
                                processprocessing();
                                holdIn("processing",
                        }
                }
                if(this.messageOnPort(x, "instart",
                        if(phaseIs("passive" )){
                                processactive();
                                holdIn("active",5.0)
                        }
                }
        }
}
```

Figure 3.3 FDDEVS GUI.

As seen in Figure 3.3, the FDDEVS GUI allows the creation of Java and XML-expressed FD-DEVS models; it allows the user to easily specify the states, time advance, internal/external transitions, incoming/outgoing messages for the models to be generated. It also permits one to employ the generated atomic models to create coupled models as described in the previous section.

As an alternative to the template-based design, the creation of FDDEVS models can be done using a constrained natural language input that has the following statement forms [Mit08]:

to start hold in PHASE for time SIGMA !
hold in PHASE for time SIGMA !
after PHASE then output MSG !
from PHASE go to PHASE' !
when in PHASE and receive MSG go to PHASE' !

For convenience, some of these statements can be compounded as phrases in the following statements:

hold in PHASE for time SIGMA then output MSG and go to PHASE' !
hold in PHASE for time SIGMA then go to PHASE' !

The semantics of these statements is provided by the following table along with examples of how to use them.

Natural Language Phrase	**XFD-DEVS Specification**	**Example**
to start hold in PHASE for time SIGMA! where SIGMA is 0, a positive real, or the symbol Infinity (other variants such as Inf are allowed)	TimeAdvanceTable(PHASE) = SIGMA and also set PHASE as initial state	to start hold in waitForJob for time Infinitiy! This can also be expressed as: to start passivate in waitForJob!
hold in PHASE for time SIGMA where SIGMA = 0, a positive real, or the symbol Infinity(other variants such as Inf are allowed)	TimeAdvanceTable(PHASE)=SIGMA	hold in sendOut for time 0.1 !
after PHASE then output MSG!	OutputTable(PHASE) = MSG	after sendOut then output Job !
from PHASE go to PHASE!	InternalTransitionTable(PHASE) = PHASE'	from sendOut go to waitForJob !
when in PHASE and receive MSG go to PHASE'!	ExternalTransitionTable(PHASE,MSG) = PHASE'	when in waitForJob and receive Job go to processing!
when in PHASE and receive MSG go to PHASE' eventually !	ExternalTransitionTable(PHASE, MSG) = PHASE' with a mark to note that this input is schedule-preserving in this state	when in waitForJob and receive Job go to processing eventually !

The semantics of compound statements are composed of the semantics of the individual phrases.

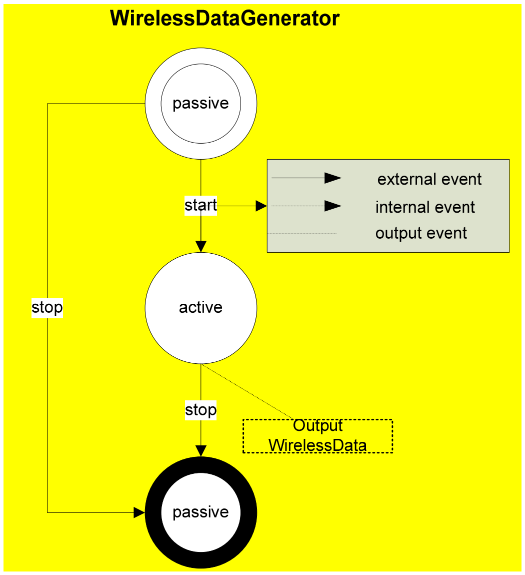


Figure 3.4 WirelessDataGenerator State Diagram.

FDDEVS follows a XML schema that facilitates the development of state machines' internal and external transition behaviors. For instance, the development of a wireless data generator FDDEVS model represented by the state diagram presented in Figure 3.4, can be translated into the following FDDEVS natural language:

WirelessDataGenerator: to start passivate in passive!
WirelessDataGenerator: when in phase passive and receive Start go to active!
WirelessDataGenerator: hold in active for time 1 then output WirelessData and go to active!

The generator outputs WirelessData every 1 unit of time. In general, FDDEVS allows use of general DEVS timing capabilities except those requiring non-finite states. For example, the remaining time to transition after an input arrival cannot be saved and reused.

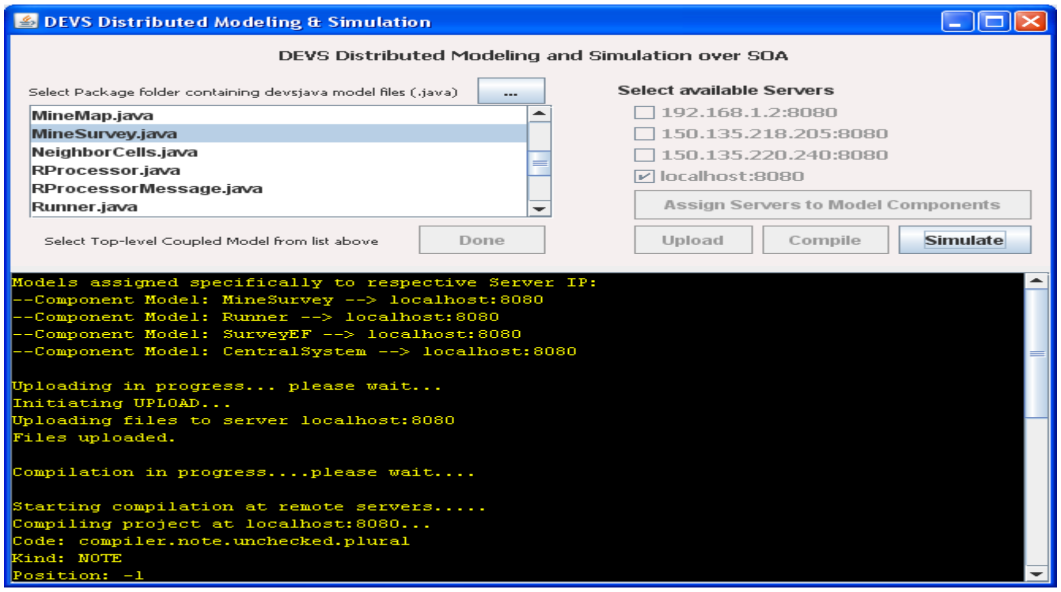
3.4 DEVS/SOA

DEVS/SOA [Mit07] is a prototype simulation framework that has been implemented using SOA technology. The central point resides in executing the simulator as a web service. The development of this framework helps to solve large-scale problems and guarantees interoperability among different networked systems and specifically DEVS-validated models. DEVS/SOA makes the DEVS simulation process transparent in the model-design cycle, allowing the modeler not to be concerned with the simulator compatibility or any platform issues as in earlier developments like DEVS/C++, DEVSJAVA, DEVS/RMI, and DEVS/CORBA. With this Simulation Service platform the designer is able to execute the model over the Internet through web services, using SOA as the communication protocol. This framework is able to execute DEVSJAVA models, and is extensible to other simulation platforms.

A DEVS/SOA client takes the DEVS models package and through the dedicated servers hosting simulation services, it performs the following operations:
1. Upload the models to specific IP locations
2. Run-time compile at respective sites
3. Simulate the coupled-model
4. Receive the simulation output at client's site

The DEVS/SOA client as shown in Figure 3.5 below operates in the following sequential manner:

1. The user selects the DEVS package folder at his machine
2. The top-level coupled model is selected as shown in Figure 3.5
3. Various available servers are selected. Any number of available servers can be selected.
4. The user then uploads the model by clicking the Upload button. The models are partitioned in a round-robin mechanism and distributed among various chosen servers
5. The user then compiles the models by clicking the Compile button at server's end
6. Finally, Simulate button is pressed to execute the simulation using the Simulation service hosted by these services.
7. Once the simulation is over, the console output window displays the aggregated simulation logs from various servers at the client's end.


DEVS Distributed Modeling & Simulation
DEVS Distributed Modeling and Simulation over SOA
Select Package folder containing devsjava model files (.java)
...
MineMap.java
MineSurvey.java
NeighborCells.java
RProcessor.java
RProcessorMessage.java
Runner.java
Select Top-level Coupled Model from list above
Done
Select available Servers
192.168.1.2:8080
150.135.218.205:8080
150.135.220.240:8080
localhost:8080
Assign Servers to Model Components
Upload
Compile
Simulate

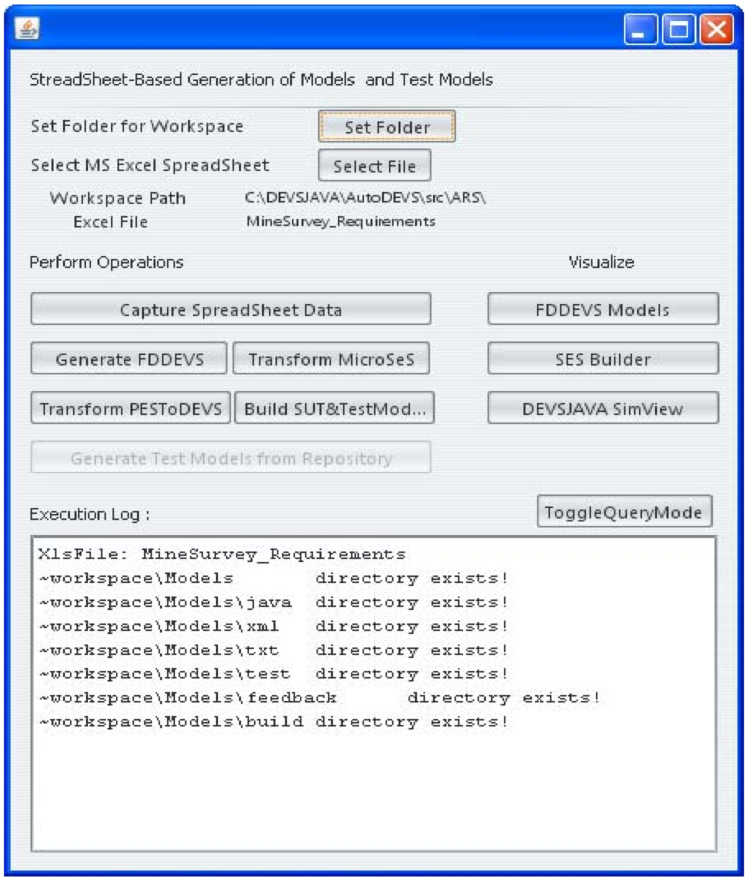
```
Models assigned specifically to respective Server IP:
--Component Model: MineSurvey --> localhost:8080
--Component Model: Runner --> localhost:8080
--Component Model: SurveyEF --> localhost:8080
--Component Model: CentralSystem --> localhost:8080

Uploading in progress... please wait...
Initiating UPLOAD...
Uploading files to server localhost:8080
Files uploaded.

Compilation in progress....please wait....

