Agent Implemented Experimental Frames for Net-Centric Systems Test and Evaluation

Bernard P. Zeigler,
Arizona Center for Integrative Modeling and Simulation

Dane Hall
BAE Systems

Manuel Salas
Modular Mining Systems

Prepared for:

Agent-Directed Simulation and Systems Engineering
Eds: Levent Yilmaz, Tuncer I. Ören,
Series on Systems Engineering
Wiley-VCH

http://www.eng.auburn.edu/~yilmaz/ADSSE.htm

Contents

1. Introduction

This chapter is concerned with test and evaluation of net-centric systems. Such systems are characterized as compositions of nodes distributed over data networks with information exchanges among nodes playing a role as essential as the processing within nodes. As with their conventional counterparts, the design of net-centric systems should follow a structured life cycle development methodology. Particularly, we emphasize a methodology that emphasizes the role of requirements and the progression from requirements to both implementation, and testing of implementations, in an integrated manner. As illustrated in Figure 1, the development and testing methodology bifurcates the process into two main streams – system development and test suite development – that converge in the system testing phase which includes test and evaluation, validation, and verification. The methodology provides a process to transform a requirements specification to a Discrete Event Systems Specification (DEVS) [Zei00] representation supporting evaluation and recommendations for a feasible design.

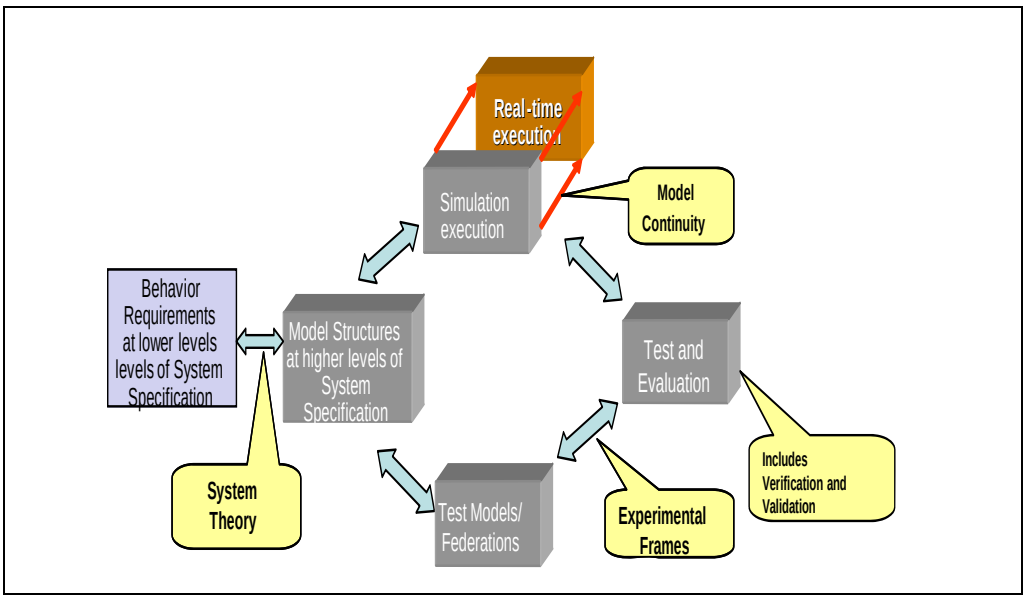


Figure 1. Bifurcated Development and Testing Methodology

The bifurcated life cycle process combines systems theory [Wym93], modeling and simulation (M&S) framework, and model-continuity concepts. The system development includes the definition of requirements, capture of specifications to map to formalized DEVS model components, and creation of a reference master model. Model continuity is exploited to execute the model using the DEVS real-time execution protocol. The test suite development includes execution of test models to run against the system under test whether the system is developed by leveraging models (i.e., via model-continuity) or is a system developed from the requirements existing independent of the executable models. In particular for net-centric systems, the test models constitute experimental frames [Öre79, Öre05], which can be distributed to observe the behaviors of, and information exchanges among, the nodes of the system. The process is iterative, allowing return to modify the reference master DEVS model and the requirements specifications. Model continuity minimizes the artifacts that have to be modified as the process proceeds and increases the coherence of the artifacts across development stages [HuX04, HuX05, Mit07].

In what follows, we elaborate on the role of requirements in the bifurcated methodology and the progression to development of test models for net-centric system testing. We then show how the test models are implementable using a concept of agent technology, where agents are DEVS models that can be distributed over an infrastructure that supports the DEVS simulation protocol. The Appendix discusses a software tool that supports the methodology being discussed. More details can be found in [Sal08].

2. Verification and the Need for Verification Requirements.

Requirements are written to ensure that the process of design and implementation eventually provides the needed system. The plan for verification should be written with, or in, the requirements. Grady [Gra 06] suggests the following:

“Failure to identify how it will be verified that the product satisfies its requirements will often lead to cost, schedule, and performance risks that we gain insight into only late in the program…[as] proven on many programs that the later…faults are exposed, the more it costs in dollars and time to correct them. Alternatively, these failures may require so much cost to fix that it is simply not feasible to correct them, forcing acceptance of lower than required performance.

Further,

“Verification of the requirements for components does not necessarily guarantee the next-higher-level item will satisfy its requirements. The Hubble Space Telescope will forever offer an example of this failure.” [Gra 06] This statement points out the need for verification of the integration of elements as the project proceeds toward system acceptance.

“Each requirement must be verified and verification may occur throughout the design, development and operational phases. Any changes…to requirements will cause re-verification. [Hoo94]

Figure 2 shows the relationship of system level specifications to other documents such as test plans and test procedures. Specifications for items have often had separate sections for verification requirements. Verification requirements need to state precisely how the system or subsystem will be verified whether or not the design completely complies with the requirements.

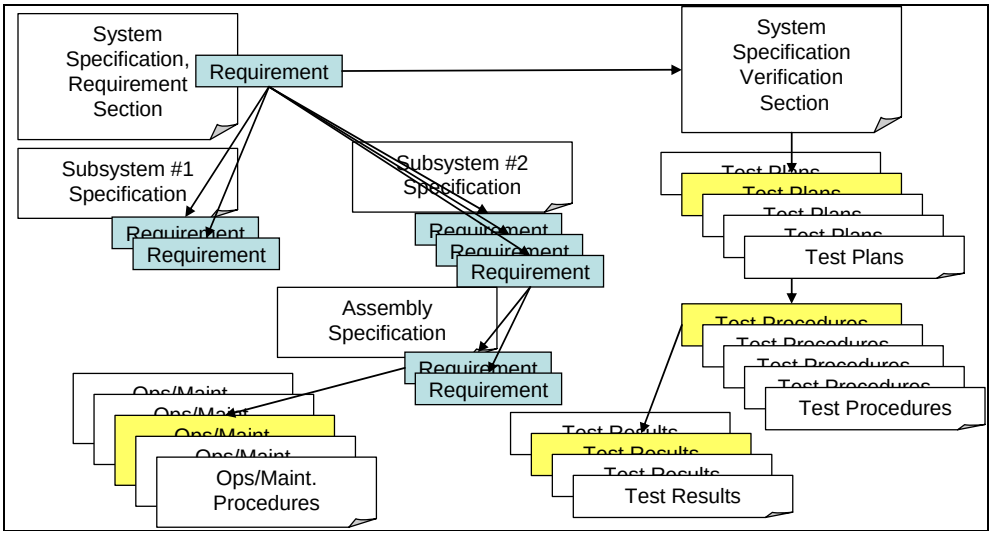


Figure 2 Requirements branch to other documents (taken from Hooks [Hoo94], fig. 2)

Grady [Gra 06] encourages having one or more verification requirements for each functional requirement in the requirements section. A better approach would be to have the requirements include the verification requirements, rather than have separate verification requirements. Table 1 shows a verification requirement under the behavioral requirement. This table suggests that writing verification requirements together with writing requirements can be an iterative process that can provide greater assurance that the requirements are adequate.

Table 1. Functional Requirement with Verification Requirement Broken Into Conditions, Behavior, and Standards

Case	Typical statement	Conditions	Behavior	Standards
Functional requirement	The item shall be capable of entering, residing in, and exiting a mode signaling North-South auto passage as appropriate as defined in <figure> for source and destination states and <table> for transition logic . [Gra 06]		entering, residing in, and exiting a mode signaling auto passage	Transitions appropriate per <figure> source and destination states <table> transition logic
Verification requirement	The item shall be connected to a system simulator and stimulated sequentially with all input conditions that should trigger mode entry and mode exit and the desired mode shall be entered or exited as appropriate. While in this mode all possible input stimuli other that those which should cause exit shall be executed for periods between 100 and 1000 milliseconds resulting in no mode change. Refer to <figure> and <table> for appropriate entry and exit conditions.	item connected to system simulator and stimulated sequentially with all input conditions that should cause desired mode transitions. In desired mode all possible input stimuli other that those which should cause exit shall be executed	Enter desired mode	Enters modes per entry conditions in <fig> and <table>. Stays in mode for periods between 100 and 1000 milliseconds
			Exit mode	Exits only per exit conditions in <figure> and <table>

Conditions, behavior, and standards provide a comprehensive description of what is required. These elements map to a formal description of a system, such as provided by the DEVS formalism. Basic DEVS models completely describe discrete event behavior of a system. They may be coupled together to form larger models (e.g., assemblies, subsystems and systems) in hierarchical fashion.

3. Representing Requirements as Experimental Frames and System Entity Structures

A requirements specification is among the earliest models of a new system. "…a model is conceived of any physical, mathematical, or logical representation of a system, entity, phenomena, or process." [DoD 94].To describe the relation of models with systems, a framework for M&S was developed in [Zei76] and is illustrated in Figure 3. A comprehensive structure for a requirement and for a requirement's validation and verification should represent the entities of this framework: experimental frame, source system, model and simulator, each of which can be described as systems in the underlying systems theory framework. An experimental frame is a system that interacts with a system to be tested under specific conditions to gather data to answer a question about the system. In tests the model represents the system of interest.

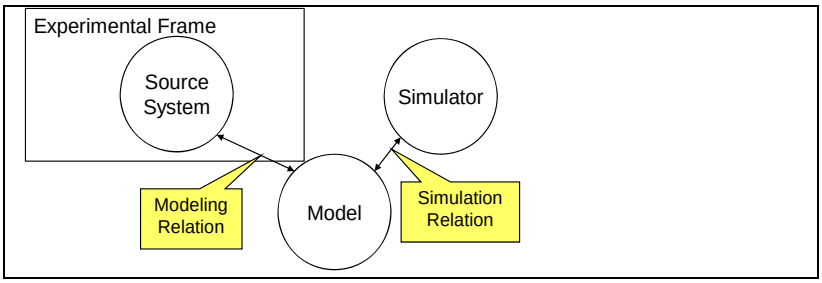


Figure 3. Framework for Modeling and Simulation

The simulator executes the model to generate its behavior. At the genesis of a new system the only simulator available may be the minds of the designers, and the only model of the system to be may be the set of system requirements (specification) under consideration. Grady [Gra 06] strongly warns against failing to have good verification requirements corresponding to all requirements. He recommends making a lone, principal engineer responsible for each specification "[who] should ensure that all of the verification requirements are prepared together with the item requirements." This emphasis on verification requirements emphasizes the importance of the experimental frame, which specifies the conditions under which the model will be exercised.

In the M&S framework [Zei76], an experimental frame is the operational formulation of the objectives that motivate a modeling and simulation project. A frame is realized as a system that interacts with the system of interest to obtain the data of interest under specified conditions. Whether used in a simulation or not, an experimental frame is an important tool for both build-time verification that the system is being built to the specification and for design-time validation that the requirements for the system can be met and represent a system that will meet the need.

Experimental frames may be implemented in modeling and simulation in various ways. One way is to divide the work of the experimental frame among a *generator*, a model that generates inputs to the system, and another component, a *transducer*, which observes inputs to the system from the generator and

observes and analyzes the system's outputs. A transducer can also inform an *acceptor* that sees whether the desired experimental conditions are met. Figure 4 shows such an experimental frame designed to measure the "what and how well" of the requirements for the system. The experimental frame and a representation of the system (model) are operated by the simulator to provide a prediction of how well the model meets requirements for the system.

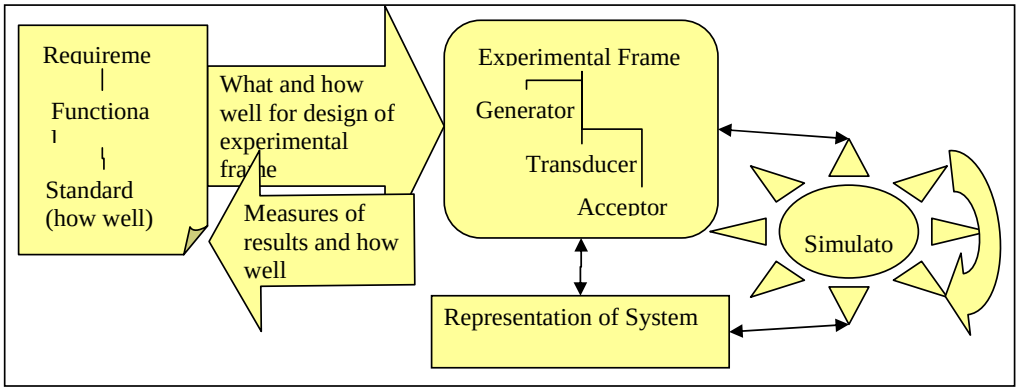


Figure 4. Experimental Frame and Model of System Driven By Simulator

Table 2 shows how the elements of the systems formalism and M&S Framework map to requirements and their conditions, behaviors, and standards. Although components of the experimental frame (being models themselves) can be specified by a DEVS, the table is filled in from the perspective of the system representation being validated or verified.

