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

Introduction

- Interoperability is defined as
 - The ability of two or more systems or components to exchange information and to use the information that has been exchanged.
- Interoperability is a critical issue for DoD C2 systems.
- Particularly, the recent emphasis on Network-Centric Warfare (NCW) placed interoperability as a priority item.

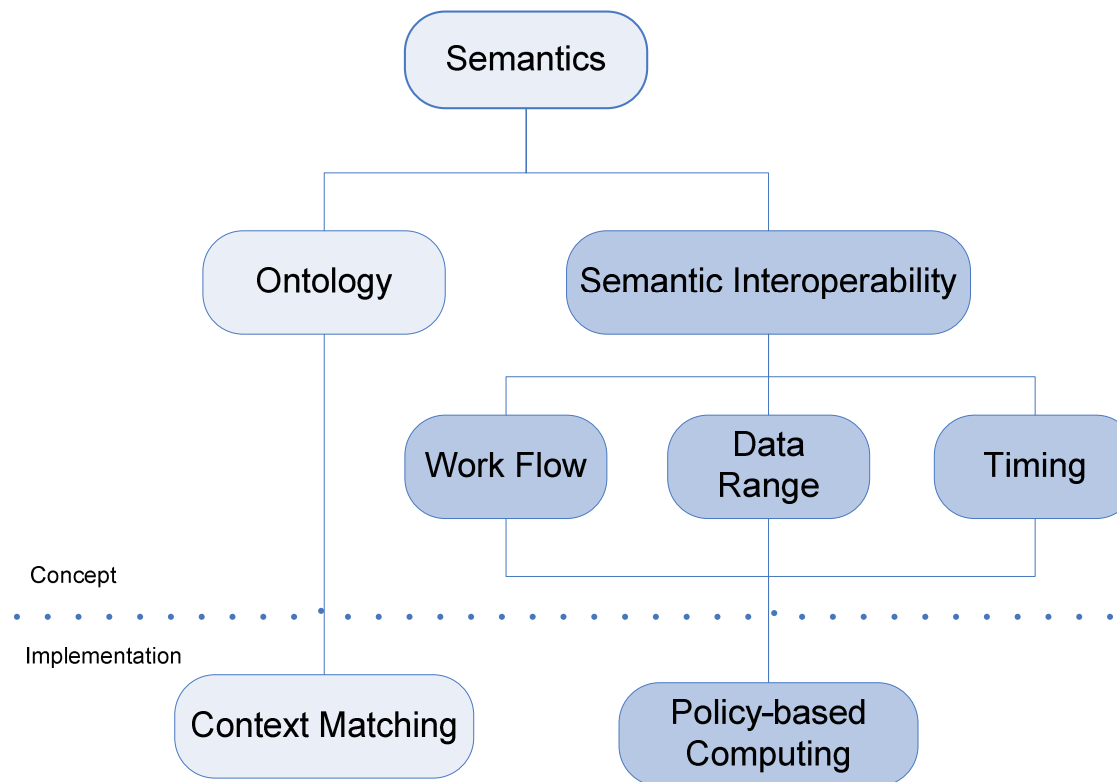
Introduction (Cont.)

- In spite of extensive studies on interoperability, 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 concept representation used for specification and matching; and
 - Service-Oriented Architecture (SOA) and other related technologies such as standard protocols (SOAP, UDDI, and etc.), interface definitions (WSDL, OWL-S, and etc.), registration and publication of interfaces, wrapper, automated composition.

Introduction (Cont.)

- 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?

General Semantics and Semantic Interoperability



General Semantics and Semantic Interoperability (Cont.)

- Generally speaking, semantics defines the meaning of the constructs of a language, or what happens during the execution of a program or program part.
- 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 general semantics.

Semantic Interoperation

- Semantic interoperation
 - Deals with how the data exchanged among the services can or cannot be used.
- There exist different kinds of the semantic interoperability
 - Workflow;
 - Data range; and
 - Timing.
- Semantic interoperability can be
 - Represented as constraints; and
 - Implemented by policy based computing

DoD C2 Requirements

- 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 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 is the proper calling sequence among methods of this service
 - What is the dependency among methods of a service or another service.

Scenario-based System Semantic Interoperability

- Following the concept of SOA, each sub-system in the composed complex system
 - Is a self-contained autonomous system;
 - Provides services;
 - Collaborates with each other; and
 - Loosely couples with other systems.
- To achieve interoperability, each system needs to be able to
 - Exchange data and services in a consistent and effective way.
 - Provide universal access capacities independent of platforms.

Scenario-based System Semantic Interoperability (Cont.)

- 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 some 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.

