BULGARIAN ACADEMY OF SCIENCES

CYBERNETICS AND INFORMATION TECHNOLOGIES • Volume **13**, No 2

Sofia • 2013

Print ISSN: 1311-9702; Online ISSN: 1314-4081
DOI: 10.2478/cait-2013-0011

Modeling and Simulation of Systems of Systems – a Survey

Anjel Tzanev

University of Chemical Technology and Metallurgy (UCTM) – Sofia. Dept. of Industrial Automation
Email: atzanev@uctm.edu

Abstract: *The paper surveys the current state of the art and perspectives in the theoretical and practical achievements at the area of Systems of Systems* (*SoS*) *and SoS Engineering* (*SoSE*). *The conceptions about SoS simulation modeling, subsystem integration and practical implementation issues in the face of main three tendencies for development* (*academic, military and industrial*) *are considered. Comprehensive generalized methodology for design, testing and efficiency evaluation of SoS, named DEVS Unified Process* (*DEVSUP*), *is briefly summarized.*

Keywords: *Systems of Systems, distributed simulation, experimental frames, System-of-Systems engineering, simulation modeling, discrete-event systems.*

1. Large-scale systems and Systems of Systems

1.1. State of the art in large-scale complex systems development

Since the early 70-es of the past century the industrial, technical and social systems have changed into large-scaled and highly sophisticated technological production conglomerates. These systems have restrictions on information structure and critical sensitivity to risks [19]. In reply to the emerging features and increased demands for control, the paradigm of *Large-Scale Systems* (LSS) has appeared in system theory.

A system is considered as large-scale, when it submits three general LSS concepts: ability to decomposition, centrality theorem for geographical distribution and property of complexity. These features require new system solutions for

operation in intensive network environments and need of integrating modern technologies to diverse applied aspects (economic, social, technical, natural, etc.).

Traditional LSSs deal with power, gas and water supply; robotics and CIMS; transportation systems; industrial processes; communication/computer networks; space/military missions. Three groups of theoretical control problems for LSS exist:

(i) Define specific performance indicators [33]: coordinability, identifyability, decentralizability, connectability, global and internal functional stabilizability.

(ii) Multicriterial and polycomponent efficiency indices.

(iii) Solving emerging problems: intelligent network systems for decision and control of critical infrastructures, autonomous AGVS, security systems, etc.

Generally said, large-scale systems become larger and more sophisticated. Practical solutions are still necessary, aiming at to include new technical progress.

1.2. The concept for Systems of Systems

The notion for *System of Systems* (SoS) is a new idea in *Systems Engineering* (SE). It arises after the endless complications of LSS. SoS is a natural extension of LSS in systems engineering. The paradigm of SoS represents a mix of independently operating and actively interacting large systems, integrated with sophisticated goal(s). SoS applications refer to *Air Traffic Management Systems*, Space exploration, ground support flight equipment, robotic space colonies, AGVS [22], communications and Internet [3, 24, 37], scientific researches (*Modeling and simulation of SoS*) [25], coast and border guard, modern army combat systems, etc.

The investigations at SoS are mainly due to the changing aerospace and defense industries, on the large-scale system integration by solutions to complex problems. SoS consists in a mix of intensively interacting independent systems with common goals during task execution. No firmly acknowledged definition for SoS exists up to now. A non-strict postulation has been given by K e a t i n g et al. [18]: *Metasystems, comprised of multiple autonomous embedded complex systems (probably LSSs), that diverse in technology, operation, context, geographical disposition and conceptual frame (but aimed to a common general goal)* [18].

The specific problems of SoS can be generalized in the following directions:

- Determine appropriate list of independent LSSs for execution of particular task;
- Heavily uncertain environment during SoS operation;
- Operative compatibility (interoperability) has to exist between SoS components.

The distinctions between SoS and LSS have been declared initially by [21] in March, 1998, and consist in the following general properties and characteristics:

(i) *Operational independence of the components in a SoS*.

(ii) *Evolutionary development*. Necessary changes in a SoS could be made. New functions are adding, existing ones removing or updating. SoS undergoes persistent evolution, in contrast to multi-step non-cyclic irreversible design of LSS.

(iii) *Emergent*, suddenly changing and non-predictable behavior [32].

(iv) Geographical distribution. Eisner [13], in 1993, denoted SoS as large geographically dispersed complexes. This property is adopted in all SoS publications.

(v) *Guaranteed interoperability* between two or more systems and components to exchange information and operate independently by utilizing this information [16]. Only a SoS is able to interoperate its component systems.

(vi) *Complementarity*. Each system should complement the other members within SoS. The multiple perspectives ensure robust approach and design [19].

(vii) *Guaranteed integrity and On-line membership optimization*.

(viii) *Holism*. The overall entity is more informative than the sum of its parts.

The comparison between LSS and SoS modeling processes shows opposite directions. The typical LSS modeling process evolves from top to bottom of the hierarchy. On the contrary, considerable difficulties arise during the SoS modeling, where the appropriate development is oriented "from bottom to top" of the system.

The problems to SoS progress are set by the general consumers and promoters:

- Tendency of fragmented perspectives for practical development;
- Lack of rigorous investigations and extensive theoretical backgrounds;
- Domination of IT and technical achievements in comparison to theoretical results;
- Limitations by the adopted conception of SE, focusing on a single system;
- Lack of theory or methodology for integrated systems analysis.

2. System of Systems Engineering

The first descriptive characteristic for the notion of the System of Systems Engineering (SoSE) is made primarily by Keating et. al. [18] and Jamshidi [17], both giving the following suggestions:

"Design, deployment, operation and transformation of metasystems, that must function as an integrated complex system to produce desirable results."

Although there is no firmly accepted definition for SoSE up to now, actually SoSE is an integrated approach to upgrade existing systems to newer, more powerful and improved systems for command, control, communications, computer hardware intelligence, surveillance, reconnaissance and innovative logistic support. So, the main research areas of SoSE are organized around the following topics [28]:

(i) Defining structure optimization, combinatorial design solving and control.

(ii) Assessing, uncertain decision making in stochastic operating environment.

(iv) Domain-specific modeling and simulation. Identify the areas of potential risk and additional analysis. Sets the operational development, mission rehearsal.

The field of SoSE does not seem to comply precise logical organization for the different descriptions, conceptions and suggestions of what constitutes a SoS and what SoSE is [18, 31]. A divergence in SoSE between *hard system* solutions and *soft system* inquiry has been achieved [31]. It consists mainly in the following [19]:

- General omission of what constitutes a SoS. It is currently a subject of disputes;

- No strict definition of SoSE. The methods and standards are not widely ratified;
- Research works continue to be fragmented and mainly application-oriented;
- The relationships of SoSE to the cognitive area of SE are rather weak;
- The significance of information technology (interoperability) is strongly overestimated, while underrating human, social, organizational and other policies.

SoSE development is a complex task, faced by a staff of system researchers. Main characteristics of this domain are consisted in the following items:

- Information systems and technologies, originally not dedicated to SoS tasks;
- Various local operators with incompatible requirements and distinct objectives;
- Insufficient, varying uncertain resources and potential instabilities during tasks;
- Shifting conditions and necessity of holding the emerged events and effects. Thus, the stability requirements to life cycle driven approaches is unreliable;
- Technology advances from the capabilities and infrastructure compatibility, necessary to support the development, integration, maintenance and progress of SoS;
- Urgent demand for response actions to prevent task crisis and failures;
- Increasing complexities and uncertainties, calling in question the ability of classical systematic approaches to effectively deal with SoS problems.

These specifications are not reachable in the near future and are expected to become more complicated. Shift from the classical SE approaches is mandatory.

This examination gives variety of perspectives and applied developments, but considers the fragmented nature of SoSE. A general problem remains the invention of a framework for creating rough structure to SoSE developments. It must provide an overview on the central design problems and their implication for progress.

The SoSE area encompasses three main perspectives: military, academic and industrial. This reveals the source of fragmentation and future divergent developing.

The term “SoS” has already been adopted with significant generality, but insufficient number of sources provide reliable tests for approving what can be accepted as “System of Systems”. Actually, SoSE paradigm has no large succession to be distinguished from other related analysis and design areas, like SE for instance. SoSE appears to be a necessary extension and evolution of classic SE. The general distinctions between SE and SoSE refer to the following points [28]:

- Greatly expanded SoS requirements for ranking level of strictness and rigor;
- Centralized in substance (although distributed in space) control structure in LSS versus decentralized control and component disposition of SoS;
- Comparing a standard stand-alone large-scale system with well-defined final states, fixed budget (resources), clearly planned schedules, technical baselines and homogenous structure, to a typical SoS with vaguely defined end states, periodical budget variations (supply, delivery) and heterogenous structure of equipment.

The research on SoSE exists in a lot of tasks. Although the area lacks conceptual completion, a conclusion has to be made for its commonly accepted assumptions:

- Higher level of interactions among component systems;
- SoS provides services, behavior or performance, impossible to any single system;
- Each subsystem operates independently to its own goals, different by that of SoS;
- A SoS is often constituted by multiple LSSs, functioning in a common mission;
- Intended and random chains of events are emerging as a result of the interactions between the LSSs, jointed in a SoS.

The SoSE requirements are summarized in ensuring compatibility and interoperability of the system components with highest levels of operational efficiency.

Three perspective directions are examined as a proof of the observed field fragmentation [11]: academic, military and industrial developments:

(i) *Military perspective*. It is issued by the US *Department of Defense* (DoD) *Architectural Framework* DoDAF and focused on the interoperability of technical command and control systems. The set of individual systems requires integration of the separate "technology" components in an overall SoS.

(ii) *Academic perspective*. It is the weakest trend of progress, but demonstrates potential for future rigorous development in philosophic and theoretical aspects.

(iii) *Industrial perspective*. It possesses a more robust view of SoSE and considers an industrial enterprise as a SoS, existing well beyond the technology.

2.1. Military perspectives of SoSE

The SoSE investigations are most closely linked to the military applications. In addition, the military perspective has evidently dictated to a large extent the former directions in SoSE progress, thus concentrating them in four main topics:

(i) Adopting the *technological aspects* as primary. The SoS in military applications is focused on supporting the command and control activity through integrated use of SoS technologies to achieve success in the planned operations.

(ii) Imposing *interoperability* as a central design objective. In addition to technical perfection, the military specifications require all technical (sub)systems in a SoS membership to be interoperable.

(iii) Extrapolation from SE. Related to demands, traceability and architecture.

(iv) Strong attention on technical devices. The military perspective of SoSE is interesting mainly for obtaining (technical) equipment, able to operate jointly.

The military orientation complicates the advances on the SoSE area. In recent US DoD publications [10, 20], the SoSE future is recognized by the next notions:

(i) There is no clear difference between SE for systems and SoS;

(ii) The single difference consists only in the fact, that no one controls a SoS;

(iii) There is no novelty in a SoS, moreover, any system composed by subsystems, is a SoS. This tendency originates since the statement of SE.