Table 2 Mapping System Formalism and M&S Framework to Conditions, Behaviors, and Standards

Elements of Systems Formalism and the M&S Framework	Requirement Element (i.e., Conditions, Behaviors, Standards, and Infrastructure)
Experimental Frame	Conditions and Standards
Generator – sends input values to the model	Conditions
Transducer – receives outputs of the model and receives copies of the inputs to the model; using these inputs it provides calculations to the acceptor.	Standards. Monitors inputs to the model and outputs of the model for the use of the Acceptor.
Acceptor – monitors the experiment to see whether the desired experimental conditions are met	Standards
Source System	Realization of the set of all system requirements
Model	Behavior
Time Base	Behavior—time indexed input/output data
Set of Input Values	Contain inputs that the model must receive

Set of States	Represent modes and states
Set of Output Values	Contain outputs that the model must send
Transition Function	Determines state transitions induced by inputs and passage of time
Output function	Behavior—generate outputs
Simulator	Infrastructure for executing the V&V

The generator, transducer, and acceptor shown in Table 2 realize an experimental frame specification, which includes the following dimensions:

- *input stimuli*: specification of the class of admissible input time-dependent stimuli. This is the class from which individual samples will be drawn and injected into the model or system under test for particular experiments.
- *control*: specification of the conditions under which the model or system will be initialized, continued under examination, and terminated.
- *metrics*: specification of the data summarization functions and the measures to be employed to provide quantitative or qualitative measures of the input/output behavior of the model. Examples of such metrics are performance indices, goodness of fit criteria, and error accuracy bound.
- *analysis*: specification of means by which the results of data collection in the frame will be analyzed to arrived at final conclusions. The data collected in a frame consists of pairs of input/output time functions.

An experimental frame can be viewed as a system that interacts with the system under test (SUT) to obtain the data of interest under specified conditions. As indicated earlier, a frame typically has three types of components (as shown in Figure 5 a): *generator* that generates input segments to the system; acceptor that monitors an experiment to see the desired experimental conditions are met; and *transducer* that observes and analyzes the system output segments.

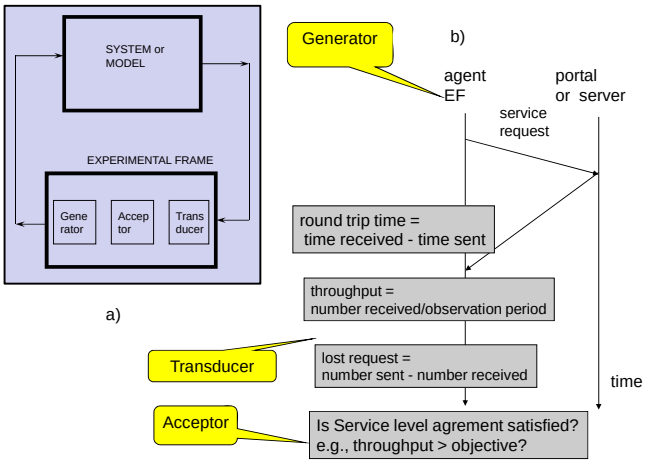


Figure 5. Experimental Frame and Components

, whether the throughput exceeds the desired level and/or whether say 99% of the round trip times are below a given threshold.
Figure 5 b) illustrates a simple, but ubiquitous, pattern for experimental frames that measure typical job processing performance metrics, such as relate to round trip time and throughput. Illustrated in the web context, a generator produces service request messages at a given rate. The time that has elapsed between sending of a request and its return from a server is the round trip time. A transducer notes the departures and arrivals of requests allowing it to compute the average round trip time and other related statistics, as well as the throughput and unsatisfied (or lost) requests. An acceptor notes whether performance achieves the developer's objectives, for example

When combined with the System Entity Structure (SES) ontology framework [Zei07], the experimental frame provides a means to represent *complete* requirements – i.e., it answers the question, what would a requirement in which all relevant features have been addressed look like? Figure 6 presents an SES for a behavioral requirement with verification (experiment) *conditions* and post conditions (*standard* for acceptable verification). This SES exploits a key feature of SES: specialization. Children of specializations are entities representing variants of their parents. The items with tildes (~) under the choices for metrics are variables, e.g., type of units, and threshold (minimum acceptable) performance levels and objective (maximum desired) levels expressed in the units specified.

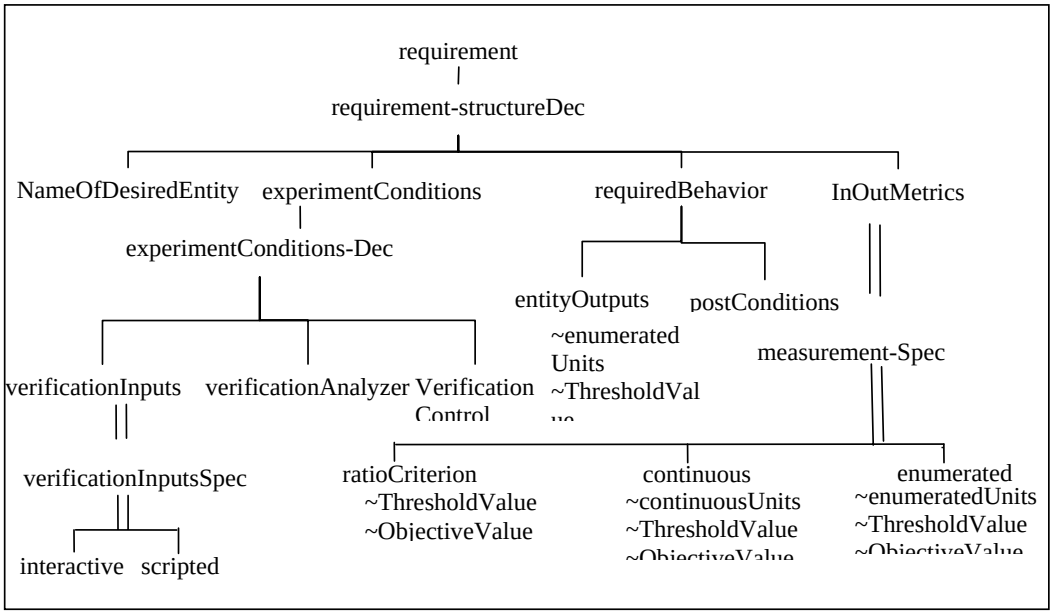


Figure 6 SES for a Requirement Including Verification Conditions and Standards for Verification

Since it can capture all the aspects needed for completeness, we propose that the experimental frame concept is a desirable way to express requirements. Such an SES can be expressed as an XML Schemata using a tool such as the SESBuilder [RTS07].

Once adopted and standardized, this form can be widely promulgated within an organization or among a wider community of interest. Modern information-technology based systems need to be represented by multiple, top level requirements. These should be characterized by their respective experimental frames. Such a concept could enable vendors to describe their products' performance and interface specifications

on the World Wide Web per the standard XML Schema, bringing vendors and system architects together more effectively and efficiently. This marketplace would, also, improve the expression of requirements through the examples that requirement developers would find on the Web. The greater effort and rigor needed to comprehensively express requirements in this manner would be encouraged by such a marketplace and facilitated by being able to learn from requirements posted by others. Another advantage of this approach is that due to the "specialization" feature of the SES, a requirements SES for a generic product should facilitate designing product lines for it.

Since experimental frames are realizable as test systems and/or tools, a database of requirements expressed as experimental frames should be easily related to the set of test items required for a project. This should improve estimating of verification costs and the verification of requirements.

Expressing requirements and experimental frames as XML instances complying with standardized XML schema would mitigate against falling into the trap of unwittingly embedding specific solutions within the requirements (perhaps over-constraining the designer's freedom). For this purpose software tools that support higher levels of validation and analysis of XML instances against standardized XML schema should be developed.

4. Need to Facilitate the Decomposition and Design of System Architecture.

"Requirements are derived through engineering processes that include design and analysis, trade studies, concept development and other activities, such as prototyping." [Hoo94] The difficulty with this major activity is shown in the following quote from [Gra06]:

"Many organizations find that they fail to develop the requirements needed by the design community in a timely way. They keep repeating the same cycle on each program and fail to understand their problem. …Commonly, the managers in these organizations express this problem as, 'We have difficulty flowing down system requirements to the designers.' …the flowdown strategy is only good for some specialty engineering [e.g., reliability engineering and weight management] and environmental design constraints. It is not a good strategy for interface and performance requirements…Performance requirements are best exposed and defined through the application of a structured process for exposing needed functionality and allocation of the exposed functionality to things in the architecture."

Grady, also, discusses why flowing requirements down is difficult and often needs simulation. "There are many requirements that will not yield directly to the allocation process, however. And, these are commonly the most difficult requirements to quantify. Given…that we have a requirement for aircraft maximum airspeed, how shall we determine the engine thrust needed?...The mathematical relationships between…parameters is complex and cannot accurately be resolved on the back of an envelope. Commonly, we have to apply some form of computer simulation to try various combinations of values working toward identification of a good and achievable mix of values."

Figure 7 provides context for requirement decomposition and design. Activities step down and then up a V-like structure. Decomposition into lower levels of detail and design is at left. At the right of the V, integration and verification step from component up to system level. While the V-shaped concept has been advocated by others, we augment it with the above Experimental Frame and SES concepts. In the center of the V is a SES representing the concept of a system under development built from smaller component systems. Such a hierarchical composition can employ levels such as assemblies, modules, packages, and so on.

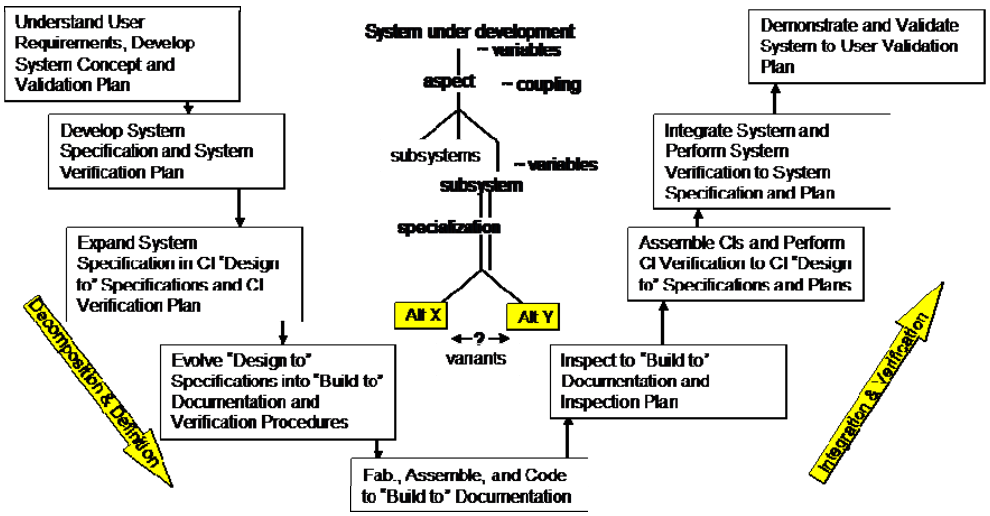


Figure 7. SES-Structured M&S Superimposed on the Center of the Vee Model of Systems Engineering [Forstberg]

Forstberg [For05] and coauthors suggest a Decomposition Analysis and Resolution Process as "the framework for proactive requirements management" to be applied at each level of the overall process of decomposing and defining requirements to provide the "design to" and verification specifications for the next lower level, down to the lowest level of "build to" documentation.

Figure 8 shows the activities and results of Decomposition Analysis and Resolution [For05].

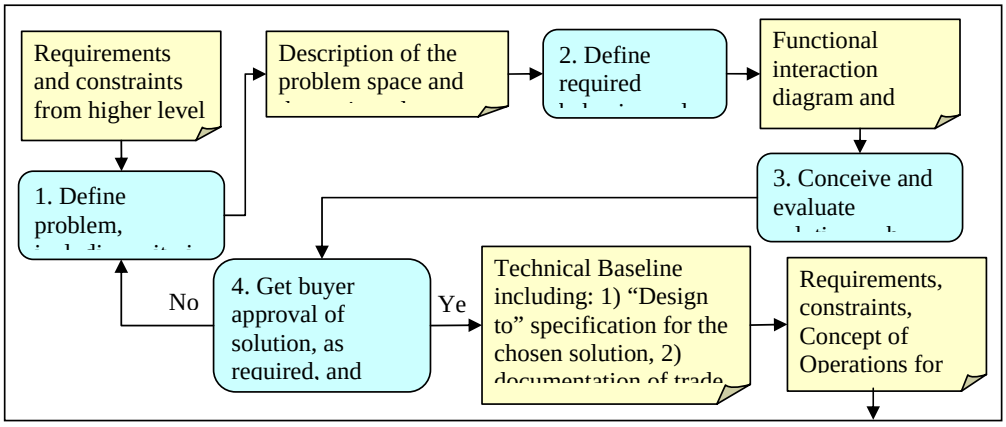


Figure 8. Decomposition Analysis and Resolution

Steps 2, 3, and 4 of Decomposition Analysis and Resolution are prime activities for the use and reuse of M&S.