System Service Specification

- 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
 - Describes the calling parameters and return values of the system.
 - The ACDATE model in E2E automation provides the capability for interface specification
 - System Scenario & Use Scenario
 - Describe how the system works and how to work with this system.

ACDATE / Scenario Overview

- The ACDATE (Actors, Conditions, Data, Actions, Attributes, Events) modeling specification
 - A language for modeling and specification in the domain of system engineering and software engineering.
 - 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
 - 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 requirement V&V.

Use Scenarios

- Use scenarios
 - The use scenario is an extension to UML's *use case* and David Parnas' concept of *use*.
 - It specifies how a service or system is used by other services or systems.
 - It focuses on the work flow part of the semantic interoperability.
 - It defines how a particular function can be used in a stepwise fashion.
 - Current interoperability definition of systems mainly specifies the functions and the syntax of calling the services.

Use Scenarios vs. System Scenarios

- A system scenario describes the behavior of a system when the system is activated with a specific input,
- 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.

Use Scenario Specification -- syntax & semantics

- 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.
 - <>:
 - Any entities enclosed by <> are *parameter entities*.
- With sub-scenarios, the use scenario can describe the interoperation of hierarchical systems in different levels.

Use Scenario Analyses

- With the use scenario specified, we can perform
 - Automated interoperation scenarios generation
 - Interoperation scenario correctness checking
 - Interoperability cross checking
- 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.

Use Scenario Example

- This example use scenario is specified for a control system that is in charge of battle tank in a simple C2 system
 - The control system, *Tank*, which has 5 functions:
 - Start
 - Move
 - LocateTarget
 - Fire
 - Stop
 - The control system, *BattleControl*, which has 1 function:
 - OrderToFire
 - The control system, *Security*, which has 1 function:
 - VerifyPassword

Use Scenario Example (Cont.)

- Use scenario for *Tank*:

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

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.

Automated Interoperation Scenarios Generation -- Example

- Automated generated interoperation scenarios:
 - < Start, Move, Stop > ,
 - < Start, LocateTarget, Stop > , and
 - < Start, Fire, Stop > .
- When interoperate with *BattleControl*, following interoperation scenarios can be generated:
 - < Start, *BattleControl.OrderToFire*, *LocateTarget*, Stop >
 - < Start, *LocateTarget*, *BattleControl.OrderToFire*, Stop >

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 generated interoperation scenarios are correct sequence according to the constraints specified.
- By the constraints checking we can identify the interoperation scenarios that do not satisfy the constraints.
 - precondition checking;
 - postcondition checking; and
 - critical region checking.

Updated Use Scenario Example

- The use scenario is updated by adding a *Critical Region* .
- Updated use scenario for *Tank*:

```
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
```

Interoperation Scenario Correctness Checking -- Example

- With the *Critical Region* constraint specified for the Tank use scenarios, not all interoperation scenarios are correct.
- Interoperation scenario for Tank and BattleControl:
 - < Start, LocateTarget, BattleControl.OrderToFire, Stop > is a correct interoperation scenario.
 - < Start, BattleControl.OrderToFire, LocateTarget, Stop > is **NOT** a correct interoperation scenario.
 - BattleControl.OrderToFire can not be put in the section tagged as *criticalreg*.

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.

Updated Use Scenario Example

- The use scenario is updated by adding *Preconditions* .
- Updated use scenario for *Tank*:

```
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
```

Use Scenario Example (Cont.)

- The use scenario for security control system is:

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

Interoperability Cross Checking -- Example

- In the use scenarios specified above, system Tank requires verifying password before all following operations on the Tank:
 - Move
 - LocateTarget
 - Fire
- 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.

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.
- The following information can be added:
 - Dependency information;
 - Categorization; and
 - Hierarchical use scenarios.

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
 - Describes other systems that need to be included for this system to function. Compatible components list
- Compatible components list
 - 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.

Dependency Information -- 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.
 - From information above, the users will know it is compatible to put helicopters, fighter planes, and scouts on the aircraft carrier but **not** the battle tanks.

Categorization

- For better organization, the use scenarios need to be categorized since
 - A system can provide multiple services.
 - Different services provided by the system may have different use scenarios.
 - A system working with different systems may have different use scenarios.
- 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.

Categorization -- 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.
 - Use scenarios for Fleet1
 - Use scenarios for Fleet2
 - Use scenarios for Fleet3

Hierarchical Use Scenario

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

Hierarchical Use Scenario -- 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.
- The use scenario hierarchy:
 - Use scenarios for command center
 - Use scenarios for army
 - Use scenarios for infantry
 - Use scenarios for 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.

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.