The promotion and perspective development of SoS is mainly pursued by the US military department, as declared in the *Systems Engineering Guide for SoS* [11], where there is a suggestion for SoSE as an extension of SE. This tendency is approved by some important definitions about the military perspectives of SoS and SoSE:

Definition 1. SoSE deals with planning, analyzing, organizing and integrating the potentialities of a group of existing and new systems into SoS capability, thus exceeding the sum of resources in the constituent elements.

Definition 2. The general objective for developing a SoS consists in satisfying the demanded system capabilities, that can only be created with a mix of multiple, autonomous and interacting systems.

Definition 3. A set of interdependent systems, that are related or connected together to provide a given capability, is a SoS. The loss of any system component will significantly degrade the overall performance or potentials. The development of a SoS is a compromise between the systems and their individual system performance.

The military oriented concept of SoSE is focused only on extrapolation from SE. Influence from the military perspective to SE and SoSE also exists (Table 2.1).

Table 2.1. Important areas in SE and SoSE [34]

Characteristic	Classic SE	SoSE
Aim	Development of a single system submitting the client's requirements and necessary preset performance	Obtaining new SoS capabilities by activating the force of coherent action between the participating systems and emerging properties
System architecture	Adopted early in the life cycle period. The set of expectations remains relatively stable	Dynamic adaptation when emergent needs appear
System interoperability	Interface requirements are defined and implemented for integration of the components in the system	All component systems can operate independently in a SoS by suitable manner. To enable an interoperable system, the protocols and standards are of essential importance
System properties	Reliability, maintainability, availability. Coordinability, detectability, identifiability, internal functional stability, decentralized stabilizability [33]	Extension to special properties: flexability, adaptability, composability [33], integriability, structural controllability, connectability, global and structural stabilizability
Acquisition and management	Centralized acquisition and management of the system	Separate inclusion of the component systems, still remaining independently managed and operated
Anticipated needs	Determination of the system needs at the concept phase	Intensive concept phase analysis, followed by continuous anticipation and forecasting, aided by real-time experiments
Related costs	Single or homogenous financial group with stable cost/funding profile and similar measures of success evaluation	Multiple heterogenous financial groups with unstable cost/funding profiles and different measures for success evaluation

2.2. The academic perspectives of SoSE

The academic researches represent a distinguished and more comprehensive approach to the SoSE area. These developments are in three main directions:

(i) *Investigate theoretical and conceptual differentiations*. The investigations on this problem are insufficient till now, with exception of a few works [19, 20, 31].

(ii) *Theoretical exploration of the phenomena*. In contrast to the other two SoSE perspectives, the academic interests are taking more analytical orientation. In the existing theoretical works the investigations, inquiry and analysis of SoSE phenomena with their origins and explanations [19, 31] are treated.

(iii) *Foundations in systems theory*. The principles, laws and concepts of this theory provide the background for academic development of SoSE and links to theoretical knowledge, but the SoSE practical developments are largely ignored.

The academic contributions to SoSE are performed in some definitions [9]:

"The design, installment, operation and transformation of metasystems, that operate as an integrated complex system to perform new desirable results. The metasystems are comprised of autonomous embedded complex systems, that can diversify in technology, context, operation, geography and conceptual frame."

2.3. Industrial trends in SoSE development

The industrial development of SoSE provides considerable distinctions. The enterprise perspectives are omitting the engineering activity from the contributors on SoS. This brings in a distance between the SE and the enterprise applications on SoS. So, an enlargement of the SoS engineering is obtained. Some features include:

(i) *Expanding the technical specifications*. The enterprise developments often include strategic and social components by a technology oriented environment.

(ii) *De-emphasizing the engineering tendency*. The industrial tendency in SoSE diminishes the significance of engineering approaches.

(iii) *Domination of architecture*. A central level of the enterprise SoS trends is occupied by the system architecture, that is a dominant paradigm in SE [21].

These tendencies are the framing perspectives for SoSE. Due to the inability to reconcile the different flows, SoSE becomes a fragmented set of clustered areas.

2.4. Continuing divergence in the SoSE area

While military, academic and private industrial tendencies prove the potential needs and use of SoSE, the related references offer diverse but divergent nature. The ongoing distinctions within SoS perspectives are met in the following topics [19]:

- Omitting some statements, no broad agreement for SoS composition is accepted;
- The SoSE philosophy, methodology or standards are not generally acknowledged;
- Research methodology in SoS and SoSE is fragmented, with many applications, but having little theoretical developments in the fundamental knowledge;
- Current goals, originating from the SoS area, are not convincingly argued.

2.5. Framing suitable structure to SoSE area

A particular framework, proposed for organization, integration and understanding the R&D activities to SoS, takes a holistic (voluntary) form and simultaneously orientates the area towards several fundamental directions. On Fig. 2.1 the holistic relationships for integrated knowledge development [19, 20] are performed.

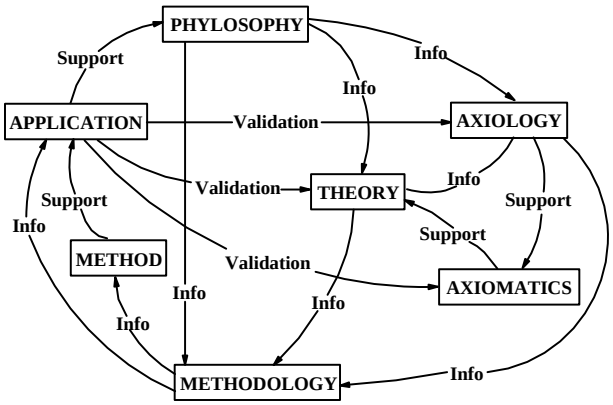


Fig. 2.1. Directions of development in SoSE area

Following, a brief review of the various interrelated topics in SoSE is provided:

- **Philosophy.** Its aim is research directed to developing theoretically consistent meanings of the paradigms in SoSE in the near future for reaching the maturity.
- **Theory.** Concerns the explanation of all phenomena in relation to SoSE, investigation of explanatory models and testable conceptual frameworks.
- **Axiological developments.** Axiology has a human-subjective component for establishing the underlying valuables via value judgment frameworks and belief propositions, that are fundamental to understand the perspectives of the SoSE area.
- **Axiomatic point of view.** Axiomatic concerns with investigating new principles, concepts and laws that constitute the basic knowledge of the prospective area [19].
- **Methodology considerations.** They concern the research activity to develop theoretically relevant frameworks that provide powerful guidance for design, analysis, implementation and advances of SoSE projects [19, 20]. There is no SoSE methodology that is dominating in the area. Each SoSE concept is only appropriate for particular application, but it is not necessarily the best one.
- **Methods for investigation.** They concern developing specific models, technologies, processes and tools for practitioners in SoSE. Most of the used approaches are adapted or extended from other research domains to be applied at the SoSE area.
- **Application development.** It considers the progress in practical implementations of SoSE via installations of science-based technologies and methods. Untill now, there is not known any large list of illustrative examples, the so called “benchmarks” on the topic.

To appreciate whether or not an application belongs to the SoSE area, the clarifying of some conceptual understandings must be considered:

- **Problem:** nature, features and definition of contents in an applied SoSE task;
- **Context:** aspects of the applied domain, embedding SoS and SoSE problems;
- **Overview:** compatibility of the SoSE application, that define the problem framing, approach and interpretation of results;
- **Approach:** creation of an appropriate methodological approach to solve a problem.

Limitations on SoSE applicability to particular problems should also be placed. Evidently, not each problem is suitable for SoSE application. But up to now, there is no valid set of distinguishing features to classify the problems as suitable.

3. Fundamentals of the Modeling and Simulation for SoS

It is necessary to show how Modeling and Simulation (M&S) helps in solving particular SoS problems.

3.1. Model based engineering

The process of model-based software engineering is generally referred to as *Model-Driven Architecture* (MDA) or *Model-Driven Engineering* (MDE). The basic idea in this approach is to develop a conceptual model before the actual *state of the art* or product is been designed and then transform the model into an actual product. The MDA approach defines the system functionality using *Platform-Independent Model* (PIM) and domain-specific language. Then, given a *Platform Definition Model* (PDM), the PIM is translated to one or more *Platform-Specific Models* (PSMs). MDA encompasses various standards – UML, *Meta-Object Facility* (MOF), *XML Metadata Interchange* (XMI), *Common Warehouse Model* (CWM), etc. [20]. An MDA tool is used to develop, interpret, compare, align, etc., models or metamodels (CWM metamodel). This tool may be one or more of the following types:

- *Creation utility.* Used to generate initial models and/or edit derived models;
- *Analysis utility.* Used to check models for completeness and inconsistency;
- *Transformation utility* to other types of models or into software code;
- *Composition utility*. Serves to group few source models into a large metamodel;
- *Testing utility.* Helps to test models. Gives a mechanism how the test cases are derived from a model, that describes some features of the system under test;
- *Simulation tool.* Contains a mechanism to execute system models;
- *Reverse engineering tool.* Transforms a particular inheritance or information product to a completed model.

A utility needs not to contain all the features of MDE. For instance, UML in MDA is a small subset of the general UML. Being a subset of MDA, the UML is

bounded to its specific UML metamodel. Recent developments have obtained executable UML models, but without wide industrial acceptance yet, due to some limitations. Potential problems with the current MDA achievements include:

- MDA is based on technical standards, yet to be specified (e.g., executable UML);
- Utilities developed by different vendors are not interoperable;
- MDA is a theoretical approach, lacking iterative features for software engineering;
- The practical realization of MDA needs skillful industrial engineers. The design imposes strict engineering rigor, not generally available by applied programmers;
- The supported by *CORBA* MDA initiative failed to become an accepted standard.

Model-based testing is a fitting procedure, organized on explicit behavior models, giving intended system reactions under the assumed environmental behavior [23]. The input/output pairs of the models are interpreting as test cases for the implementation: the model output is the expected output of the system under test. This methodology considers added abstractions and design goals, dealing with various summarized aspects, as these cannot be tested separately via the developed model. The model-based testing process consists in the following steps (Fig. 3.1):

(i) Test system model with desired abstraction is built upon the specifications;

(ii) Test selection criteria detect more or less severe faults at plausible costs;

(iii) The selection criteria are coded to test specifications. The documentation is turned into operational instructions. Automatic test case generators are also used;

(iv) A separate test module is built via the system model and test specifications;

(v) Various trial cases from the generated test module are running on the test system after appropriate priority selection. Each simulation run produces an appreciation of "*passed*" or "*failed*", or "*inconclusive*".