At each point in progressing down the left side of the V (refer to Figure 7), system engineers should discover and document how to verify later during the work leading up the right side of the V that the

solution is being implemented as specified. Also, the project team should be validating that the solution will perform as required (i.e., the right solution being built). M&S can contribute by building a tree of experimental frame leafs for M&S as decomposition proceeds down the hierarchical levels. With the use of real time simulators, physical implementations can gradually replace models in the simulations. So in addition to being used for the validation of requirements and for choosing between alternatives on the decomposition and definition side of the V, the experimental frames may be reused on the right side of the V in stimulating and measuring the response of the components, assemblies, subsystems, and finally the system. Dual use of an experimental frame is shown in Figure 9, which shows M&S used in an integral way with requirement management, validation, design, prototyping, and verification.

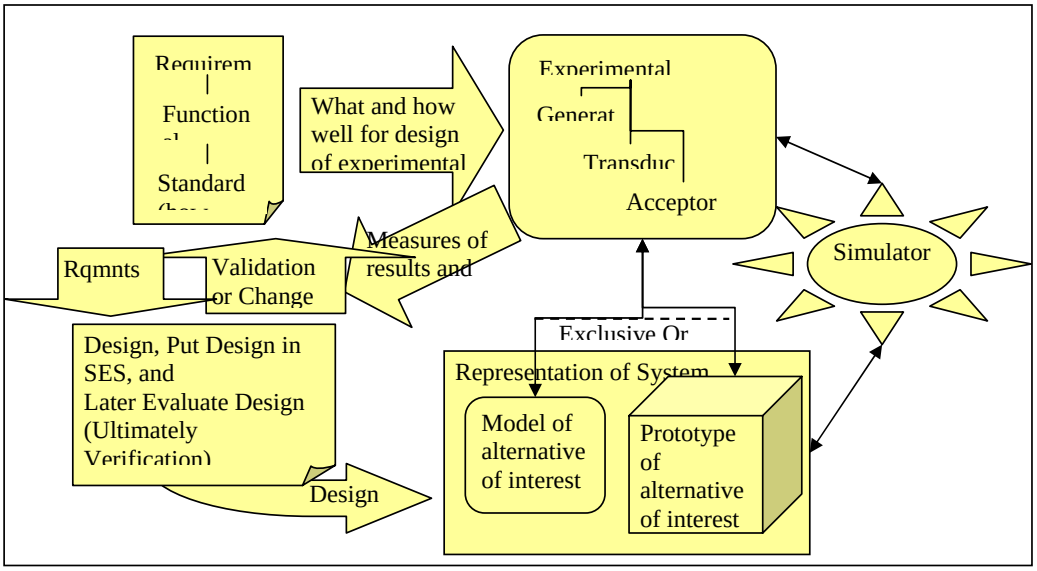


Figure 9 M&S Integral with Requirements Management, Validation, Design, Prototyping, and Verification

So the process of examining the breakdown of the desired system into the best alternative subsystems, assemblies, and components can be viewed as resulting in a tree of experimental frame – alternative choice pairs. The ability to express these pairs and to select best alternatives and know the cost and schedule to realize them is a measure of subject matter expertise. The SES and the consistent development of models can provide a significant part of a structured approach to capturing this expertise. A scheme that uses the SES for managing a collection of models for simulating and evaluating alternative configurations of systems as shown in Figure 10.

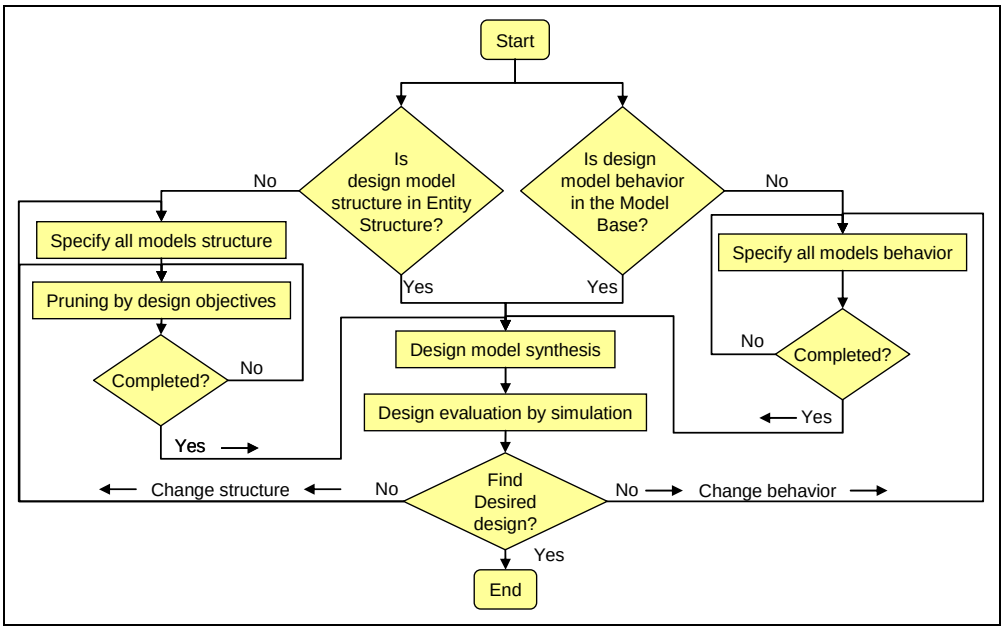


Figure 10. Design Using a System Entity Structure and a Behavior-Model-Base

Special expertise is required to derive requirements for a set of component systems that will work together to satisfy a given requirement. For example, posed with a requirement to transport a 10,000 pound payload 5,000 miles away in 2 hours, expert designers would have to be able to derive requirements for components to provide propulsion, control, and navigation. In the case of the navigation system for example, the requirements for navigation would be transformed to an experimental frame specific to that subsystem, and navigation experts and/or navigation system vendors would contribute models and/or specifications representative of the state of the art. Figure 11 illustrates the hierarchy of experimental frames and alternatives juxtaposed with the product breakdown and work breakdown.

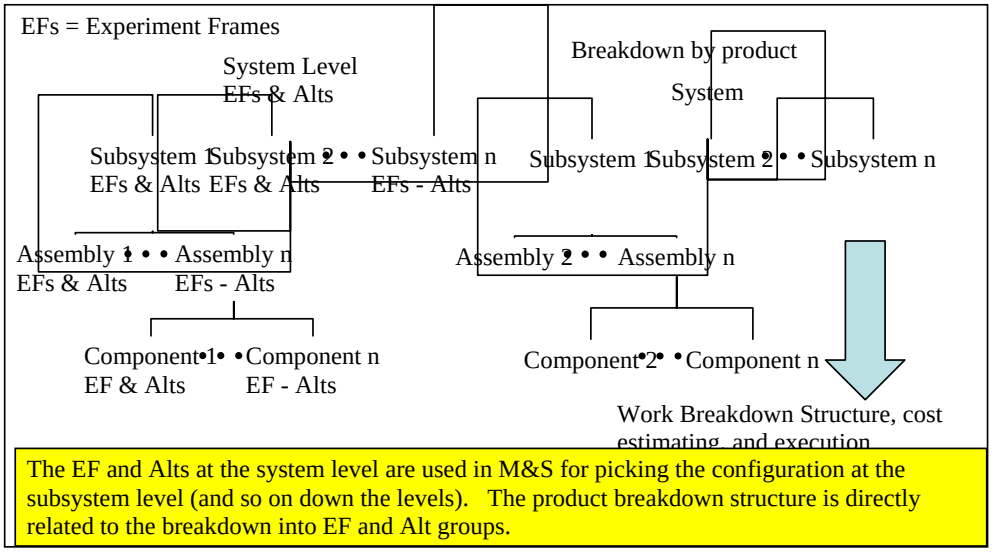


Figure 1. M&S Is Useful for Requirement Decomposition and Generating the Product Breakdown Structure

We now propose combining requirement management via M&S with the concept of web services facilitated by open standards. Consider a scenario in which already developed components are described with requirements written to an open standard for requirements. Then, with web service-enabled models of the components written to an open standard and a standard simulation protocol [Zei08], designers could search and discover such models in an open registry. Vendors could have their products modeled in a ready-to-simulate form in a web catalyzed marketplace. Vendors could register web service enabled models of their products in an on line registry, such as registries conforming to the open standard for Universal Description Discovery and Integration (UDDI). To facilitate registering their models, vendors would describe their web service enabled models using Web Service Description Language (WSDL) using the XML schema discussed above as the schema for expressing the specifications that their products meet. Models can use these WSDLs to develop simulations. Also under this concept, vendor web services would upon suitable request, return the specification for their product in XML, including interface requirements according to an open standard (i.e., the XML schema for requirements/experimental frame). Interface requirements are closely related to coupling of a representation (model) of a system of interest with an experimental frame, which would be required to demonstrate what the product will do and accomplish. This system of requirements and models should be supported by a standard lexicon or ontology. For example, ISO 9001, "Quality management systems – Requirements" is accompanied by ISO 9000, "Quality management systems -- Fundamentals and vocabulary." [OSI08]

This system should extend the idea of SES-enabled model base discussed above with Figure 10. Additionally, this **system should capitalize on the technology that provides service oriented distributed modeling and simulation** over the World Wide Web [Mit07, Mit08, Mit09]. This technology would enable designers to federate suitable vendor models with other models to validate the performance of a proposed architecture under consideration.

5. Employing Agents in M&S-Based Design, Verification and Validation

In this article, the term "agent" (or "mobile agent") refers to a discrete thread of computation that is deployable in distributed form over computer networks where it interacts with local computations and communicate/coordinate with other such agents. The modular nature of software objects together with their behavior and interaction with other objects, led to the concept of agents which embodied increased autonomy and self-determination. A wide variety of agent types exists in large part determined by the variety and sophistication of their processing capacities – ranging from agents that simply gather information on packet traffic in a network to logic-based entities with elaborate reasoning capacity and authority to make decisions. The step from agents to their aggregates is natural, thus leading to the concept of multi-agent systems or societies of agents, especially in the realm of modeling and simulation. Distributed Artificial Intelligence systems [Gla87] integrate concepts from concurrent programming [Agh85] and knowledge representation to coordinate agent ensembles in distributed problem solving. Cognitive agents represent beliefs and intentions and flexible mechanisms for communication among agents [Roa95]. Reactive agents employ procedural reasoning and automatic control theory to interact with their environments. Hybrid agents combine higher level goal-directed planning with lower level automatic control to integrate deliberative and reactive processes in real time [Fir92]. Endomorphic agent concepts, first defined in [Zei90], provide a framework for including the distributed knowledge that arises from providing agents with internal models of the environment, themselves, and other agents. We will employ these concepts below.

Given this review of agent concepts and technology, we address the question of how to effectively employ agents in the M&S approach to requirements-based design, verification and validation of systems. There are various roles that agents could take in supporting the activities summarized in Figure 11. Here we focus on the possibility of employing agents to implement the experimental frames derived from requirements for a distributed system. Such agents will be expressed as DEVS models that may implement experimental frame components, may communicate and/or coordinate with each other using underlying DEVS-based distributed simulation infrastructure and may include endomorphic models of the system under test. We now discuss these dimensions of DEVS agent technology.

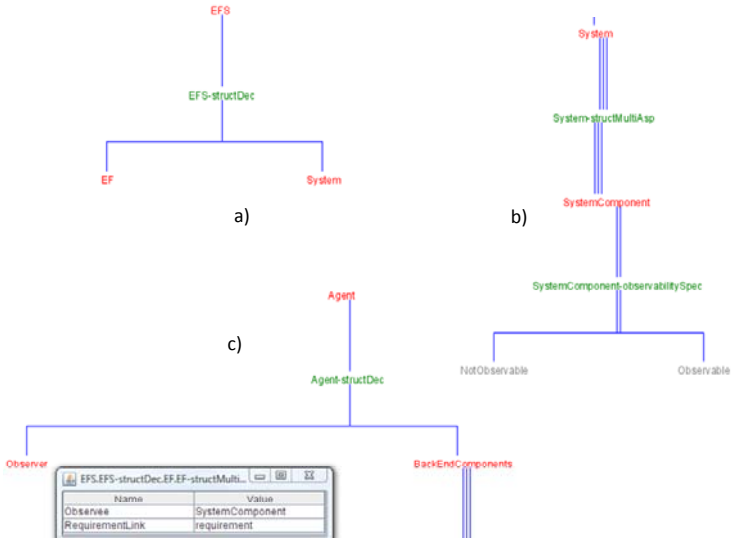


Figure 12. Experimental Frame, System and Agents

Starting top down, the composition of a System and an Experimental Frame is represented in Figure 12 a). Using a natural language specification [Zei07], we can express this composition as:

From the structure perspective, EFS is made of EF and System

For purposes of observing the interactive behavior of components, we can indicate that a System consists of components, each of which may be observable or not observable, as illustrated in Figure 12 b) and expressed as:

From the observability perspective, a System is made of more than one SystemComponent
A SystemComponent can be Observable or NotObservable in observability

A system component is *observable* if it accommodates attachment of an agent to intercept its communications with other components. Indeed, an essential constraint on experimentation in net-centric systems is that components may not support agent attachment for various reasons including security, proprietary concerns, machine loading, and others.