Starting compilation at remote servers.....
Compiling project at localhost:8080...
Code: compiler.note.unchecked.plural
Kind: NOTE
Position: -1
```

Figure 3.5 GUI snapshot of DEVS/SOA client hosting distributed simulation

In accordance with the model continuity requirement, both logical time (time-managed) and real-time discrete event simulation are supported by DEVS/SOA. The user can determine the simulation mode of a model by selecting which simulation protocol to be used, whether logical or real-time, without changing the model itself.

4. AutoDEVS Description

As indicted earlier, AutoDEVS has been created to increase productivity in systems development by automating the life cycle process of a system in all the different phases. The name, AutoDEVS, comes from its support for automation of DEVSJAVA model generation. It provides methodologies to develop systems by generating models and testing models from a spreadsheet containing requirements specification that turn into an executing real-time system, see Figure 4.1.



```
XlsFile: MineSurvey_Requirements
~workspace\Models        directory exists!
~workspace\Models\java   directory exists!
~workspace\Models\xml    directory exists!
~workspace\Models\txt    directory exists!
~workspace\Models\test   directory exists!
~workspace\Models\feedback       directory exists!
~workspace\Models\build directory exists!
```

Figure 4.1 AutoDEVS Graphical User Interface

4.1AutoDEVS Lifecycle

AutoDEVS process is iterative allowing return to modify the reference master DEVS-model and the requirement's specifications. Model continuity minimizes the artifacts that have to be modified as the process proceeds. The design methodology provides a process to transform the requirement's specifications to a DEVS representation supporting evaluation and recommendations for a feasible design.

As seen in Figure 4.2, AutoDEVS life cycle process combines system theory, M&S framework, and model-continuity concepts. As illustrated, the tool bifurcates the process into two main streams – system development and test suite development – that converge in the system testing phase. The system development includes the definition of requirements, capture of specifications to map formalized DEVS model components and create a reference master model, and use of model continuity to execute model in the DEVS real-time execution protocol, i.e. SimView. The test suite development includes development of test models, and execution of test models against the system under test to provide a feasible design from simulation and analysis results [Mit07].

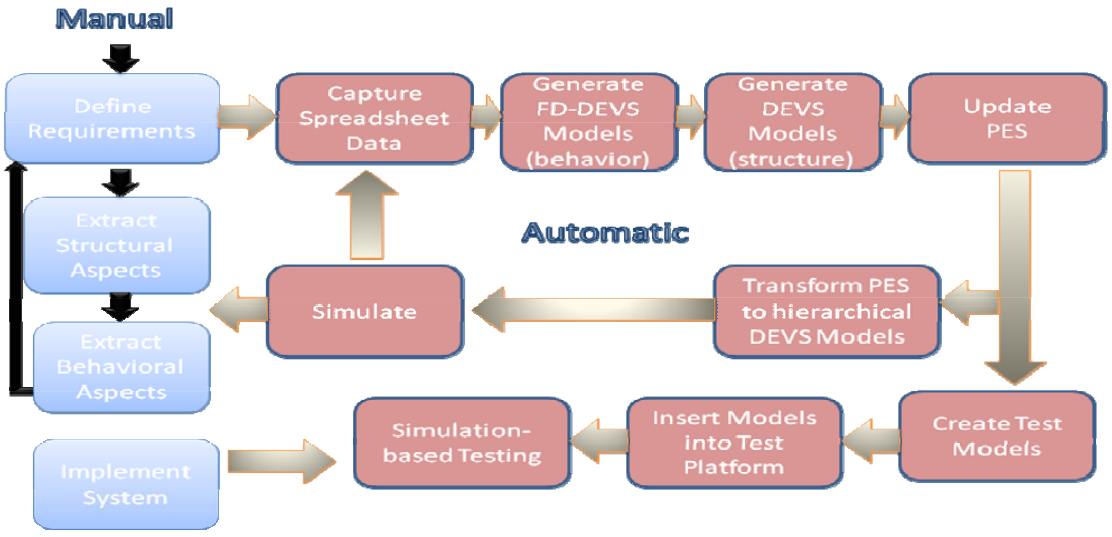
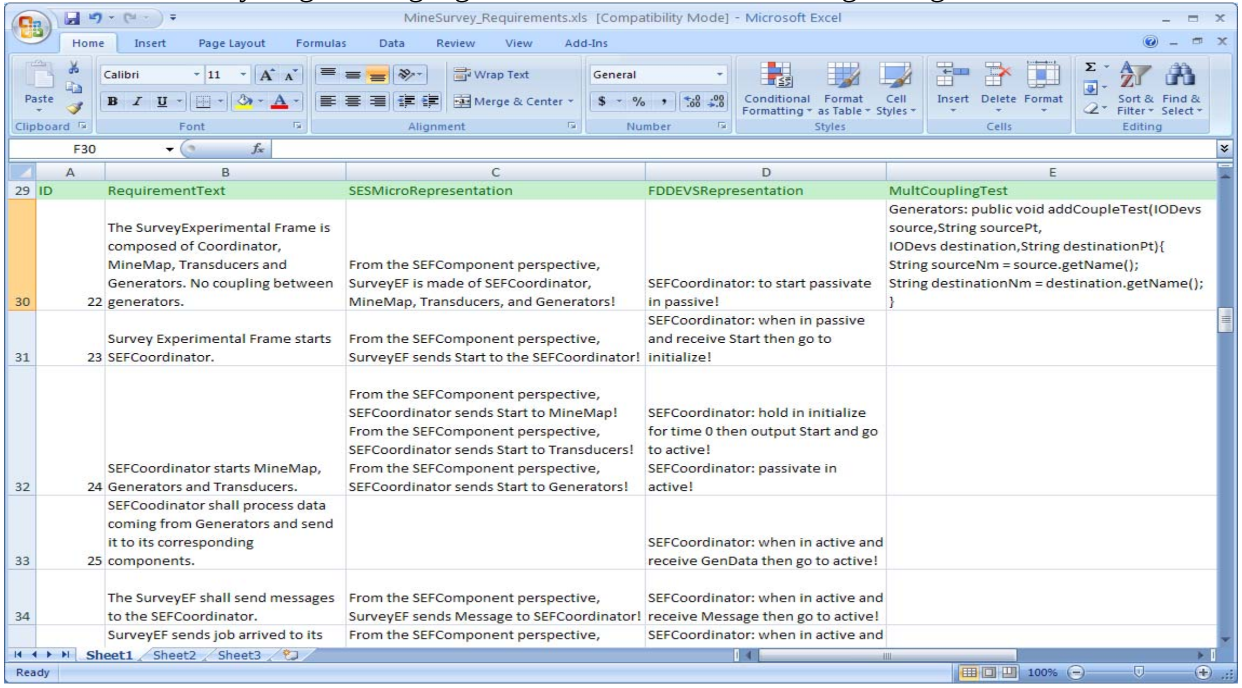

Manual
Define Requirements
Extract Structural Aspects
Extract Behavioral Aspects
Implement System
Capture Spreadsheet Data
Generate FD-DEVS Models (behavior)
Generate DEVS Models (structure)
Update PES
Automatic
Simulate
Transform PES to hierarchical DEVS Models
Simulation-based Testing
Insert Models into Test Platform
Create Test Models

Figure 4.2 Integrated Development and Testing Methodology

4.2 AutoDEVS Tool

AutoDEVS uses Natural Language (English) Specification of an SES as the preferred means of specifying discrete event systems and defining the structural aspects of the models being developed; see Figure 4.3 "SESMicroRepresentation" column. This Natural Language is bounded by rules that encompass all the possible interactions related to any message type. These rules also limit the way English language is used in terms of removing ambiguous statements.



	A	B	C	D	E
29	ID	RequirementText	SESMicroRepresentation	FDDEVSRepresentation	MultCouplingTest
30	22	The SurveyExperimental Frame is composed of Coordinator, MineMap, Transducers and Generators. No coupling between generators.	From the SEFComponent perspective, SurveyEF is made of SEFCoordinator, MineMap, Transducers, and Generators!	SEFCoordinator: to start passivate in passive!	Generators: public void addCoupleTest(IODevs source,String sourcePt, IODevs destination,String destinationPt){ String sourceNm = source.getName(); String destinationNm = destination.getName(); }
31	23	Survey Experimental Frame starts SEFCoordinator.	From the SEFComponent perspective, SurveyEF sends Start to the SEFCoordinator!	SEFCoordinator: when in passive and receive Start then go to initialize!	
32	24	SEFCoordinator starts MineMap, Generators and Transducers.	From the SEFComponent perspective, SEFCoordinator sends Start to MineMap! From the SEFComponent perspective, SEFCoordinator sends Start to Transducers! From the SEFComponent perspective, SEFCoordinator sends Start to Generators!	SEFCoordinator: hold in initialize for time 0 then output Start and go to active! SEFCoordinator: passivate in active!	
33	25	SEFCoodinator shall process data coming from Generators and send it to its corresponding components.		SEFCoordinator: when in active and receive GenData then go to active!	
34		The SurveyEF shall send messages to the SEFCoordinator.	From the SEFComponent perspective, SurveyEF sends Message to SEFCoordinator!	SEFCoordinator: when in active and receive Message then go to active!	
		SurveyEF sends job arrived to its	From the SEFComponent perspective,	SEFCoordinator: when in active and	

Figure 4.3 AutoDEVS: Requirements Specification

The basic idea is as follows. The entity is considered as a collection of various message streams. It has been observed in complex systems that an entity node can act as receiver and sender simultaneously. It is logical to consider that a node may be processing more than one messages at a given instant. Consequently, developing a framework where the entity node model can operate with multiple message streams is the objective of this type of requirement specifications [Mit07].

The rules that provide a binding to this type of requirement specifications are provided in Table 4.1. The designer can specify each node's behavior as a sender and a receiver with respect to any specific message type.

Rules:		
1	Copula	"is" and "are" are treated the same.
2	Compounds	x and y; x, y, and z; NOTE: the commas are mandatory for 3 or more constituents x, y, z, and w; x or y; x, y, or z; x, y, z, or w;
3	Determiners	"a","the",… are removed from the input before processing
4	End of Sentence	use "!" instead of "."
5	Sentence Order	· The entity mentioned first in the first sentence becomes the root of the SES. · A variable must be attached to an entity before giving it a range specification (see below). · Otherwise sentences can be in any order.
6	Forms	In the following, CAPITALs indicate variables, lower case indicate mandatory key words in the order shown.
7	Specialization	THING can be VARIANT1, VARIANT2, or VARIANT3 in CLASSFAMILY!
8	Aspect	From VIEW perspective, THING is made of COMPONENT1, COMPONENT2, and COMPONENT3 !
9	MultiAspect	From multiple perspective, THINGS are made of more than one THING !
10	Attached Variables	THING has VAR1, VAR2, and

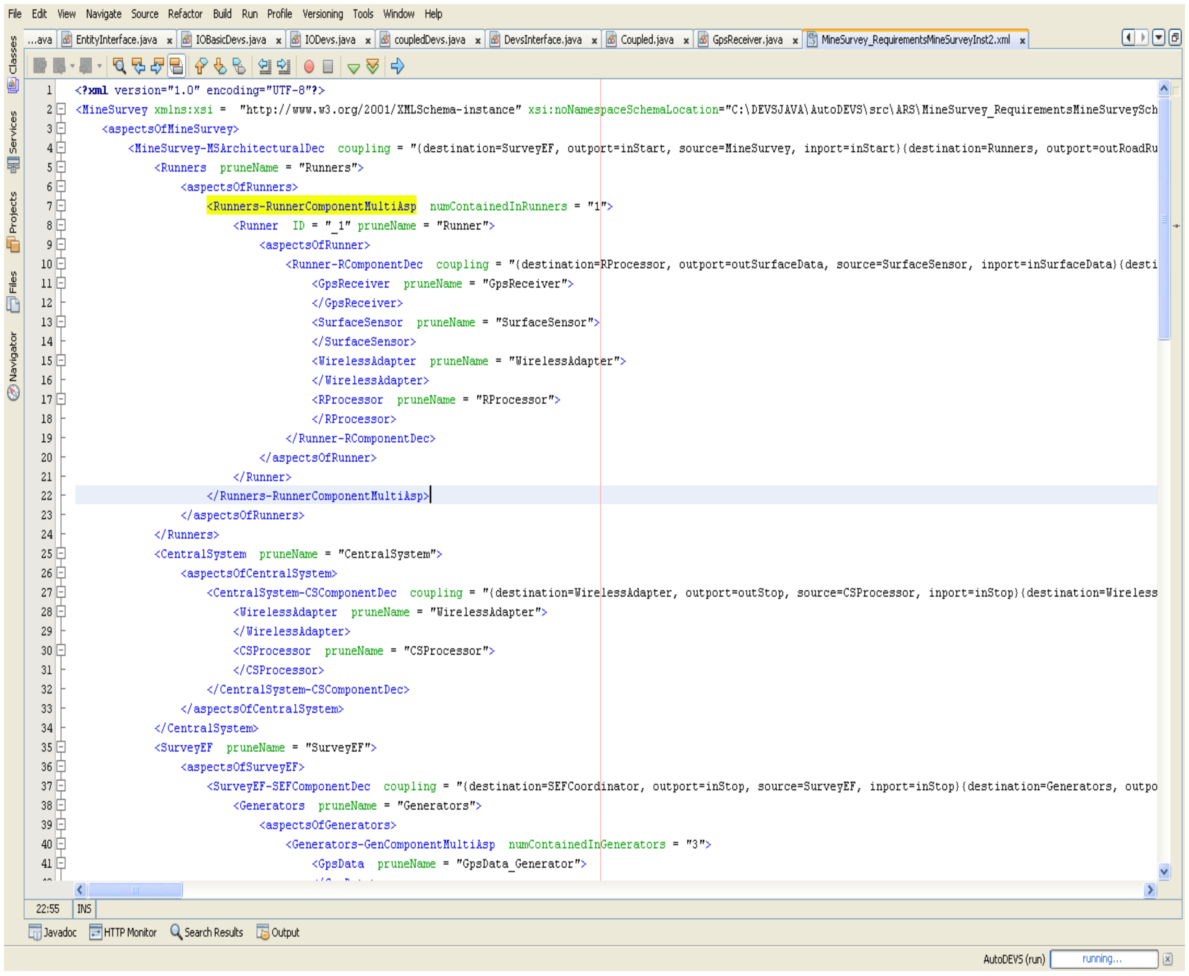
		VAR3!
11	Range specifications of variables	The range of THING's VAR1 is RANGE!

Table 4-1 Rules for Restricted NLP based Requirement Specifications

AutoDEVS then uses Finite-Deterministic DEVS Natural Language to describe the behavior aspect of the system being developed. This FDDEVS Natural Language defines the different states, internal and external transitions between the models being developed as described in section 3.3.

See the "FDDEVSRepresentation" column of Figure 4.3. It contains the behavioral aspects of the system under development. AutoDEVS makes use of this column to automate the development of the behavioral aspects for the DEVS models being created.

After constructing the structural and behavior aspects of the system, AutoDEVS then executes automatic pruning on the models created based on the different specialization identified, see Figure 4.4. The PES can be transformed into a composition tree, see Figure 4.5 and eventually synthesized into a simulation model. AutoDEVS also provides the capability to modify the PES created by the tool and then simulate it to validate the models pruned, see Figure 4.6.



