

Rapid and Adaptive End-to-End Test and Evaluation of Systems of Systems

Raymond Paul
Department of Defense
Washington, DC

W. T. Tsai and L. Yu
Department of Computer Science and Engineering
Arizona State University
Tempe, AZ 85287

Abstract

This paper presents DoD End-to-End (E2E) Test and Evaluation (T&E) for Systems of Systems (SoS). The E2E T&E is based on the following techniques: scenario modeling and specification, automated test script generation, dependency analysis, completeness and consistency analysis, risk analysis, pattern analysis, usage analysis, timing analysis, simulation, automated distributed test execution, statistical analysis, ripple effect analysis, regression testing. The key concepts of the E2E T&E are reusability, tight integration of system analysis and testing, and hierarchical scenario specifications.

1 Introduction

Modern military systems consist of multiple parties structured in a hierarchical manner to form systems of systems (SoS) connected by possibly multiple networks such as satellites and wireless networks. Individual systems of SoS accept the control and management of warfare centers, and also collaborate with each other to achieve the mission.

Testing computer systems is often difficult, and testing a distributed SoS is especially difficult and costly because of the complexity of the system, the real-time safety-critical nature of the mission, and technology used to implement the system as well as the associated warfighting doctrines keep on changing. Furthermore, integration system at the system level is often the most difficult and time-consuming aspect of system. Unfortunately, a review of the existing literature on integration testing shows that most integration testing techniques are mostly principles, such as incremental integration, top-down, and bottom-up integration [15], or explore

the programming or design structure of the program [9][11][17]. The techniques that explore programming or design structure are useful, however, they are applicable to software written using the related techniques. For example, an integration testing technique for an Object-Oriented (OO) program using Java may not be applicable to testing a legacy program using COBOL. It is even possible that the testing technique may not be applicable to testing an OO program using C++ because Java has no pointers but C++ does.

Due to these considerations, DoD needs an integration Test and Evaluation (T&E) technique that are applicable to a wide variety of applications implemented in different programming languages such as COBOL, FORTRAN, ADA, Java, C, and C++, and design techniques such as OO design patterns [7] and architecture. The integration testing technique must be applicable to both legacy applications (such as data management using COBOL) as well as modern applications (such as web-based Enterprise Resource Planning using CORBA and Java). Thus, the integration testing must be mostly functional (or black-box), however, it should be able to extend to include certain white-box information such as network characteristics and overall system architecture.

Recently, DoD proposed an End-to-End (E2E) integration T&E process to address these issues [18][19][20][21][25][22][24]. Specifically, the E2E T&E has the following features:

- a. **Scenario-based testing:** Testing scripts are developed based on specifying system scenarios, and scenarios are formalized as a sequence of events, actions and associated

pre-/post-conditions. Because scenarios are developed directly from requirements, it is independent of any programming languages, design techniques, or development processes such as the waterfall model, or agile development processes such as Extreme Programming [2].

- b. **System analysis and modeling using scenarios:** A system can be specified and then analyzed using scenarios by various analysis techniques [19][25][26] such as completeness and consistency analysis, risk analysis, timing analysis, usage analysis, and dependency analysis. These analyses facilitate the designer to make informed and intelligent decision in the requirement and design phases of project development. In other words, the E2E T&E tool can be used early in the life cycle during and throughout the rest of life cycle including operation and maintenance.
- c. **Simulation:** In addition to static analyses mentioned, the E2E process also supports dynamic analysis by simulation. Once scenarios are available, the E2E tool can simulate the system by tracing the conditions and actions in the scenarios. Simulation can be used together with other analysis to determine the appropriate system design. For example, simulation can be used together with timing analysis to determine if the SoS satisfies the timing requirements. Furthermore, multiple scenarios can be simulated at the same time to determine the interaction of these scenarios.
- d. **Rapid test script generation by verification patterns:** The E2E process supports rapid test script generation by classifying system scenarios into patterns, and each pattern has a corresponding test script that can be parameterized to test all the system scenarios belong to the pattern. This approach promotes test script reusability and reduces the cost of test script generation significantly [29]. This approach has been used successfully to test commercial real-time safety-critical embedded medical devices such as pacemakers and defibrillators.
- e. **Distributed automated test execution:** The E2E tool support distributed test execution by providing an architecture with a test master and test agents. The test master is responsible for managing test scenarios and test scripts, and sending test commands to

test agents for remote execution. Test agents are responsible in sending test commands to the System Under Test (SUT) for test execution, and collecting and analyzing data, and reporting test results to the tester master. Because a typical SoS is a distributed system, the test master and test agents often reside on remote sites and communicate via various communication protocols such as TCP/IP or SNMP.