On the other side, as illustrated in Figure 12 c), an experimental frame in this context will be composed of agents that carry out its distributed functionality. In the following description, each agent has an assigned component to observe and a link that traces back to one or more requirements (as described in Figure 9) from which its role in the testing and evaluation was derived. An agent is composed of an observer module and one or more "backend" components such as generators, acceptors and transducers that implement its functionality:

From the structure perspective, EF is made of more than one Agent
From the structure perspective, Agent is made of Observer and BackEndComponents
An Observer has an Observee and a RequirementLink
The range of Observer's Observee is SystemComponent
The range of Observer's RequirementLink is requirement
From the structure perspective, BackEndComponents are made of more than one BackEndComponent
A BackEndComponent can be Generator, Acceptor, or Transducer in function

From this high top level view we proceed to drill down to uncover some critical details in making the concept of agent-implemented frames operational.

6. Experimental Frame Concepts for Agent Implementation

In the realm of service-oriented architecture for web services, mission threads for a proposed system of services are intended to encapsulate how well the system's capabilities will be used in real world situations. The capability to measure effectiveness of missions requires the ability to execute such mission threads in operational environments, both live and virtual. For testing mission threads, Experimental Frame concepts offer an approach to enable simulated and real-time observation of participant system information exchanges and processing of acquired data to allow mission effectiveness to be assessed. Relevant Measures of Performance (MOPs) include quality of shared situational awareness, quality and timeliness of information, and extent and effectiveness of collaboration. Measures of Effectiveness (MOEs) are measures of the desired benefits such as increase in combat power, increase in decision-making capability, and increase in speed of command. Such metrics must be modeled with mathematical precision so as to be amenable to collection and computation with minimally intrusive test infrastructure to support rigorous, repeatable and consistent testing and evaluation (T&E).

An example of a mission thread is the Theater Strategic Head Quarter Planning at the Combatant Command Level. As illustrated in Figure 13, this thread starts when the Combatant Command HQ receives a mission which is delegated to HQ Staff. The staff initiates the Mission Analysis Process which returns outputs and products to the Commander. Then the Joint planning group and the development teams develop courses of action and perform effects analysis, respectively. The thread ends with the issue of operational orders. The MOP of interest might include various ways of measuring extent and effectiveness of collaboration, while the MOE might be the increase in speed of command. These measures might be collected in assessing the value added of a collaboration support system in an operationally realistic setting. An informally presented mission thread can be regarded as a template for specifying a large family of instances. As illustrated in Figure 13, such instances can vary in several dimensions, including the objectives of interest (including the desired MOP and MOE), the type of application, the participants involved, and the operational environments in which testing will occur. Furthermore, mission threads can be nested, so that for example, Mission Analysis is a sub-thread that is executed within the Theater Planning thread.

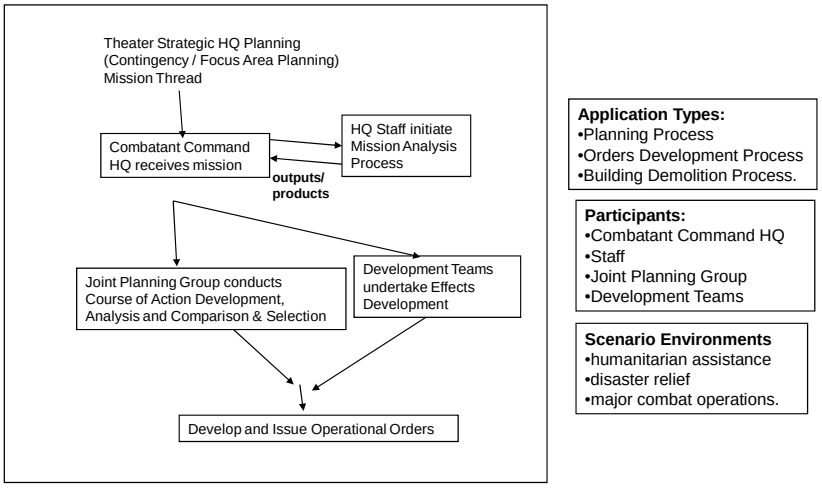


Figure 13. Illustrating a Joint Mission Thread and its dimensions for variation

The upper part of Figure 14 illustrates the process for deriving the types of experimental frame components and their parameter values. The test event objectives, as embodied in the MOEs and/or MOPs of interest in the mission thread to be tested, determine the experimental frame – this is a specification of the experimental conditions to be established in the network environment to support the test. The experimental frame consists of a selection of observers – the agents that listen to web service requests, generators that can simulate user actions or inject background web traffic, transducers that gather statistics and acceptors that check for conditions that must pertain for valid experimentation to proceed. Not all of these components need be present for a test – for example, generators would be absent in a test in which only live users participate without virtual counterparts.

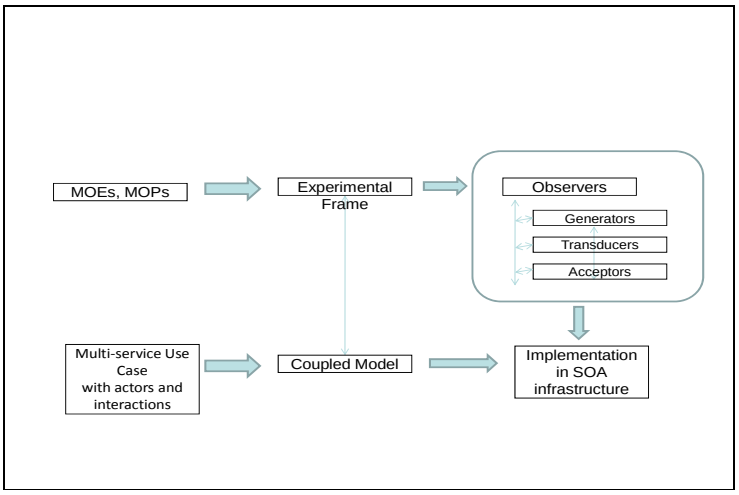


Figure 14. DEVS Concepts for Mission Thread-based Testing

The lower part of Figure 14, shows how an experimental frame sets up the conditions for testing the effectiveness of a service oriented architecture (SOA) for mission threads involving multiple SOA services. An instance of collaboration is modeled as a coupled model in DEVS – a model which specifies components and how they interact by being coupled together by specified output to input connections. In a model of collaboration, the components are participants in the collaboration and couplings represent the possible information exchanges that can occur among them. This formulation offers an approach to capturing and implementing mission threads in the form of test model federations that can be deployed in a net-centric environment. The experimental frame, as derived from the test event objectives, determines which participants in the collaboration will be observed, the types of web service requests that will be listened to, and the particular field values in the request Simple Object Access Protocol (SOAP) messages that will be extracted and examined. The Web Service Description Language (WSDL) structures associated with such requests provide the basis for accessing these messages and their contents.

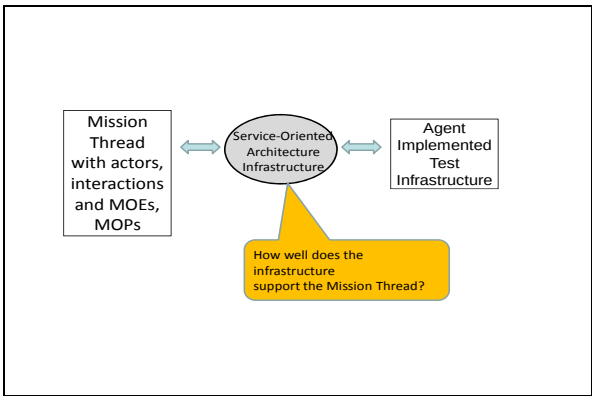


Figure 15. Mission thread-based testing the net-centric infrastructure

As shown in Figure 15, the design of an agent-implemented test infrastructure is aimed at the problem of assessing how well a service-oriented infrastructure supports the collaboration and information exchange requirements of a mission thread. Figure 15 depicts how a SOA infrastructure offers a way to deploy the

participants in a mission thread together with network and web services that allow them to collaborate to achieve the mission objectives. Formulation of a mission thread instance as a coupled model allows us to provide rigorous methods for realizing the use case within the SOA infrastructure. At the same time, the MOEs and MOPs that have been formulated for assessing the outcome of the mission execution are translated into an appropriate experimental frame, with observers of the participant activities and message exchanges as well as the generators, transducers and acceptors as discussed above. However in the context of net-centric operation, the distributed nature of the execution will require the experimental frame to be distributed as well. This distribution is implemented by deploying agents and their associated components among a selection of nodes of the network that is specified by the tester.

7. Agent-Implemented Experimental Frames in Distributed Test Federations

A DEVS distributed federation is a DEVS coupled model whose components reside on different network nodes and whose coupling is implemented through middleware connectivity characteristic of the environment, e.g., SOAP for Global Information Grid (GIG)/SOA. The federation models are executed by DEVS simulator nodes that provide the time and data exchange coordination as specified in the DEVS abstract simulator protocol.

As discussed earlier, in the general concept of experimental frame (EF), the generator sends inputs to the system under test (SUT), the transducer collects SUT outputs and develops statistical summaries, and the acceptor monitors SUT observables making decisions about continuation or termination of the experiment [Zei00]. Since the application is composed of service components, the EF is distributed among application components, as illustrated in Figure 6**.** Each component may be coupled to an EF consisting of some subset of generator, acceptor, and transducer components. As mentioned, in addition an observer couples the EF to the component (web server/client). We refer to the DEVS model that consists of the observer and EF as a *test agent*.

Net-centric Service Oriented Architecture (SOA) provides a currently relevant technologically feasible realization of the concept. As discussed earlier, the DEVS/SOA infrastructure enables DEVS models, and test agents in particular, to be deployed to the network nodes of interest. As illustrated in Figure 16, in this incarnation, the coupling between observer and the service is facilitated by the interface provided by the web service description language (WSDL). The network inputs sent by EF generators are SOAP messages sent to other EFs as destinations; transducers record the arrival of messages and extract the data in their fields, while acceptors decide on whether the gathered data indicates continuation or termination is in order [Mit07].

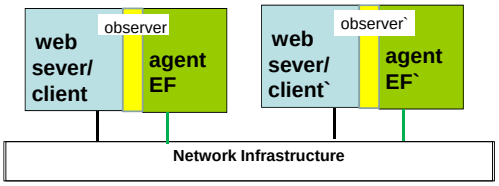


Figure 16 Deploying Experimental Frame Agents and Observers

Since EFs are implemented as DEVS models, distributed EFs are implemented as DEVS models, or agents as we have called them, residing on network nodes. Such a federation, illustrated in Figure 17, consists of DEVS simulators executing on web servers on the nodes exchanging messages and obeying time relationships under the rules contained within their hosted DEVS models.

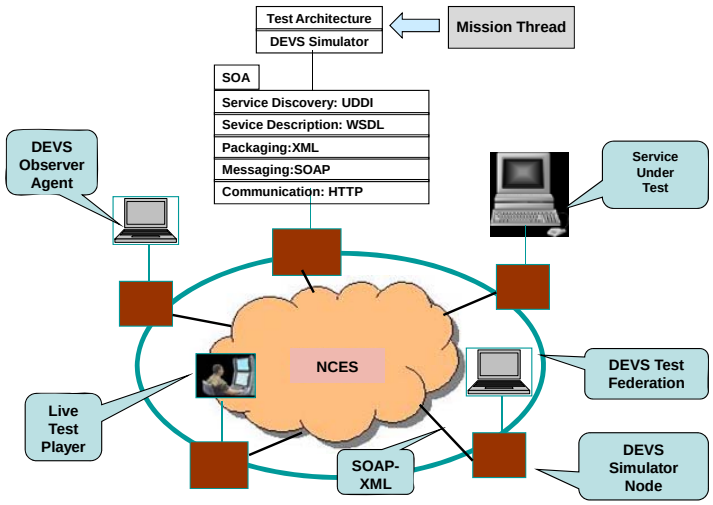


Figure 2 DEVS Test Federation in GIG/SOA Environment

8. DEVS/SOA: Net-centric Execution using Simulation Service

The fundamental concept of web services is to integrate software applications as services. Web services allow the applications to communicate with other applications using open standards. We are offering DEVS-based simulators as a web service, and they must have these standard technologies: communication protocol (Simple Object Access Protocol, SOAP), service description (Web Service Description Language, WSDL), and service discovery (Universal Description Discovery and Integration, UDDI).

The complete setup requires one or more servers that are capable of running a DEVS Simulation Service, The capability to run the simulation service is provided by the server side design of DEVS Simulation protocol supported by the DEVSJAVA [ACI06]. The Simulation Service framework has two layers. The top-layer is the user coordination layer that oversees the lower layer. The lower layer is the true simulation service layer that executes the DEVS simulation protocol as a Service [Mit07, Mit08].

Automation of Agent Attachment to System Components

Having reviewed the underlying infrastructure, we return to the top level view of Figure 14 to consider how to automate the attachment of agents to observable components and their deployment in a net-centric environment. We continue by limiting our discussion to the case of mission threads as illustrated in Figure 13 and refer to Figure 18 to frame the problem of automating the attachment of agents to observable components. In Figure 18 a), we are given a DEVS coupled model of a mission thread and consider mapping such a specification into a coupled model consisting of agents as defined in Figure 18. Such a mapping can include information concerning the message exchange paths among actors in the mission thread for runtime use by the observing agents.

Comment [DRH1]: 14?