```xml
<?xml version="1.0" encoding="UTF-8"?>
<MineSurvey xmlns:xsi =  "http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespaceSchemaLocation="C:\DEVSJAVA\AutoDEVS\src\ARS\MineSurvey_RequirementsMineSurveySch
    <aspectsOfMineSurvey>
        <MineSurvey-MSArchitecturalDec  coupling = "{destination=SurveyEF, outport=inStart, source=MineSurvey, inport=inStart}{destination=Runners, outport=outRoadRu
            <Runners  pruneName = "Runners">
                <aspectsOfRunners>
                    <Runners-RunnerComponentMultiAsp  numContainedInRunners = "1">
                        <Runner  ID = "_1" pruneName = "Runner">
                            <aspectsOfRunner>
                                <Runner-RComponentDec  coupling = "{destination=RProcessor, outport=outSurfaceData, source=SurfaceSensor, inport=inSurfaceData}{desti
                                    <GpsReceiver  pruneName = "GpsReceiver">
                                    </GpsReceiver>
                                    <SurfaceSensor  pruneName = "SurfaceSensor">
                                    </SurfaceSensor>
                                    <WirelessAdapter  pruneName = "WirelessAdapter">
                                    </WirelessAdapter>
                                    <RProcessor  pruneName = "RProcessor">
                                    </RProcessor>
                                </Runner-RComponentDec>
                            </aspectsOfRunner>
                        </Runner>
                    </Runners-RunnerComponentMultiAsp>
                </aspectsOfRunners>
            </Runners>
            <CentralSystem  pruneName = "CentralSystem">
                <aspectsOfCentralSystem>
                    <CentralSystem-CSComponentDec  coupling = "{destination=WirelessAdapter, outport=outStop, source=CSProcessor, inport=inStop}{destination=Wireless
                        <WirelessAdapter  pruneName = "WirelessAdapter">
                        </WirelessAdapter>
                        <CSProcessor  pruneName = "CSProcessor">
                        </CSProcessor>
                    </CentralSystem-CSComponentDec>
                </aspectsOfCentralSystem>
            </CentralSystem>
            <SurveyEF  pruneName = "SurveyEF">
                <aspectsOfSurveyEF>
                    <SurveyEF-SEFComponentDec  coupling = "{destination=SEFCoordinator, outport=inStop, source=SurveyEF, inport=inStop}{destination=Generators, outpo
                        <Generators  pruneName = "Generators">
                            <aspectsOfGenerators>
                                <Generators-GenComponentMultiAsp  numContainedInGenerators = "3">
                                    <GpsData  pruneName = "GpsData_Generator">
```

Figure 4.4 AutoDEVS: Automated PES

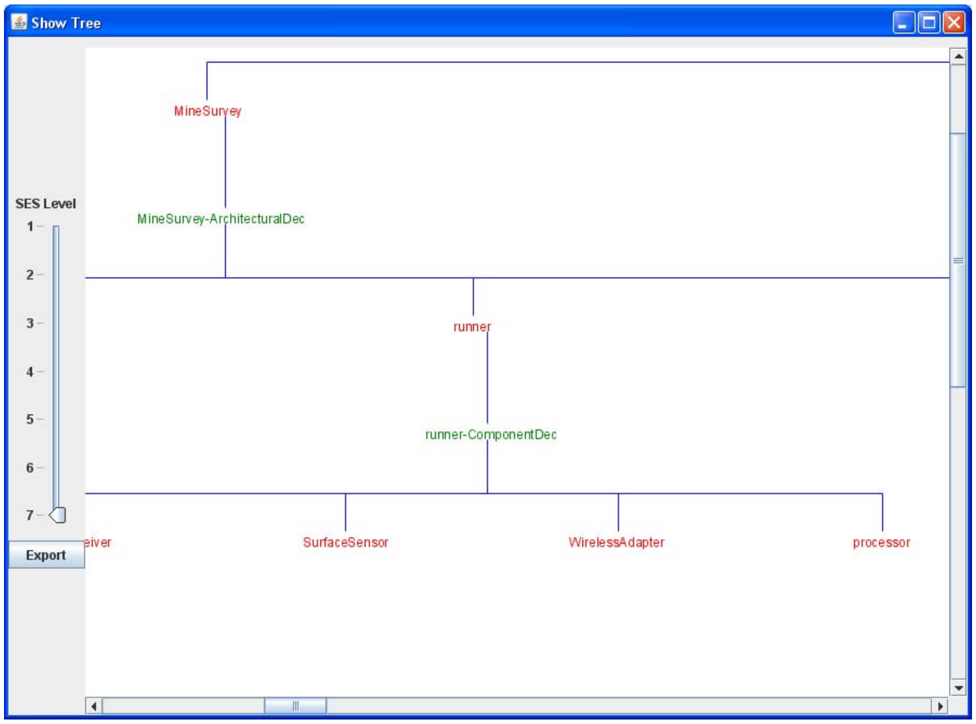
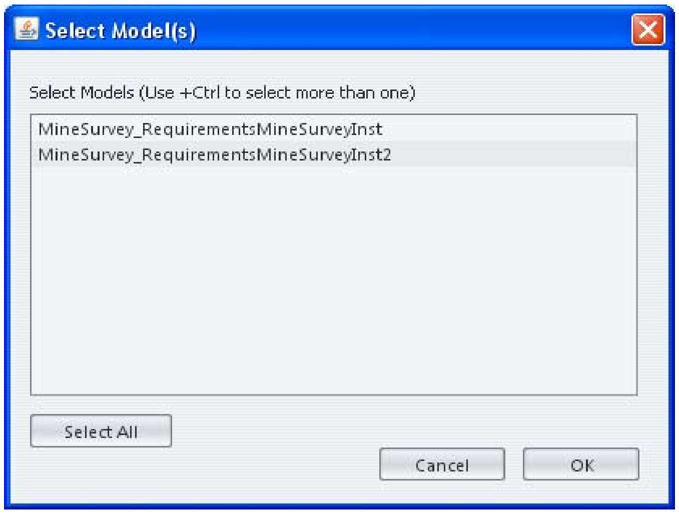


Figure 4.5 AutoDEVS: tree representation of the PES created.


Select Model(s)

Select Models (Use +Ctrl to select more than one)

MineSurvey_RequirementsMineSurveyInst
MineSurvey_RequirementsMineSurveyInst2

Figure 4.6 AutoDEVS: PES user selection

After AutoDEVS has created all the models from the derived requirement specifications, it allows the user to automatically create DEVS test models for the system being developed. These test models are created with the objective to verify the correctness of the DEVS models. The test methodology is based on minimal testable I/O pairs restricted to messages, and assuming they are the only automatable observables available for testing. The DEVS test models are in the form of an experimental frame and allow the developer to perform experiments against the System Under Test (SUT). The test engineer analyzes the requirements (from the spreadsheet) and creates the test scenarios which describe the behaviors of the SUT. The requirements are written in minimal testable input/output representation, and the test models are created by applying the model mirroring concept that reverse the minimal testable I/O pairs.

Both the minimal testable file and test models are written in XML format and represented by SES, allowing for the transformation between the two XML files. The inputs/output pairs are now represented by three *primitive* atomic models: *holdSend*, *waitReceive*, and *waitNotReceive*. Since the input/output are in sequential order, only one atomic model is active each time, and the rest of the atomic models are passive. In order to try out these test models against the real system, they are converted to software programming source code, refer to [Mit07] for more details. This testing methodology is still under development and will be available soon in the AutoDEVS tool.

Finally, the AutoDEVS tool provides an interface to the FDDEVS tool, SESBuilder, and DEVSJAVA SimView. Refer to Section 4 for the description of FDDEVS and SESBuilder tools. The DEVSJAVA SimView is a program included in the DEVSJAVA framework to view and check that all the models and couplings are built as expected. In addition, SimView allows the developers to run, observe and evaluate the real-time execution of the system under development, see Figure 4.7.