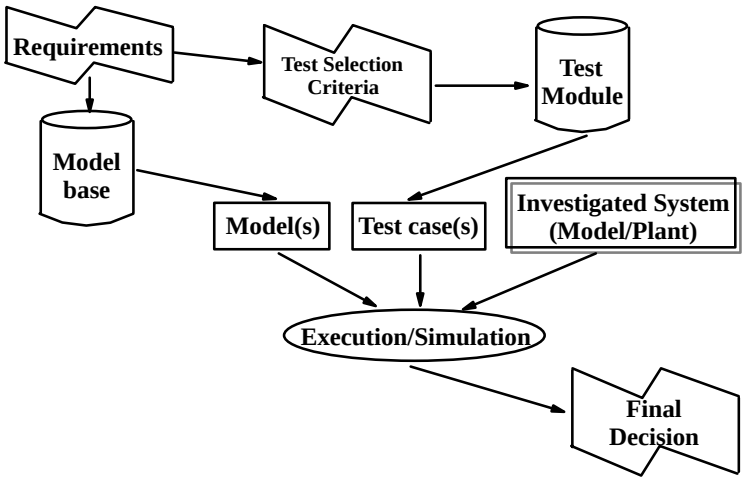


Fig. 3.1. Model-based testing process

3.2. Levels of interoperability

The SoS concept is proposed as a method to describe the use of different interconnected systems, integrated to achieve specific goals, unattainable to each of them separately [8, 9]. A common attribute of a SoS, distinguishing it from a LSS, is its interoperability, or the lack of thereof between the constituent disparate systems.

The variety of perspectives for the SoS problems is evident in [26, 28] and suggests that interoperability may take the form of integration to the constituent systems or interoperation between the composing subsystems (e.g., two or more separately independent elements or systems without established hierarchy).

Systems theory, as formulated by W y m o r e [35], provides a conceptual basis about formulating the interoperability problems in SoS. Systems are considered as components having to be coupled together to form a higher level system, i.e., SoS. The components have input and output ports, that provide coupling to be defined for information flows from the output ports to the input ports.

The review of interoperability for distributed simulation and the linguistic approach to SoS interoperability is given in [4, 9]. In [29] a parallel between the structure of SoS as a federation of systems and a set of systems, supported by the *High Level Architecture* (HLA), is suggested. HLA is an IEEE standard, proposed by DoD to enable composition of simulations [7]. It is a network middleware layer, that supports message exchanges between simulations (federates) in a neutral format. HLA provides also a set of services to support the executing of a number of simulations. The practical use of HLA is very unsatisfactory, so two distinct notions for interoperability are defined: the data exchange between heterogenous simulations (*technical interoperability*), and exchanging meaningful data to realize concerted interactions between system federates (*substantive interoperability*).

The notion for *Levels of Conceptual Interoperability Model* (LCIM) sets up seven levels of interoperability between co-operating systems. These are developments of the operational interoperability, defined in [8].

To attain interoperation in a federation of SoS, during development LCIM is mapped onto three pseudolinguistic levels: *syntactic*, *pragmatic* and *semantic* [37]. The pragmatic level describes how the information in the messages is used. The semantic level represents a common understanding of the meaning in the messages. The syntactic level provides common rules for managing contents and transmission of messages.

In the interoperability framework, the M&S problem is how all three linguistic levels of interoperability to be achieved. The formal foundations of M&S allow discussing the support, now available, and that awaiting in the future.

3.3. Modeling and simulation in the SoS framework

The theory of M&S [8] provides conceptual framework and computational approach for M&S methodology. It supplies a set of elements (real system, model, simulator, experimental frame) and due relations between them (model validity, simulator correctness) to perform ontology of the M&S domain.

The computational approach lies on the mathematical *Theory of systems* and features object orientation and other programming paradigms. It provides

manipulation on the framework elements and ability to derive logical relationships between them in simulation modeling. The framework elements are due to the system specifications in system theory. The framework relations are formulated in terms of morphism (preservation laws) between the system specifications. Conversely, the abstractions, supplied by mathematical systems theory, require interpretation, as provided by the framework, applicable to practical problems.

In a computational aspect, the modeling and simulation theory lies upon the formalism of *Discrete Event Systems* (DEVS). It is implemented in various object-oriented environments. By using the *Unified Modeling Language* (UML) technique, the M&S framework is described as a set of classes and relations (Figs 3.2, 3.3). Various implementations have the feature to support different subsets of classes and relations. The implementation of DEVS via the *Service Oriented Architecture* (SOA) environment (DEVS/SOA) is further outlined [23].

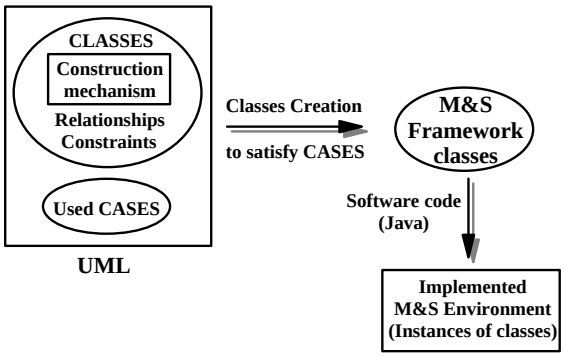


Fig. 3.2. M&S framework expressed by UML

The building components in SoS interact to each other due to interoperability, incorporated during the overall system integration. The interactions are achieved by reliable data interchanges between the systems via peer-to-peer communication or through centralized coordinator in SoS. Due to operational independence of the systems in SoS, interactions between them are generally asynchronous. A simple robust solution to serve such asynchronous interactions (message receiving) is by setting events at the receiving end to pick up the messages from the contacting systems. These system interactions can be described efficiently by discrete-event models.

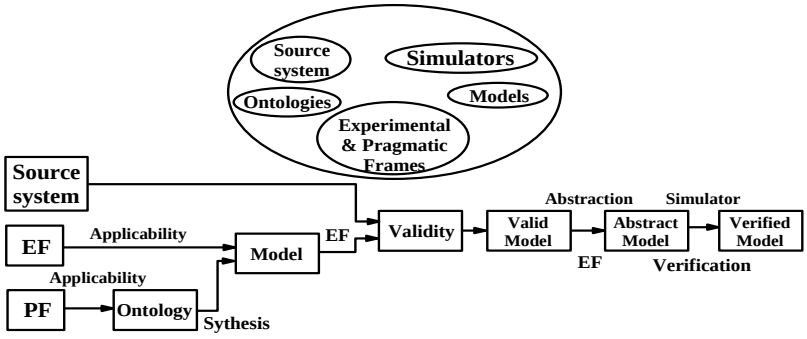


Fig. 3.3. M&S framework classes and relations in UML form

3.4. Modeling of SoS architectures

A preferred medium for structure specifications in industrial applications is the UML for its comprehensive multi-model power. But the UML primitives are not sufficient for full specification of SoSE operations. The better organization and support of the SoSE process require a more extensive architectural framework. Examples for implementing UML in the SoSE concept are the *Wymore's Framework* [35, 36], the *DoD Architecture Framework* (DoDAF) [12], the *Zachman's Framework*, *IDEF* functional modeling method, etc. UML has gained large support as a powerful graphical performer of multiple SE subprocesses in SoSE frameworks.

M&S takes integrated role in the SoSE design with respect to the theory of systems engineering. The DEVS technology serves as a mediator, placed between the I/O requirements and the technological specifications, aiming to provide M&S design level much before any design is considered as feasible. DEVS is a component-based modeling and simulation formalism, based on "port" recognition of events. It intersects the three regions of Wymore's theory for systems Engineering (SE) – Fig. 3.4.

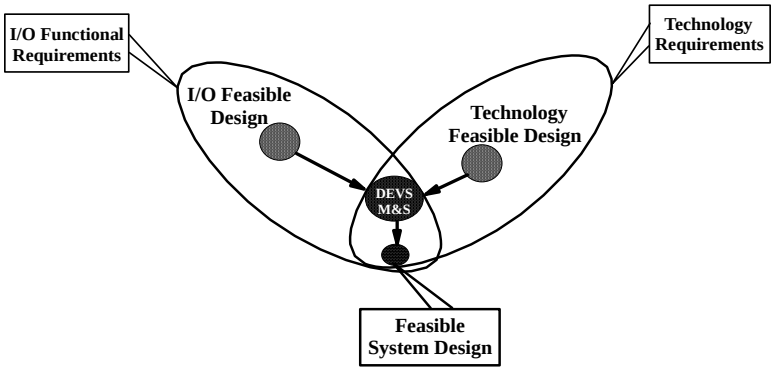


Fig. 3.4. Role of M&S in the three-component theory for Systems engineering

3.5. DoDAF design standard for SoSE

DoDAF (US *Department of Defense Architectural Framework*) is a conception for integrated architectures (*DoD Instruction 5000.2* [10]). It provides specifications to operational, system and technical levels of design. Integrated architectures are the background for interoperability in the *Joint Integrated Utilities and Development Systems* at service in the US army. DoDAF aims to overcome the variety of old-fashioned design models. Integration of such models is necessary because systems, families of systems and SoS have heterogenous capabilities and integration of the models, developed on languages with various syntax and semantics, is a serious problem. Another goal of DoDAF is adopting the SOA paradigm, supported in the *Network Centric Enterprise Service* (NCES). DoDAF conception consists of three components: *Operational Level* (OL), *System Level* (SL) and *Technical Level* (TL).

OL is a description of the tasks and activities, operational elements and information transfers, necessary to accomplish DoD missions. Such missions combine

both war-fighting targets and business tasks, together with humanitarian constraints. All the components are further decomposed to separate mission threads.

The SL itself is a set of graphical and textual products for describing systems and interconnections that support general mission functions. SL associates all system resources, available to the tasks and produced by the OL.

TL is a minimal set of rules and knowledge, arranging the organization, interaction and interdependency of the system components or elements. Its purpose is to ensure that a particular system satisfies the special set of requirements.

M&S design tasks help to link up the desired goals and their possible relations, i.e., organize the transition from abstraction (OL specifications) to reality (SL implementations). M&S contributes also to the system design process.

Although the current DoDAF specification provides an extensive methodology for system architectural development, it is suffering some important shortcomings: absence of integrated M&S support for model-continuity principles throughout the development process and lack of associated testing support. To overcome these disadvantages, specification of DoDAF architectures within a development environment, founded on DEVS-based M&S, is introduced. The enhanced DoDAF specifications [23, 37] contain M&S as a tool for developing "executable architecture" and providing detailed DoDAF to DEVS mapping in simulation and feasibility analysis. The result is an improved system lifecycle development that contains *model-continuity based design and testing* in an integral form.

4. DEVS and DEVS for SoS

The modeling formalism DEVS underlies the high performance simulation-based environment for SoS. The main concepts of the DEVS formalism [39] and simulation methodology describe the modeling/simulation phenomena via four basic objects:

Object 1. *The real system*, existing or designed, is regarded as a general data source.

Object 2. *The model* is a set of program instructions for generating relevant data, compatible to observations from the real system. The structure of the model is transferred by a set of instructions. The model behavior contains a set of all possible data that can be generated by correct executions of the model instructions.

Object 3. *The simulator* executes the model instructions to generate the actual feasible behavior of the system.

Object 4. *Experimental frames* establish how the researcher objectives influence the model structure, experimentation practice and validation results. The frames are formulated as model objects, such as the general models. So, the pairs "model – experimental frame" are composing coupled model objects to simulate model behaviors.

All these basic objects are bound together by two relations.

(i) *Modeling relation*. It links the real system and the model by defining how well the model represents the system or the subject being modeled. In general, a model can be approved as valid if the data, generated by this model, approaches sufficiently near the data obtained by the real system in an experimental frame.