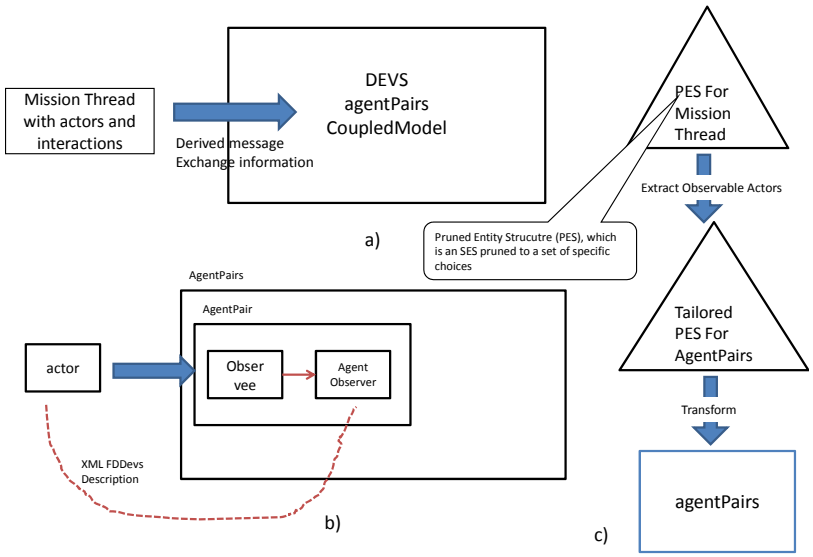


Figure 3 Mapping mission thread coupled models to agent pair models

To formulate the process that will attach agents to clients (as in Figure 16), we introduce the abstraction of an *agentPair* as in Figure 18 b). Here the DEVS model for an observable actor in the mission thread is called an Observee. It is to be interfaced to an Observer to form an *agentPair*. These pairs form the components of the coupled model, *agentPairs*, a composition that will represent the final net-centric environment, as in Figure 17. Information concerning the structure and behavior of the actor can be extracted from its model to help its agent in its observation. Information of this kind, at the both the system and component model levels, endows the agents with endomorphic models of the system (mission thread) they are tasked with observing.

To construct *agentPairs* we write the SES specification:

> From a structure perspective, agentPairs are made of more than one agentPair
> From a message perspective, the agentPair is made of observer and observee

which is illustrated in Figure 19.

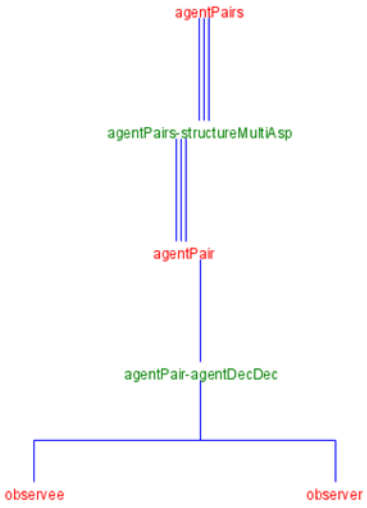


Figure 19. SES for agent pair models

DEVS-Agent Communications/Coordination

We indicated earlier that the mapping in Figure 18 can include information concerning the message exchange paths among actors in the mission thread for runtime use by the observing agents, endowing the agents with endomorphic models of the mission thread under observation.

We use a greatly simplified example of a mission thread to illustrate how this can be implemented.

Example: Mission Thread

In this thread, a truncated version of a much longer one, a commander, MajSmith, posts a request for intelligence on a target he has been assigned to attack. The request is posted using a web service hosted at a Net-Enabled Command Capability server farm, NECC, and relayed to an intelligence unit, IntelCell, which collects the needed data and returns its estimate to MajSmith via web services of the NECC.

From a message perspective, the MissionThread is made of MajSmith, IntelCell, and NECC

From the message perspective, IntelCell sends threatEstimate to NECC
From the message perspective, NECC sends threatEstimatePost to MajSmith
From the message perspective, MajSmith sends ThreatEstReqPost to the NECC
From the message perspective, NECC sends IntelReq to IntelCell

To describe the possibility of observing these actors we state that:

MajSmith can be Observable or NonObservable in observability
NECC is like MajSmith in observability
IntelCell is like MajSmith in observability

This description is illustrated in Figure 19.

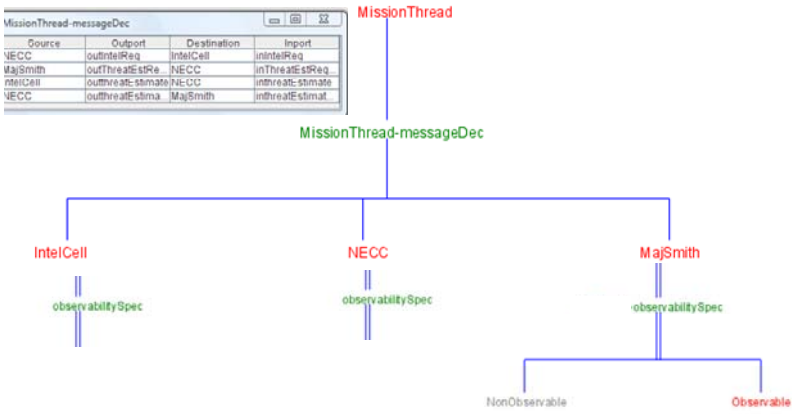


Figure 19. SES for observability designations

Consider the assumption that we can only attach DEVS agents to clients of the web services at the user sites not at the servers. Then only messages sent and received by the MajSmith and IntellCell can be observed in actual testing. This constraint can be expressed by selecting Observable from observabilitySpec for both MajSmith and IntellCell while selecting NonObservable for the specialization under NECC. Using the transformation process [Zei07] this pruned entity structure is transformed to a DEVS coupled model of the example mission thread illustrated in Figure 20. The component names, Observable_MajSmith, NonObservable_NECC, and Observable_IntellCell carry the information about observability that was determined in the pruning process.

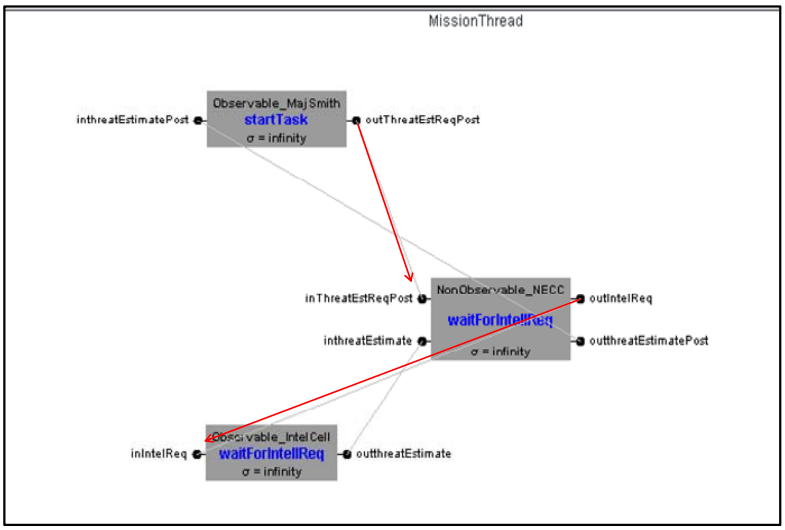


Figure 20. Example mission thread

While the grey lines in Figure 20 indicate couplings prescribed by the SES of Figure 19, the red arrows are superimposed to indicate web service requests that can be observed by agents to be coupled to the observable components. For example, when MajSmith posts an intelligence request an agent connected to the attached client can view this request. Similarly, when IntellCell requests the details of the posting via a web service, this request can be observed by the associated agent. However, because the NECC server is not observable, agents cannot directly watch for arrivals or departures of request messages at this server. This means that in order to verify that MajSmith's request for intelligence actually reached IntellCell, we have to make it possible for MajSmith's agent to inform IntellCell's agent that a request has been posted. This in turn will enable IntellCell's agent to correlate any observed request for details that it might see with the prior posting. Recall that interaction among participants in a mission thread is taking place in the context of ongoing use of the web services so that information exchanges have to be properly identified to provide useful information.

DEVS-Agent Endomorphic Models

The process is illustrated in Figure 21. First we prune the mission thread SES to select the observable actors, as in Figure 19 (shown by red coloring, e.g., Observable is selected from MajSmithObservabilitySpec. After extracting the observable actors, SES fragments can be generated that set up the assignment of agents to observable actors, both at the level of agentPair and each of its components:

An agentPair can be MajSmith or IntelCell in observerType
An observee can be MajSmith or IntelCell in observability
An observer can be MajSmith or IntelCell in observability

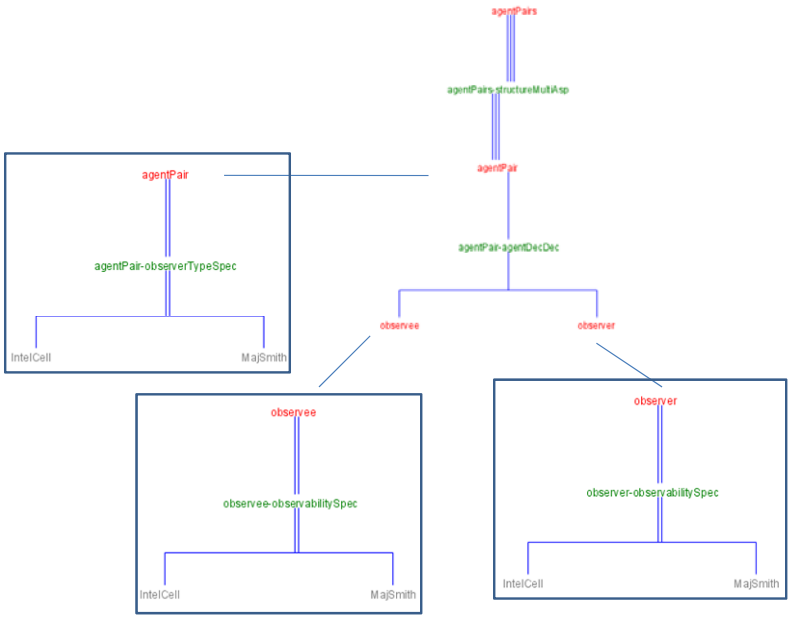


Figure 21. Merger of SES segments into the SES for AgentPairs

These SES fragments, shown boxed in Figure 19 for the example, are automatically merged into the agentPairs SES of Figure 21. Then in the pruning process, an agentPair is selected for each of the observable actors, and the agentPair's observee and observer are pruned to the same actor. For example, we obtain MajSmith_AgentPair which includes MajSmith_observee, and MajSmith_observer.

When the pruned entity structure, shown in Figure 22a), is transformed into a DEVS model, the result is the DEVS coupled model shown in Figure 22b). This model contains agentPairs each of which consist of an observer and observee. For example, the MissionThread of Figure 20 is embedded into the model shown in Figure 22 that contains agentPairs for each of the two observable components. Each observee is an instance of a class that inherits from the associated observable class of the MissionThread. For example, MajSmith_observee inherits its DEVS functionality from the MajSmith class whose instance is in the example MissionThread. The coupling specification for the agentPairs' observee components reflects the coupling between the corresponding observable components in the MissionThread. In addition there is coupling between an observee and observer within the same agentPair, and between observers in different agentPairs. The latter represents the coupling that enables agent observers to coordinate and exchange information about the actors they are observing.

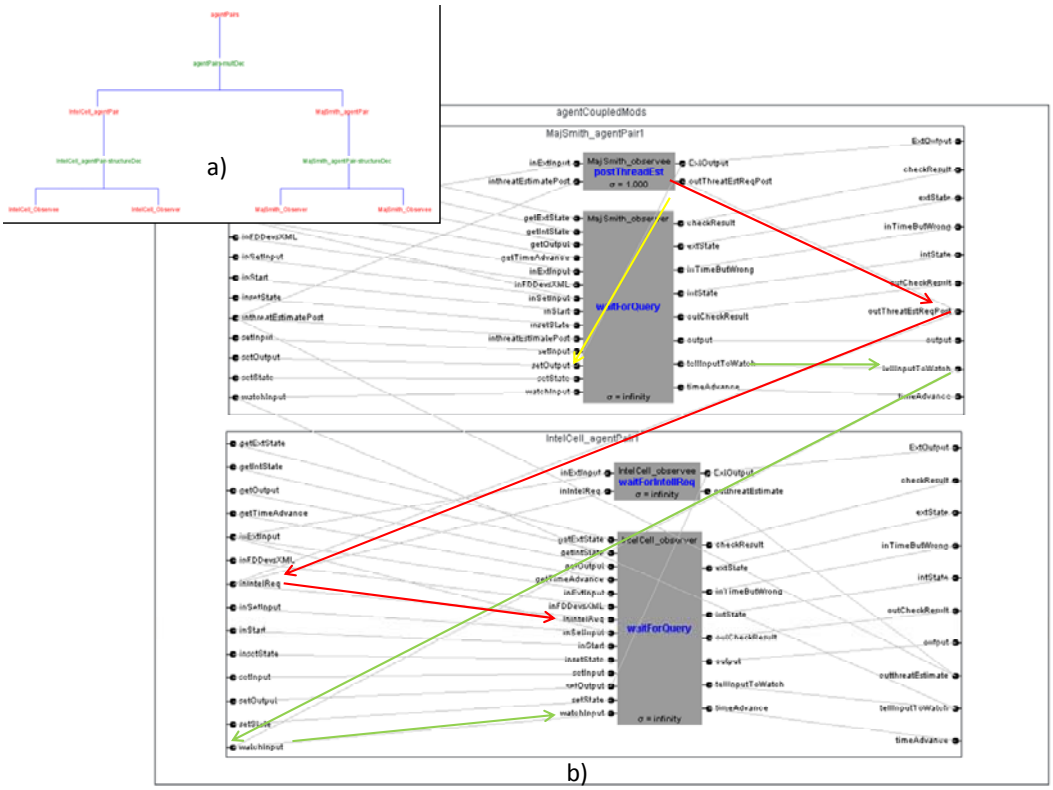


Figure 22. The AgentPairs model (b) generated by the transformation of the pruned entity structure of (a)

These types of couplings are illustrated in Figure 22. For example, the red arrow sequence from MajSmith_Observee to IntelCell_Observee is transferred from the corresponding red-colored sequence in Figure 20. Note that the ThreatEstimatePost output port of MajSmith_Observee is coupled to the setOutput input port of MajSmith_Observer representing its ability to intercept service requests in the actual test environment. The green arrow sequence from MajSmith_Observer to Intel_Observer represents the sequence of couplings that enable the former (after observring MajSmith's intelligence request) to inform the latter of the request to watch for in the near future.

