

# Semantic Interoperability and its Verification & Validation in C2 Systems

W. T. Tsai, R. Paul\*, Hai Huang, Bingnan Xiao, Yinong Chen

Department of Computer Science and Engineering  
Arizona State University, Tempe, AZ 85287-8809  
(wtsai@asu.edu)

\*OSD NII, Department of Defense, Washington, DC

## Abstract

Interoperability is a critical issue for DoD C2 systems. Current research has mostly focused on the data interoperability and ontology of context. While these studies are important and useful, they have not addressed other important issues on semantic interoperability and its verification & validation. This paper proposes a new technique called *use scenario*, which specifies the workflow of passing parameters among different services or the semantics of interoperation. For a C2 system, once the use scenario is specified, a family of automated analysis, verification, and validation techniques is available for testing and evaluating the system and its interoperability.

**Keywords:** Semantic Interoperability, Use Scenario, Modeling, Simulation

## 1. Introduction

Interoperability is defined as the ability of two or more systems or components to exchange data and to use the data in each other's operations [3], which is a critical issue for DoD C2 systems. Particularly, the recent emphasis on Network-Centric Warfare (NCW) placed interoperability as one of the priority issues. In spite of extensive studies on this topic, the focus has been mostly on

- Data interoperability including data schema, meta-data, and database integration;
- XML, such as using XML to represent data schema and as a means for data interoperability;
- Ontology as a means for context representation and matching in a given taxonomy; and
- Service-Oriented Architecture (SOA) and other related technologies such as standard protocols (SOAP, HTTP, and etc.), interface definitions (WSDL, OWL-S, and etc.), registration and publication of interfaces (UDDI), wrapper, automated composition.

While these studies are important and useful, they have not addressed other important issues on interoperability, namely

- Semantic interoperability: How can C2 systems or services actively collaborate to achieve a mission (not just exchange data)?
- Interoperability verification and validation particularly related to semantic interoperability:

How can we know the semantic interoperability meet the specification?

In this paper, we focus on the semantic interoperability and its verification & validation.

Generally speaking, semantics defines the meaning of the constructs of a language, or what happens during the execution of a program or program part [4][5]. Current research on ontology belongs to semantics because it deals with the context match of terminologies such as synonyms and acronyms based on taxonomy of a language. However, ontology is only a part of the semantics. As shown in Figure 1, we propose to study the semantic interoperation which deals with how the data exchanged among the services can or cannot be used. There exist different kinds of the semantic interoperability, for example, workflow, data range, and timing. Semantic interoperability can be represented as constraints and implemented by policy based computing [16]. In this paper, we focus on the work flow part of the semantic interoperability by introducing the concept of use scenario. A use scenario specifies how a service or system is used by other services or systems.