(ii) *Simulation relation* associates the model and the simulator by representing how successfully the simulator can execute the program instructions of the model.

The basic data, generated by a system or a model, are multiple *time segments*. They represent mappings from defined specified time intervals to values of different variables. The variables can either be measured or observed by software observers.

The model structure may be expressed in a particular mathematical language, called *formalism*. In brief, formalism defines how to generate new values for the variables and the particular time moments when these new values should appear.

The discrete event modeling formalism generates changes in the variables as piecewise constant time segments. So, a discrete event is a sudden jump in values of the variables, occurring instantaneously. Important feature of the DEVS formalism is that the time intervals between the discrete random events are varying, in contrast to the discrete-time simulation with generally fixed quantized time steps.

The independence from fixed time steps gives valuable advantages to DEVS regarding heterogenous modeling and simulations. Multiprocess models contain a number of sub-processes, operating at different time scales. The calculation of such set of models is difficult under a common discretization time step, because the simulation procedure would behave inherently inefficient with the uniform smallest time increment to all states of the processes and frequent updating would be time-consuming to the slower sub-processes. In contrast, when a discrete-event model is used, every component organizes its own time control till the next internal event. Thus, the component models demand processing resources just at an extent imposed only by their own dynamics or necessary speed to responses on external events.

DEVS are regarded to the formalism of H o [15] for *Discrete Event Dynamic Systems* (DEDS) [40]. DEVS were introduced for discrete-event dynamic modeling and simulation [39]. Due to their theoretical backgrounds, DEVS are adopted as formal abstract notation for specifying systems with sudden jumps (discontinuities) in their piecewise continuous trajectories of inputs, outputs and states.

Discrete event models provide also a natural theoretical framework to include discrete formalisms for intelligent systems – neural networks, fuzzy logic, qualitative reasoning and expert systems. But the conventional differential equation models remain basic paradigm for performing technical environments to intelligent agents. The DEVS-based systems theory of mixed discrete-continuous formalisms provides a general powerful framework for modeling, simulation, design and analysis of technical and computerized systems. Any causal dynamic system, possessing piecewise continuous inputs and outputs, belongs to the class of *DEVS-representable systems*. In particular, systems specified by differential equations, are equally used to describe both the controlled system and the controller, the latter demonstrating a genuine DEVS behavior as decision making element.

DEVS approach supports creation of new combined models by interconnecting stand-alone models as components. This interconnection is specified in a well-defined manner, contained in the formalism of the *coupled models*. The ***closure*** property under coupling guarantees that coupling of a class of instances results in a system of the same class. The class of DEVS-representable dynamic systems is closed under coupling [40]. Closure is a substantial property since it justifies

hierarchical, modular structures of both DEVS models and the original system (continuous or discrete) they describe.

4.1. Discrete-event modeling and simulation of SoS

The DEVS formalism [5, 38], based on systems theory, provides a computational framework and tool to support systems concepts in application to SoS. In discrete-event modeling, the events are generated at random time intervals. More precisely, the state variations of a DEVS appear only when events are detected (or generated), though not necessarily appearing at constant times. So, the DEVS model is a feasible approach for simulating the overall SoS framework and its interactions.

Information flow in DEVS formalism is implemented on an object-oriented basis by the DEVS message concept via *container* classes for bounding *port-value* pairs. In a message, sent from component **A** to component **B**, a *port-value* pair contains an output port of **A**, and the value is an instance of the base class (or any of its subclasses) from a DEVS implementation. A *coupling* is a successive quadruple of the form

(1) (*send component* **A**, *output port* of **A**, *receive component* **B**, *input port* of **B**)

So, the coupling sets up a path through which a value is placed by an output function to an output port of **A** and immediately (in zero time) is transmitted to the input port of **B** for further processing by the latter's external transition function.

In systems or simulation tasks, running in DEVS environments, the *port-message-coupling* (*p-m-c*) concept is explicitly coded. However, for systems or simulation implementations, created earlier without the systems theory background, in existing or non-DEVS environments, the *p-m-c* concept is not straightforward and needs to be designed with the constructs of the particular environment.

In SoS engineering, application of existing components is very often used. It is then suitable to start with clear concepts and modern methodology, based on systems theory and DEVS formalism, to deal with the interoperability requirements, then translate the whole problem back to the non-DEVS outdated frameworks.

A. Implementation of confluent DEVS simulators

The main goal in designing DEVS-based high performance simulation environment for optimization of large complex systems is the *portability of models* across different platforms, though at deep abstraction level. So, DEVS models are not to be recoded for execution on serial, parallel or distributed environments. Ideally, such platform independence exists at deep abstraction level, e.g., set theory, where DEVS models are defined. But the computational platform, encompassing all these features does not yet exist, although some efforts appear. Close to these requirements is the computer ability to port DEVS models, coded on a common programming language across various platforms. There are numerous advantages following the portability:

(i) Models developed on serial machines with their whole convenient development support, can be easily transferred after appropriate verification on parallel systems for executing high performance simulation runs (Fig. 4.1);

(ii) In both parallel or distributed environment, the identical models can be applied to perform interactions between the components, executed within serial

nodes and in the same way for parallel interactions of the model components on different nodes.

Object-Oriented Programming (OOP) approach is the key for achieving DEVS portability, while retaining software flexibility on lower programming costs. The most notable feature of OOP is its ability to separate behavior from implementation, thus enabling distinct implementations with the same behavior to coexist [6].

As shown on Fig. 4.1, DEVS are implemented in an object-oriented form to enable execution on serial or parallel platforms. The DEVS formalism is expressed as objects and their interactions with details of serial or parallel implementations, are hidden within the objects. The user interacts only with interfaces, ensured in the DEVS structures while being encapsulated by the ultimate execution environment.

The DEVS formalism is modified to enable parallel execution of heterogenous large-scale simulation problems. This revised version of DEVS is the basis of a high-performance simulation environment, denoted as DEVS-C++ [5]. Due to its rapidly growing availability, C++ language is usually applied as an object-oriented descriptive instrument. DEVS is implemented via a set of classes (*containers*). In their simulated serial version, such classes provide a well-defined tool for specifying list data structures and their manipulation. A more abstract and generalized description of the container functionality provides services to group the objects into collections and coordinate the activity within groups.

B. Specification of containers classes in OOP descriptions to DEVS [6]

In Fig. 4.2, the primitives to coordinate the object behavior in a container are shown:

- *tell-all*. Send uniform command to each object in a container;
- *ask-all*. Send query to each object and return a container, holding the responses;
- *which?* Return subcontainer with all object responses *TRUE* to a Boolean query;
- *which one?* Return a container object, responding *TRUE* to a Boolean query;
- *reduce.* Aggregate the object responses in a container to a single one (e.g., sum).

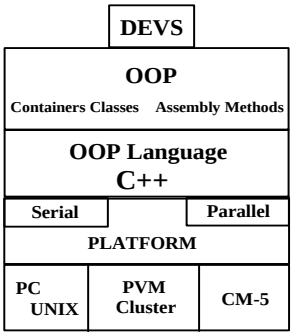


Fig. 4.1. Object-oriented implementation of DEVS on various platforms with C++

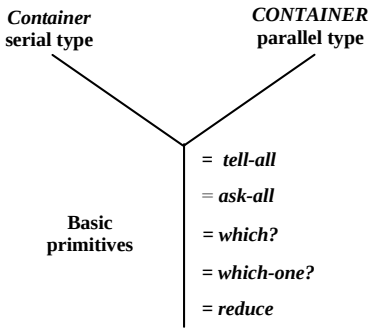


Fig. 4.2. Containers classes with basic primitives

These *Assembling methods* seem rather parallel than sequential by nature, but having abstract specifications, independent from the particular implementation. So, applying the C++ polymorphism, two abstract container subclasses are defined:

- **Lower-case class:** implement a set of assembling methods in a serial form;
- **Upper-case class:** implement same assembling methods in a parallel form (Fig. 4.2).

The serial implementation can be executed in any architecture having C++ compiler. In particular, if the nodes of a system can execute C++, then the serial type *Containers* will be compatible with them. However, the implementation of a parallel type *CONTAINERS* involves physical (as opposed to virtual) message passing between objects on different nodes. This kind of message exchange must be realized within the communication primitives, admissible by the parallel or distributed system. For example, a parallel *CM-5* implementation applies the message-passing library *CMMD* in *CM-5*. Similarly, a network of workstations, linked with *PVM* technology, offers communication primitives supplied by the *PVM* itself.

C. DEVS engines and platforms

The computational organization of modeling the subsystem processes in program simulation environments is shown in Fig. 4.3. The internal structure and system interrelations between subsystems, as well as the simultaneous functioning on different time-scales, are clearly denoted. In a separate discrete time-period during the simulation run of each subsystem, a specific data exchange takes place and the computational processes start inside its particular (micro)time-scale between t and $t+h$ time intervals. Evidently, the system in the federation of LSS has also complex structure, built up by the component modules (sub-subsystems), executed in a definite step. This shows the binding condition for *hierarchicity* in simulation of the overall task. The large time-scale of the system is the range between T_0 и T_1 time intervals.

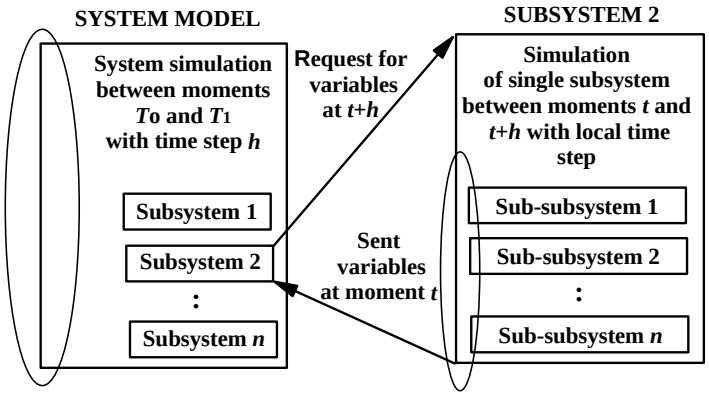


Fig. 4.3. Functional diagram for subsystem simulation of a large-scale system

A Java-based application of DEVS formalism – DEVS-JAVA, is used to atomic or coupled models [30]. In addition, distributed tasks for simulating multiple hetero-genous systems at the SoS framework, implement the DEVS-HLA environment.

Up to now, multiple discrete-event simulation engines are known (XDEVS, Matlab Simulink, NS-2, OMNET++, etc.). They are suitable to simulate interactions in heterogenous groups of independent systems. The advantage of DEVS to distributed simulation tasks is its powerful mathematical description and feasible support using middleware products, such as the software suit *HLA* of DoD.

4.2. Application of DEVS in a test and evaluation of SoS

The capabilities of DEVS for SoS Testing and Evaluation (T&E) lie in the *DEVS Unified Process* (DEVSUP). The T&E process is included in SOA due to numerous tasks by DoD. In Table 4.1 the general DEVS features to T&E are performed.

