The Impact of SOA Policy-Based Computing on C2 Interoperation and Computing

Raymond Paul
OSD NII
Department of Defense
Washington, DC

W. T. Tsai
Department of Computer Science and Engineering
Arizona State University
Tempe, AZ 85287-8809

Jay Bayne
Echelon 4 Corporation
Mequon, WI 53092-3477

Abstract

This paper presents issues related to policy-based computing in a service-oriented architecture (SOA) for network-centric warfare. Service-oriented computing and its associated architecture represent a new paradigm of computing and numerous issues need to be addressed. One of important issues is distributed policy computing. Issues such as policy specification, analysis, enforcement, synchronization, and communication are all related.

Keywords: Service-oriented architecture, service-oriented computing, enterprise C2 systems, policy-based computing.

1. Introduction

DoD is moving into GIG-based service-oriented network-centric policy-based enterprise computing. These new C2 systems will be drastically different from the traditional C2 systems due to their

- Network-centric nature;
- Service-oriented architecture (SOA) and associated the Service-Oriented Computing (SOC);
- Policy-based computing.

Specifically, network-centric operation means that each C2 system must be network ready, and must be able to interconnect and interoperate with other C2 systems. The SOA adds another new dimension to this problem as SOC is fundamentally different from traditional procedural programming as well as the OO computing. The requirement that future C2 systems will support policy-based computing adds another complexity to these

challenges. The technology for policy-based computing is at the development stage only and it will take years before the technology is mature enough to be applied.

This paper addresses the impact of SOA on modern network-centric enterprise and policy-based C2 systems. Network-centric operation is already a difficult issue, but SOA involves additional issues such as C2 policy specifications, V&V and enforcement. In an SOA system, a service can be discovered, bound, executed, and verified and validated at runtime and in real-time, and the entire system is organized as a loosely coupled system of services. While this kind of loose coupling provides flexibility and adaptability, it is difficult to enforce C2 policies in such a network. It is difficult because each computation unit in such a system is actually a collection of services possibly composed in real time and at runtime. So, how can one enforce C2 policies in such an environment as it is not known what service will be used before application?

While consensus exist on SOA-based EC2 (Enterprise C2) systems such as

- Policies must be formally specified and embedded in the enterprise C2 system because these policies may be dynamically composed in real time and at runtime;
- Policies must be enforced in a distributed manner involving policy synchronization and coordination;
- Actions and tasks must be checked and enforced against stated C2 policies in real time and at runtime.

However, there exist differences in treating policies in an SOA. This paper will discuss those issues.

This paper is organized as follows. Section 2 briefly overviews SOC paradigm; section 3 presents policy-based computing; section 4 shows two different schools of thoughts, both of them use the SOA but the ways they treat policies are drastically different. These two approaches provide a good comparison and contrast of policy-based computing in an SOA.

2. Service-Oriented Computing

The key concepts of SOC are as follows [9]:

- Each computing unit is considered as a service, and each service is considered self-contained and independent of other services;
- A service publishes its interfaces using a standard language WSDL;
- The service provider registers its services with an intermediate broker called UDDI server.
- A client that wishes to obtain a service inquires the UDDI server and requests specific services that it needs;
- After receiving the user request, the UDDI matches the need with the most appropriate service in its repository, and sends the ID of the service to the client;

- The client then makes a call to the service requesting specific kind of tasks to be performed; and
- All communications among all parties are through standard protocols such as XML and SOAP

SOC is fundamentally different from OO Computing (OOC) even though SOC evolves from OOC. SOC may looks similar in many ways to OOC, but it is a different computing paradigm. In the past, some people mistakenly thought that OOC is not much different from procedural computing because procedural languages already have the concept of data abstraction and procedure calls. They did not realize that even though OOC may look similar to procedural computing in numerous ways, the fact that designers think in terms of classes and objects fundamentally changes the way of thinking. And from the OOC, many new concepts emerge such as UML, design patterns, object composition, dynamic binding, and design architecture.