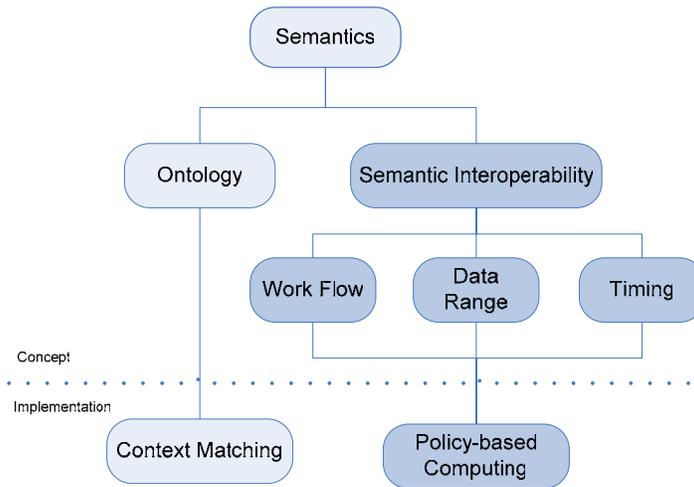


Figure 1: Semantics and semantic Interoperability

Future DoD C2 systems not only need to exchange data and interoperate with their fellow C2 systems with respect to data, but also need to collaborate with other C2 systems in terms of tasks and missions. While the current interoperability technologies such as standard interface, data representation, and ontology are critical for semantic interoperability, they are not sufficient because the current interface technologies provide method signatures only for a single service. These method signatures do not provide sufficient information for another new system or user to properly use the service, e.g., what the proper calling sequence is among methods of this service and what the dependency is among methods of a service or another service. For example, the user's manual of a product, say a cellular phone, describes both the functional specification and the operational procedure. The former lists what can be done, e.g., make phone call, receive phone call, send and receive text messages, store 120 contact items, and play MP3 music. The latter, on the other hand, instructs how to use each of the functions in a stepwise fashion. Current interoperability definition of systems mainly specifies the functions and the syntax of calling the

functions. The use scenario studied in this paper defines how a particular function can be used in a stepwise fashion.

This paper proposes a new technique for semantic interoperability called *use scenario*, which defines the workflow of the exchanged data. The use scenario is an extension to UML's *use case* [6] as well as David Parnas' concept of *use* [1]. An innovative idea is that once the use scenario for a service is specified, a family of analyses and verification techniques can be developed for the system semantic interoperability, including:

- Automated sample interoperability templates;
- Automated interoperability test case generation;
- System simulation; and
- Completeness and consistency analyses.

These automated techniques and tools are possible because the use scenario can be specified using a semi-formal specification language supported by automated tools. A *use scenario* is also different from a *system scenario*. A system scenario describes the behavior of a system when the system is activated with a specific input, while a use scenario describes a possible sequence of actions to activate a service provided by the system. The use scenario, once specified, can greatly reduce the time needed for C2 systems to collaborate by properly calling each other in the specified order.

The automated system interoperability framework can significantly reduce the manpower spent for reasoning, verifying, and validating a large C2 system with respect to semantic interoperability. This paper presents the detailed techniques and illustrates them using a sample C2 system.

## **2. ACDATE Modeling Overview**

The ACDATE language, named based on its model entities Actor, Condition, Data, Action, Timing, and Event, has been developed as a generic modeling and specification language in the domain of system engineering and software engineering [16]. It facilitates the specification, analysis, simulation, and execution of the requirement and therefore the system.

A Scenario is a semi-formal description of system functionality, written in pseudo code text. It is a sequence of events expected during operation of system products which includes the environment conditions and usage rates as well as expected stimuli (inputs) and response (outputs).

ACDATE entities are the building blocks for Scenario specification. After one's system requirements have been decomposed into ACDATE entities, one can then specify Scenarios. This ACDATE/Scenario model allows for system modeling and provides the capability to perform various analyses of requirements and their verification and validation [7][10][11][12][13][14].

### **3. Scenario-based System Semantic Interoperability**

Software applications are becoming distributed. There are several kinds of distribution architectures like client/server architecture, multi-agent architecture, and SOA. No matter what distributed architecture is adopted by a complex software application, there must be more than one components collaborating with each other to achieve a given mission.

Following the concept of SOA, which is also the trend of DoD Network Centric Warfare, each constituent system in a composed complex system, or system of systems, is a self-contained autonomous system which provides services and is loosely coupled with other systems. To achieve interoperability, each system needs to be able to exchange data and services in a consistent and effective way. Also, they may be deployed on to different hardware or software platforms. Thus, it is important for the systems to provide universal access capacities independent of platforms.

Most current research projects focus on the data aspect of interoperability. E.g., XML and meta-data modeling. However, to fully achieve interoperability, handling the data exchange only is not sufficient because:

- Data exchange is a small part of interoperability only;
- Systems need to interact with each other at run-time;
- One system may use the services provided by others; and
- Systems may need to work with legacy systems.

To make heterogeneous systems working with each other, we need to have a framework which provides support for platform independent system service specification, system wrapping for legacy systems, and system composition and re-composition.

#### **3.1 System Service Specification**

In a system of systems, each constituent system (or subsystem) in the composite system can be considered to be an agent which can send and receive messages. Each agent provides an abstract set of functionalities that are called services.