- f. **Statistical modeling:** The E2E tool has a statistical model Assurance-Based Testing (ABT) that can evaluate the test performed [16][13]. ABT was initially designed to evaluate Y2K testing but it is applicable to general system testing.
- g. **Accommodation to changes:** When the SUT is changed, the user just needs to modify/update the corresponding scenarios specification. The framework will re-generate the test script automatically to accommodate the change and execute the new test script generated. This is supported by design patterns Strategy, Template Method, Composite, and Command [5]. The framework also supports regression testing to ensure that changes will not introduce undesirable ripples.
- h. **Uniform usage:** The E2E T&E tool uses a uniform process to perform testing: *setup*, *execute*, and *verify*. A tester has the freedom to add new testing strategies such as a new regression testing selection algorithm, but the tester is not allowed to change the overall testing process, i.e., *setup*, *execute*, and *verify*. In this way, test scripts developed can be reused while at the same time allowing new testing strategies or techniques to be introduced. This also minimizes misuses and mistakes.

This paper presents these features of the E2E T&E. Specifically, Section 2 covers scenario specification; Section 3 presents static analyses such as completeness and consistency checking; Section 4 covers pattern analysis where the verification patterns are explained; Section 5 presents automated test execution by distributed agents; Section 6 shows how simulation of concurrent scenarios can be executed; Section 7 presents change management such as regression testing and ripple effect analysis; Section 8 gives an overview of the E2E tool; Section 9 shows an experimentation of a military system. Section 10 concludes this paper.

considered a sequence of events, with their pre-conditions, actions, and post-conditions.

By matching the same conditions and events from different scenarios, a tester can merge all the scenarios together to form a tree, called State/Event Tree (SET) [20][25]. The root of a SET is the initial system state. Each path from root node to a leaf node represents a thread, consisting of a sequence of time ordered (explicit or implicit) events, actions, with associated lists of pre-/post-conditions, and incoming data and outgoing data (expected output). Linking a sequence of atomic scenarios obtains complex scenarios, and they can be used to explore the interaction between various scenarios and for stress testing.

Once scenarios are specified using the state model, it is possible to perform various analyses as presented in Section 3.

2.3 Constraint Identification

Constraints are those special requirements where trade-offs cannot be made. Typical constraints in system requirements are:

- ?? Sequence: triggered/trigger-by
- ?? Logical constraint: mutually exclusive
- ?? Functional constraints
- ?? Timing constraints
- ?? Concurrency constraints
- ?? Synchronization/Asynchronous operations
- ?? Physical constraints

Identifying constraints helps perform completeness and consistency check. It is usually unnecessary or impossible to exhaustively test all the combination of all the conditions, events, and actions. Thus, it is important to analyze the relationships among them so that a tester can build certain test criteria and select the most typical combinations to test effectively within available time and budget. For example, it may be necessary to test the combination of related conditions only because those not related may not affect each other. In this way, the number of possible combinations of conditions can be reduced in testing.

3 Static Analyses

Completeness Analysis

The E2E specifications need to be complete with respect to the external requirements. A tester can determine this by checking if any system scenario is missing in the specification.

However, if some scenarios are missing in the external requirements, a tester will not be able to identify all the missing parts. One way to address this problem is to simulate the system requirements, and this is discussed in Section 6. Another way to do this is to check all the combinations of conditions in the scenario specification, and this can be done in both the scenario model and the SET model. Specifically, the following algorithm enumerates all the combinations of conditions, and a tester can easily identify those missing scenarios from comparing the output of the algorithm with the SET.

```
For each state
  collect  $(c_{pre_i})$  , a set of preconditions
  associate with state  $s_i$ ;
  enumerate all the combinations of  $(c_{pre_i})$  with
  each condition;
  scan transition set  $t_i$  that already exists;
  report the uncovered transition  $t_i'$  , such that
   $t_i \neq t_i'$  are all the combination of transitions
  associated with state  $s_i$ ;
End
```

Consistency Analysis

It is important that scenarios specified are consistent with each other. A tester can perform this by checking relationships and constraints between events and conditions in scenarios. A typical constraint is mutual exclusion. For example, one scenario cannot exist while another scenario is active. For example, a scenario of active progress cannot exist at the same with another scenario of rollback recovery because these two scenarios are mutually exclusive with each other. In scenarios are specified using the SET model, there should no path from the root to any leaf node with two mutually exclusive states.