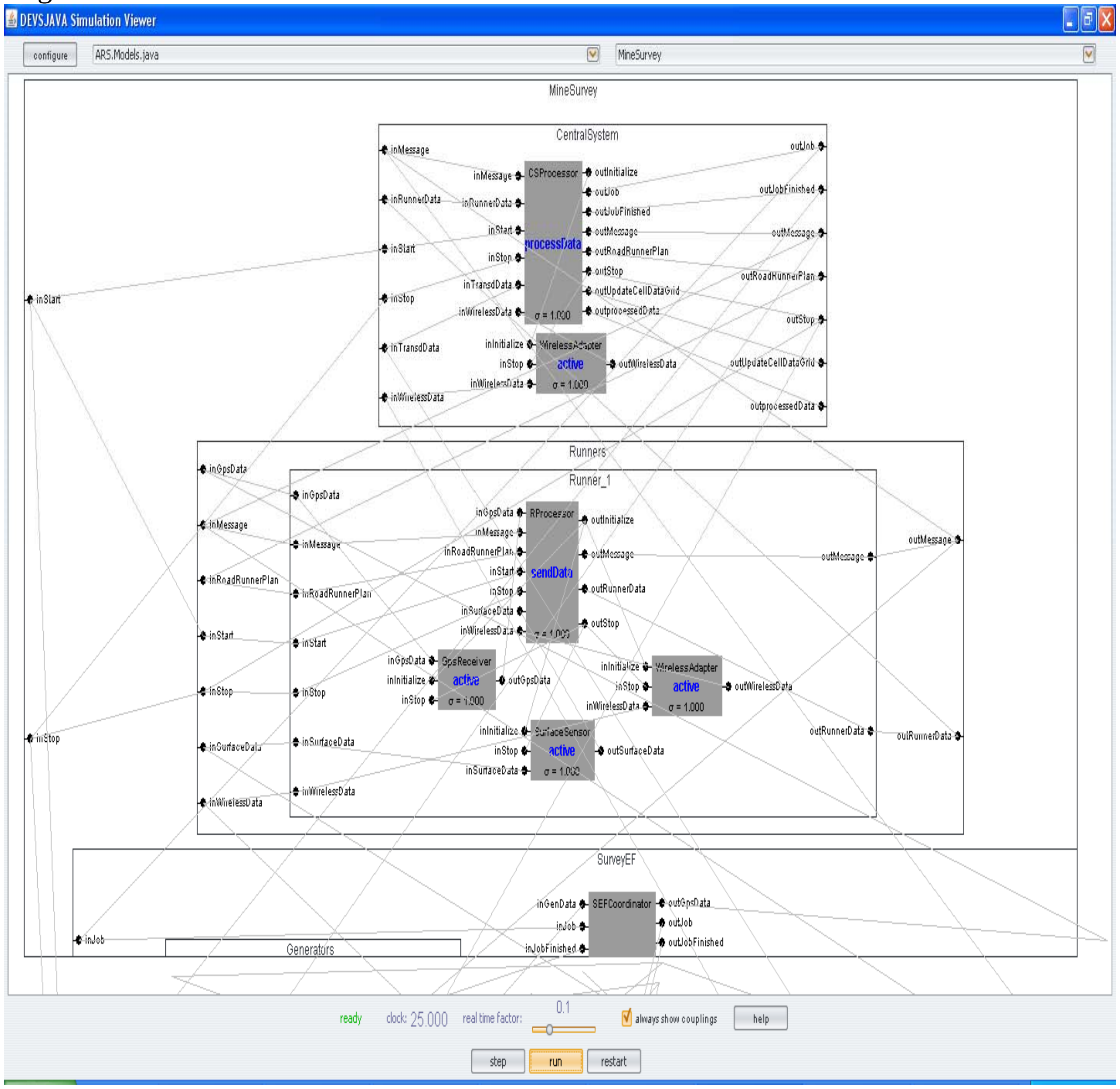


Figure 4.7 DEVSJAVA SimView running system under development

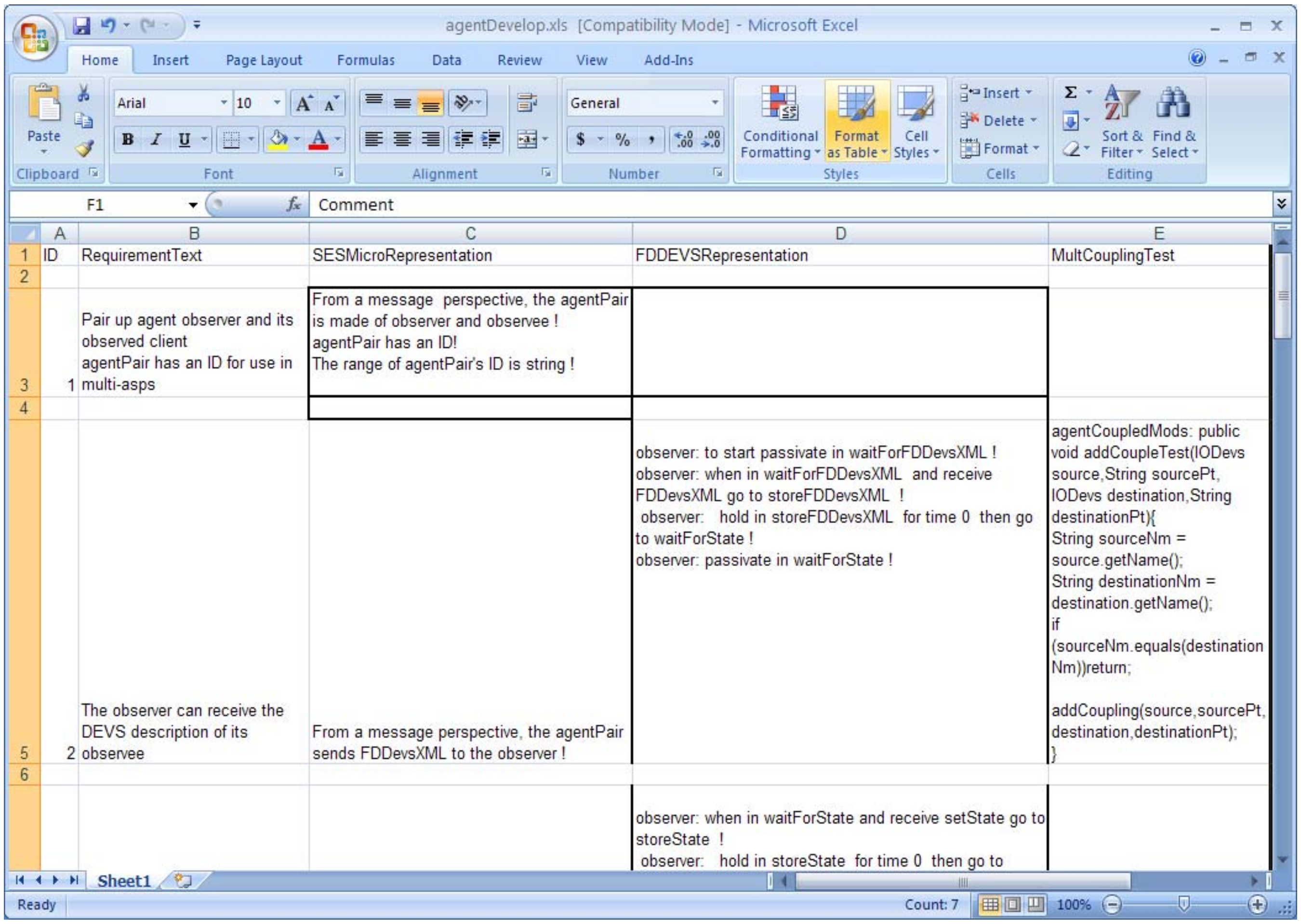
As seen in Figure 4.7, SimView allows the user to stop, run and restart the execution of the model being simulated. This helps the developers detect flaws easily when simulating and testing the system, as it shows the transactions and state transitions for each of the models that are being executed. SimView provides the flexibility to choose and run the desired model and simulate different models without having to reopen the SimView application. In addition, DEVSJAVA SimView allows the developer to simulate hierarchical models. These models are coupled models with components that may be atomic or coupled models that constitute a part of the entire system. This permits to simplify the scope of testing and find problems at different stages of the development, i.e. execute unit simulation to analyze models more concretely.

One very useful feature of the AutoDEVS tool is that it displays the model's debugging messages into an execution log textbox which is integrated in the tool. This permits the developer to quickly debug the system and resolve any bugs encountered during simulation or just monitor the application in execution. SimView also provides the capability to adjust the simulation speed of the models by increasing or decreasing the execution time by a scale factor. For example, a scale factor equals 1 means the simulation will run at the same speed as a wall clock; time scale factor equals to 0.5 means the simulation will run twice as fast as the wall clock, and so on. This allows the designer to analyze the data to determine if the system under test fulfills the logical behavior as desired.

4.3 AutoDEVS & Model Continuity

AutoDEVS is a software development methodology that supports model continuity for distributed real-time software development. This methodology is based on the DEVS modeling and simulation framework. Corresponding to the general "Design-Test-Execute" development procedure, this tool provides a "Modeling-Simulation-Execution" process which includes several stages to develop real-time software. During these stages, a model's continuity is maintained because the same control models that are created will be tested by simulation methods and then deployed to the target system for execution.

Next, is a description of the different stages that AutoDEVS methodology utilizes to develop systems showing model continuity. This description takes advantage of an Agent Development System simple example. The first stage of the AutoDEVS methodology is defining user requirements for the Agent Development System:

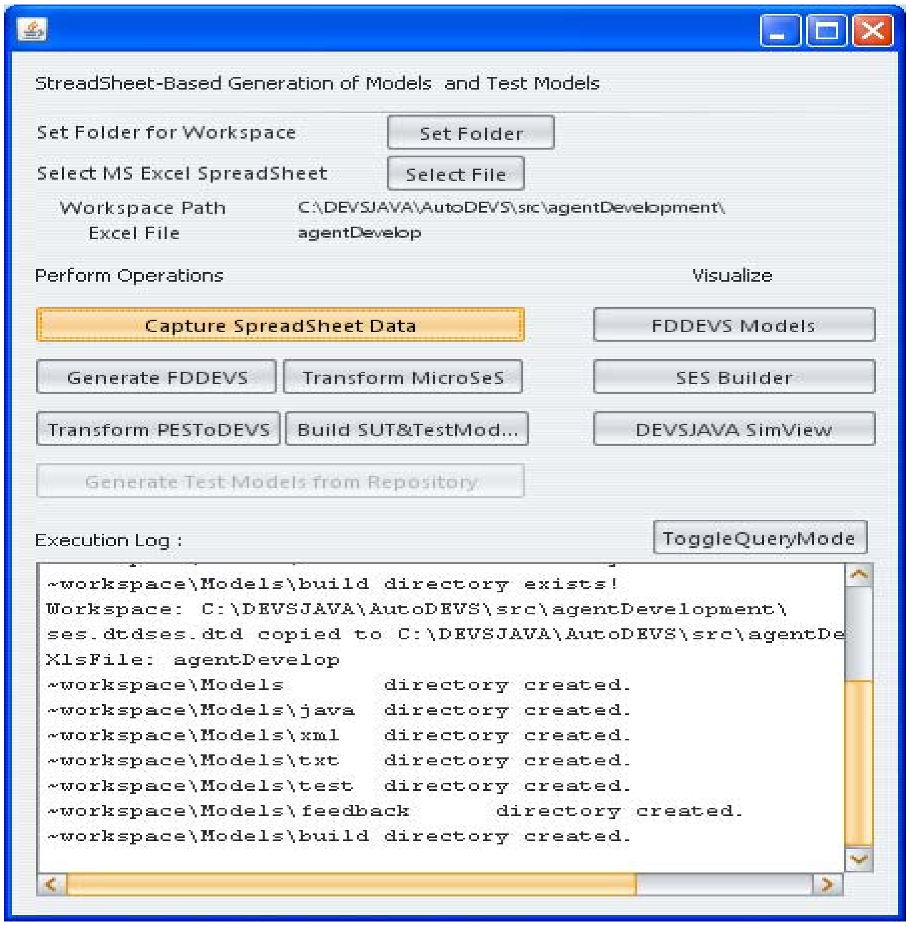


F1 | Comment

	A	B	C	D	E
1	ID	RequirementText	SESMicroRepresentation	FDDEVSRepresentation	MultCouplingTest
2					
3	1	Pair up agent observer and its observed client agentPair has an ID for use in multi-asps	From a message perspective, the agentPair is made of observer and observee ! agentPair has an ID! The range of agentPair's ID is string !		
4					
5	2	The observer can receive the DEVS description of its observee	From a message perspective, the agentPair sends FDDevsXML to the observer !	observer: to start passivate in waitForFDDevsXML ! observer: when in waitForFDDevsXML and receive FDDevsXML go to storeFDDevsXML ! observer: hold in storeFDDevsXML for time 0 then go to waitForState ! observer: passivate in waitForState !	agentCoupledMods: public void addCoupleTest(IODevs source,String sourcePt, IODevs destination,String destinationPt){ String sourceNm = source.getName(); String destinationNm = destination.getName(); if (sourceNm.equals(destinationNm))return; addCoupling(source,sourcePt, destination,destinationPt); }
6					
				observer: when in waitForState and receive setState go to storeState ! observer: hold in storeState for time 0 then go to	

Figure 4.8 Agent Development Example: Define Requirements

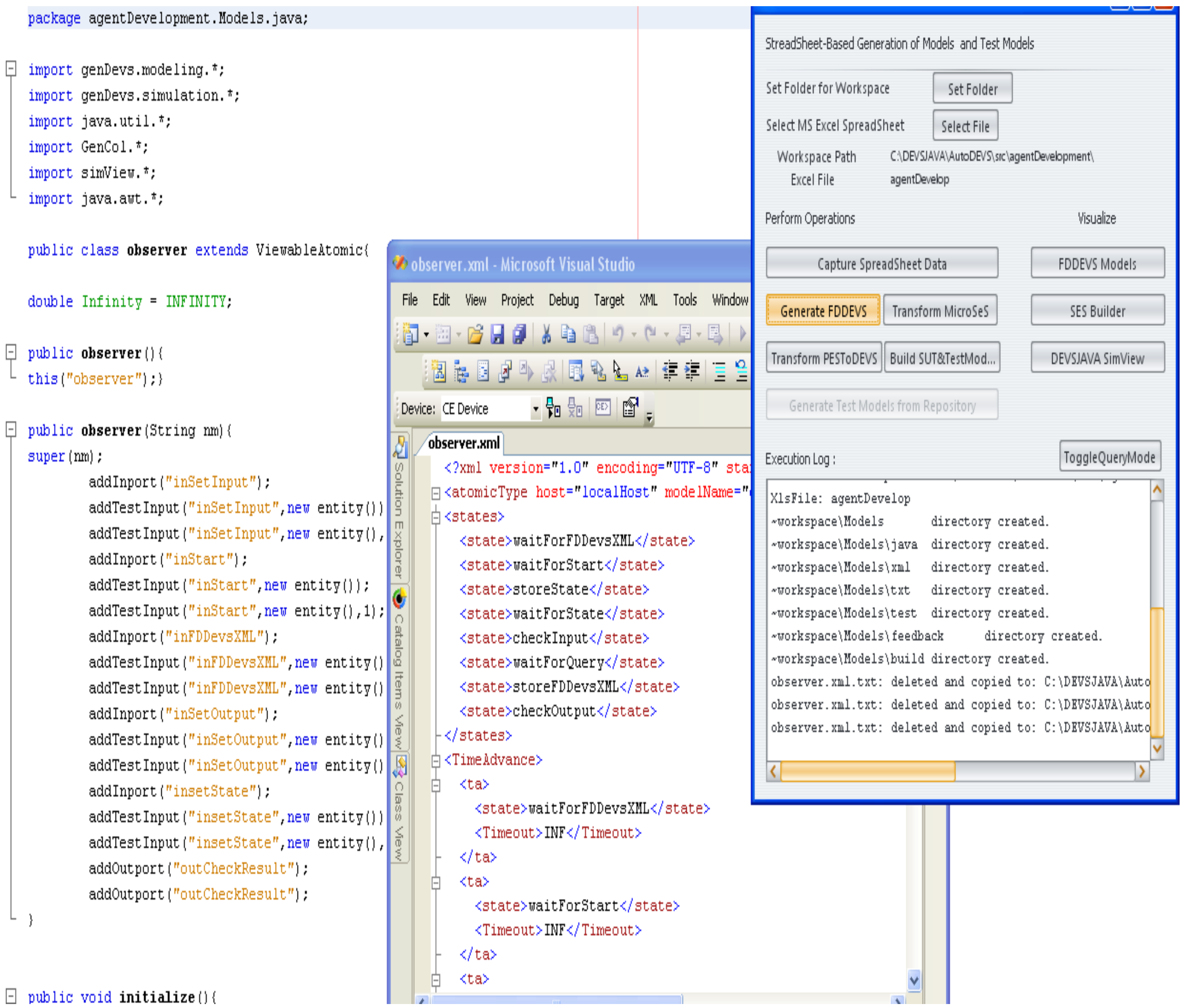
As seen in Figure 4.8, the requirements for the new system are collected and organized in spreadsheet table, i.e. “RequirementText” column. The second stage is describing the structural aspects for each of the requirements of the Agent Development System, i.e. fill the “SESMicroRepresentation” column of the spreadsheet. The third stage is describing the behavioral aspects for each of the requirements, i.e. fill the “FDDEVSRepresentation” column of the spreadsheet. The fourth stage is defining the multi-aspect coupling for the coupled models which automates the coupling generation for multi-aspect models. This is defined in the “MultiCouplingTest” column of the spreadsheet, i.e. agent input/output coupling. The fifth stage is running the AutoDEVS tool to capture the spreadsheet data, i.e. agentDevelop.xls, see Figure 4.9.