For different systems to be interoperable with each other, system's service specification needs to conform to a common standard. Services designed using the same service specification language can have a higher level of interoperability.

System service specification is a system profile which provides information of what the system is. The profile includes following information:

- Interface Specification

Interface specification is critical for system interoperability. It describes the calling parameters and return values of the system. The ACDATE model in E2E automation provides the capability for interface specification. To be more specific, Events in

ACDATE are the key elements for system interaction. The Events-Actions handling relation and Data define the interface information for a system, which will be elaborated in the following sections..

- System Scenario & Use Scenario

The scenarios describe how the system works and how to work with this system.

### 3.2 System Scenario vs. Use Scenario

System scenario is a semi-formal specification of a system's functional behaviors. It specifies what the system's control logic is for a service to handle a specific input. System scenarios are derived from system functional requirements directly. A system scenario focuses on how the system will behave given it is activated with an input.

The system scenarios are essential to a given system since they specify the system behaviors. Once a system is specified with system scenarios, it is able to provide services to handle different inputs. It is sufficient for a system to have system scenarios only if it works alone without being connected to other systems. If we want to connect the system to other systems to collaborate, we need to provide information on how this system works with other systems.

Use scenario is a semi-formal specification of how the system can be used by other systems. It does not specify anything about the system's internal control logic but focuses on how the other systems can use the services provided by this system. To be more specific, a use scenario indicates the temporal logic and / or timing constraints on a sequence of function calls. It provides a usage pattern on how to invoke the interfaces for the system.

### 3.3 Use Scenario Specification and Analysis:

This section defines the syntax and semantics of *use scenario*. We define the following new structural constructs:

- *choice*{ *option*[] *option*[] ... *option*[] }: *choice* means that the interoperation can select any single sub-scenario (listed as *options*) to continue the control flow.
- {} *precond*: *precond* indicates the preconditions before a particular action.
- *postcond* {}: *postcond* indicate the postconditions after a particular action.
- *criticalreg* {}: *criticalreg* indicate a critical region such that no other actions can take place to interrupt the execution of actions within the critical region. Any action sequence outside a critical region can be intervened by any sub-scenario.
- $\diamond$ : Any entities enclosed by  $\diamond$  are *parameter entities*. It can be instantiated by any entity that satisfies the requirements described in the use scenario, such as, must have some actions, data, or satisfy some conditions.

With sub-scenarios, the use scenario can describe the interoperation of hierarchical systems in

different levels.

With the use scenario, we can perform

- Automated interoperation scenarios generation

If more than one systems specified with use scenarios are to be put together to compose a complex system, the interoperation scenarios can be generated by intervene the use scenario for individual systems. Please refer to example 1 for detailed introduction.

- Interoperation scenario correctness checking

There will be quite a lot of interoperation scenarios can be generated or specified by intervene the individual use scenarios for different systems. But not all intervenes for the systems are correct interoperation sequence. By the constraints checking such as precondition checking, postcondition checking, and critical region checking, we can identify the interoperation scenarios that do not satisfy the constraints. Please refer to example 2 for detailed introduction.

- Interoperability cross checking

The constraints may be specified in different use scenarios. If one wants to put the systems together, the interoperability cross-checking needs to be done to identify potential inconsistencies. Please refer to example 3 for detailed introduction.

With the support of the analytic techniques mentioned above, users can verify the correctness of use scenario. This can further enhance the semantic interoperability of systems. In the rest of the paper, we will show several examples to depict how to perform the reasoning and the analysis.

### 3.4 Use Scenario Example

The first example applies use scenario in a simple C2 system and shows how to automatically generate interoperation scenarios. The use scenario of a control system that is in charge of battle tank is listed below. The control system is modeled as an actor Tank, which has 5 functions, Start, Move, LocateTarget, Fire, and Stop.

#### Example 1:

```
do ACTION:Tank.Start
choice {
  option [
    do ACTION:Tank.Move
  ]
  option [
    do ACTION:Tank.LocateTarget
  ]
  option [
    do ACTION:Tank.Fire
```

```

    ]
}
do ACTION: Tank.Stop

```

The following use scenario defines the skeleton of an interoperation with Tank. It is straightforward to extract 3 different interoperation scenarios from the use scenario following the 3 options in the choice:

```

    < Start, Move, Stop >,
    < Start, LocateTarget, Stop >, and
    < Start, Fire, Stop >.