Dependency Analysis

Dependency analysis is useful for change management including regression testing and ripple effect analysis. Both the scenario model and the SET model facilitate dependency analysis because both models contain input and output, conditions, actions, and preconditions and post-conditions. They can be used to identify the following kinds of dependency:

- ?? Input/output dependency
- ?? Functional dependency

- ?? Execution/control dependency
- ?? Precondition/post-condition dependency

For example, a scenario is input dependent on other scenario if it uses an output from another scenario as its input. Two scenarios can have input/input dependency if both of them use the same data as their input. This needs to be identified because if the data is changed, both scenarios must be examined for possible ripples. The E2E tool automates dependency analysis for both the scenario model and the SET model.

Usage Analysis

Usage information can be useful in prioritizing test activities, e.g., highly used scenarios should be tested more to ensure the system reliability. This follows the Pareto's principle: 20% of scenarios will be executed 80% of time. Thus for a highly reliable SoS, it is important to thoroughly test those 20% of scenarios.

If the system is specified using the SET model, one can consider the usage from two points of view: 1) static view by examining the internal SET structure; and 2) dynamic view by examining the usage externally. For example, each node in a SET belongs to some scenarios, and one can determine the static usage of a state k as follows:

$$SU_k = \sum_j \frac{p_i}{n_j}$$

where p_i is equal to 1 if the node k is used in path i , otherwise zero; n_j is the total number of nodes in path j . Let DU_k be the dynamic usage of node k . The criticality of node k can be defined as follows:

$$C_k = SU_k + DU_k + \sum_j \frac{p_i}{n_j} + DU_k$$

Because a test scenario is a path from the root to a leaf node, one can thus determine the criticality of summing all C_k for all the node k in the path. In this way, the tester can spend more effort on critical scenarios.

Risk Analysis

Risk analysis can guide the tester in prioritizing the testing effort. In the E2E guidebook [4], one may assign risks to scenarios as well as the conditions in the scenarios. The risk can be estimated by the probability of occurrence multiplied by the consequence. Furthermore, risks associated with a scenario can be dynamically changed during the development,

i.e., whenever there is a change in the system, the risk associated with the changed scenarios as well as those that have dependency relationships with the changed scenarios are automatically raised.

Concurrency analysis

Many scenarios may be active at the same time, and it is necessary to verify that concurrent execution of these scenarios will not cause the system to deviate from its intended behavior. This can be done by checking data sharing among concurrent scenarios. Specifically, concurrent scenarios with readers only are safe, but a single writer may make concurrent scenarios at risk of running into synchronization problems.

4 Pattern Analysis

Even though a system may have hundreds of thousand scenarios in the state/event, it may have only few scenario patterns. For example, a commercial defibrillator has hundreds of thousand scenarios, however, most (95%) of these scenarios can be classified into just *eight* scenario patterns [29]:

- ?? Basic Pattern (40%),
- ?? Key-Event Driven Pattern (15%),
- ?? Timed Key-Event Pattern (5%),
- ?? Key-Event Time-Sliced Pattern (7%),
- ?? Command-Response Pattern (8%),
- ?? Look-back Pattern (6%),
- ?? Mode-Switch Pattern (8%), and
- ?? Interleaving Pattern (6%).

This provides an excellent opportunity for rapid verification because scenarios that belong to the same pattern can be verified using the same mechanism except perhaps with different parameters such as timing and state information. And this can save significant time and effort for implementation of verification software.

For example, suppose a system has 7,000 scenarios, and if 15% of these scenarios belong to a specific pattern, these 1050 (7,000 x 0.15) scenarios can be tested using the same verification software except with individualized parameters. Thus, the productivity gain can be significant and industrial application of this approach showed that 25% to 90% effort reduction is possible [29].

Another significant advantage of this approach is the size reduction by using this approach. The industrial experiment indicates that average code for test scripts have reduced from 1380 LOC per scenario to 143 LOC per scenario by using this approach, or about 89.6% size reduction! If we assume an expert test engineer can develop 1,000 LOC of test script each week, the effort reduction by using this approach is significant.

5 Automated Test Execution by Distributed Agents

Once scenario specifications are available, the E2E tool generates thin-threads and test scripts, and executes the test automatically.

5.1 Thin Thread Generation

Scenarios describe the system behaviors responding to stimuli under different conditions. Test execution however needs one and only one execution path of a scenario. A thin thread describes the single execution path of the system, and each path in a scenario corresponds to a thin thread. Thus thin threads can be obtained by parsing scenarios.