Similarly, SOC is fundamentally different from OOC because now designers are thinking in terms of services, runtime service discovery and composition [6][7], runtime system re-composition, and runtime system verification and validations. These are different from the OOC concepts.

Furthermore, services are available and accessible on-line. A user or a software agent can use runtime search to discover new services. A user may not need to buy and install, instead just call and use the service remotely. Software upgrade will become easier because once the service is upgraded, the new service will immediately replace all the old versions instantly, saving significant cost of un-installing and re-installing software.

The SOC also has a significant impact on the system structure and dependability such as system reliability, system composition (and re-composition), and security. These mechanisms are different from the corresponding OOC mechanisms. For example, instead of static composition (with dynamic objects and dynamic binding) in OOC, SOC allows dynamic composition in real time and at runtime using services just discovered, and with knowledge of the service interface only [8][14]. Because multiple new services that meet the same specification can be discovered, SOC also needs a runtime ranking, interoperability evaluation, testing, and selection mechanisms. In case of system failures, the SOC has a distributed reconfiguration [8][14], and such protocol is different from the reconfiguration protocol in OO software. In SOC, a faulty service can be easily replaced by another standby service by the DRS (Dynamic Reconfiguration Service), and the DRS itself is also a service that can be monitored and replaced. The key is that each service is independent of other services, and thus replacement is natural. Only the affected services will be removed and this allows mission-critical applications to proceed with minimum interruption.

This will be difficult to achieve in OOC because the replacement code must be developed and put into the place via dynamic binding. The current OO dynamic binding mechanism allows methods that belong to a family of classes can replace each other at runtime, yet

SOC allow an unrelated service to replace an existing service as long as the new service has the same specification.

The SOA is robust and survivable because most of binding is done at *runtime*, thus services can be removed and/or added to the service pool without changing the overall architecture and protocols among all the parties. If a service is removed from the service pool, the UDDI server will select a new service provider at runtime; similarly if a new service wishes to join the service pool, it simply registers its services with the UDDI server.

- Essentially, SOC will change the whole landscaping in system and software development from requirement to V&V. Specifically, in the **requirement** stage, knowledge of existing services is critical as reusability will be the key enabler. Thus, rather than construction of new requirements, reuse of existing requirements including existing service specifications will be critical.
- In the **design** stage, the loosely coupled SOA allows dynamic composition, dynamic re-composition, and dynamic reconfiguration. In the sense, the design is never completed because new service may arrive after the current design is completed, and the new service may replace the already selected services at runtime in real time.
- In the **implementation** phase, majority of work will be composition (or linking) rather than code development as services will be reused.
- In the **V&V** phase, CV&V (Collaborative Verification and Validation) [10][11][13][18] will be used rather than IV&V as the source code of many services may not be available. Service specification will be extensively used because often a service customer has access to specification and URL only, no source code or even binary code would be available.

Various SOA system engineering techniques are available, including:

- Automated completeness and consistency checking [15].
- Verification and analysis techniques including both static and dynamic verification techniques [19];
- Automated state model generation [20];
- Formal model checking [2][3];
- Runtime verification and constraint checking with simulation [16];
- Automated test case generation [17];
- Test coverage analysis based on the service specification and risk analysis;
- Automated dependency identification and analysis;
- Automated concurrency analysis; and
- Autonomous distributed test execution via remote agents [12].

Acceptance of SOA within DOD

SOC has recently gained acceptance within DOD. Specifically, two important projects, namely NCES and GIG-ES are based on SOC. Specifically, NCES has the following critical services:

- Discovery;
- Messaging;
- Collaboration;
- Brokering;
- Storage;
- IA (information assurance) and Security;
- Domain Name Service (DNS);
- Enterprise service management (ESM); and
- Mediation.