```

There is no precondition and postcondition for any actions. And there is no critical region, which means that any actions can intervene with the sequence. For example, suppose there is another control system BattleControl that has one function OrderToFire. The function OrderToFire can occur at any point in time within any sequence, such as

```

    < Start, BattleControl.OrderToFire, LocateTarget, Stop >, and
    < Start, LocateTarget, BattleControl.OrderToFire, Stop >

```

If we put a critical region to enclose Start and OrderToFire, then no other actions, particularly OrderToFire in this example, can occur between them.

```

criticalreg {
  do ACTION:Tank.Start
  choice {
    option [
      do ACTION:Tank.Move
    ]
    option [
      do ACTION:Tank.LocateTarget
    ]
    option [
      do ACTION:Tank.Fire
    ]
  }
}
do ACTION:Tank.Stop

```

Hence, between the two sequences, only

```

    < Start, LocateTarget, BattleControl.OrderToFire, Stop >

```

is a legitimate one, which describes a typical battle control procedure. Note that there is other ways to achieve the same effect, such as putting LocateTarget as the precondition of

OrderToFire.

### **3.5 Extended Use Scenario**

Use scenarios are useful for efficient system composition. Yet, additional information can be added to use scenario to improve the system's selection and composition effectiveness and scalability. Specifically, the following information can be added:

- Dependency information;
- Categorization; and
- Hierarchical use scenarios.

#### **3.5.1 Dependency information**

In addition to the information specified in use scenarios for how to use the given system, it is useful to add dependency information.

- Dependencies Specification

Dependencies specification describes other systems that need to be included for this system to function. Some system may not provide full function by itself. It needs to use the services provided by other systems. Thus, the other systems that are used together with this system need to be included in the system dependencies list.

- Compatible components list

The compatible components list is a list of other systems that are known to be able to work with the system. With this list, the system composition and re-composition can be done more efficiently.

In a C2 system, the dependency list and compatible component list are very important for composing a C2 system. For example, for an aircraft carrier:

- Dependencies: Destroyer, Frigate, and Submarine.
- Compatible components: Helicopter, Fighter plane, and Scout.

With the information specified above, the composition process will be greatly eased. When putting an aircraft carrier into a C2 system, users will know that the destroyer, frigate and submarine are also needed. Also, the users will know it is compatible to put helicopters, fighter planes, and scouts on the aircraft carrier but not the battle tanks.

### **3.5.2 Categorization**

A system can provide multiple services. Different services provided by the system may have different use scenarios. Furthermore, a system working with different systems may have different use scenarios. For better organization, the use scenarios need to be categorized. A set of use scenarios describing the usage of one specific service provided by this system can be put into the same category. Each system can be assigned with a category tree of use scenarios.

For example, in a C2 system, there is usually a command center which controls the overall battle. Since multiple units, say Fleet 1, fleet 2, and Fleet 3, are all involved in the battle, the command center needs to coordinate the battle and provides services for the Fleets, respectively. To better organize the design, the use scenarios must be categorized accordingly.

### **3.5.3 Hierarchical Use Scenario**

Use scenario can be hierarchical. A higher level use scenario can call lower level use scenarios. Also, a higher level use scenario may specify the use of more than one subsystem. In this case, the high level use scenario specifies the overall process and can be broken down into several low level use scenarios by scenario slicing.

For example, in the service provided for the Army in the command center, it controls the battle on ground. The use scenarios are specified to coordinate infantry and battle tanks. In this case, the use scenario in the command center invokes the Army use scenarios which in turn invokes the use scenarios specified for infantry. The use scenario hierarchy is constructed in this way.

## **3.6 System Composition**

Complex mission often requires collaboration among multiple participating systems. Each participating system (subsystem) in a complex system (system of systems) focuses on handling one aspect of the overall mission. It is important for each subsystem to be specified with system scenarios as well as use scenarios.