Aiming to solve the problem for war mission thread testing at the second and third system levels, a *Collaborative Distributed Environment* (CDE) is proposed in [3]. It is a combined toolbox of new and existing utilities from commercial, military and non-profitable issues. In this environment, M&S technologies are applied to support model-continuity and *Model-Driven Design* (MDD) development, making the test and evaluation an integral part of the design and operations life-cycle.

The development of such a distributed testing environment has to comply with the recent DoD trends, that require DoDAF to be adopted to describe all high-level system and operational requirements and architectures [10]. Unfortunately, DoDAF and DoD network-oriented specifications bring in significant difficulties in T&E process, since these requirements must be clearly estimated to guarantee meeting the objectives, but they all are not expressed in a strict form for evaluation.

DoDAF however, does not provide a formal algorithm to support this integration. Without such process DoDAF is non-applicable to SOA ideas and GIG in particular. There are some efforts to map DoDAF products to SOA, but with no clear methodology to develop a SOA project directly from DoDAF, though nothing is mentioned to their testing and evaluation.

Table 4.1. General DEVS resources and capabilities

M&S capabilities for T&E of SoS	DEVS solutions
Support to DoDAF for executable architectures via M&S (mission testing for GIG/SOA)	DEVSUP [23] methodology and SOA infrastructure for integrated development/testing, extending DoDAF requirements [25].
Interoperability and platform-independent M&S using GIG/SOA	Layered simulation architecture to accomplish technology advances and run various technological versions [23, 26]. Network-oriented composition and integration of validated via DEVS models using *Simulation Web Services* [23].
Automatic test generation and implementation in distributed simulation problems	Model separated from the simulation process (executed on single or multiple distributed platforms). Randomized test and development process.
Testing product continuity and traceability between phases of system development	Rapid tool for applying model-continuity principles like “simulation reality”
Real time observation and manipulation of test environments	Dynamic variable-structure component modeling to perform control and reconfiguration of the simulations at run. Dynamic simulation tuning, interoperability testing and benchmarking applications.

5. SoS Test and evaluation via DEVS modeling and simulation

SoS test and evaluation is accomplished by modeling and simulation of the large sets of systems, performed as DEVS with particular properties.

5.1. Branching Model-Continuity based Life-Cycle Methodology for SoSE design

The generalized solution of SoSE design is obtained by combining the system theory, M&S framework and model-continuity concepts. This introduces the *Branching Model-Continuity based Life-Cycle process* at the following steps:

- *Behavior requirements at lowest levels of system specification*. The hierarchy of system specifications [38] sets well-defined goals for the system behavior. The process is iterative and follows improved formal description.
- *Model structures at higher level specification*. The formal requirements become model implementations, e.g., DEVS-based transformation in C++, Java, C#, etc.
- *Simulation execution*. The developed model is copied from the *Software Program base* to the simulation module. Separation of the model from the simulator allows independent development, contrasting to the tight coupling in some older systems.
- *Real-time execution*. The simulation must be executable in Real-time mode and according to *Model-Continuity* principles. The model is a final application code.
- *Test models/Federations*. By working out the *branching system model process*, the *formalized models*, which might be developed at the atomic or the coupled level, turn into *test models* and become organized federations. For checking the system specifications, experiments and test cases are accomplished. DEVS aids the invention of experimental frames at this step of developing a test tool.
- *Verification and validation*. The simulation provides correctness check of the system specifications over a wide range of execution platforms. The test tool is verifying the consistency of such implementations in a testing infrastructure. Both operations consist in *Verification and Validation* design step.

5.2. Analysis capabilities

A DEVS coupled model allows the use of analysis tools to obtain useful information on the testing support. A mission thread model contains upper and lower bounds on the time duration for individual activities. Time windows are derived for the random events and entering of the messages. During analysis, more strictly than in simulation, it is important to select the events and messages of particular interest, derived by the performance and efficiency measures in the experimental frame. This is necessary for appropriate constraining the analysis, since it suffers from “global state explosion”. For this reason, the algorithms are so developed, that under restrictions on the model class, they derive time window specifications to inform the DEVS agents during monitoring of the related mission thread performers [23, 41].

6. Experimental frames for SoS

6.1. Experimental frames concepts

An *Experimental Frame* (EF) is a specification of the conditions for observation orexperiment, which must be conducted with the system. So, EF is an operational definition of the objectives in a modeling and simulation project.

For a particular system (source system or model), several experimental frames might be defined, and vice-versa, a single experimental frame may be applied to different systems. This is due to the various designer objectives in modeling a particular system, or having identical goals to different systems. This conception considers the *frame* as a definition of the data element types, contained into a data base.

Another formulation suggests the *frame* as a system that interacts with the tested system to obtain data of interest under specified conditions. Here the frame is described by its implementation as a measurement system or state observer.

In this treatment, a frame contains three types of components (Fig. 6.1 (a)):

- *Generator* for generating input sequences to the system;
- *Acceptor* for experiment monitoring: meeting desired experimental conditions;
- *Transducer* for observing the output sequences and analyzing the system behavior.

In Fig. 6.1 (b), illustration of a standard type experimental frames is shown. In the web context, a *generator* produces messages for service request at a given rate. The elapsed time between sending a request and its return back from a given server is the RTT. Data throughput for the observation period T_o is measured.

The *Transducer* notifies the departure and arrival times of the requests and computes the average RTT, statistics, throughput DT and the unattended requests.

The *Acceptor* monitors performance and achieved objectives (e.g., throughput).

The experimental frames transform the objectives into precise experimental conditions [38], regarding the original system or its valid models. For given objectives, there is a suitable preciseness level in model description. Thus the appropriate model abstraction depends on the objectives and experimental frame.

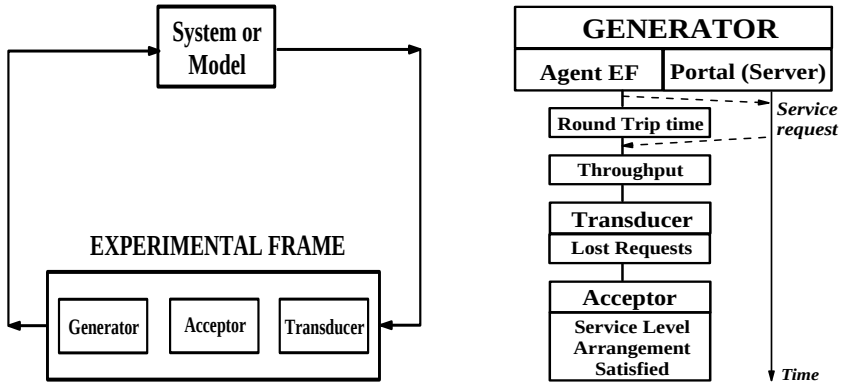
In SoSE, modeling objectives determine the development and testing of SoSs. The Measures Of Efficiency (MOE) in SoS allow the evaluation of the architectural and design alternatives. MOEs are computing via model output variables, which are obtained during repeated simulation runs. The mapping of the output variables into the outcome measures is performed by the *transducer* in the experimental frame. For some systems, more than one layer of parameters between the outputs and outcome measures exists. In sophisticated control solutions (third hierarchical level), the Measures Of Performance (MOP) (integral error, damping, settling time) are output parameters of the system operation and directly enter the MOE.

The interoperability conception of DoD [10, 11] for planning military missions with mutual participation of various services (army, navy, air force, etc.) offers

numerous SoS problems to investigate how the joint critical threads will perform in real situations. Obtaining numerical efficiency measures for joint missions requires multiple runs of the mission threads in live or virtual simulation environments.

The EF approach enables simulated and real-time data in the system information exchange and processing. Relevant MOPs refer to shared situational clarification, quality and non-delayed operative information, range and effectiveness of co-operation. The efficiency measures cover power increase, decision-making capabilities and command speed. Such abstract indices are to be modeled mathematically and precisely computed to support repeatable and consistent T&E.

The thread starts when the *Combat Command* HQ receives a mission order, exposed to the *HQ Staff*. The *Staff* performs mission analysis with results back to the commandment (Fig. 6.2). Then the *Joint Planning Group* and the *Development Teams* investigate the operations and predict the expected effects. The thread ends with an issue of orders. The performance indices include measures of collaboration and interrelations. The efficiency estimates the speed of command and goals execution.



(a) (b)

Fig. 6.1. Experimental frames for SoS testing and evaluation: building components (a); functionality (b)

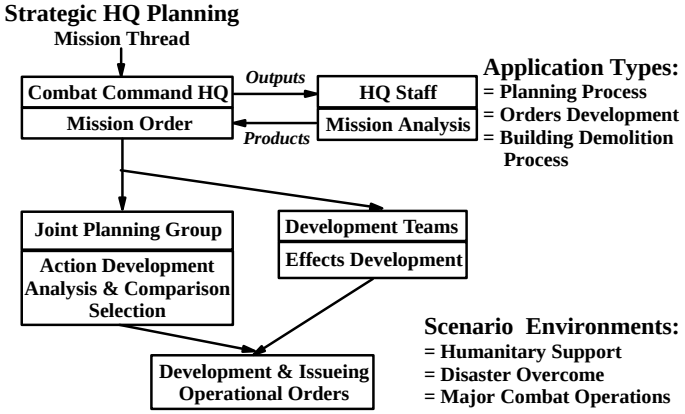


Fig. 6.2. Joint mission thread and its components

All measures might be combined together to assess the collaboration support system in an operational real treatment. The performed mission thread is further used as a template for specifying a large set of instances (Fig. 6.3). As shown, the instances can vary in several directions, including the objectives of interest (desired MOP and MOE), the type of application, the participants involved and the operational environment for testing. Various mission threads can be nested one into another. For example, *Analysis* is a sub-thread, executed within the *Planning* thread.

A collaborative mission thread can be described as a coupled model in DEVS, where all components are participants in the collaboration and the coupling between them represents the available information exchanges that can occur.

The implementation of the joint mission threads as test model federations is simulated in *network-oriented integrated infrastructures* as GIG/SOA [4]. This infrastructure ensures an environment for the mission thread components together with network and web services, allowing collaboration to them (Fig. 6.3). The formulation of a mission thread as coupled model ensures the thread simulation within the infrastructure. The defined MOEs and MOPs for assessing the mission execution need to be transformed to a distributed EF, possessing observers for check of the component activities and message exchanges, as well as *generators*, *transducers* and *acceptors*. The assessment of an integrated infrastructure to support the collaboration of a mission thread is presented in Fig. 6.3 as design of an *Instrumented Test System* (SoS). It can be solved with the M&S approach.