Each thin thread contains *pre-conditions*, *events*, *actions*, and *post-conditions*, which are mapped to *setup*, *execution*, and *verification* parts in an Execution/Verification Template (EVT) by following the steps:

- ?? Map pre-conditions and events in a thin-thread to the *setup()* of the EVT;
- ?? Map the sequence of actions in a thin-thread to the *execution()* of the EVT; and
- ?? Map the post-condition in a thin-thread to the *verify()* of the EVT.

The EVT is designed and implemented using the Template Method pattern [5] that allows the tester to customize thin threads. The E2E tool supports automatically generating thin threads from scenario, and exports them as XML files (Figure 2).

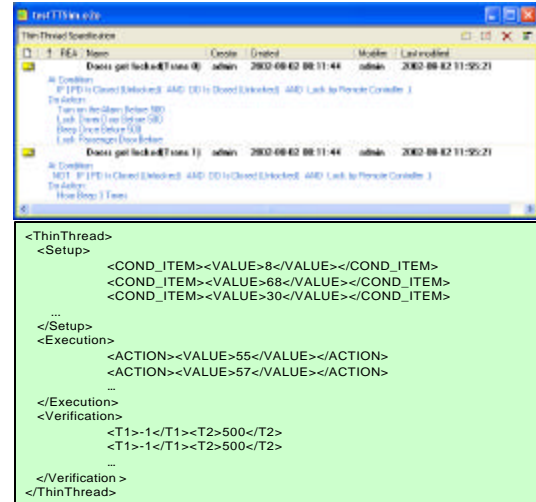


Figure 2. Thin Threads Generated from Scenarios

5.2 Test Case/Script Generation

Once thin threads are available, the E2E tool generates test cases and test scripts for test execution. First, the tester need to supply data related to the input of thin threads, once this is done, the tool generates test inputs based on different testing techniques:

- ?? Random testing: the tool will take random sample data from the data set.
- ?? Partition testing: the tool will select sample data from each partition of the data in the data set.
- ?? Boundary value testing: the tool will take the boundary values of the partitions of data in the data set.

The tool also generate test scripts for test execution, if the SoS is a distributed system, the tool also generate distributed test commands so that remote systems can be exercised.

5.3 Distributed Test Execution

The E2E tool provides distributed test execution by having an architecture with a test master and associated test agents. A test master performs the following functions:

- ?? Generates test scripts and test cases from scenarios, and initiates the test by sending commands to agents either locally or remotely using TCP/IP or SOAP protocols;
- ?? Manages the dependency and interaction information between systems in the SoS;
- ?? Collects the test result data from agents, and verifies them;
- ?? Verifies interaction patterns identified.

While an agent provides the following functions:

- ?? Upon receiving commands from the test mater, it executes the test by first setting the SUT to the state specified by the precondition in the test script, generating events or actions according to script, and verifying results returned by the SUT;
- ?? It also traces the interaction of the SUT with other systems in the SoS. This is a useful function because a distributed system often can exhibit behaviors that are difficult to trace, and having each agent tracking the communication between its SUT and the rest of the SoS helping in identifying the cause of system failures; and
- ?? Sends any related information to the master.

The test master and test agent collaborate in the following manner:

- ?? **Request System Function:** The master requests local and/or remote agents to execute functions according to the scenario specification. The agent receives the request, sets up the SUT for test, and then issues test commands to execute the requested function, and then reports the results to the master.
- ?? **Request and Response of Message:** The master requests agents to collect information of exchanged messages among sub-systems including requests and their responses, and each agent sends information back to master asynchronously.
- ?? **Request Change Configuration:** The master requests agents to change configuration of the system to start a new round of testing. Each agent re-sets it's own state and reports to the master.
- ?? **Notice State Change:** The test master adds listeners (this is Observer design pattern[5]) to the agents. When the system state is changed either triggered by internal or by external events, the agent notices this change to the master asynchronously.
- ?? **Verify Results:** The master receives the information such as exchange messages and state changes from agents asynchronously, and collects the data related to specific test scenarios. Then the master verifies the test results with the expected output.

The architecture can be illustrated using the following figure.

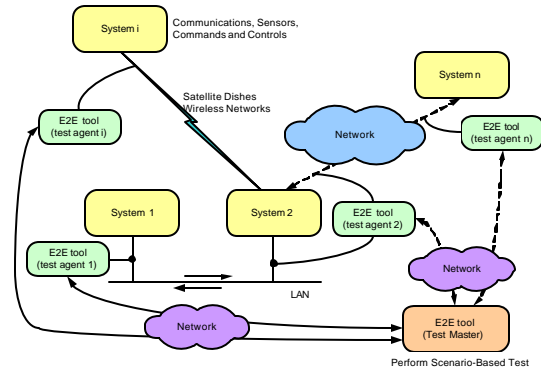


Figure 3. Testing a SoS

An important characteristic of testing a SoS than testing an individual system is that sometimes it is often easier to develop system scenarios for individual systems than to develop system scenarios for a SoS. One way to derive system scenarios for a SoS is as follows:

- ?? Derive scenario specification for each individual system in the SoS;
- ?? Specify the interaction between each pair of systems in the SoS;
- ?? Derive scenario specification of the SoS by combining scenarios for individual systems with interaction specification.