### **3.6.1 System Composition Approach**

The bottom-up approach is more efficient to build a new composite system once the system scenarios and use scenarios are known.

With system scenarios, multiple analyses (dependency analysis, C&C analysis, event analysis, simulation, model checking) can be done to evaluate the system. Furthermore, automated system testing and verification with verification patterns can provide us with confidence of the quality assurance of the selected system. Once we have verified and validated the individual subsystems, we can build complex system on top of them.

Firstly, we need to find appropriate systems for system composition. The system discovery and selection can be done by analyzing the system scenarios. Then, we can compose the individual subsystems into the complex system we need by connecting the systems according to the use scenarios.

If a use scenario calls the use scenarios of subsystems, it specifies the interoperation among several different subsystems. In this case, the use scenarios play the role of system composition pattern.

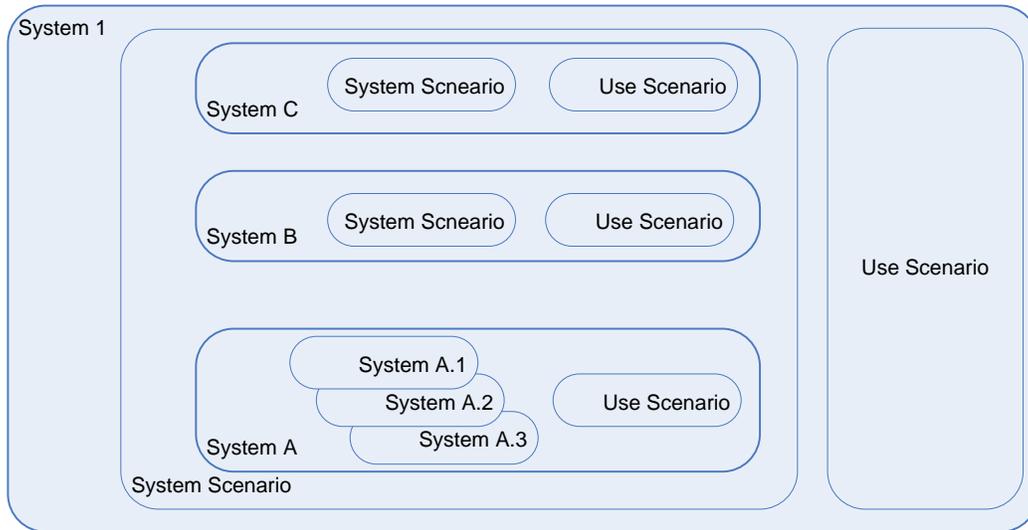


Figure 2: System Specification: System Scenarios and Use Scenarios

Figure 2 shows the composition information of a complex system 1 and three subsystems. In system 1, there are 3 subsystems: system A, system B and system C, each of which is specified with system scenarios and use scenarios. System scenarios for each subsystem provide information on what services this system provides. The system scenarios are useful for system selection. Assume complex system 1 requires 3 major functions and these functions can be provided by the systems A, B, and C, respectively. In this case, these systems are selected to be the subsystems of system 1. Once we have system A, B, and C, we need the information on how to put them together so that they can work with each other to provide the higher level functionalities needed by system 1. If we have interface information only for systems A, B, and C, we may not obtain the functionalities required by system 1 because we do not know how to call the interfaces of each subsystem. On the other hand, if each subsystem is specified with use scenarios, the integration becomes possible.

A use scenario can have templates. In such a template, we may specify what concrete system we need. If a system provides the functionalities specified in the system scenario, it can be used as subsystem. In Figure 2, there are three subsystems, system A.1, A.2, A.3 provides the same functionalities. Each of these subsystems can be a valid candidate for the composition. These subsystems may be ranked with different criteria by the automated testing tools. Appropriate subsystem can be added to the composition system according to different system performance requirements. What system to be chosen will be decided at the system run-time (Dynamic Binding).