9. Summary and Conclusions

We have shown how the DEVS modeling and simulation framework provides an integrated development and testing methodology for net-centric systems, particularly those based on service-oriented architecture. The bifurcated system development methodology organizes the transition from user-stated requirements to both implementation and testing of implementations, in an integrated manner. The system development includes the definition of requirements, capture of specifications to map formalized DEVS model components and create a reference master model. The test suite development includes execution of test models to run against the system under test. In particular for net-centric systems, the test models

constitute experimental frames that can be distributed to observe the behaviors of, and information exchanges among, the nodes of the system. The process is iterative, allowing return to modify the reference master DEVS model and the requirements specifications. We elaborated on the role of requirements in the bifurcated methodology and the progression to development of test models for net-centric system testing. We showed how the test models are implementable using a concept of agent technology, where agents are DEVS models that can be distributed over an infrastructure that supports the DEVS simulation protocol.

The test methodology is currently being implemented to support testing of the Department of Defense's rapid transition to net-centric operation of all its agencies, contractors, and personnel over its global internet, the Global Information Grid/Service Oriented Architecture (GIG/SOA). The Joint Interoperability Test Command (JITC) has the responsibility to test for GIG/SOA compliance for such projects as Net-Centric Enterprise Services (NCES) and Net-Enabled Command Capability (NECC). NCES is a major acquisition program of the Department of Defense (DoD) to deploy core services that enable information sharing by connecting people and systems that have information (data and services) with those who need information. These services are vital to Net-Centric Operations and Warfare. DoD's Net-Centric Environment (NCE) is radically different from commercial environments, particularly in the security requirements. Commercial tools have not been developed with DoD's stringent and demanding security and war-fighting environment in mind.

In its fully operational state, agent-implemented test methodology will provide testers the ability to deploy software agents to monitor, observe, and measure how well the NCES core services support mission threads developed by testers to emulate actual operational activities in critical DoD applications. DoD's project implementing this methodology will evaluate the effectiveness of the tools and underlying methodology in meeting the test and evaluation requirements before fielding the system in actual operational testing. This evaluation will reveal the extent to which the agent-implemented M&S concepts and bifurcated test and development methodology succeed in supported the demanding application to net-centric test and evaluation.

References

[Zei90] Zeigler, B.P., Object Oriented Simulation with Hierarchical, Modular Models: Intelligent Agents and Endomorphic Systems, Academic Press, Orlando, FL, 1990.

[Sal08] Manuel C. Salas, AutoDEVS: A Methodology for Automating Systems Development, Electrical and Computer Engineering Dept., University of Arizona, 2008

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

[HuX05] X. Hu, 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

[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.

[Mit08] Saurabh Mittal, "DEVS Unified Process for Integrated Development and Testing of Service Oriented Architectures" http://www.acims.arizona.edu/PUBLICATIONS/PDF/Thesis_Mittal.pdf accessed May 4, 2008

[Mit09] Saurabh Mittal, B. Zeigler, J .Martín, , F. Sahin, and M. Jamshidi, “Modeling and Simulation for Systems of Systems Engineering , to appear in “Systems of Systems Engineering “, http://www.acims.arizona.edu/PUBLICATIONS/PDF/Mo_Chapt_5_MittalZeiglerJoseFeratV4.pdf accessed May 4, 2008

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

[Zei76] Zeigler, B.P., Theory of Modelling and Simulation, Wiley, N.Y., 1976

[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.

[Gra06] System Requirements Analysis (Hardcover) Jeffrey O. Grady (Author) Academic Press; 1 edition (February 6, 2006)

[Hoo94] Hooks, I.F, “Guide for Managing and Writing Requirements,” Compliance Automation, Inc., 1994

[DoD 94] Department of Defense (DoD) Directive 5000.59, “DoD Modeling and Simulation Management,” January 4, 1994

[Zei08] Zeigler, B.P., “TOWARDS A NEXT-GENERATION STANDARD FORMODELING AND SIMULATION INTEROPERABILITY", Military Modeling and Simulation, Sigapore , 2008.

[Zei07] Modeling & Simulation-Based Data Engineering: Introducing Pragmatics into Ontologies for Net-Centric Information Exchange (Hardcover)
by Bernard P. Zeigler (Author), Phillip E. Hammonds (Author) Academic Press (August 3, 2007)

[For05] Forsberg, Mooz, and Cotterman, “Visualizing Project Management: Models and Frameworks for Mastering Complex Systems” 3 edition (September 1, 2005)

[OSI08]http://www.iso.org/iso/search.htm?qt=iso+9001&searchSubmit=Search&sort=rel&type=simple&published=on and http://www.iso.org/iso/search.htm?qt=iso+9000&published=on&active_tab=standards accessed May 4, 2008

[ACI06] http://www.acims.arizona.edu/SOFTWARE/software.shtml

[Gla87] L. Gasser, C. Braganza, and N. Herman. Mace: A extensible testbed for distributed AI Research. Distributed Artificial Intelligence - Research Notes in Artificial Intelligence, pages 119-152, 1987.

[Agh85] G. Agha and C. Hewitt. Concurrent programming using actors: Exploiting large-Scale Parallelism. Proceedings of the Foundations of Software Technology and Theoretical Computer Science, Fifth Conference, pages 19-41, 1985.

[Roa95] A. S. Rao and M. P. George_. BDI-agents: from theory to practice. In Proceedings of the First Intl. Conference on Multiagent Systems, San Francisco, 1995.

[Fir92] R. J. Firby. Building symbolic primitives with continuous control routines. In Procedings of the First Int. Conf. on AI Planning Systems, pages 62{29, College Park, MD, 1992.

[Wym93] Wymore, A.W., Model-based Systems Engineering: An Introduction to the Mathematical Theory of Discrete Systems and to the Tricotyledon Theory of System Design. 1993, Boca Raton: CRC.

[Öre79] Ören T.I. and Zeigler, B.P. (1979). Concepts for Advanced Simulation Methodologies. Simulation, 32:3, 69-82.

[Öre05] Ören Tuncer, “Maturing Phase of the Modeling and Simulation Discipline”, Ottawa, Ontario, Canada, 2005, http://www.site.uottawa.ca/~oren/pubs-pres/2005/pub-0512-maturing.pdf

Appendix AutoDEVS: A Tool Supporting the Bifurcated Methodology for Test Agent Development

AutoDEVS is a software tool that supports the integrated bifurcated methodology for DEVS-based test agent development [sal08]. This methodology is based on the DEVS modeling and simulation framework. It supports several stages of development that can be traversed initially in waterfall manner and subsequently in arbitrary steps while iterating until satisfactory results are obtained. Next, is a description of the different stages that AutoDEVS supports.

The first stage of the AutoDEVS methodology is defining user requirements for the system:

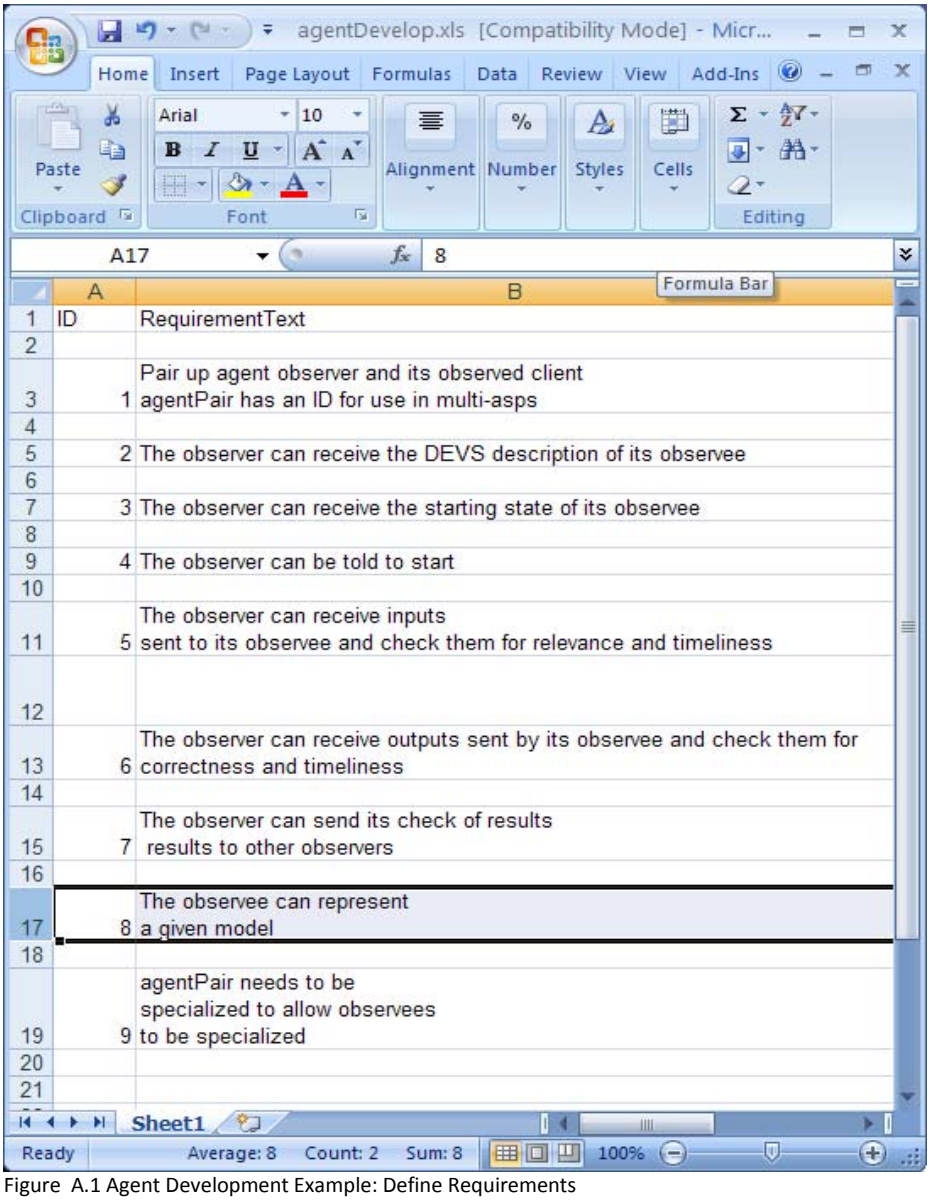
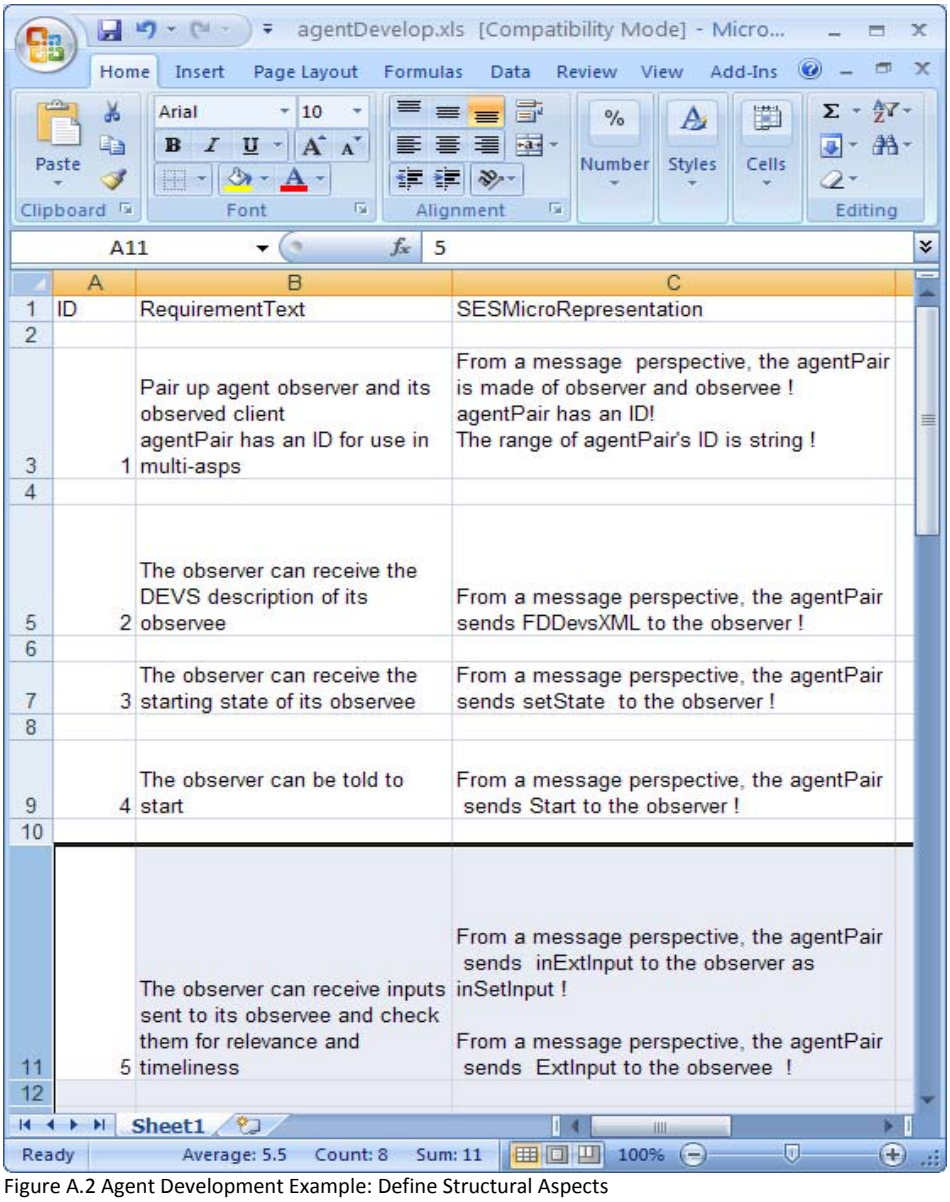


Figure A.1 Agent Development Example: Define Requirements

As seen in Figure A.1, 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, i.e. fill the “SESMicroRepresentation” column, refer to Figure A.2.