```
~workspace\Models\build directory exists!
Workspace: C:\DEVSJAVA\AutoDEVS\src\agentDevelopment\
ses.dtdses.dtd copied to C:\DEVSJAVA\AutoDEVS\src\agentDe
XlsFile: agentDevelop
~workspace\Models       directory created.
~workspace\Models\java  directory created.
~workspace\Models\xml   directory created.
~workspace\Models\txt   directory created.
~workspace\Models\test  directory created.
~workspace\Models\feedback       directory created.
~workspace\Models\build directory created.
```

Figure 4.9 Agent Development Example: Capture Spreadsheet Data

Notice that the user needs to set the folder and spreadsheet file to be captured in the AutoDEVS tool, i.e. “Set Folder” and “Select File” buttons. In addition, notice that subsequent to capturing the data from the spreadsheet, AutoDEVS encodes the captured data into an XML schema/document type definition (XSD or DTD), i.e. Sheet1agentDevelopRowsSchema.xsd, ses.dtd. The sixth stage is to generate FDDEVS models based on the schema type definition and the captured XML data from previous stage, i.e. behavioral aspects of the Agent Development System (FDDEVSRepresentation column).



```
package agentDevelopment.Models.java;

import genDevs.modeling.*;
import genDevs.simulation.*;
import java.util.*;
import GenCol.*;
import simView.*;
import java.awt.*;

public class observer extends ViewableAtomic{

double Infinity = INFINITY;

public observer(){
this("observer");}

public observer(String nm){
super(nm);
        addInport("inSetInput");
        addTestInput("inSetInput",new entity())
        addTestInput("inSetInput",new entity(),
        addInport("inStart");
        addTestInput("inStart",new entity());
        addTestInput("inStart",new entity(),1);
        addInport("inFDDevsXML");
        addTestInput("inFDDevsXML",new entity()
        addTestInput("inFDDevsXML",new entity()
        addInport("inSetOutput");
        addTestInput("inSetOutput",new entity()
        addTestInput("inSetOutput",new entity()
        addInport("insetState");
        addTestInput("insetState",new entity())
        addTestInput("insetState",new entity(),
        addOutport("outCheckResult");
        addOutport("outCheckResult");
}

public void initialize(){
```

observer.xml - Microsoft Visual Studio

```
<?xml version="1.0" encoding="UTF-8" sta
<atomicType host="localHost" modelName="
<states>
  <state>waitForFDDevsXML</state>
  <state>waitForStart</state>
  <state>storeState</state>
  <state>waitForState</state>
  <state>checkInput</state>
  <state>waitForQuery</state>
  <state>storeFDDevsXML</state>
  <state>checkOutput</state>
</states>
<TimeAdvance>
  <ta>
    <state>waitForFDDevsXML</state>
    <Timeout>INF</Timeout>
  </ta>
  <ta>
    <state>waitForStart</state>
    <Timeout>INF</Timeout>
  </ta>
  <ta>
```

StreadSheet-Based Generation of Models and Test Models

Set Folder for Workspace	Set Folder
Select MS Excel SpreadSheet	Select File
Workspace Path	C:\DEVSJAVA\AutoDEVS\src\agentDevelopment\
Excel File	agentDevelop