Use scenarios for a lower level subsystem can be converted to system scenario for a higher level system. In Figure 2, when the use scenarios of system A, B, and C are combined together, we can

generate system scenarios of system 1. Also, use scenarios for system 1 may be generated automatically or be specified by system designers. With the system scenarios and use scenarios for system 1 are identified, we can build a higher level system using system 1 as subsystem.

### 3.6.2 System Composition Examples

We use a simple C2 system here to illustrate how to compose a complex system of systems using existing components. In this example, we have three constituent systems: Tank control system, Security control system, and Policy control system. These three systems need to collaborate with each other after being deployed in the battle field. Without use scenario, the commanders may not know how to make these three systems working together since they have no idea about when to do the security check or / and when to apply the policy checking during the battle in progress. If the use scenarios have been specified, the commanders can simply follow the predefined sequence, if no better choices are available, for controlling these two systems.

#### Example 2:

This example shows how to perform interoperation scenario correctness checking and compose the correct complex system. Assume that control systems Tank, Security, and Policy are involved.

This is the use scenario for tank control system:

```
do ACTION:Account.Open
choice {
  option [
    {do ACTION:<Security>.VerifyPassword} precondition
    do ACTION:Tank.Move
  ]
  option [
    do ACTION:Tank.LocateTarget
  ]
  option [
    {do ACTION:<Policy>.ApprovePolicy} precondition
    do ACTION:Tank.Fire
  ]
}
do ACTION:Tank.Stop
```

This is the use scenario for security control system:

```
do ACTION:Security.VerifyPassword precondition
{
  do ACTION:<Tank>.Move
  do ACTION:<Policy>.ApprovePolicy
}
```

This is the use scenario for Policy control system:

```
{do ACTION:<Security>.VerifyPassword} precondition
  do ACTION:Policy.ApprovePolicy
```

There is an interoperation sequence for the system involved:

```
do ACTION:Tank.Start
do ACTION:Policy.ApprovePolicy
do ACTION:Tank.Fire
do ACTION:Tank.Stop
```

The interoperation scenario satisfies the use scenarios of Tank and Security. But in Policy, it requires that before ApprovePolicy, it needs to perform VerifyPassword. Hence the interoperation scenario is not correct. The correct one is:

```
do ACTION:Tank.Start
do ACTION:Security.VerifyPassword
do ACTION:Policy.ApprovePolicy
do ACTION:Tank.Fire
do ACTION:Tank.Stop
```

### Example 3:

This example shows how to perform the system composition and interoperability cross checking. Assume that the control system Tank is to interoperate with another control system, the Security, which performs specific security check.

This is the use scenario for tank control system:

```
do ACTION:Tank.Start
choice {
  option [
    {do ACTION:<Security>.VerifyPassword} precondition
    do ACTION:Tank.Move
  ]
  option [
    {do ACTION:<Security>.VerifyPassword} precondition
    do ACTION: Tank.LocateTarget
  ]
  option [
    {do ACTION:<Security>.VerifyPassword} precondition
    do ACTION:Tank.Fire
  ]
}
do ACTION:Tank.Stop
```

The use scenario for security control system is then:

```
do ACTION:Security.VerifyPassword postcond
{
    do ACTION:<Tank>.Move
    do ACTION:<Tank>.Fire
}
```

The use scenario of Tank and Security are listed above. With the precondition or / and postcondition specified, we can know that the sequence

< Start, Security.VerifyPassword, Move, Stop >

is a correct calling sequence but

< Start, Move, Security.VerifyPassword, Stop >

is not a correct one since the action VerifyPassword needs to be done before anybody can Move the tank according to the use scenarios.

The interoperability cross checking can detect potential inconsistency for the use scenarios. In the use scenarios specified above, system Tank requires verifying password before any operation on the Tank, while Security enables Fire and Move after verifying password, without mentioning LocateTarget. A cross checking shows a potential inconsistency, which is not necessary an error. Either Account enforces an unnecessary strong precondition on LocateTarget, or Security enables an insufficient weak postcondition on VerifyPassword.