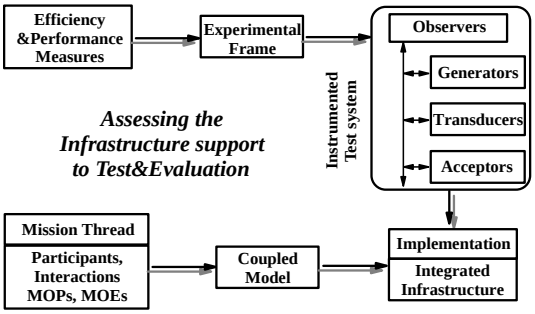


Fig. 6.3. Mission thread implementation in an integrated environment

6.2. Distributed test federations

A distributed DEVS federation consists of coupled DEVS models with components residing on different network nodes and coupling guaranteed by the middleware connectivity of the programming environment, such as *SOAP* for GIG/SOA. The federation models are executed on DEVS simulator nodes, with time and data exchange coordination as specified in the DEVS simulator protocol. According to the general concept of EF, the *generator* sends inputs to the tested SoS, the *transducer* collects outputs of the SoS to calculate statistics, and the *acceptor* monitors SoS for decisions to continue or terminate the test [41]. The EF is distributed between all SoS components (Fig. 6.4). Each system might be coupled to a particular EF, consisting of generator, transducer and acceptor subsets. An observer couples the EF to the component via interface from the integrated infrastructure.

Thus the DEVS model is built by an observer and EF, both embedded in a test agent.

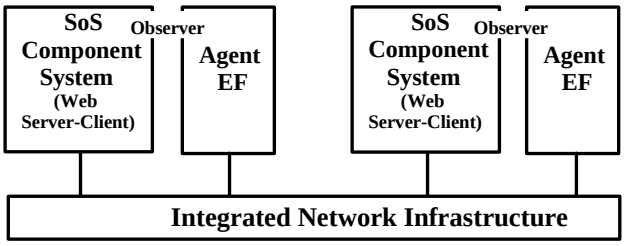


Fig. 6.4. Distributing experimental frame agents and observers

The network-based SOA provides a technical realization of the concept. The DEVS/SOA infrastructure enables DEVS models and test agents, located on prescribed network nodes. In Fig. 6.4 the network inputs from the EF generators are *SOAP* messages, sent to other EFs. The distributed EFs are arranged as DEVS models (software *agents*) on many network nodes (Fig. 6.5): emulated on web servers DEVS simulators exchange messages on a specified time within the models.

6.3. Distributed multi-level test federations

The linguistic interoperability provides structure improvements to test system. Test federations operate simultaneously at *syntactic*, *semantic* and *pragmatic* levels [32].

A. Syntactic level – network fault monitoring

From a syntactic level point of view, the testing consists in assessing the possibilities of an infrastructure to support the speed and accuracy, necessary for higher level information exchange by multimedia data types. This is a continuous assessment whether the network is sufficiently reliable to support the planned collaboration.

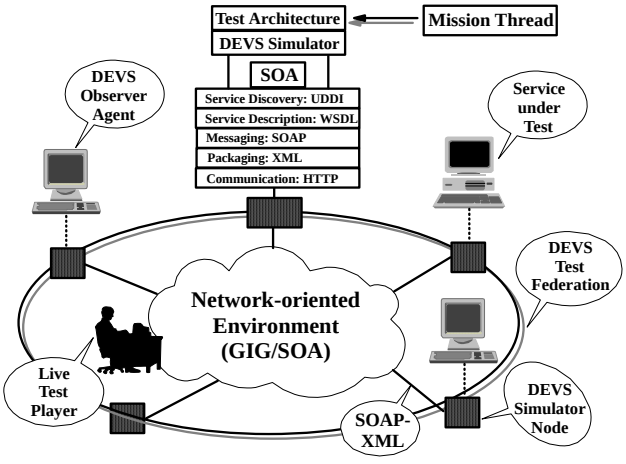


Fig. 6.5. DEVS test federation in GIG/SOA distributed environment

Nodal generator agents activate the check of the *Quality of Service* (QoS) specifications according to the information, supplied by the higher layer test agents. These probes return statistics and alarm monitoring data to the *transducer-acceptors* at the DEVS fault diagnosis layer, which cancels the virtual experiment at the test layer when QoS measures are violated. In an EF for real time network diagnostics, the test system is the network infrastructure (OSI layers 1-5) It supports higher session and application layers. QoS measures have values for meaningful testing at the higher layers: transit times and other statistics for quality. For messages in XML and transferred by *SOAP* middleware, such measures are directly produced by DEVS generators and utilized by the DEVS *transducers*/*acceptors*. Such messages investigate the network secrets and bottlenecking conditions, experienced by the message exchange procedure between higher level web servers/clients. Under certain QoS conditions, however, video streams and other types of data packets, may experience different conditions than the *SOAP*-produced messages. For these cases, lower layer monitoring is necessary under control of the nodal EFs.

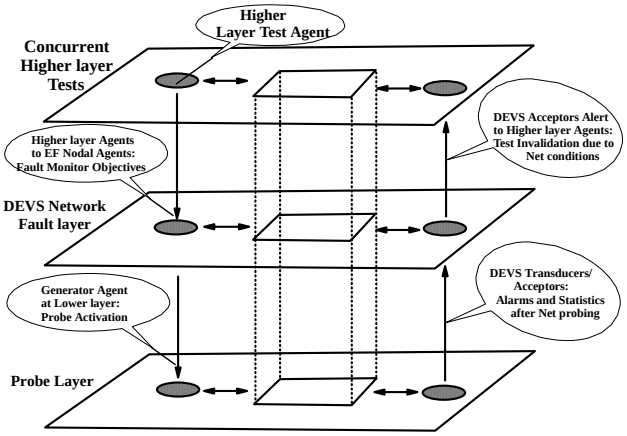


Fig. 6.6. Multi-layer testing with Network fault monitoring

The agent-based EFs have the objective to assess the network functionality according to QoS and provide for concurrent higher level tests. In this way a distributed EF is being informed about the nature of a concurrent test.

B. Semantic level – information exchange in collaborations

Mission threads consist of sequences of discrete information exchanges. A collaboration service supports such exchanges by enabling the participants to use a variety of media instruments, such as text, audio and video, in multiple combinations. At the next stage, the service supports establishing of producer/user relationships. For example, the graphical/audio combination might be directed to one or more participants interested in that particular topic. From a multilevel perspective, testing of such exchanges involves pragmatic, semantic and syntactic aspects all together. From a pragmatic point of view, however, the benefit of an exchange is how well it contributes to the successful and in-time completion of a mission thread. From a semantic perspective, the performance measures involve the

speed and accuracy of sending an information item from the producer to the user. Accuracy may be estimated by comparing the received item to the sent one, using appropriate metrics. To automate the process of comparing, metrics must be selected to be both discriminative and fast for computing. If translation takes place, the meaning of the item must be preserved as declared. The delay, involved in sending an item from the sender to the user, must be within preset limits, defined by human abilities. Such limits are more severe where random exchanges are observed on immediate priority, such as in a conversation. The instrumentation of such tests is similar to one at the syntactic level. There must be understanding that the complexity of accuracy and speed testing is of higher order at the semantic level.

C. Pragmatic level – mission thread testing

A test federation observes the agreement of web services to verify if the message flow traffic among participants conforms to information exchange requirements. A mission thread is a series of activities, executed by the operational nodes and engaging the information processing functions of the web services. Test agents watch the messages, sent and received by the services that host the participating operational nodes. Depending on the mode of testing, the test architecture may, or may not have information about the running mission thread under test. When thread knowledge is available during execution of a mission thread, the test process has more opportunities than in case of lacking information.

D. Measuring mission thread performance execution

The ultimate test for efficiency of an integrated infrastructure is its ability to support successful goal achievement of mission thread executions. To measure such efficiency, the test instrumentation system must be informed about the possible events and messages to be expected during execution, including those that provide acknowledgement for success or failure. The test system must also be able to detect and track these events and messages during the execution. It is often considered that success of a mission depends on the obtained relevant information. So, a performance objective is the ability of an integrated infrastructure to measure the extent of delivering the right information to the necessary place at the precise time moment.

6.4. Layered high-performance environment for SoS simulation

The complexity of behavior, that modern large systems can exhibit, demands computing power for simulation-based design, far exceeding that of standard workstation technology. In order to be able to solve such hard computing problems using high-resolution and large-scale representations of both natural and artificial subsystems, high performance simulation-based design environments are used with two levels of intensive knowledge and information processing. At the *Decision-making level*, searches are performed through vast problem parameter spaces of alternative design configurations and associated model structures. At the *Execution level*, simulations generate and evaluate complex candidate model behaviors, possibly interacting with human participants in real time. To represent both continuous and discrete processes, the DEVS modeling formalism is used. This is due to its significant performance and conceptual advantages. The simulation-based

task solver is a layered system of functions, containing modeling, simulation, optimization and decision making. Decision makers draw their inferences on experiments with alternative strategies (reducing the risk, minimizing the time of task execution, etc.), where the best ones, according to some criteria, are put into analysis. Experiments on models are preferred to those, carried out in reality. For realistic models such experiments cannot be worked out analytically, therefore they require direct simulation. The design of an environment to support all these activities is based on a layered collection of services, where each level uses the options of lower levels to realize its functionality. To provide generic and robust search capabilities, special algorithms are implemented for the searcher in the model space. The optimization layer employs the searcher to find reasonable or even optimal system designs. The experience with different computing environments shows, that only a large number of interconnected processing nodes can provide memory to hold the enormous amount of knowledge or data for modeling complex systems and simulation speed to provide reliable solutions in reasonable time. Currently, such large numbers of computing nodes, dedicated to a single problem, can be organized only on scalable, high-performance platforms, like *Connection Machine*, *CM-5* or *IBM SP2*, containing up to 1024 processors. However, at least a million times increase in either speed or numbers of nodes is necessary for such systems to support optimization of large-scale models. Unfortunately, the cost of such platforms is beyond most of the potential users. The other solution to networking large number of distributed computing resources (network clusters) results in 130 workstations, connected with *Parallel Virtual Machine* (PVM) over Ethernet [27], which is less than the parallel computer platforms. So, the technical barriers in the design of simulation environment are heterogeneity, portability and dimensions. A multilayered hierarchical environment for high precision simulation of large complex systems is applicable only if meets the high-level requirements towards its computational abilities and quality of results. The structure of such environment consists of three mayor layers: *modeling*, *simulation* and *searching* layer.

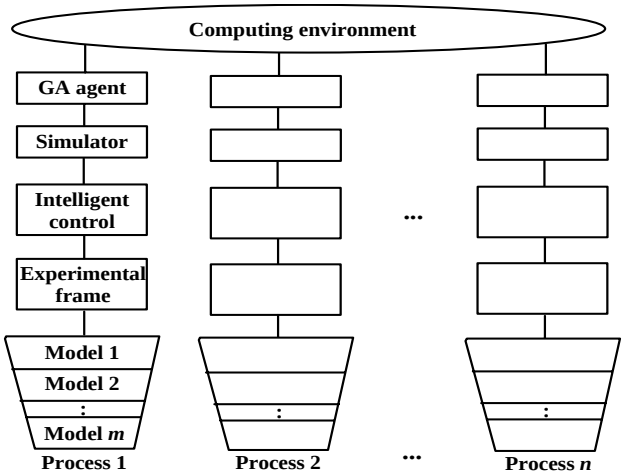


Fig. 6.7. High performance simulation environment for distributed modeling and optimization of LSS

The simulation layer executes concurrent processes in heterogenous and distributed computing multiagent environment. Single process optimization is run by an independent agent on separate algorithm in individual simulator for a model-based experiment (Fig. 6.7). Although each simulator is shown as a separate stand-by element, it could be allocated everywhere in a multiprocessor distributed computational system. Thus an experiment is consisting of numerous simulation trials, aiming to determine how well the particular intelligent controlling (monitoring, managing) agent operates. The functional environment is performed as a simulation model, governed and observed by the agent via "experimental" frame.