And GIG-ES adds the additional services:

- Command and control;
- Focused logistics;
- Battlespace awareness;
- Force applications;
- Protection;
- Expedient and cross-domain COI (Community of Interest) including business and warfighting COIs;
- Human resource management;
- Strategic planning and budgeting;
- Accounting and finance;
- Logistics; and
- Acquisition.

3. Policy-based Computing

A policy is a statement of the intent of the controller of some computing resources, specifying how he wants them to be used. A policy allows or denies subjects (such as processes) that satisfy some conditions to perform designated actions on objects (such as files) [1]. The common pattern of a policy takes the form of:

{allow | deny | require} {subject} performs {action} on {object} when {conditions}

Policies range from simple (accept users with correct passwords) to complex (reject users who read a file three times over the past 24 hours). A policy-handling system must

provide a flexible and powerful means to specify various policies. Runtime mechanisms must be provided as well, so that the system can enforce specified policies. Policies are also changeable, which requires the system be able to dynamically revise policies. Consequently, *policy specification*, *enforcement* and *revision* are the three basic mechanisms a policy-handling system must provide.

> ***Policy***, n. 1. a course or general plan of action. 2. a contract of insurance governing a plan of action. 3. the rules or constraints governing a general plan of action.

Within the context of the EC2 the term policy refers to the set of rules (constraints) that govern C2 actions within a given C2 *policy domain*. Specifically, a policy is a statement of the intent of a controller of some resource specifying how such resources are to be used. In an SOA, a policy will be interpreted and executed in real time at runtime to meet the agile and changing requirements.

Policy Specification Languages

Policies are ubiquitous in most, if not all, computing systems. However, one might not be aware of their existence, because most policies are coded into a system's implementation by functional requirements, language features, and design decisions [4][5]. For example, if a policy says, "passwords must be at least 8 characters long", there must exist a segment of code in the system that checks the length of passwords.

This traditional approach to implementing a policy-handling system has limitations:

- It does not separate policy specification from policy implementation.
- Policies are difficult and expensive to change. Adding new policies or updating / removing existing policies requires modifying the policies, recompiling and redeploying these policies into the system.

If policy specifications and implementations are separated, any change of policies can be easily implemented without changing the architecture of the system, and this significantly simplifies the entire process. Policy specification languages are an attempt to meet the above requirements. A policy is thus a binding of entities and their attributes to specified actions. What entities, attributes and actions can be represented depends on the concrete system in which policies are to be specified.

Some of benefits of employing policy specification languages to implement a policy-handling system are:

- Policy specification languages enable policies to be defined, independent from a concrete system implementation.
- Policy specification languages are to be interpreted by a policy engine at runtime, which makes dynamical policy changes possible.
- Policy specification languages formalize the intent of the controller into a form that can be read and interpreted by systems.

- Policy specification languages are high-level languages, which makes it easy to learn and use by policy makers who are normally non-programmers.

Policy-Based Computing in GIG

Policy-based computing is motivated by GIG as it has many policy specification and enforcement requirements:

- **Policy-Based Networking**: Various network operations will be possible but they will be controlled by policies in an EC2 SOA.
- **Common Open Policy Service**: this is a query and response protocol that can be used to exchange policy information between a policy server and its clients.
- **Routing Protocol Specification Language**: this allows a network operator to specify routing policies at various levels.
- **Internet Protocol (IP) Security Policy**: this is a repository-independent information model for supporting IP security policies.