```
XlsFile: agentDevelop
~workspace\Models       directory created.
~workspace\Models\java  directory created.
~workspace\Models\xml   directory created.
~workspace\Models\txt   directory created.
~workspace\Models\test  directory created.
~workspace\Models\feedback      directory created.
~workspace\Models\build directory created.
observer.xml.txt: deleted and copied to: C:\DEVSJAVA\Auto
observer.xml.txt: deleted and copied to: C:\DEVSJAVA\Auto
observer.xml.txt: deleted and copied to: C:\DEVSJAVA\Auto
```

Figure 4.10 Agent Development Example: Generate FDDEVS

As seen in Figure 4.10, DEVS java models are automatically generated, including an XML representation of those models, i.e. observer.java, observer.xml. Based on the MicroSESRepresentation column defined in the spreadsheet, the seventh stage is to add the structural aspects on the DEVS models created in the previous stage. During this stage a SES representation of the models is created and parsed into an XML file, i.e. agentDevelopagentCoupledModsSeS.xml. Notice that this file could serve to see the SES representation as a tree view, see Figure 4.11. In addition, an automatic PES that represents the logically possible set state descriptions consistent with the SES is created, i.e. agenDevCoupModInst.xml.

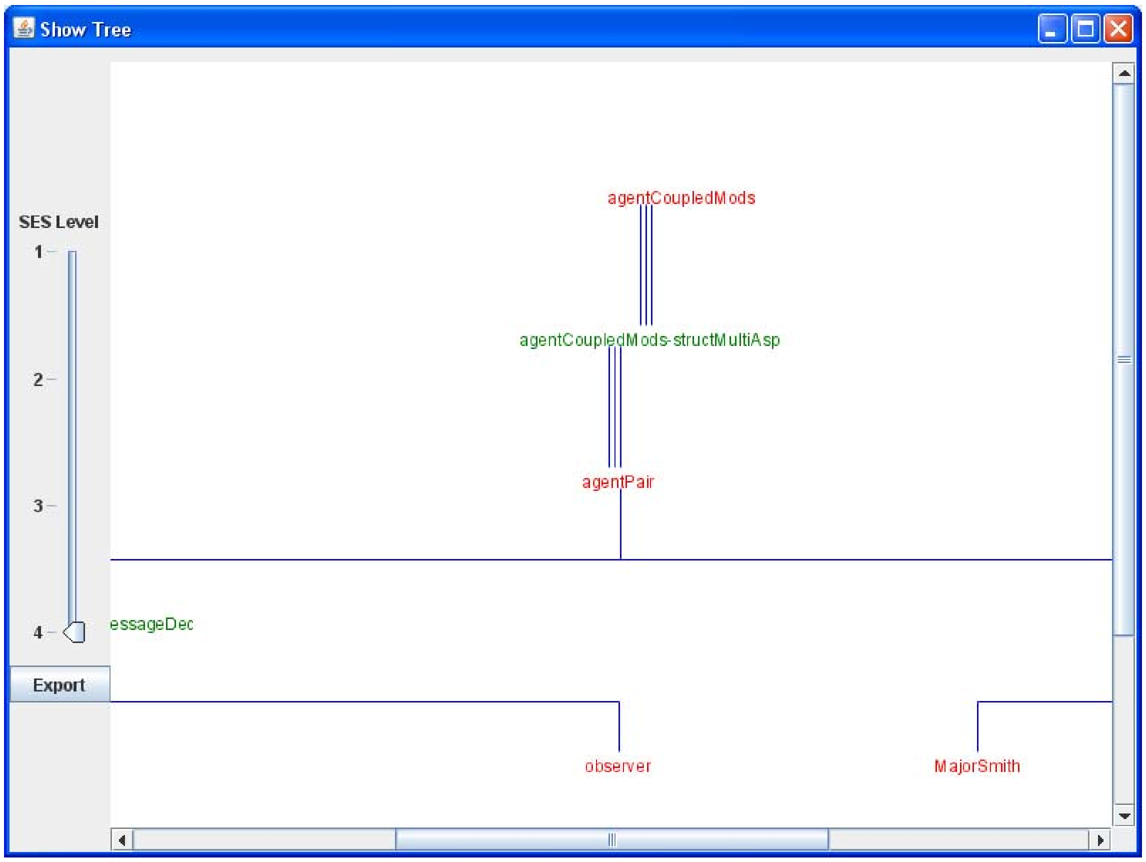
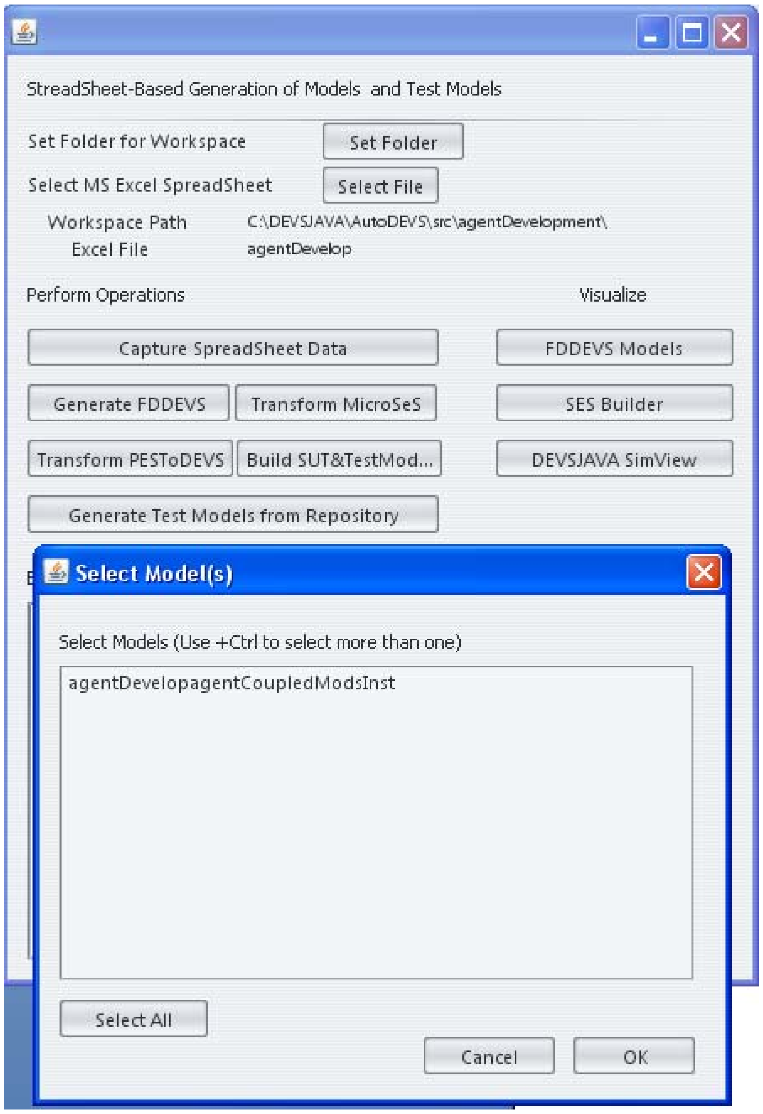


Figure 4.11 Agent Development Example: SES Tree View

The eighth stage is to run the PES that was automatically created in the previous stage, i.e. Transform PEStoDEVS. During this stage the specialized models are created and the DEVS models are updated with the corresponding structural aspects, i.e. PES transformed into DEVSJAVA models. This PES can also be modified by the developer to create his own pruning and analyze the models of interest. The AutoDEVS tool allows choosing the PES desired and then run it using the tool, as shown in Figure 4.12.



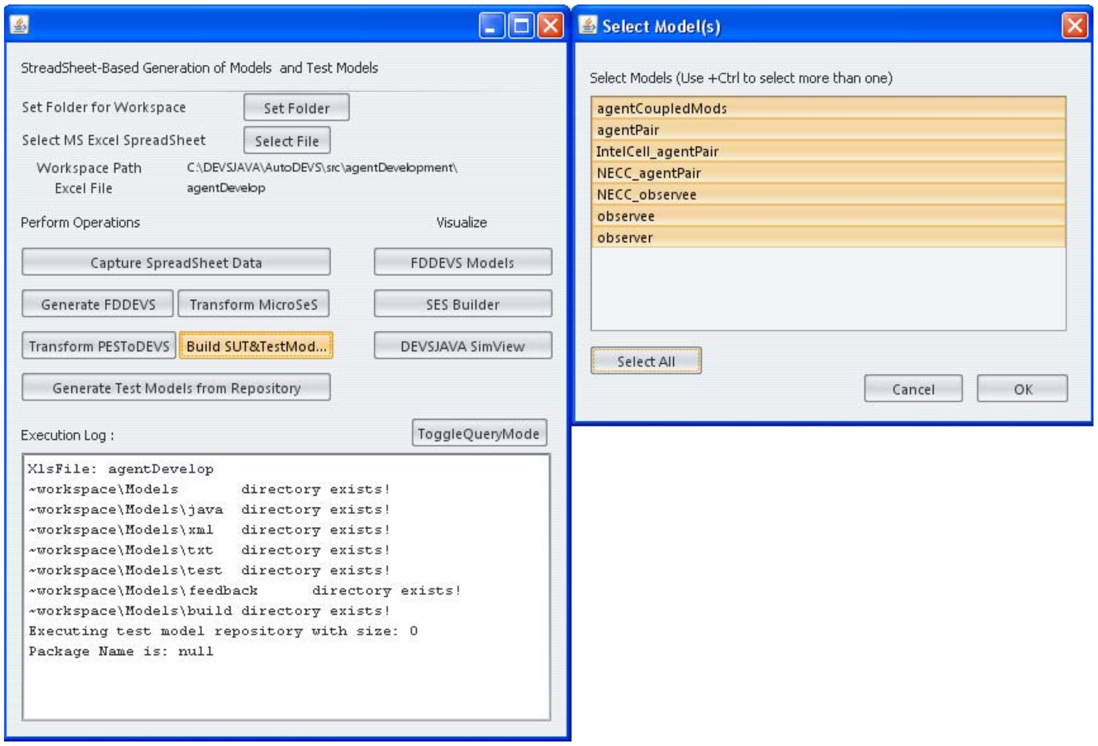
Workspace Path C:\DEVSJAVA\AutoDEVS\src\agentDevelopment\
Excel File agentDevelop

Select Models (Use +Ctrl to select more than one)

agentDevelopagentCoupledModsInst

Figure 4.12 Agent Development Example: Choosing a PES

The ninth stage is to create a set of test models to validate the system under development, see Figure 4.16.



Workspace Path C:\DEVSJAVA\AutoDEVS\src\agentDevelopment\
Excel File agentDevelop

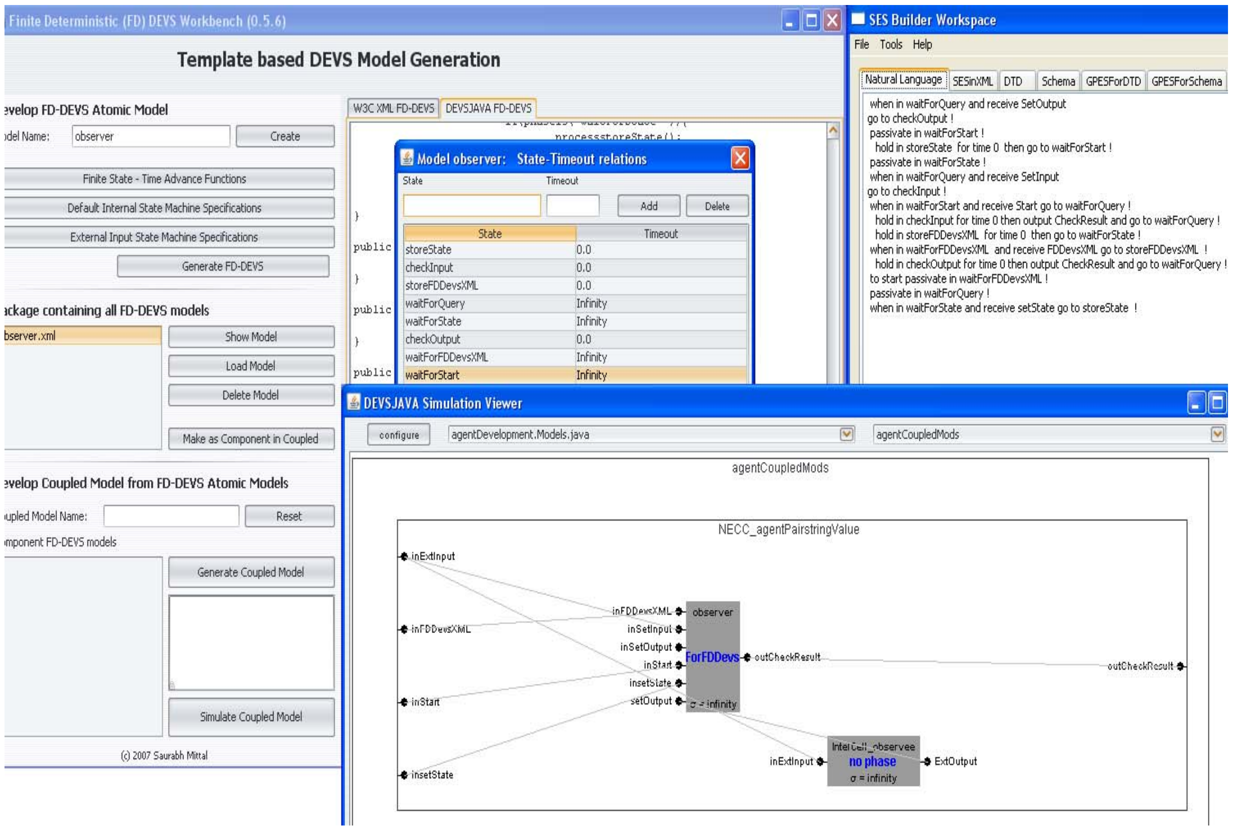
```
XlsFile: agentDevelop
~workspace\Models       directory exists!
~workspace\Models\java  directory exists!
~workspace\Models\xml   directory exists!
~workspace\Models\txt   directory exists!
~workspace\Models\test  directory exists!
~workspace\Models\feedback      directory exists!
~workspace\Models\build directory exists!
Executing test model repository with size: 0
Package Name is: null
```

agentCoupledMods
agentPair
IntelCell_agentPair
NECC_agentPair
NECC_observee
observee
observer

Figure 4.13 Agent Development Example: Generate Test Models

As seen in Figure 4.13, AutoDEVS lets the developer choose the test models to create from a list given. As described previously, these test models are based on minimal testable I/O pairs restricted to messages and are created with the objective to verify the correctness of the DEVS models. This feature is being developed and will be available in the near future.

The tenth stage is to verify the models created in the FDDEVS Models and SES Builder DEVSJAVA SimView applications, i.e. Figure 4.14. Then modify the models accordingly to the desired needs and start simulation, see Figure 4.15.