In this way, scenarios for a SoS can developed in an incremental manner with scenarios for individual systems developed first. This approach also support changes in a SoS. When a new system is added to a SoS, it is possible to reuse the existing scenarios for other systems and interaction specifications to re-develop the overall scenarios for the modified SoS.

6 Simulation

A SoS is usually complex and involves lots of systems including hardware and software. Thus, even an exercise of the entire system can be costly. Thus, if possible, a T&E of a SoS should be carried out first in a completely simulated mode, and then in a simulated mode but with partial participation from individual systems in the SoS before an exercise of the complete system.

The E2E tool allows a tester to simulate both the system and its environment. The reason that the tool allows two separate simulations is that often

it is not possible to control external events while internal mechanisms can be fully designed and managed. For example, an aircraft tracking system is designed to track any incoming aircrafts in the monitored airspace, however, it certainly does not control the number of aircrafts or birds that may come at any given time. Thus, separating simulation of a SoS from simulation of its environment allows a thorough evaluation of the SoS under various external situations and loads.

Simulation of the environment simulate both the user's behaviors as well as other issues such as arrival of new objects, communication network failures, and weather condition. Event can be discrete (e.g., the number of users'/signal arrival within a given period) or continuous (e.g., timer events).

The E2E simulation uses the information stored in the scenario specification for execution. Specifically the conditions and actions in the specification. Conditions specify the state that a given scenario can be activated, and actions specify the input used, output produced including the state change. The tool automatically analyze scenarios specified and produced a system decomposition like the one in Figure 3.

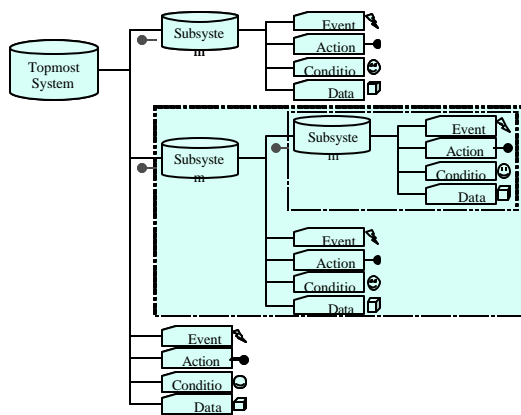


Figure 3. Simulation Architecture

Simulation is performed by tracing three types of relationships:

- a) *Reaction* relations: This is between an environmental event with a system action, and this is used to trigger an action after an external event happed;
- b) *Impact* relations: This is between actions and conditions, and this updates the values of

variables after an action is performed. This often changes the system state because variables are used to define the state of the system;

- c) *Value* relations: This is between conditions and data, and this updates a condition whenever the variables associated with the condition have new values.

A tester can also specify simulation to be performed with respect to certain sub-systems in the SoS, or with respect to certain paths, objects, events, or data so that a focused simulation rather than a complete simulation of the SoS. This is done by disabling those unnecessary conditions and actions during simulation.

A unique feature of this approach is that a scenario will be activated whenever its conditions are met, and thus multiple scenarios may be active at the same time during simulation. This is useful because in a SoS, multiple scenarios can indeed be active at the same time, and system specifications often focus on single threads, and thus it is difficult to see the impact of the interaction between concurrent threads or scenarios. The E2E simulation allows the interaction among concurrent threads to be examined carefully before system implementation.

The E2E simulation allows a tester to predict system behaviors, determine system performance before its implementation, and evaluate a new system without the exercise the entire SoS. An important evaluation is timing analysis. Many DoD systems have strict timing requirements, and timing requirements can be added to system scenarios, and simulated using various time distributions for each action. As the system is simulated, timing constraints can be checked at runtime to see if the stated requirements are met.

7 Change Management

Requirement changes due to dynamic market needs; technique changes due to global competition; and process changes due to the characteristics of products. E2E testing and evaluation follow Agile Software Development process, allow rapidly adapting to system evolution. Whenever changes occur, the tester just needs to re-specify the system, then E2E tool will automatically perform dependency analysis, ripple effect analysis, generate new test cases/scripts, and execute the testing. Furthermore, it runs regression testing using built-in test strategies, such as risk-based, usage-

based, and time-wised; also it allows the tester to specify a new test strategy.

