GMU C4I SEMINAR

Tomorrow’s Needs - Yesterday’s Technology: DoD’s Architectural Dilemma

Dr. Raymond A. Paul
Raymond.Paul@osd.mil
“There are things we know that we don’t know – the known unknowns. And there are unknown unknowns – the things we do not yet know that we don’t know”
- Donald Rumsfeld, U.S. Secretary of Defense
This presentation expresses the views of Ray Paul. The thoughts do not represent the thinking from the minds in the DoD.
Historical Prediction: 2000

The evolutoinal and technological shift from products to services means that value is moving from the technology itself to how the technology is being applied. Value (in terms of increased productivity, total cost of ownership, improved efficiency and effectiveness, return on investment formula or benchmark, project completion time, and increased revenue) must be explicit and reflected from the technology’s investment.
C4I SEMINAR Outline

- Software as a service - as it is being embraced within DoD
- Differences between Service-Oriented Computing and OO Computing
- Change in the entire Software Engineering lifecycle
- Conclusion
Service-oriented computing including service-oriented architecture (SOA) and Web Services (WS) are receiving significant attention.

Most major corporations and government agencies (including DoD and NASA) are pushing this technology.

The idea that a software program as a service can be discovered, matched, composed, executed, verified, and monitored in real time and at runtime provides a new paradigm of computing.
New Service Model for Software

- Web services are available on-line
- No need to buy and install software
- No need of maintenance overheads
- Automatic upgrades
- Payment based on usage
- Microsoft’s licensing model with XP – where you have to register the software or it stops functioning in 30 days – is a step in this direction

*Use and pay … not buy and install*
Three parties are involved:

- **Service providers**
  - Have access to design and implementation as well as the interface WSDL
  - May host services

- **UDDI/ebXML**
  - Provide searching, updating
  - May have access to WSDL only

- **Clients**
  - Customer, may not have access to design and implementation
  - May have access to WSDL only
Many Assurance Challenges Ahead

In 2003, at ISSRE as keynote speaker, I presented the impact of software services on Internet computing, and suggested many challenges including

- Trustworthy issue will become important
- Need a new paradigm for V&V, i.e., instead of IV&V, we need collaborative V&V
- Reliability model for WS and SOC
- Security Vulnerability will be critical
- While most research projects today focus on composition and ontology (discovery).
Almost Four Years Later

- SOA and WS are getting acceptance these days.
- Four years ago, when we mentioned SOA within DoD, people were puzzled and said “service what?”
- But during the last two years, DoD started several SOA projects (to name just a few) including all the services:
  - NCES (Network-Centric Enterprise Services) (NII & DISA)
  - GIG-ES (NII & DISA)
  - JBMC2 (OSD, JFCOM)
  - FORCEnet (Navy)
  - FCS (Army)
  - JBI (AF)
- SOA is gaining momentum within DoD.
- The challenges in SOA and WS remain, and we’re not addressing important issues.
Service Oriented Computing is much more than OO Computing

Some people still think SOA is just a minor variant of OO computing;

- People said similar things when OO was introduced (such as “OO is just a minor increment of data abstraction”).
- However, the system structure and its implications to system composition, reliability, verification and validation, security, reconfiguration capabilities are drastically different from OO computing.
Differences

- Instead of static composition (with dynamic objects and dynamic binding) in OO, SOA allows dynamic composition in real time, at runtime, and with knowledge of the service interfaces only:
  - Include dynamic discovery and matching
  - Runtime ranking and selection of services
  - Runtime interoperability verification
Differences (continued)

- In case of system failure, the reconfiguration strategy will be rather different for OO and SOA
  - In OO, it is necessary to develop the reconfiguration strategy manually.
  - In SOA, a faulty service can be easily replaced by another standby service by the DRS (Dynamic Reconfiguration Service), and the DRS also is 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 shut down and this allows the mission-critical application to proceed with minimum interruption.