Policy-based computing means that each computing unit within an EC2 has the following capabilities or rules:

1. It has a formal and self consistent set of C2 policies,
2. It has a set of C2 policy execution processes (mechanisms),
3. It has a set of C2 process performance measures and measurement processes, and
4. It has an accountability structure governing allocation of assets required in and consumed by the execution of any given policy or set of policies

In an EC2, a policy is specified and executed within a policy domain as defined below:

> ***Policy Domain***, n. 1. a region of *policy space* where a particular set of policies are applied in the implementation (execution) of one or more plans of action (POA)

Within the context of an EC2 enterprise, there are many policy domains, organized according to the extant *command and control accountability structure* (hierarchy). The EC2 accountability structure defines a policy tree spanning selected activities of the joint field combatant commanders up through the President of the United States. This "nested" set of interdependent policy domains constitutes a *containment hierarchy*. Generally, enterprise policy trees include high-level strategic, mid-level operational and low-level tactical policy domains.

Another important feature is that a policy in an EC2 SOA must be executable as policies will be enforced at runtime in real time, and thus it is essential policies can be executed at runtime and in real time to enforce their constraints. The execution capability of a policy language enables numerous tasks that were not possible before, including the ability to

- Simulate the effects of adding new or modifying existing policies prior to their deployment throughout relevant policy domains, and

- Provide a formal means of *verification and validation* that policies are "correct" with respect to quality of service (QOS) and service level agreements (SLA) established for their respective containment domains at runtime and in real time.

Policy Enforcement and Operations

A policy language must allow policies specified to be enforced either statically or at runtime. Both static and runtime policy enforcement follows a 3-step process:

- Policy C&C checking: This ensures that policies specified are complete with external requirements and regulations, as well as consistent with each other before application of these policies to evaluate the concerned system. Sample security policies can be Bell-LaPadular security policies or Chinese Wall security policies.
- Embed policies with the system specification: Once the policies specified are determined to be of reasonable quality, the policies specified using PSEL can now be embedded into the system specification for evaluation
- Evaluation of the system with respect to policies specified: This step evaluates the system to see if it satisfies the policies specified.

Similarly, a policy language must allow policies to be updated, simulated, and analyzed in real time and at runtime. Also, note that these must be done while the EC2 is still running, possibly during an important mission.

4. Different Schools of Thoughts on SOA for Enterprise C2 Systems

One school of thought (we label it School A) is that SOA is simply an implementation technology just like other programming languages or design techniques such as OOC, and thus the impact of SOA to EC2 is limited as the fundamental C2 principles remain the same regardless of the implementation technology used. In this way, policies can be treated as static XML text. It is the role of some NCW policy management service that creates, publishes, and maintains a federation of policy databases. Furthermore, each enterprise node must be able to establish and maintain its own set of local policies as well as be aware of the global policies under which it must comply. As a consequence, the NCW policy services are simultaneously local and global. V&V is first and foremost a local matter (self-consistency), and secondarily a global matter (mutual consistency). Resolution of local and global policies is subjective and the purview of enterprise managements. Essentially, policies are treated as

Policy := {{Rules}, {Penalty Functions}}

Another school of thoughts (School B) believes that even though the fundamental C2 principles remain the same, the technology used can have a drastic impact on EC2. While both Schools A and B are new, they have different opinions on the following issues:

School A treats policies as *passive* entities and they are used mainly for constraint checking and enforcement. In this way, the C2 processes used to implement the policies

need to be synchronized. Furthermore, because C2 policies are mainly static objects, the creation, specification, and enforcement will be easier to manage. The main focus is on developing the generic EC2 architecture that is independent of the implementation technology and can use a hodgepodge of infrastructure to implement EC2 if necessary, including a mixture of traditional C2 systems and SOA C2 systems. This school maintains that the EC2 architecture remains the same and should not be affected by any implementation technology such as SOA.

School B treats policies as *active* services that will be called in real time to perform enforcement. In this way policies can be published, disseminated, executed, and monitored just like any other service in the system. While the new school also ensures the fundamental EC2 principles remain the same, the impact of SOA can be immeasurable. In fact, the changes can be drastic because C2 policies can be updated and validated in real time and at runtime to give decision makers and warfighters maximum flexibility for the modern agile warfighting.