The model in each simulator is selected by a set of related models with different abstraction levels – from the lowest to the highest resolution of the phenomena. The optimization agent performs initially coarser searching, starting with the more abstract problem DEVS models, before going on to more precise slower descriptions of high resolution. The experimental modeling frame ensures various inputs to the process models for observing intermediate and final results. The test effectiveness of all spatially distributed configurations is estimated and reported by a GA agent and manipulated by distributed functional agent.

7. Service Oriented Application of the DEVS unified process

The *Branching Model-Continuity* process undergoes further refinement and integrates together with various elements as automated DEVS model generation tool or automated test-model generation utility via network-oriented simulation procedure over SOA to produce the DEVSUP [23], built over the *Branching Model-Continuity based life-cycle* methodology. The design of test simulation framework is carried out in parallel with the model simulation of the system under design. The DEVSUP process consists in the following components:

(i) Automated DEVS model generation in various requirement specifications;

(ii) Collaborative model developed by *DEVS Modeling Language* (DEVSML);

(iii) Automated generation of a test image from DEVS simulation model;

(iv) Network-oriented execution of the model and the test image over SOA.

Various forms of requirement specifications – state-based, natural language based, rule-based, BPMN/BPEL-based and DoDAF-based are analyzed and the automated process, used by each one to deliver DEVS hierarchical models and DEVS state machines, is evaluated [33]. In this case the simulation execution is more sophisticated than simple execution of a model on a single machine. With grid applications and multiprocessor collaborative computing, a network-oriented platform using XML as middleware results in an infrastructure, that supports the distributed collaboration and model reuse. The infrastructure provides for platform-free specification language, DEVSML and network-oriented execution using *Service-Oriented Architecture* DEVS/SOA [24]. Both DEVSML and DEVS/SOA provide a novel approach to integrate, collaborate and remotely execute complex system models on SOA. This infrastructure supports automated procedures for test-case generation leading to test models. Using XML for the system specifications in a rule-based format, a new tool (*Automated Test-Case Generator* (ATC-Gen)) is developed to facilitate the automated development of test models [37].

The integration between DEVSML and DEVS/SOA is shown in Fig. 7.1. Various model specification formalisms are supported and mapped onto DEVSML models: UML *state charts*, *exhibit-driven state-based approach* [24], *Business Process Modeling Notation*s (BPMN), *Business Process Execution Language* (BPEL) [2] or DoDAF-based specifications [25]. A translated DEVSML model is supplied to the DEVSML client that makes coordination together with the DEVSML server utility. When the client operates with DEVS-JAVA models, a DEVSML server can be used to compile the client's model with the other models available at other sites to obtain an enhanced integrated DEVSML file that can produce a coupled DEVSML model. The adopted DEVS/SOA server can use this integrated DEVSML file to distribute the component models at selected DEVS web-server simulation engines. The resulting distributed simulation, or alternatively, a real-time distributed run of the coupled model is the final point of the design process.

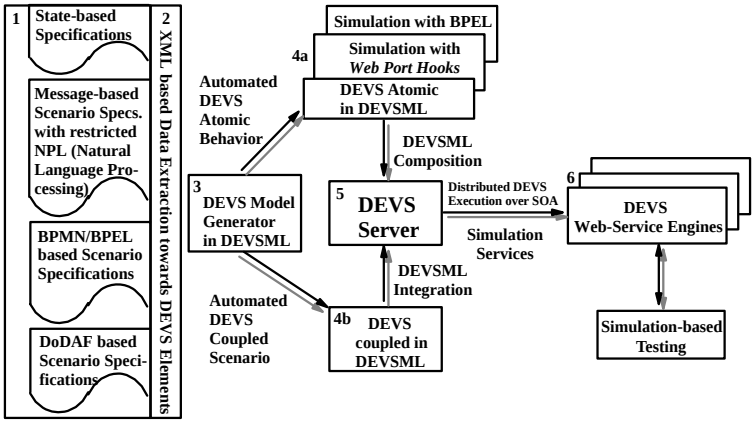


Fig. 7.1. Network-oriented collaboration and execution using DEVSML and DEVS/SOA

7.1. DEVSML collaborative development

DEVSML represents DEVS models in XML language format. The DEVSML utility is built on JAVAML (*Java Modeling Language*), which is XML implementation of *Java*. The current development popularity of DEVSML follows from the underlying JAVAML, that serves for specifying the "behavior" logic of atomic and coupled models in a SoS application. The DEVSML models have the unique feature of being transformable back and forth to *Java* and to DEVSML. This feature provides interoperability between various models and an option to create dynamic scenarios.

The DEVSML concept is represented in Fig. 7.2. It demonstrates a layered architecture. At the top level, the application layer is situated. It contains a system model in one of the several formats at DEVS-JAVA or DEVSML. The second level contains the DEVSML layer. It provides integration, composition and dynamic scenario constructing, that results in fully completed portable DEVSML models. They are transferred to any remote location via the network-oriented infrastructure and are able for execution there. Another advantage of such capability is the total simulator transparency during model execution on network-oriented infrastructures.

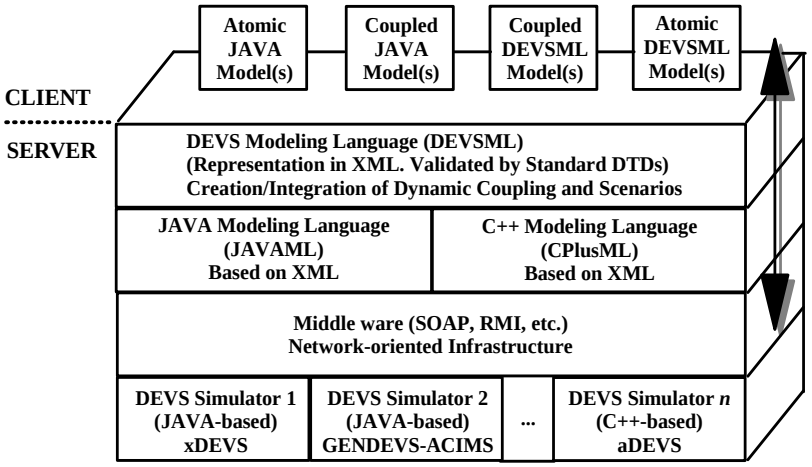


Fig. 7.2. DEVS *Transparency and Network-oriented Model Interoperability* using DEVSML. *Client* and *Server* categorization is done for DEVS/SOA implementation

The DEVSML model files in XML description contain meta-data information about the compliance with other simulation components or software versions to provide interoperability between various simulator engines. This property is achieved for at least a couple of independent simulation engines, as they have DEVS protocol to contact each other. This feature is realized by applying a single atomic DTD and a single coupled DTD for validating the DEVSML descriptions, generated by these two implementations. This type of run-time interoperability provides advantages when composing larger coupled models from models on different nodes using the DEVSML integration capabilities [24].

7.2. DEVS/SOA: Network-oriented execution using simulation services

The fundamental concept of web-based services aims to integrate the software applications as series of tasks. The use of Web services allows the application to communicate with others via open standards. The DEVS-based simulators are applying as Web service. They must incorporate the following standard options:

- Communication protocol (*Simple Object Access Protocol*, SOAP);
- Service description (*Web Service description Language*, WSDL);
- Service discovery tool (*Universal Description, Discovery and Integration*, UDDI).

The complete set of components requires one or more servers for DEVS simulation services (Fig. 7.3). The ability of running simulations is provided by the server design with DEVS *Simulation protocol*, supported by DEVS-JAVA, Version 3.1 [1]. The framework for simulation is organized as a two-layered application. The top level contains the user coordination layer, which executes the task for observation the subordinate lower layer. The lower level contains the simulation service layer that executes the DEVS simulation protocol as a dedicated service.

At the next stage of the multistep DEVS model generation is the simulation of generated models. The DEVS/SOA client obtains set of DEVS models to perform series of operations via dedicated servers for web hosting of simulation services:

(i) Upload the models to particular IP locations, i.e. make partitioning;
(ii) Performing run-time compilation at respective sites;
(iii) Simulating the coupled model;
(iv) Obtaining the simulation output at client's finish.

The specified main server houses the top level coupled model and emulates a coordinator that generates simulators in the same residence and/or on other servers.

An important feature of DEVS/SOA is the multi-platform simulation capability. It organizes distributed simulation among different DEVS platforms or simulator utilities, (DEVS-JAVA, DEVS-C++, etc.), on *Windows* or *Linux* platforms. The system is supplied with simulation services on specific platforms, managed by a single coordinator. In simulation, the overall model is partitioned according to the application platforms and executed by the appropriate simulation service. This is a form of interoperability, where multi-platform simulations are executed with DEVSML integration facilities. DEVSML is used to describe the whole hybrid model. At this stage, the message passing is realized by means of an adapter pattern in the design of the "message" class [24]. In Fig. 7.4, the *Interoperability DEVS Simulation protocol* is shown. The platform-specific simulator generates messages or event flags, but the simulation services transform them into PIMs, according to DEVS/SOA. Hence, the DEVS/SOA framework can be extended for network testing. It needs to have multi-platform simulation capabilities to achieve interaction as *Services* for concurrent test models on any DEVS environment (e.g., Java, C++).

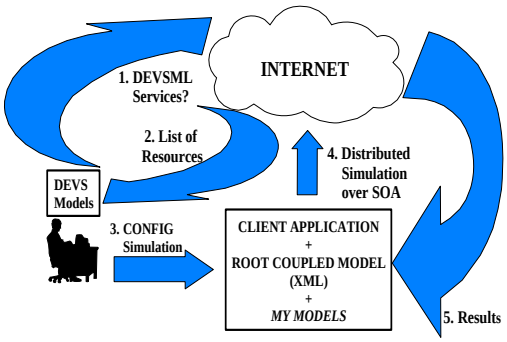


Fig. 7.3. Executing DEVS/SOA-based M&S task

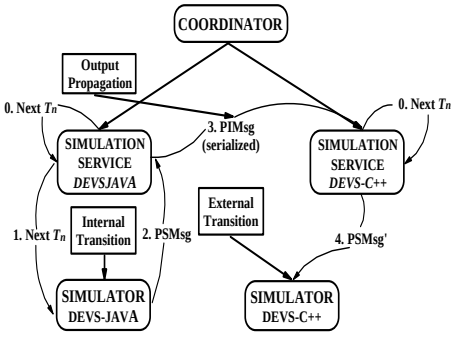


Fig. 7.4. Interoperable DEVS Simulation protocol in DEVS/SOA

7.3. Overall DEVS unified process

The complete Overall DEVS Unified Process (DEVSUP) design procedure [32] consists in the following general steps:

(i) Developing system requirements in format BPMN, DoDAF, *Natural Language Processing* (NLP), UML or DEVS-based;

(ii) By the DEVS-based automated model generation utility, generate atomic and coupled DEVS models from the requirement specifications via XML;

(iii) Validate obtained PIMs by DEVS V3C to prove their network capability for collaborative development and implement via DEVSML;

(iv) Following *Step (ii)*, either coupled model is simulated via DEVS/SOA, or a test image of the DEVS models is generated;

(v) The simulation is running in real-time or in logical time on a stand-by machine or distributed environment using SOA middleware for network execution;

(vi) The test image of the DEVS models is executed as described in *Step (v)*;

(vii) The results are comparing in a *Verification and Validation* process.