### 3.7 System Re-configuration and Re-composition

After a complex system is composed using subsystems, it may be re-composed statically or dynamically. When a composite system is running, the individual subsystems may not satisfy the performance or functional requirement for achieving a mission. When a subsystem is considered as not satisfying, we can re-compose the composite system by replacing the individual subsystems or adding new subsystems. The dynamic re-configuration needs to be done dynamically during the system run-time [15].

In Figure 2, suppose we have chosen system A.1 as the subsystem to compose system 1. During the system run-time, system A.1 may not satisfy the performance requirement. In this case, the malfunction of system A.1 will be detected and subsystem A.2 or A.3 will be plugged into the system to replace subsystem A.1.

Users may want to change the architecture of the composition system. The re-composition still needs to follow the specification in the use scenarios. Once a system is re-composed, it can be deployed rapidly. Also, it is possible that the users can add a new subsystem into the composed system or remove and / or replace a non-active subsystem in the system runtime.

## 4. Conclusion

Based on use scenarios, we proposed the concept of semantic interoperability which extends the general semantics beyond the concept of ontology. Once a system is specified with use scenarios, it can be used by other systems by simply following the steps defined in the use scenarios. Furthermore, the analysis capabilities included in the use scenarios can be used to automatically verify and validate the correctness of system composition, which significantly increases the confidence and reduces the effort to verify and validate the system..

## References

- [1] Parnas, D.L., "Designing Software for Extension and Contraction", Proceedings of the 3rd International Conference on Software Engineering (10-12 May 1978), pp. 264-277.  
Later published in IEEE Transactions on Software Engineering, March 1979, pp. 128-138.
- [2] Craig Larman, *Applying UML and Patterns – An Introduction to Object-Oriented Analysis and Design and Iterative Development*, third edition, Prentice Hall, 2004
- [3] Institute of Electrical and Electronics Engineers. *IEEE Standard Computer Dictionary: A Compilation of IEEE Standard Computer Glossaries*. New York, NY: 1990.
- [4] Kenneth C. Loudon, *Programming Languages – Principles and Practice*, Thomson Learning 2nd ed.,2002.
- [5] Yinong Chen, *Introduction to Programming Languages*, Kendall-Hunt Publishing, 2003.
- [6] Daryl Kulak and Eamonn Guiney, *Use Cases – Requirements In Context*, Addison-Wesley, 2000
- [7] 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.
- [8] Global Information Grid (GIG) Enterprise Services (ES), <http://ges.dod.mil/>.
- [9] Net Centric Enterprise Services (NCES), <http://www.disa.mil/ca/buyguide/feature/nces.html>
- [10] 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.
- [11] W. T. Tsai, R. Paul, Z. Cao, L. Yu, A. Saimi, and B. Xiao, "Verification of Web Services Using an Enhanced UDDI Server", Proc. of IEEE WORDS, 2003, pp. 131-138.
- [12] W. T. Tsai, C. Fan, R. Paul, and L. Yu, "Automated Event Tree Analysis from Scenario Specifications", Proc. of IEEE ISSRE, 2003, pp.240-241.
- [13] W. T. Tsai, L. Yu, R. Paul, C. Fan, and X. Liu, "Rapid Scenario-Based Simulation and Model Checking for Embedded System", Proc. of International Conference on SEA, 2003, pp 568-573.
- [14] W. T. Tsai, R. Paul, and L. Yu, A. Saimi, and Z. Cao, "Scenario-Based Web Service Testing with Distributed Agents", IEICE Transaction on Information and System, 2003, v.E86-D, no. 10, pp.2130-2144.
- [15] W. T. Tsai, Weiwei Song, Ray Paul, Zhibin Cao, Hai Huang, "Services-Oriented Dynamic Reconfiguration Framework for Dependable Distributed Computing", COMPSAC 2004 , September 2004, Hong Kong, pp.554-559.
- [16] W.T. Tsai, X. Liu, Y. Chen, R. Paul, "Dynamic Simulation Verification and Validation by Policy Enforcement," 38th Annual Simulation Symposium, San Diego, CA, April 2005
- [17] W3C, Web Services Architecture Working Group. <http://www.w3.org/2002/ws/arch/>.