Dependence analysis provides the foundation for regression testing and ripple effect analysis. Ripple Effect Analysis (REA) is used to analyze and eliminate side effects due to changes and to ensure consistency and integrity after changes are made to software [19][4]. It is an iterative process of change request, software modification, impacts identification and validation. It ends when there are no more ripples. For example, when there is a change to a thin-thread, the tool finds out all the thin-threads that are dependent on that changed thin-thread (Figure 4). The changed thin-thread and affected thin-threads are the candidates for further testing.

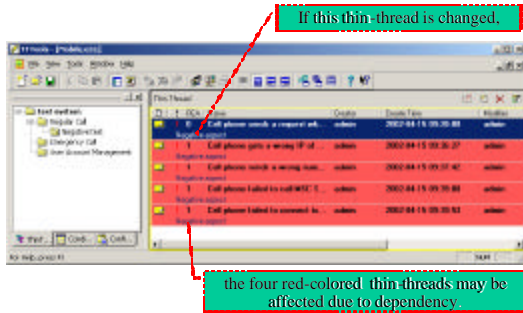


Figure 4. Dependency & Ripple Effect Analysis

8 Tool Support

The E2E tool support many of activities described in this paper, specifically scenario specification, test design, thin thread generation, simulation, dependency analysis, distributed test execution, remote testing, regression testing, ripple effect analysis, verification pattern, test result analysis, and report generation. Figure 5 shows the main components and the three-tier architecture of the E2E tool.

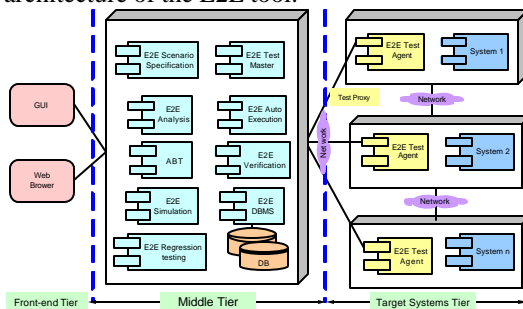


Figure 5. E2E Tool Components and Architecture

?? The front tier is the presentation layer. It provides users with various graphical views of the test data and analysis results that are obtained from the middle tier either through Web browsers or standalone GUI applications. This allows everyone involved in the software development – analyst, designer, tester, and manager – to get useful information efficiently. The interface components access the services provided by the middle tier components through standard network protocols such as HTTP, SOAP, IIOP, RMI, or COM/DCOM.

?? Middle tier: two internal tiers are at middle tier: front-middle and back middle tier, which perform the core functions of the E2E testing and evaluation. Front-middle tier organizes scenario specification in OO fashion: such as creating testing scenario objects, test case objects, input data objects, method signature objects, and complex scenario objects. Analyzer in front-middle tier performs a variety of analysis, such as completeness and consistency check, dependency analysis. Test master executes testing, runtime verifies results, performs ABT, regression testing and simulation. The test master manages testing distributed systems by coordinating the distributed test agents via network using TCP/IP, SOAP protocols. Test master and test agent are considered as E2E middleware part. The front-middle tier accesses test data either through direct database connection using ODBC, JDBC, or through the APIs provided by data management components. The back-middle tier facilitates access to database for storing all test specification and test results.

?? Target systems tier: Test agents act as proxies of test master: runtime binding remote objects on the associated systems under test (SUT), dynamically invoking testing methods. Test agents carry out testing execution by collaborating with each other, runtime verifying the results, and reporting the results to the test master.

9 Experimentation

The tool has been implemented using Microsoft Visual C++, and is currently being experimented on a military SoS. Currently, three high-level threads have been specified with 17 scenario groups, 42 scenarios, 42 events, 103 conditions,

103 actions, and 85 thin threads. Figure 6 shows the dependency analysis and ripple effect analysis on these scenarios.