School A treats policies as passive services affecting only the implementation, not the architecture or design. This school thinks that frequent changes in policies create significant overhead such as synchronization among various C2 processes, changes in action plan generation, and changes in asset management. These changes will make C2 processes complicated and thus slow down C2 processing. Furthermore, updating C2 policies in real-time can be problematic as it may complicate the decision process as C2 policies are often related to the surrounding context.

School B considers flexibility and adaptability are the key factors; treating C2 policies as active services and providing the flexibility needed in the modern agile warfighting. The added overhead should be rather minimal. The SOA already has this overhead of service interfaces: publishing, discovery, composition, execution, verification, and monitoring, and thus treating policies as active services should not increase the overhead much, however, the benefits of treating C2 policies as active services add significant flexibility to warfighters that were not available before.

	Policies as passive objects (School A)	Policies as active services (School B)
Policy Specification	Policies specified as data such as XML files.	Policies specified as services
Policy Enforcement	Policies will be enforced by execution services, different enforcement algorithms and software can be used.	Policy services are active services that can perform enforcements. Multiple services may cooperate together to accomplish service enforcement.
Policy Overhead	As policies are treated as static objects, the overhead of policy is the overhead of accompanied computation in managing and enforcement of policies.	Policy services is just like another services. The overhead is the added complexity of activating policy services while doing regular computing.

Policy Update	Policy can be updated by updating the related policy files, and possibly policy processing services as well.	Policy can be updated by replacing the existing policy service by a new policy service.
Policy V&V	Various static and dynamic V&V mechanisms can be used, e.g., completeness and consistency checking, and policy simulation.	Policy V&V will be equivalent to service V&V, a variety of static and dynamic V&V techniques can be used including test case generation, simulation, and runtime monitoring.
Policy Simulation	Policy can be simulated by running the system using policy data.	As a policy service is just another service in an SOA, policy simulation can be carried using any SOA simulation.
Policy communication	Two systems can send policies to each other like sending a regular electronic message as policies are treated as data.	Two systems can communicate their policies by sending service specifications or the URL of the related services.
Policy synchronization	Policy synchronization is needed whenever a policy is updated or the system is reconfigured. Formal synchronization protocols need to be employed to ensure completeness and consistency.	Policy synchronization is needed whenever a policy service is updated or system is reconfigured. Policy services may actively pursue their own service synchronization protocols.
System Structure	Enterprise C2 systems will be organized in an SOA, but policies will be treated as objects or data used by services.	Enterprise C2 systems will be organized in an SOA-manner and policy services will be treated like a regular service.
System Reconfiguration	Systems can be easily reconfigured using a DRS (dynamic reconfiguration service) and policy files need to be updated to ensure that new policies are consistent with the new system structure.	System can be easily reconfigured using a DRS, and policy services will be a part of reconfiguration process.

5. Other Consideration Issues and Open Issues

What is the likely change rate of policies? Different policies are likely to have different change rate. Any time a policy is updated, the C2 system may need to go considerable changes, e.g., the C2 SOA system may need to be re-composed to fit the new requirements of the updated C2 policies. Because of such overhead, it is important that the enterprise C2 system will be organized in a hierarchical manner, and thus minimizing the impact of the change once C2 policies are updated.

The EC2 SOA still face many open problems such as

- A formal yet easy to use policy specification language that can be used to specify, analyze, simulate and enforce distributed policies in an EC2 SOA federated systems;
- Distributed policy synchronization and enforcement; and
- Policy dependency analysis to assist real-time policy evolution;

These and other related issues must be addressed before policy-based computing can be realized.

References

[1] N. Damianou, A. Bandara, M. Sloman and E. Lupu, “A Survey of Policy Specification Approaches”, Technical Report, Department of Computing at Imperial College of Science Technology and Medicine, 2002.

[2] H. Huang, W.T. Tsai, R. Paul, and Y. Chen, "Automated Model Checking and Testing for Composite Web Services", to appear in 8th IEEE International Symposium on Object-oriented Real-time distributed Computing, Seattle, May 2005.