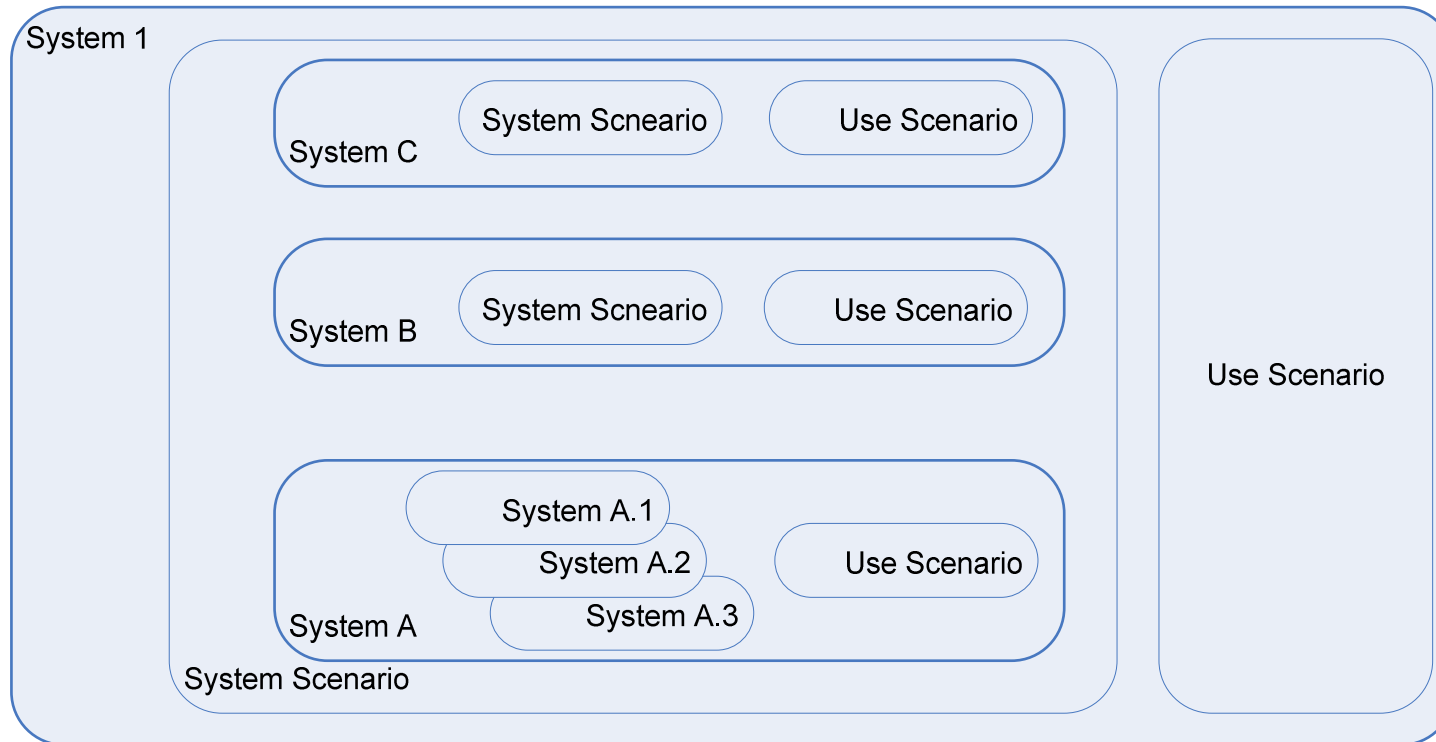
System Composition Approach

- 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.
 - 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.

System Composition Approach (Cont.)

- The system discovery and selection can be done by analyzing the system scenarios.
- 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.

System Composition Example



System Composition Example (Cont.)

- The System 1 composition figure in last page shows the composition information of a complex system 1 with three subsystems.
 - 3 subsystems for system 1:
 - System A,
 - System B, and
 - System C.
 - Each system is specified with system scenarios and use scenarios.
 - System scenarios for each subsystem provide information on what services this system provides.
 - Each subsystem is specified with use scenarios, the integration becomes possible.
 - 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.

System Composition Template

- A use scenario can have templates.
 - In such a template, we may specify what functionalities we need.
 - If a system provides the functionalities specified in the system scenario, it can be used as subsystem.
- In the system 1 composition figure:
 - 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 Scenario / System Scenario Conversion

- Use scenarios for a lower level subsystem can be converted to system scenario for a higher level system.
- In the system 1 composition figure:
 - 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.

System Re-composition

- After a complex system is composed using subsystems, it may be re-composed statically or dynamically.
 - Re-composition is needed when a subsystem is considered as not satisfying
 - Replacing the individual subsystems
 - Adding new subsystems.
- The re-composition still needs to follow the specification in the use scenarios.
- Once a system is re-composed, it can be deployed rapidly.
- 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.

Summary

- Semantic interoperability 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.
- 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.

Thanks!