The system development of DEVS by the *Branching Model Continuity-based Life-cycle* process for system engineering is performed on Fig. 7.5.

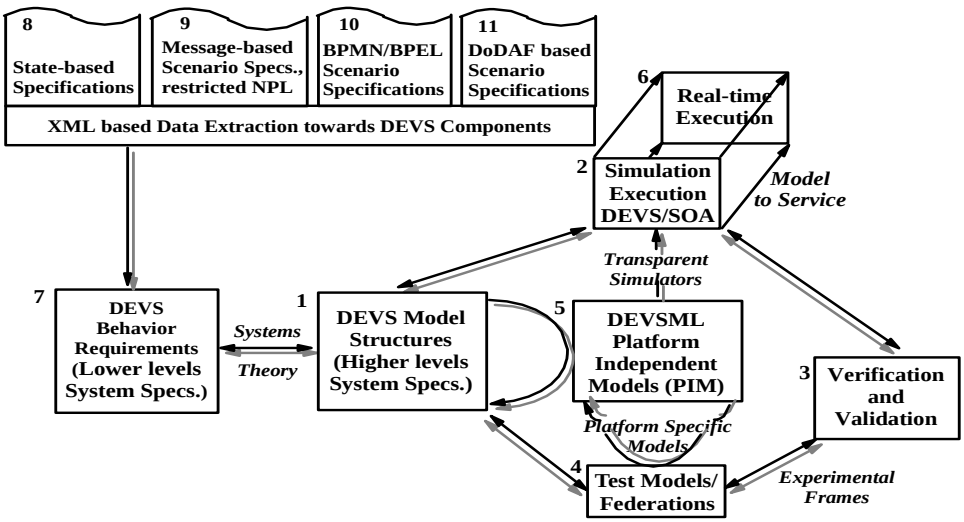


Fig. 7.5. Overall DEVS unified process development

Since up to now, a large number of case studies are based on the DEVSUP conception and many projects are at a final stage or currently active at disposal of the *Joint Interoperability Test Command* (JITC). By extending DEVS with *Branching Model-continuity Life-cycle* process, systems theory becomes able to support new generation network-based developments and software engineering test solutions.

References

1. ACIMS Software Suite.
   http://www.acims.arizona.edu/SOFTWARE/software.html
2. Business Process Execution Language (BPEL).
   http://en.wikipedia.org/wiki/BPEL
3. Carstairs, D. J. Wanted: A New Test Approach for Military Net-Centric Operations. Guest Editorial. – ITEA Journal, Vol. **26**, October 2005, No 3.
4. Chaum, E., M. R. Hieb, A. Tolk. M&S and the Global Information Grid. – In: Proc. of the Interservice Industry Training, Simulation and Education Conference (I/ITSEC), 2005. Paper 2450.
5. Chow, A., B. P. Zeigler. Revised DEVS: A Parallel, Hierarchical, Modular Modeling Formalism. – In: Winter Simulation Conference Proceedings, INSPEC Accession Number: 4869673, 11-14 December 1994, 716-722.

6. C h o, Y. Parallel Implementation of Container Using Parallel Virtual Machine. PhD. Th., Univ. of Arizona, Tucson, 1995.
7. D a h m a n, J. S., F. K u h l, R W e a t h e r l y. Standards for Simulation: As Simple as Possible, but not Simpler. The High Level Architecture for Simulation. – Simulation, Vol. **71**, 1998, No 6/7, 378-387.
8. D i M a r i o, M. J. System of Systems Interoperability Types and Characteristics in Joint Command and Control. – In: Proc. of the International Conf. on System of Systems Engineering IEEE/SMC'2006, TP3, Los Angeles, CA, April 2006, 236-241.
9. D i M a r i o, M. J. From Systems Engineering to System of Systems Engineering. SoSE Discussion Panel Introduction. – In: IEEE International Conference on System of Systems Engineering (SoSE), San Antonio, Texas, 16-18 April 2007.
10. DoD Instruction 5000.2 Operation of the Defense Acquisition System. 12 May 2003.
11. DoD. Systems Engineering Guide for Systems of Systems. Ver. 1.0. Office of the Deputy under Secretary of Defense for Acquisition, Technology and Logistics. Washington, DC, 2008.
12. DoDAF Working Group. DoD Architecture Framework. Ver. 1.0. Vol. III: Deskbook. US Department of Defense. Washigtnon, DC, 2003.
13. E i s n e r, H. RCASSE: Rapid Computer-Aided System of Systems Engineering. – In: Proc. of 3rd International Symp. of the National Council of Systems Engineering, Vol. **1**, Washington, DC, 1993, 267-273.
14. F i l i p, F l.-G. Decision Support and Control for Large-Scale Complex Systems. – Annual Reviews in Control, Vol. **32**, 2008, No 1, 61-70.
15. H o, Y.-C. Special Issue on Discrete Event Dynamic Systems. – In: Proc. of IEEE, Vol. **77**, 1989, No 1, 3-6.
16. IEEE 610.12. Standard Glossary of Software Engineering Terminology. International Organization for Standardization and IEEE. NY, 1990.
17. J a m s h i d i, M. System-of-Systems Engineering – A Definition. – In: IEEE International Conf. on System, Man and Cybernetics (SMC), Waikoloa, Hawaii, Vol. **4**, October 2005, 10-12. **http://ieeesmc2005.unm.edu/SoSE_Defn.htm**
18. K e a t i n g, C., R. R o g e r s, R. U n a l, D. D r y e r, A. S o u s a-P o z a, R. S a f f o r d, W. P e t e r s o n, G. R a b a d i. System of Systems Engineering. Engrng. – Management J., Vol. **15**, 2003, No 3, 36-45.
19. K e a t i n g, C. Research Foundations for System of Systems Engineering. – In: IEEE International Conf. on Systems, Man and Cybernetics, Waikoloa, Hawaii, October 2005, 2720-2725.
20. K e a t i n g, C., P. K a t i n a. Systems of Systems Engineering: Prospects and Challenges for the Emerging Field. – International Journal on SoSE, Vol. **2**, 2011, No 2/3, 234-252.
21. M a i e r, M. W. Architecting Principles for Systems-of-Systems. – Systems Engineering, Vol. **1**, 1998, No 4, 267-284.
22. M a n s o u r i, M., B. S a u s e r, J. B o a r d m a n. Application of Systems Thinking for Resilience Study in Maritime Transportation System of Systems. – In: 3rd Annual IEEE Systems Conference, Vancouver, 2009, 211-217.
23. M i t t a l, S. DEVS Unified Process for Integrated Development and Testing of Service Oriented Architectures. Ph. D. Thesis. University of Arizona, Arizona, 2007.
24. M i t t a l, S., J. L. R i s c o-M a r t i n, B. P. Z e i g l e r. DEVS-Based Simulation Web Services for Net-Centric T&E. – In: Proc. of the Summer Computer Simulation Conference (SCSC'07), San Diego, CA, 2007, 357-366.
25. M i t t a l, S. Extending DoDAF to Allow Integrated DEVS-Based Modeling and Simulation. – Special Issue on DoDAF Journal of Defense Modeling and Simulation (JDMC): Applications, Methodology, Technology, Vol. **3**, 2006, No 2, 95-123.
26. M o r g a n w a l p, J., A. P. S a g e. Enterprise Architecture Measures of Effectiveness. – International Journal of Technology, Policy and Management, Vol. **4**, 2004, No 1, 81-94.
27. PVM Project Team. PVM User Survey Results. 1994. **http://comp.parallel.pvm**
28. S a g e, A. P., C. D. C u p p a n. On the Systems Engineering and Management of Systems and Federations of Systems. – Information, Knowledge, Systems Management, Vol. **2**, 2001, No 4, 325-345.

29. S a g e, A. From Engineering a System to Engineering an Integrated System Family, from Systems Engineering to SoS Engineering. – In: IEEE Int. Conf. on System of Systems Engineering (SoSE), San Antonio, Texas, April 2007.
30. S a r j o u g h i a n, H. S., B. P. Z e i g l e r. DEVS and HLA: Complimentary Paradigms for M&S. – Transactions of the SCS, Vol. **17**, 2000, No 4, 187-192.
31. S o u s a-P o z a, A., S. K o v a c i c, C. K e a t i n g. System of Systems Engineering: An Emerging Multidiscipline. – International Journal of System of Systems Engineering (SoSE), Vol. **1**, 2009, No 1/2, 1-17.
32. System of Systems Engineering: Innovations for the 21st Century. Ed. Mo Jamshidi. Hoboken, NJ, John Wiley, 2008.
33. T z a n e v, A. Large-Scale Systems. Lecture Notes in AIT Specialty, Masters Engineering Degree. Published by UCTM – Sofia. 2010. 384 p. (in Bulgarian).
34. V a l e r d i, R., A. R o s s, D. R h o d e s. A Framework for Evolving System of Systems Engineering. – Crosstalk., Vol. **20**, 2007, No 10, 28-30.
35. W y m o r e, W. A. A Mathematical Theory of Systems Engineering: The Elements. Krieger. NY, Huntington, 1967.
36. W y m o r e, W. A., B. C h a p m a n, T. B a h i l l. Engineering Modeling and Design. NY, CRC Press, 1992.
37. Z e i g l e r, B. P., P. H a m m o n d s. Modeling & Simulation-Based Data Engineering: Introducing Pragmatics into Ontologies for Net-Centric Information Exchange. NY, Academic Press, 2007.
38. Z e i g l e r, B., T. G. K i m, H. P r a e h o f e r. Theory of Modeling and Simulation. NY, Acad. Press, 2000.
39. Z e i g l e r, B. P. Theory of Modeling and Simulation. NY, John Wiley, 1976.
40. Z e i g l e r, B. P., W. S a n d e r s. Preface to Special Issue on Environments for Discrete Event Dynamic Systems. – DEDS: Theory and Applications, Vol. **3**, 1993, No 2, 110-119.
41. Z e i g l e r, B. P., D. F u l t o n, P. H a m m o n d s, J. N u t a r o. Framework for M&S-Based System Development and Testing in a Net-Centric Environment. – ITEA Joutnal of Test and Evaluation, Vol. **26**, 2005, No 3, 21-34.