	A	B	C
1	ID	RequirementText	SESMicroRepresentation
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 !
6			
7	3	The observer can receive the starting state of its observee	From a message perspective, the agentPair sends setState to the observer !
8			
9	4	The observer can be told to start	From a message perspective, the agentPair sends Start to the observer !
10			
11	5	The observer can receive inputs sent to its observee and check them for relevance and timeliness	From a message perspective, the agentPair sends inExtInput to the observer as inSetInput ! From a message perspective, the agentPair sends ExtInput to the observee !
12			

Figure A.2 Agent Development Example: Define Structural Aspects

The third stage is describing the behavioral aspects for each of the requirements, i.e. fill the "FDDEVSRepresentation" column, refer to Figure A.2.

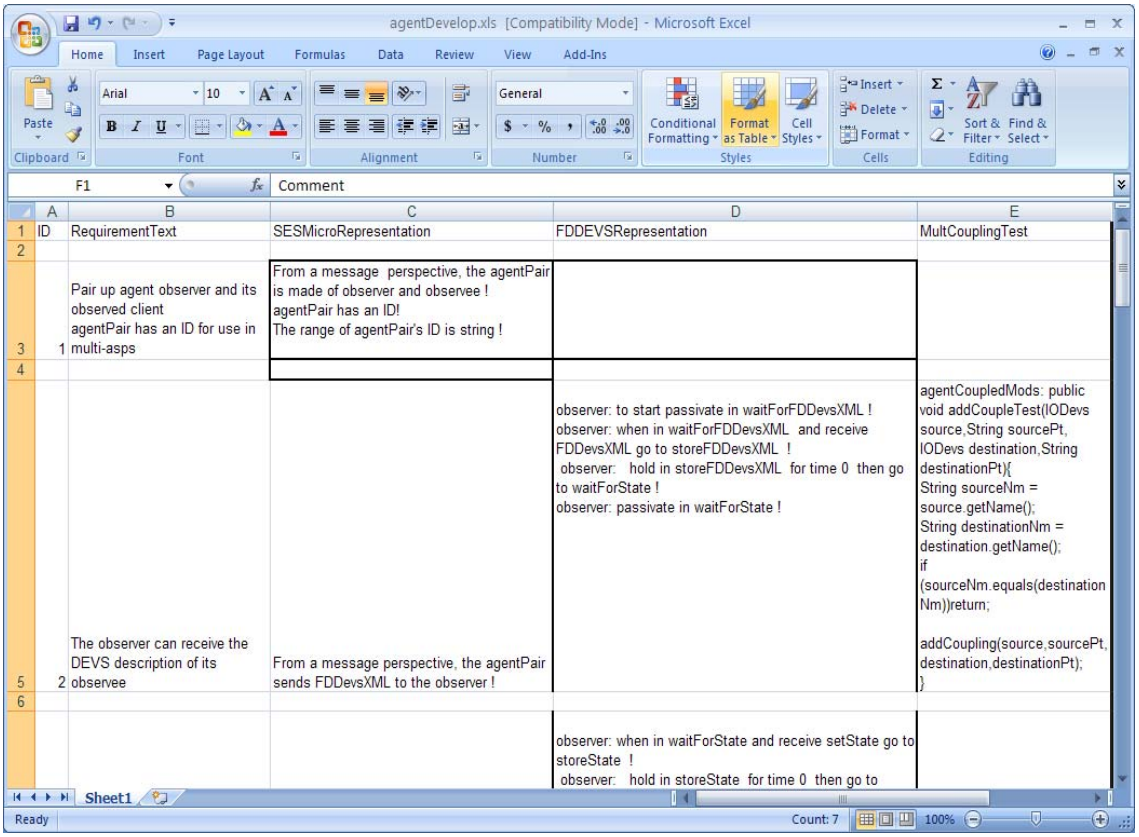


Figure A.3 Agent Development Example: Define Behavioral Aspects

Note that the second and third stages can be intermixed as the design emerges iteratively.

The fourth stage is defining the multi-aspect coupling for the coupled models, see Figure A.4. This is defined in the "MultiCouplingTest" column and automates the coupling generation for multi-aspect models.

The fifth stage is running the AutoDEVS tool to capture the spread sheet data, i.e. agentDevelop.xls, see Figure A10.

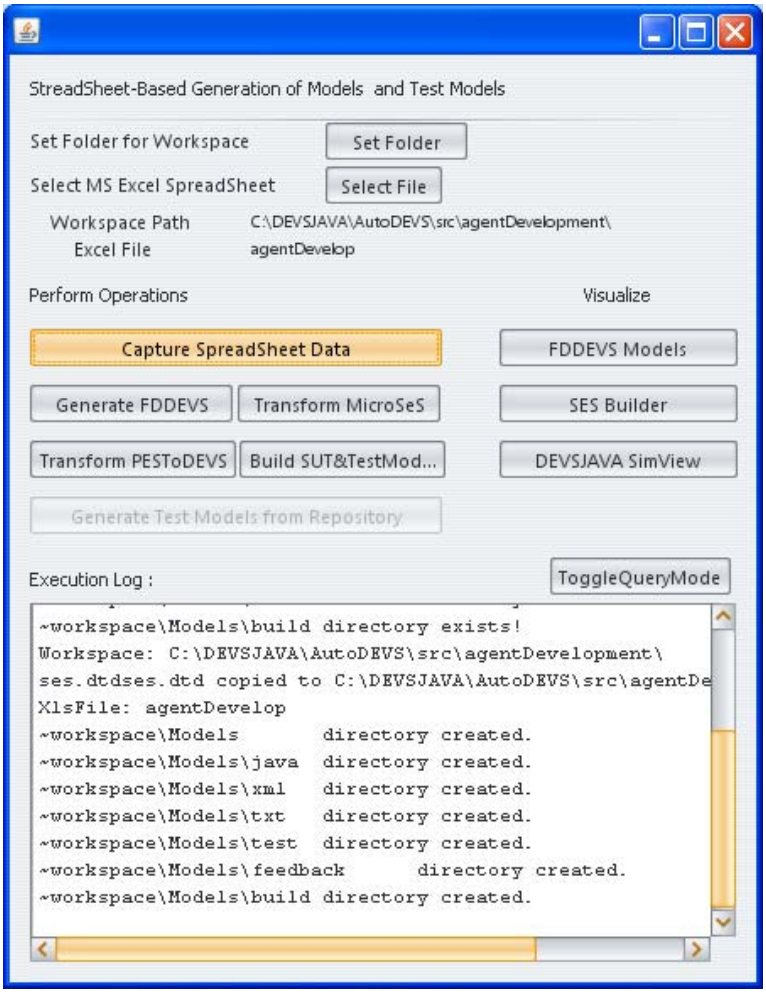


Figure A.5 Agent Development Example: Capture Spreadsheet Data

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 system (FDDEVSRepresentation column).

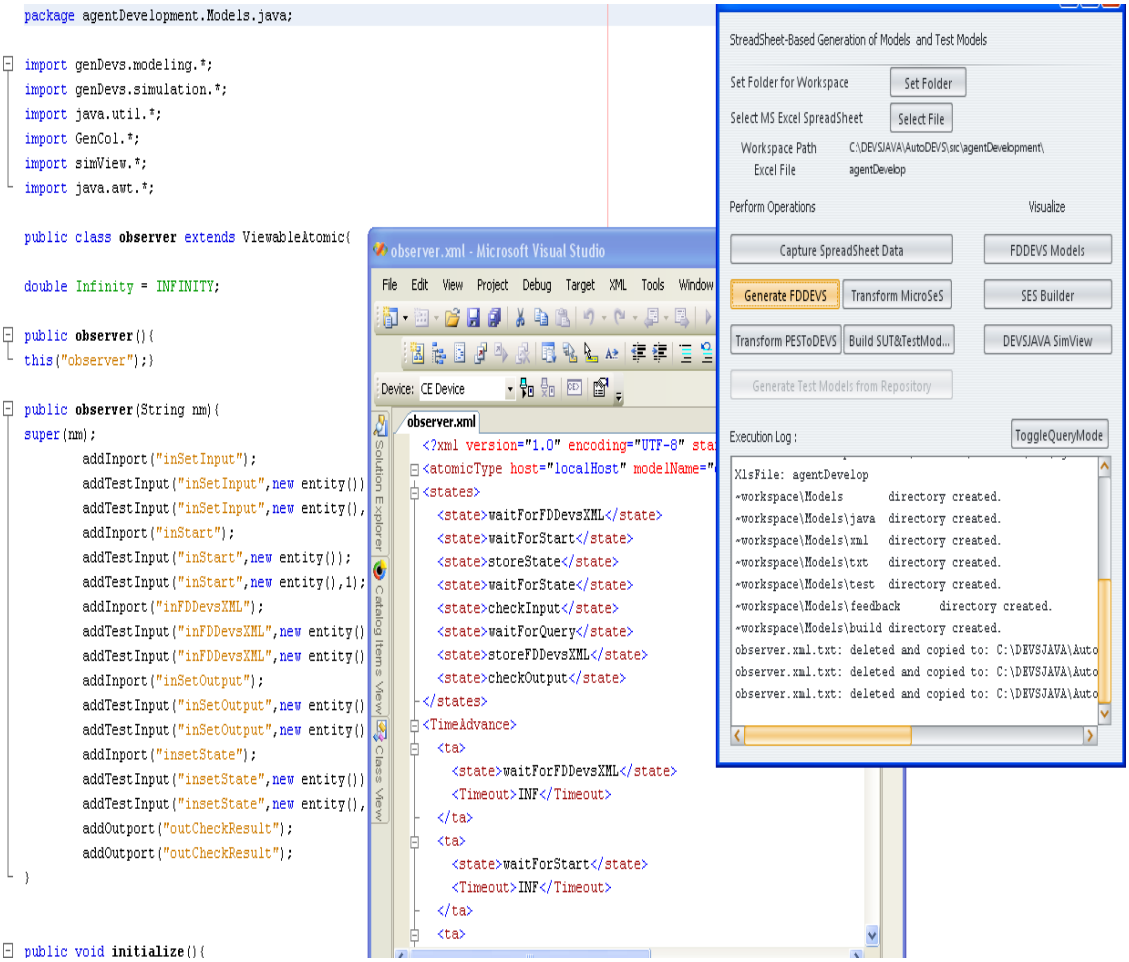


Figure A.6 Agent Development Example: Generate FDDEVS Models

As seen in Figure A.4 and Figure A.7, DEVS java models are automatically generated, including an XML representation of those models, i.e. observer.java, observer.xml. By clicking on the FDDEVS Models button, these representations can be viewed and inspected.

```
public void deltint() {
    if (phaseIs("checkOutput")) {
        passivateIn("waitForQuery");
    } else if (phaseIs("storeFDDevsXML")) {
        passivateIn("waitForState");
    } else if (phaseIs("checkInput")) {
        passivateIn("waitForQuery");
    } else if (phaseIs("storeState")) {
        passivateIn("waitForStart");
    } else if (phaseIs("waitForFDDevsXML")) {
        passivateIn("waitForFDDevsXML");
    } else {
        passivate();
    }
}

public void deltext(double e, message x) {
    Continue(e);
    for (int i = 0; i < x.getLength(); i++) {
        if (this.messageOnPort(x, "inSetOutput", i)) {
            if (phaseIs("waitForQuery")) {
                processcheckOutput();
                holdIn("checkOutput", 0.0);
            }
        }
        if (this.messageOnPort(x, "inSetInput", i)) {
            if (phaseIs("waitForQuery")) {
                processcheckInput();
                holdIn("checkInput", 0.0);
            }
        }
        if (this.messageOnPort(x, "inStart", i)) {
            if (phaseIs("waitForStart")) {
                processwaitForQuery();
```

Figure A.8 Agent Development Example: Generate FDDEVS

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 implemented by the Transform MicroSES button, SES representations of the models are 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 A.9. In addition, an automatic PES that represents the logically possible set state descriptions consistent with the SES is created, i.e. agenDevCoupModInst.xml.