[3] H. Huang, W.T. Tsai, and R. Paul, "Proof Slicing with Application", to appear in 8th IEEE International Symposium on Object-oriented Real-time distributed Computing, Seattle, May 2005.

[4] L. Kagal, “Rei: A Policy Language for the Me-Centric Project”, Technical Report, HP Laboratories, 2002.

[5] M. Kangasluoma, “Policy Specification Languages”, Technical Report, Department of Computer Science at Helsinki University of Technology, November 1999.

[6] N. Milanovic and M. Malek, “Current Solutions for Web Service Composition”, IEEE Internet Computing, Nov/Dec, 2004, 51-59.

[7] N. Milanovic and M. Malek, “Verifying Correctness of Web Services Composition”, Proceedings of the 11th Infofest, Budva, Montenegro, 2004.

[8] R. Paul and W. T. Tsai, “Service-Oriented Architecture for Command and Control Systems with Dynamic Reconfiguration, 2004 Command and Control Research and Technology Symposium, 2004. http://www.dodccrp.org/events/2004/CCRTS_San_Diego/CD/papers/183.pdf

[9] M. P. Singh and M. N. Huhns, *Service-Oriented Computing*, John Wiley & Sons, 2005.

[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, R. Paul, L. Yu, A. Saimi, and Z. Cao, “Scenario-Based Web Service Testing with Distributed Agents”, IEICE Transactions on Information and Systems, 2003, Vol. E86-D, No. 10, 2003, pp. 2130-2144.

[13] W.T. Tsai, Y. Chen, R. Paul N. Liao, and H. Huang, , “Cooperative and Group

Testing in Verification of Dynamic Composite Web Services", in Workshop on Quality Assurance and Testing of Web-Based Applications, in conjunction with COMPSAC, September 2004, pp. 170-173.

[14] W. T. Tsai, W. Song, R. Paul, Z. Cao and H. Huang, "Service-Oriented Dynamic Reconfiguration Framework for Dependable Distributed computing", Proc. of IEEE COMPSAC, 2004, pp. 554-559.

[15] W. T. Tsai, R. Paul, L. Yu, X. Wei, and F. Zhu, "Rapid Pattern-Oriented Scenario-Based Testing for Embedded Systems", in *Software Evolution with UML and XML*, edited by H. Yang, 2004.

[16] W. T. Tsai, C. Fan, Z. Cao, B. Xiao, H. Huang, X. Liu, X. Wei, R. Paul, Y. Chen and J. Xu, "A Scenario-Based Service-Oriented Rapid Multi-Agent Distributed Modeling and Simulation Framework for SOS/SOA and Its Applications", Foundations '04: A Workshop for VV&A in the 21st Century, Tempe, October 2004, http://www.scs.org/confernc/foundations/foundations04.htm.

[17] W. T. Tsai, X. Wei, Y. Chen, B. Xiao, R. Paul, and H. Huang, "Developing and Assuring Trustworthy Web Services", to appear in the Proc. of The 7th International Symposium on Autonomous Decentralized Systems (ISADS), April 2005.

[18] W.T. Tsai, Y. Chen, and R. Paul, "Specification-Based Verification and Validation of Web Services and Service-Oriented Operating Systems", Proc. of IEEE WORDS, Sedona, February 2005.

[19] W.T. Tsai, X. Liu, Y. Chen, R. Paul, "Simulation Verification and Validation by Dynamic Policy Enforcement", Proceedings of the 38th Annual Simulation Symposium, April 2005.

[20] W. T. Tsai, L. Yu, R. Paul, C. Fan, X. Liu and Z. Cao, "Rapid Scenario-Based Simulation and Model Checking for Embedded Systems", Proceeding of 7th IASTED International Conference on Software Engineering and Applications, 2003, (SEA2003).