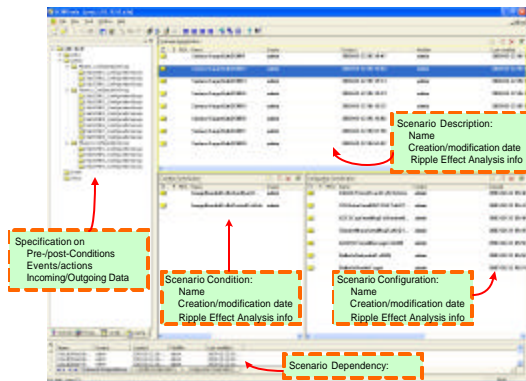


Figure 6. Scenarios of the Military SoS

10 Conclusion

The E2E T&E process was designed to be a flexible process that can adapt to various changing environments. In other words, adaptability is the primary concern. Some important attributes of *adaptability* include *speed*, *scalability*, *reusability*, *partitioning*, and *integration*. In general, a process is adaptive if it supports high-speed development, is scalable from small applications to large applications, has many reusable tools that can produce reusable components, and has an integrated process.

Concerning *speed*, the E2E T&E process is fast because the E2E tool generates test scripts automatically from system scenarios, perform distributed test execution by sending test commands to remote agents automatically, generates new test scripts after system modifications for regression testing and ripple effect analysis, and obtains system performance by simulation once the system scenarios are available early in the life cycle.

Concerning *scalability*, the E2E T&E process is scalable because it can apply to large applications as well as to small applications. Scenarios used in the E2E process are hierarchical and thus they can apply to the hierarchical structure of a SoS. For example, the scenarios specified in Section 9 are high-level scenarios with actions from multiple systems. But the same techniques can be used to specify low-level system scenarios for individual

systems, and high-level scenarios can be constructed using low-level scenarios as their components. The E2E simulation is also scalable because it depends on the scenario specification only, thus it is possible to perform high-level simulation such as at the military SoS level, or low-level simulation at the task level.

Concerning *reusability*, the E2E T&E process has many reusable tools including scenario specification tool, test case management tool, scenario simulation tool, and distributed test execution tool. One of the key benefits of the E2E tool is that scenarios specified are highly reusable and can be easily changed. System scenarios keep on changing as new requirements are known and new technology is introduced during system development, changing system scenarios with the E2E tool is much easier than re-developing scenarios by hand. In most cases, new scenarios are developed by changing the existing scenarios, and changing scenarios using the E2E tool with dependency analysis is easier than starting from the scratch without any tool support. Furthermore, once new scenarios are specified, they can be automatically analyzed by various analysis techniques such as timing analysis, and new test scripts can be rapidly generated and executed.

Concerning *partitioning*, the E2E's scenarios are partitioned in several ways: a) By functional group, this is addressed by scenario trees [1][18] where functionally related scenarios are grouped together to form a tree; b) By sub-systems, as discussed in Section 5, the overall system scenarios can be constructed by developing scenarios for individual sub-systems first, and then combining with interaction scenarios between sub-systems. These two kinds of partitioning facilitate adaptability. If a new sub-system is introduced to replace an existing sub-system, if the overall mission does not change, the high-level system scenarios need not change, but the low-level scenarios with respect to the retiring system need to be replaced by the scenarios for the new sub-system. If the high-level system scenarios are changed due to change in warfighting tactics, but the participating sub-systems remain the same, it is only necessary to change the high-level scenarios while reusing the low-level scenarios for the sub-systems.

Concerning *integration*, the E2E T&E tightly integrates system analysis and modeling with integration testing because the same techniques,

i.e., scenarios, can be used for both system analysis as well as integration testing. The importance of testing has recently being emphasized by agile development processes such as Extreme Programming. While testing is one of their main techniques, agile processes do not have such tight integration between system analysis and testing as the DoD E2E T&E. The tight integration changes the way systems are developed: Instead of performing *requirement-driven testing* only, the E2E process calls for *test-based requirement analysis*. In other words, the requirements should be developed in a way that can be used for rapid integration testing (by automated test script generation, verification patterns, and distributed test execution) and evaluation (by various analyses and simulation). In fact, the E2E T&E supports a test-based development process from requirements to operation and maintenance, and such process is compatible with agile development processes or incremental development.