```
public
}
public
}
public
```

State	Timeout
storeState	0.0
checkInput	0.0
storeFDDevsXML	0.0
waitForQuery	Infinity
waitForState	Infinity
checkOutput	0.0
waitForFDDevsXML	Infinity
waitForStart	Infinity

```
 when in waitForQuery and receive SetOutput
go to checkOutput !
 passivate in waitForStart !
  hold in storeState  for time 0  then go to waitForStart !
 passivate in waitForState !
 when in waitForQuery and receive SetInput
go to checkInput !
 when in waitForStart and receive Start go to waitForQuery !
  hold in checkInput for time 0 then output CheckResult and go to waitForQuery !
  hold in storeFDDevsXML  for time 0  then go to waitForState !
 when in waitForFDDevsXML  and receive FDDevsXML go to storeFDDevsXML  !
  hold in checkOutput for time 0 then output CheckResult and go to waitForQuery !
 to start passivate in waitForFDDevsXML !
 passivate in waitForQuery !
 when in waitForState and receive setState go to storeState  !
```

Figure 4.14 Agent Development Example: Verifying models in SES, FDDEVS, and SimView

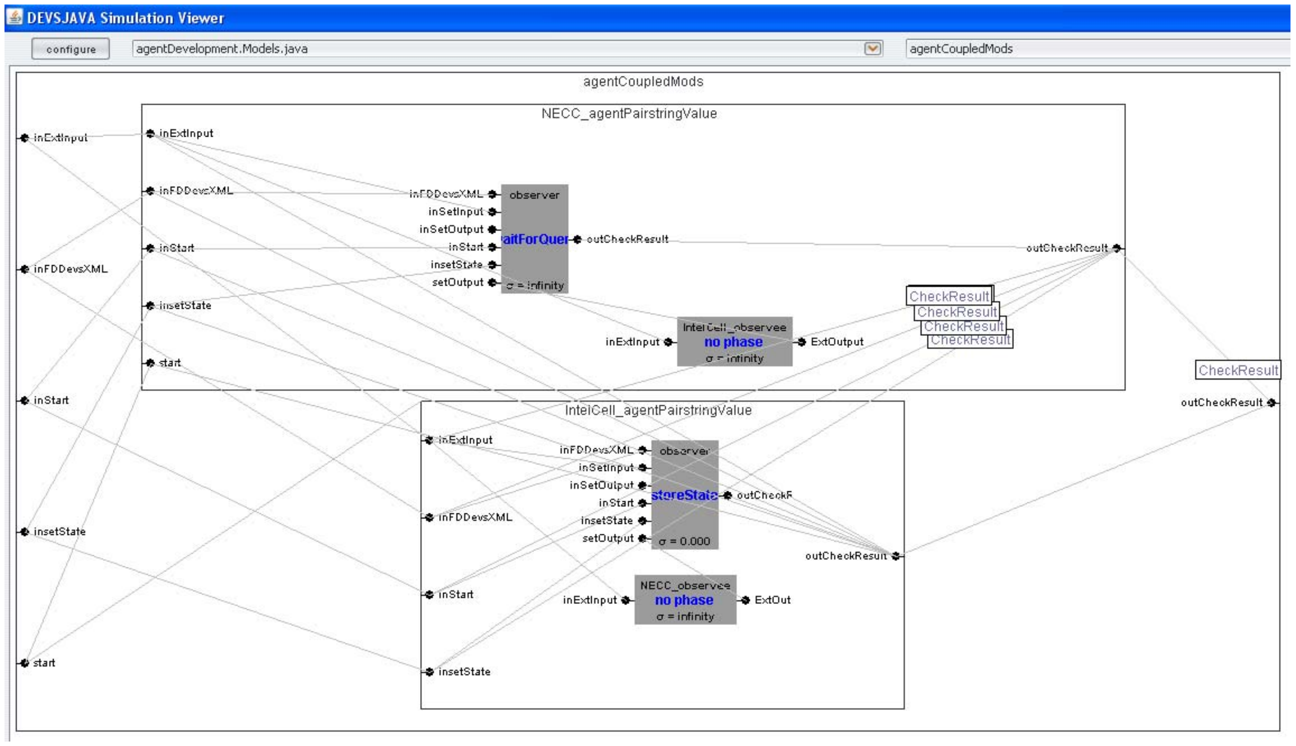


Figure 4.15 Agent Development Example: Running models in SimView

4.4 Mapping AutoDEVS-generated Models to DEVS/SOA

The DEVS/SOA tool as described in section 3.5 facilitates the developer to create transparent distributed simulation. It exploits model continuity by taking the generated AutoDEVS models and allowing the upload, compilation and simulation of the developed models in a distributed way among different available servers. AutoDEVS and DEVS/SOA tools minimize the number of modifications required to the developed source code to make possible the distributed simulation (see [Sal08] for details). By distributing the simulation of the system developed in different machines, the memory requirements on any single system can be substantially smaller than the memory used in a single-workstation simulation. The developer needs to consider optimizing the transactions between the different workstations to allow the overall execution time of the simulation to be at least as fast as the original single-workstation simulation. Networking or distributing the simulation of the generated AutoDEVS models using the DEVS/SOA tool encourages improving the speed of highly parallelizable tasks by distributing pieces of the system across many computers that together form a distributed computing simulation system.

Furthermore, AutoDEVS together with DEVS/SOA provide an infrastructure to implement structural and hierarchical real-time distributed systems. In particular DEVS/SOA supports the fourth step where all the models are deployed to their target platforms and tested in the real physical environment. In this case, the target platforms and physical environment are the SOA services and underlying network infrastructure in which DEVS is implemented. The user can determine the simulation mode of a model by selecting the simulation protocol to use, whether logical or real-time, without changing the model itself. When executing under the real-time simulation model, the model effectively becomes an executing web service that operates in real-time and can be distributed to geographically dispersed nodes as desire. An example application to support of negotiation is given by [Hwa06] .

4.5AutoDEVS Applications to M&S for Defense Test and Evaluation

Modeling and Simulation (M&S) is finding increasing application in defense such as the incorporation of M&S functionality into command and control systems [Pul08] and the use of M&S to support the development and testing of System of Systems (SoS). The SoS concept relates to the attempt to integrate disparate systems to achieve new requirements with the defining concern being interoperability, or lack thereof, among the constituent system [Pul08, Sag07]. Achieving such interoperability is among the chief SoS engineering objectives in the development of command and control (C2) capabilities for joint and coalition warfare [Jac04, Zei08]. DEVS/SOA operation over web middleware enables it to operate within the net-centric environment of the Global Information Grid/Service Oriented Architecture (GIG/SOA) which is being developed to achieve a new level of defense interoperability. The development of AutoDEVS enhances the quality of solutions provided by DEVS technology to support Test and Evaluation (T&E) of interoperability in defense SoS, particularly in net-centric environments such as GIG/SOA [Zei05, Bri08]. Since AutoDEVS can support model continuity through a simulation-based development and testing life cycle, the mapping of high-level requirement specifications into lower-level DEVS formalizations enables such specifications to be thoroughly tested in virtual simulation environments before being easily and consistently transitioned to operate in a real environment for further testing and fielding. Indeed, the DEVS Unified Process [Mit07, Mit08a] provides methodology and SOA infrastructure for integrated development and testing. Extended DoD Architectural Framework (DoDAF) views support executable architectures using M&S for application to mission based testing for GIG/SOA [Zei05a, Mit08b]. The DEVS simulation architecture is layered to support technology neutrality and execution in different technological scenarios [Mit07a, Mit08a, Sar01a]. Separating a model from the act of simulation itself, which can be executed on single or multiple distributed platforms supports automated test generation and deployment in distributed simulation [HuX03].

5. Conclusions

We have presented a new DEVS-based tool called “AutoDEVS” that automates systems development and exploits “model continuity” to maintain coherence through the development process and to support test artifact continuity and traceability through phases of system development. AutoDEVS creates models and systems that can be mapped to DEVS/SOA, a DEVS implementation compliant with a Service Oriented Architecture infrastructure such as adopted by the DoD’s Global Information Grid initiative. Therefore, it provides a rapid means of integrated development and testing of defense command and control systems for interoperability.

Achieving interoperability is one of the chief System of Systems (SoS) engineering objectives in the development of command and control (C2) capabilities for joint and coalition warfare. The importance of M&S in SoS design and evaluation cannot be underestimated. M&S can be used strategically to provide early feasibility studies and aid the design process. As components comprising SoS are designed and analyzed, their integration and communication is the most critical part that must be addressed by the employed SoS M&S framework. The integration infrastructure must support interoperability at syntactic, semantic and pragmatic levels to enable such integration [Zei07, Zei08].

Currently there are several other approaches to distributed simulation and to integration of M&S with advanced C2 systems. These approaches build on the internet or other net-centric middleware to provide component connectivity and simulation services [Pul08, Rei02]. DEVS provides a formal systems-based abstraction that can support higher level interoperability, whether alone or on top of HLA. The DEVS/SOA implementation provides a SOA implementation independent of HLA and is a viable approach to M&S integration with C2 SoS. DEVS components including decision making agents, sensor simulators, and environmental representations can bring the power of M&S to the development of C2 SoS as well as serving as support for command and control in real operation. The underlying SOA standard that facilitates this interoperation can be expected to be widely implemented following its adoption by the DoD’s Global Information Grid initiative.

6. REFERENCES

[Acc00] Accelerating Embedded e-development, Rational Rose Real Time, http://www.ghs.com/partners/rational/rose-rt.pdf

[Alb08] Jeannie Albrecht, Ryan Braud, Charles Killian, Priya Mahadevan, Kashi Vishwanath, and Amin Vahdat, An integrated software environment for distributed systems development, http://sysnet.cs.williams.edu/~jeannie/papers/plush-spie.pdf