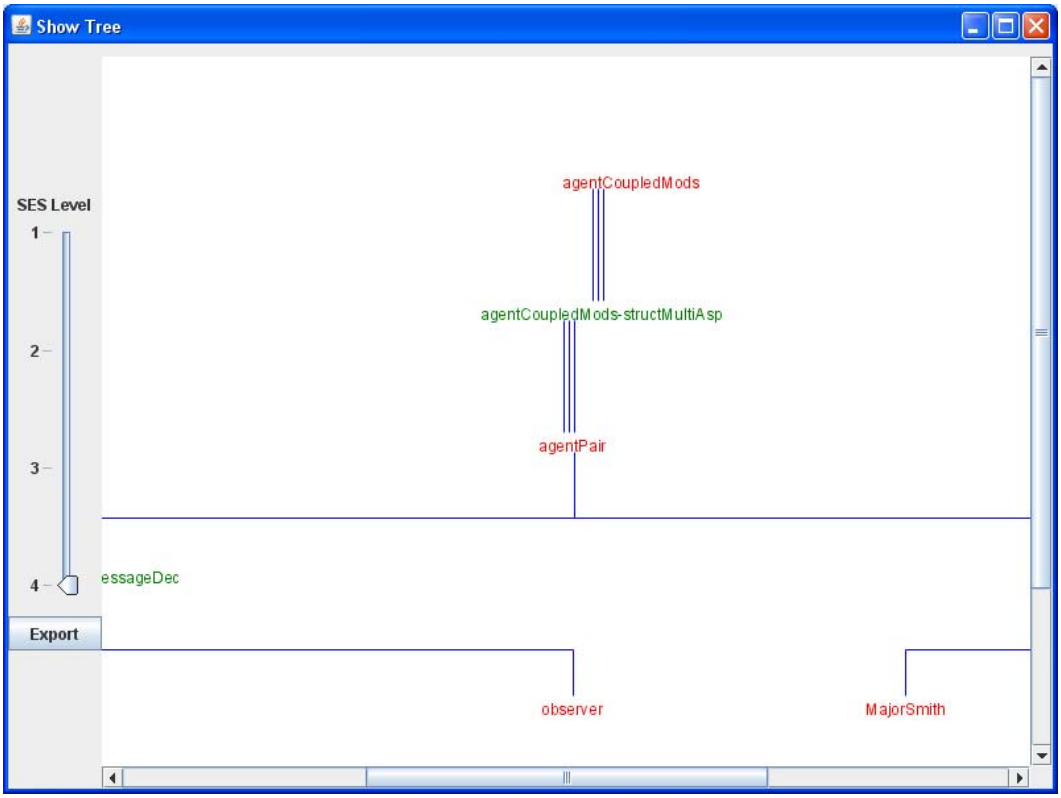


Figure A.10 Agent Development Example: SES Tree View

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 in the system, as shown in Figure A.8.

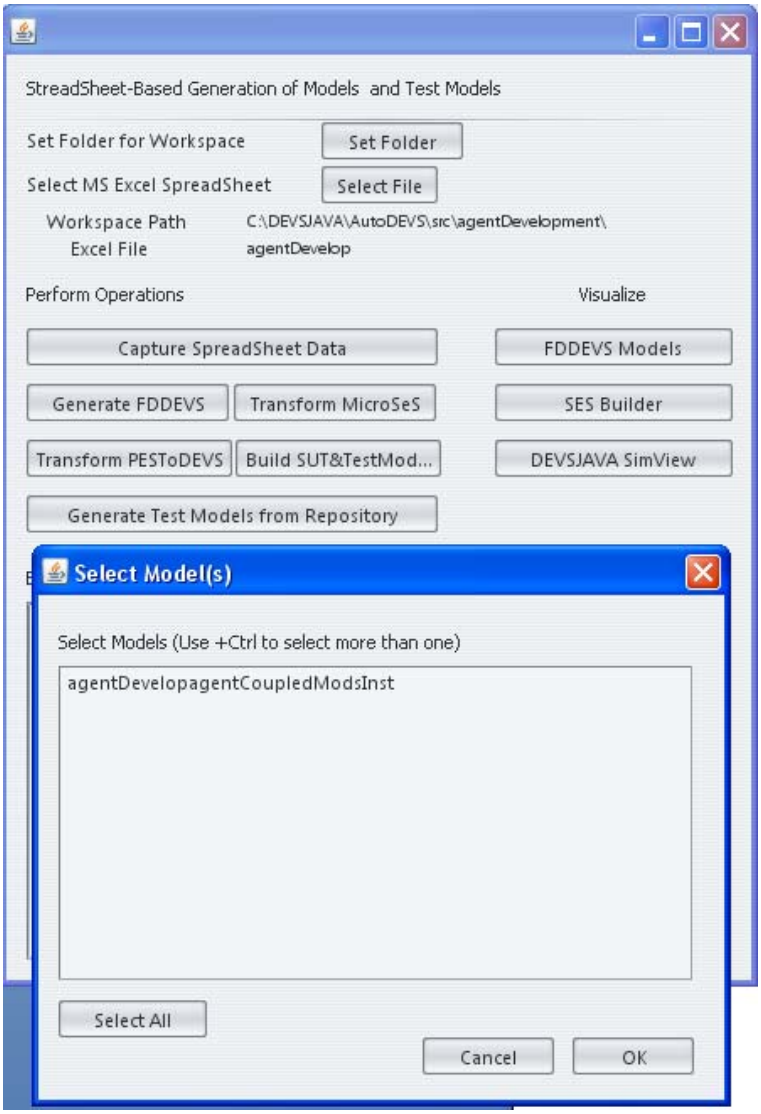


Figure A.11 Agent Development Example: Choosing a PES

The ninth stage is to create a set of test models to validate the system under development, see Figure A.12.

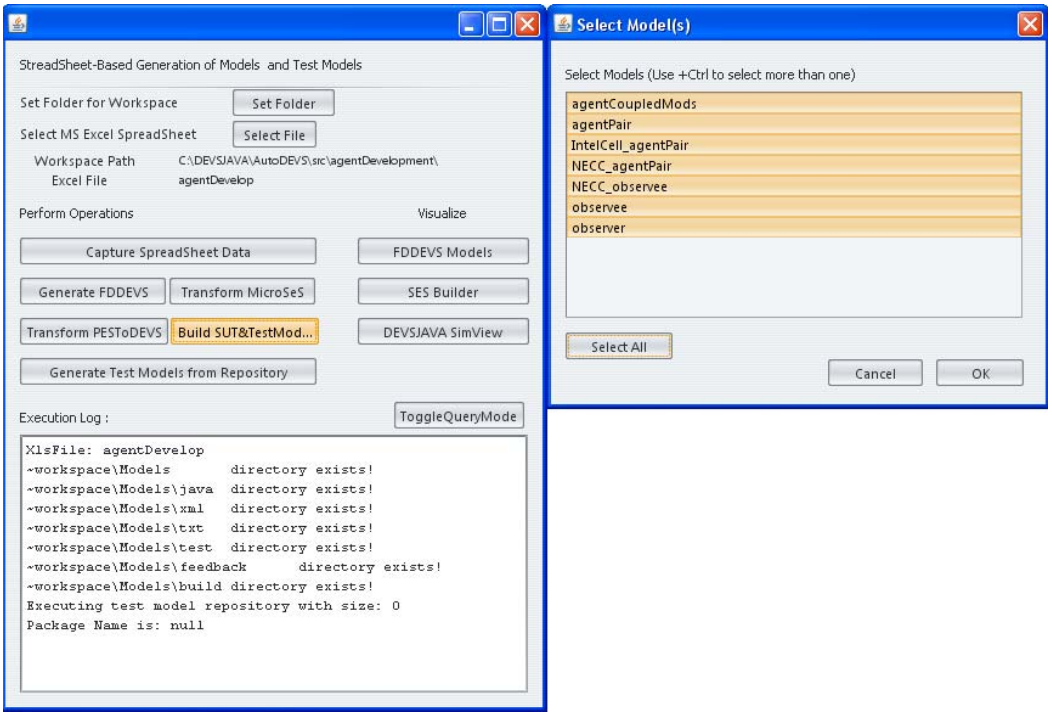


Figure A.13 Agent Development Example: Generate Test Models

As seen in Figure A.14, AutoDEVS lets the developer choose from a list the test models to create . 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 shall 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 A.11, and then to modify the models according to the desired needs and start the simulation.

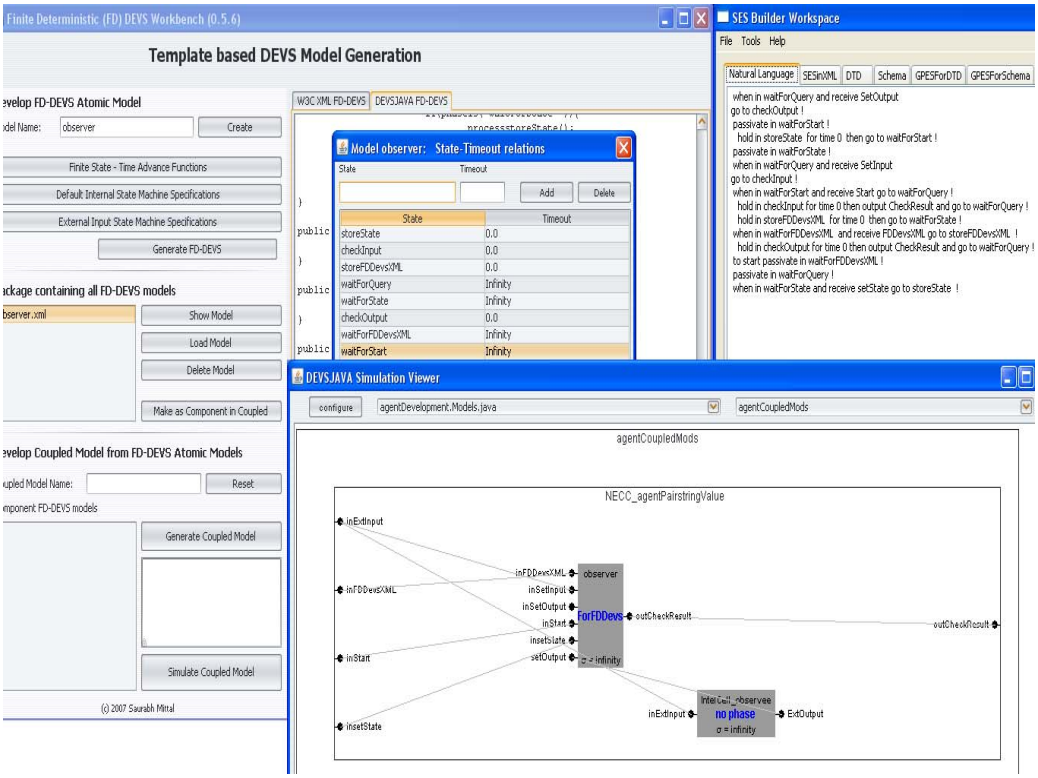


Figure A.15 Agent Development Example: Verifying models in SES, FDDEVS, and SimView

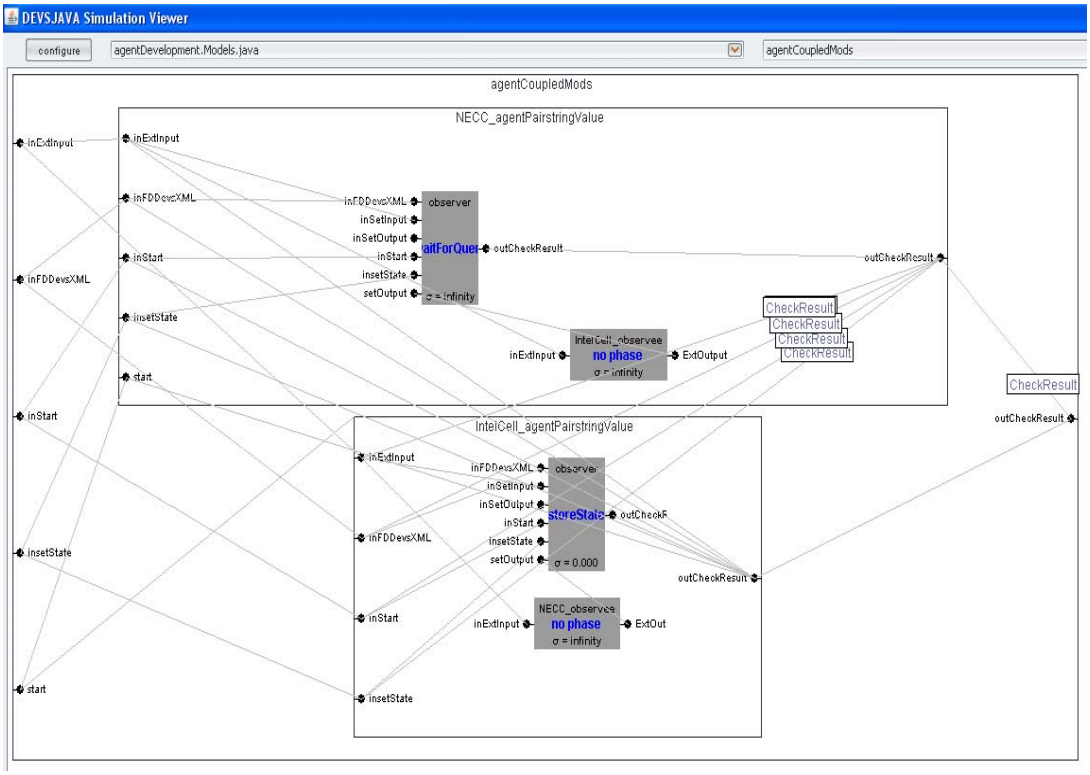


Figure A.16 Agent Development Example: Running models in SimView