11 References

- [1] X. Bai, W. T. Tsai, R. Paul, K. Feng, and L. Yu, "Scenario-Based Modeling and Its Applications to Object-Oriented Analysis, Design, and Testing", Proc. of IEEE WORDS 2002, pp. 140-151.
- [2] A. Cockburn, *Agile Software Development*, Addison Wesley, Reading, MA, 2001.
- [3] DoD OASD C3I Investment and Acquisition, "Year 2000 Management Plan", 1999.
- [4] DoD OASD C3I Investment and Acquisition, "End-to-End Integration Testing Guidebook", 2002.
- [5] DoD: "Testing for System-of-Systems Interoperability", Army Test and Evaluation Days Conference, July, 2001 (<http://www.acq.osd.mil/sts/te/speeches.html>).
- [6] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison Wesley, Reading, MA, 1994.
- [7] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison Wesley, Reading, MA, 1994.
- [8] W. E. Howden, "Good Enough versus High Assurance Software Testing and Analysis Methods," in Proc. of IEEE High-Assurance Systems Engineering (HASE), 1998, pp. 166-175.
- [9] S. Kirani and W. T. Tsai, "Method Sequence Specification and Verification of Classes", Journal of Object-Oriented Programming, 1994.
- [10] D. Kulak and E. Guiney, *Use Cases: Requirements in Context*, Addison Wesley, Boston, MA, 2000.
- [11] D. C. Kung, P. Hsia, and J. Gao, *Testing Object-Oriented Software*, IEEE Computer Society Press, Los Alamitos, CA, 1999.
- [12] V. Matena and B. Stearns, *Applying Enterprise JavaBeansTM: Component-Base Development for the J2EETM Platform*, Addison-Wesley, Boston, MA, 2000.
- [13] R. Paul and W. T. Tsai, "Assurance-Based Testing – A New Quality Assurance Technique", Proc. of Quality Week, Europe, 2000.
- [14] R. Paul, "End-to-End Integration Testing: Evaluating Software Quality in a Complex System", Proc. of Assurance System Conference, Tokyo, Japan, 2001, pp. 1-12.
- [15] R.S. Pressman, *Software Engineering: A Practitioner's Approach*, 5th edition, McGraw-Hill, New York, New York, 2000.
- [16] W. T. Tsai, R. Paul, W. Shao, S. Rayadurgam, and J. Li, "Assurance-Based Y2K Testing", Proc. of IEEE HASE, 1999, pp. 27-34.
- [17] W. T. Tsai, Y. Tu, W. Shao and E. Ebner, "Testing Extensible Design Patterns in Object-Oriented Frameworks through Hierarchical Scenario Templates", Proc. IEEE COMPSAC, 1999, pp. 166-171.
- [18] W.T. Tsai, X. Bai, R. Paul, W. Shao, V. Agarwal, T. Sheng, and B. Li, "End-to-End Integration Testing Design", Proc. of IEEE COMPSAC, 2001, pp. 166-171.
- [19] W.T. Tsai, X. Bai, R. Paul, and L. Yu, "Scenario-Based Functional Regression Testing", IEEE Proc. of COMPSAC, 2001, pp. 496-501.
- [20] W. T. Tsai, L. Yu, R. Paul, T. Liu, and A. Saimi, "Developing Adaptive Test Frameworks for Testing State-Based Embedded Systems", Proc. of IDPT, 2002.
- [21] W. T. Tsai, Y. Na, R. Paul, and F. Lu, "Adaptive Scenario-Based Object-Oriented Test Frameworks for Testing Embedded Systems", Proc. of IEEE

- COMPSAC, 2002, pp. 321-326.
- [22] W. T. Tsai, R. Paul, Y. Wang, C. Fan, and D. Wang, "Extending WSDL to Facilitate Web Services Testing", Proc. of IEEE HASE, 2002, pp. 171-172.
- [23] W. T. Tsai, R. Paul, W. Song, and Z. Cao, "Coyote: An XML-Based Framework to Test Web Services", Proc. of IEEE HASE, 2002, pp. 173-174.
- [24] W. T. Tsai, R. Paul, Z. Cao, B. Xiao, and L. Yu, "Adaptive Scenario-Based Testing Using UML", Proc. of OMG 3rd Workshop on UML for Enterprise Applications: Model Driven Solutions for the Enterprise, 2002.
- [25] W. T. Tsai, L. Yu, X. Liu, A. Saimi, and Y. Xiao, "Scenario-based Test Generation Tool for Embedded Systems", to appear in Proc. of IEEE IPCCC 2003.
- [26] W. T. Tsai, L. Yu, A. Saimi, and R. Paul, "Scenario-Based Object-Oriented Test Frameworks for Testing Distributed Systems", to appear in Proc. of IEEE Future Trends in Distributed Computing Systems, 2003.
- [27] A. W. Ulrich, P. Zimmerer, and G. Chrobok-Diening, "Test Architectures for Testing Distributed Systems", Proc. of Quality Week, 1999.
- [28] WS-I: <http://www.ws-i.org/>, "WS-I Usage Scenarios", "Supply Chain Management Use Case Model", "Sample Application Supply Chain Management Architecture".
- [29] F. Zhu, "A Requirement Verification Framework for Real-time Embedded Systems", PhD dissertation, Department of Computer Science and Engineering, University of Minnesota, Minneapolis, MN 55455, 2002.