[Ant01] Magnus Antonson, Pernilla Hansson, Modeling of Real-Time Systems in UML with Rational Rose and Rose Real-Time based on RUP, http://www.google.com/url?sa=U&start=6&q=http://rise.uni.lu/tiki/se2c-bib_download.php%3Fid%3D455&usg=AFQjCNFqw153nT_4kw1OJ8hdmvqJ7D123w

[Bri08] Steven Bridges, Bernard P. Zeigler, James Nutaro, Dane Hall, Tom Callaway, and Dale Fulton, Evolving Enterprise Infrastructure for Model & Simulation Based Testing of Net-Centric Systems, ITEA journal, 2008; Vol. 29, pps 51-61.

[Can97] Claudio A. Cañizares, Advantages and Disadvantages of Using Various Computer Tools in Electrical Engineering Courses, http://www.power.uwaterloo.ca/~claudio/papers/trace.pdf

[Cms08] CMS, Selecting a Development Approach, http://www.cms.hhs.gov/SystemLifecycleFramework/Downloads/SelectingDevelopmentApproach.pdf

[Hon97] J.S. Hong, and T.G. Kim, "Real-time Discrete Event System Specification Formalism for Seamless Real-time Software Development," Discrete Event Dynamic Systems: Theory and Applications, vol. 7, pp.355-375, 1997.

[HuX03] Hu, X., Zeigler, B.P., Mittal, S., "Dynamic Configuration in DEVS Component-based Modeling and Simulation", SIMULATION: Transactions of the Society of Modeling and Simulation International, November 2003

[HuX04] Hu Xiaolin, A Simulation-based software development methodology for distributed real-time systems, *http://acims.arizona.edu/PUBLICATIONS/PDF/Xiaolin_dissertation.pdf*.

[HuX05] Hu, Xiaolin and B.P. Zeigler, "Model Continuity in the Design of Dynamic Distributed Real-Time Systems", *IEEE Transactions On Systems, Man And Cybernetics— Part A: Systems And Humans, 35: 6, pp. 867- 878*, November, 2005

[Jac04] Jacobs, R.W. "Model-Driven Development of Command and Control Capabilities For Joint and Coalition Warfare," Command and Control Research and Technology Symposium, June 2004.

[Jad08] Joint Application Development. http://en.wikipedia.org/wiki/Joint_application_development

[Hwa06] Moon Ho Hwang and Bernard P. Zeigler, A Reachable Graph of Finite and Deterministic DEVS Networks, DEVS Integrative M&S Symposium (DEVS'06) (DEVS 2006) Huntsville, Alabama, USA, April 2 - 6, 2006

[Kil07] C. Killian, J. W. Anderson, R. Braud, R. Jhala, A. Vahdat, Mace: language support for building distributed systems, Proc. Program. Lang. Design Implem., 2007. (accessed 21 Mar 2008) http://mace.ucsd.edu

[Lew08] http://codebetter.com/blogs/raymond.lewallen/archive/2005/07/13/129114.aspx

[Mar91] Martin, James, "Rapid Application Development", Macmillan Publishing Co., Inc., 1991.

[Mat08] The MathWorks, http://www.mathworks.com/

[Mit07] Saurabh Mittal, DEVS unified process for integrated development and testing of service and testing fo service oriented architectures, Ph. D. Dissertation, Univ. of Arizona, 2007.

[Mit07a] Mittal, S., Martin, J.L.R., Zeigler, B.P., "DEVSML: Automating DEVS Execution over SOA Towards Transparent Simulators", Special Session on DEVS Collaborative Execution and Systems Modeling over SOA, DEVS Integrative M&S Symposium DEVS' 07, Spring Simulation Multi-Conference, March 2007

[Mit08] Saurabh Mittal, Bernard P. Zeigler, Moon Ho Hwang , XFD-DEVS, http://www.saurabh-mittal.com/fddevs/

[Mit08a] Mittal, S., Zeigler, B.P., "*DEVS Unified Process for Integrated Development and Testing of System of Systems*", Critical Issues in C4I, AFCEA-George Mason University Symposium, May 2008

[Mit08b] 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

[Pal06] Palaniappan, S., Sawheny, A., Sarjoughian, H.S., "Application of DEVS Framework in Construction Simulation", Winter Simulation Conference, Monterey, CA, http://acims.arizona.edu/PUBLICATIONS/PDF/constructionSim.pdf

[Pto08] The Ptolemy project, http://ptolemy.eecs.berkeley.edu/

[Pul08] Pullen, M., Wilson, L.T.C.K, Hieb, M., Tolk, A., "Extensible Modeling and Simulation Framework (XMSF) C4I Testbed," http://www.movesinstitute.org/xmsf/xmsf.html

[Rad08] Rapid Application Development. http://en.wikipedia.org/wiki/Rapid_application_development

[Ras01] Rastofer, U.; Bellosa, F., Component-based software engineering for distributed embedded real-time systems, Software, IEE Proceedings, Volume: 148 Issue: 3, June 2001

[Rei02] Reichenthal, S.W., SRML - Simulation Reference Markup Language W3C Note 18 December 2002 http://www.w3.org/TR/SRML/

[Rob00] Paul Robertson, Robert Laddaga, and Howie Shrobe, "Introduction: the first international workshop on self-adaptive software", Lecture Notes in Computer Science, 2000, pp. 1-10

[Rts08] SESBuilder, http://www.rtsync.com/services/SESBuilder.html

[Sae08] Saehoon Cheon, Doohwan Kim, Bernard P Zeigler, System Entity Structure For XML Meta Data Modeling; Application to the US Climate Normals, IEEE International Conference on Information Reuse and Integration, Las Vegas, NV, July 2008.

[Sag07] Sage, A., "From Engineering a System to Engineering an Integrated System Family, From Systems Engineering to System of Systems Engineering", 2007 IEEE International Conference on System of Systems Engineering (SoSE). April 16th -18th, 2007, San Antonio, Texas

[Sal08] Salas, M.C., AutoDEVS: A Methodology for Automating Systems Development, Master's Thesis, Electrical and Computer Engineering Dept., Univ. of Arizona, 2008. http://www.acims.arizona.edu/PUBLICATIONS/PDF/Salas_Thesis.pdf

[Sar01] Hessam S. Sarjoughian, Francois E. Cellier, Discrete Event Modeling and Simulation Technologies: A Tapestry of Systems and AI-Based Theories and Methodologies, p.31, Springer 2001.

[Sar01a] Sarjoughian, H., Zeigler, B.P., and Hall, S., "A Layered Modeling and Simulation Architecture for Agent-Based System Developmen"t, Proceedings of the IEEE 89 (2); 201-213, 2001

[Sdl08] Systems Development Life Cycle. http://en.wikipedia.org/wiki/Systems_Development_Life_Cycle

[Ses08] http://www.sesbuilder.com/

[Sho98] Shokri, E.; Crane, P.; Kim, K., An implementation model for time-triggered message-triggered objectsupport mechanisms in CORBA-compliant COTS platforms, http://ieeexplore.ieee.org/Xplore/login.jsp?url=/iel4/5419/14648/00666764.pdf?temp=x, 1998

[Soa08] Service-oriented architecture, http://en.wikipedia.org/wiki/Service-oriented_architecture

[Vah02] A. Vahdat, K. Yocum, K. Walsh, P. Mahadevan, D. Kostic, J. Chase, D. Becker, Scalability and accuracy in a large-scale network emulator, *Proc. 5th USENIX OSDI Symp.*, 2002. (Accessed 21 Mar 2008) http://modelnet.ucsd.edu

[Wel01] Wells, R.B.; Fisher, J.; Ying Zhou; Johnson, B.K.; Kyte, M.: Hardware and software considerations for implementing hardware-in-the-loop traffic simulation. Industrial Electronics Society, 2001. IECON '01. The 27th Annual Conference of the IEEE , Volume: 3 , 2001

[Woo95] Wood, Jane and Silver, Denise; Joint Application Development, John Wiley & Sons Inc,   ISBN 0-47104-299-4

[Wri08] http://www.garywwright.com/sdlc.php

[Zei00] B.P. Zeigler, T.G.Kim and H. Praehofer, "Theory of Modeling and Simulation: Integrating Discrete Event and Continuous Complex Dynamic Systems," second edition Academic Press, Boston, 2000.

[Zei03] B. P.Zeigler, DEVS Today: Recent Advances in Discrete Event-based Information Technology, MASCOTS' 03, Orlando, FL, October 2003.

[Zei05] Zeigler, B.P., Fulton, D., Hammonds, P., Nutaro, J., "Framework for M&S Based System Development and Testing in Net-centric Environment", ITEA Journal, Vol. 26, No. 3, October 2005

[Zei05a] Zeigler, B. P., Mittal, S., "Enhancing DoDAF with DEVS-Based System Life- cycle Process", IEEE International Conference on Systems, Man and Cybernetics, Hawaii, October 2005

[Zei07] B.P. Zeigler and P. Hammond, "Modeling&Simulation-Based Data Engineering: Introducing Pragmatics into Ontologies for Net-Centric Information Exchange", Academic Press, Boston, 2007. 448 pages

[Zei08] Zeigler, B.P., Mittal, S., Hu, X., "Towards a Formal Standard for Interoperability in M&S/Systems of Systems Engineering", Critical Issues in C4I, AFCEA-George Mason University Symposium, May 2008