- Thus, SOA improves reliability of systems and systems of systems.
### Differences (continued)

<table>
<thead>
<tr>
<th></th>
<th>Traditional IV&amp;V (OO)</th>
<th>Service-Oriented CV&amp;V</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Definition</strong></td>
<td>By independent team</td>
<td>Collaboration among multi-parties</td>
</tr>
<tr>
<td><strong>Approach</strong></td>
<td>Off-line field testing</td>
<td>On-line just-in-time testing</td>
</tr>
<tr>
<td><strong>Regression</strong></td>
<td>Off-line regression</td>
<td>On-line regression</td>
</tr>
<tr>
<td><strong>Integration</strong></td>
<td>Static configuration &amp; linking</td>
<td>Dynamic reconfiguration &amp; binding</td>
</tr>
<tr>
<td><strong>Testing coverage</strong></td>
<td>Structural &amp; functional</td>
<td>Specification &amp; Usage based</td>
</tr>
<tr>
<td><strong>Profiling</strong></td>
<td>Static and centralized</td>
<td>Dynamic and distributed</td>
</tr>
<tr>
<td><strong>Model checking</strong></td>
<td>Based on source code or states</td>
<td>Just-in-time dynamic model checking</td>
</tr>
<tr>
<td><strong>Reliability model</strong></td>
<td>Input domain &amp; Reliability growth models</td>
<td>Dynamic profiling and group testing</td>
</tr>
<tr>
<td><strong>Certification</strong></td>
<td>Static certification center</td>
<td>Dynamic certification based on history</td>
</tr>
</tbody>
</table>
Network-Centric Computing

- This means that when a system is developed, it must be network ready, and can interoperate with the existing and future systems in an integrated manner.
  - However, current network-centric interoperability may mean data interoperability via XML only.
  - It is still far from system operation interoperability, but this is required.
- Thus, DoD may one day require new systems to be SOA-ready in addition to network-ready.
SOA Changes the Entire Lifecycle

- From requirements engineering to V&V, SOA and WS changes the landscape.
- In the **requirements** stage, knowledge of existing services is critical as reusability will be the key enabler.
- In the **design** stage, the loosely coupled service architecture will allow dynamic composition, and dynamic re-composition.
- 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 will be used rather than IV&V as the source code of many services may not be available.
Reusability will be a key consideration during the requirements phase.
  - Searching and discovering services that can be reused will be key – this means that a broker or library will be needed
  - Profile and ranking of these services will be a key consideration
- The system will assume an architecture from the very beginning, i.e., SOA.
- From the very beginning, it is understood that the system components (services) and architecture can be changed at any given time, even during runtime.
- The requirements phase is continuous and considers the evolution plan as new services may arrive after the system is deployed.

Once the requirements are fully understood and specified, lots of code will be available immediately. This is similar to Extreme Programming or Agile processes rather than the Waterfall model. However, the difference here is, many services will be ready for reuse immediately after the requirements analysis.
Impact on Software Applications

- Mutual Interdependence
- Reliability
- Vulnerabilities
Mutual Interdependence – Virtues and Pitfalls

“Interdependence is better than independence” – Stephen P. Covey

Certainly, because

- Each can leverage the other’s strengths
- Two (or more) can do together what each can’t on his own
- The basic principle behind Keynesian economics

However, it also

- Requires a high degree of trust between collaborators
- Leaves each vulnerable to the other(s)
- Reduces degrees of freedom in decision making
Mutual Interdependence in Web Services

Web Services based applications
- By their very nature are mutually interdependent, since
  - Applications are built by leveraging the services provided by other applications;
- However
  - Smooth operation of an application, and its ability to fulfill the needs of its clients satisfactorily, depends in turn on the smooth operations of other applications whose services it is using.
- Hence
  - There is a need to have policies, mechanisms, and agreements in place to manage the risks associated with the added vulnerability.
Web Services Reliability

- Various aspects of reliability
  - Service availability – what fraction of time is the service available
  - Service quality (QoS) – quality of the service provided
    - Timeliness
    - Precision
    - Accuracy
  - Graceful degradation & recovery – how well does the service degrade and recover?
  - Non-repudiation & dispute handling – how well does the service provider handle agreements in case of a dispute?
Evaluating Web services at Runtime in Real Time

- Traditional ‘Shrink wrapped & shipped’ software has its own QA process, and integration with other software is either limited or impossible.

- Web services are different. It interacts with other software frequently and extensively, and it is necessary to evaluate its quality at runtime in real time.

- First, what are attributes of quality for web services?
  - Reliability – the service will not crash
  - Performance – the service will return results rapidly
  - Security – the service will not leak sensitive data to 3rd parties and it will not return false, malicious information back to the client
  - Safety – the service will not harm its users, mission and environment
Evaluate “Reliability” of Services at Runtime

- Can we test across inter-organizational web services in real time and at runtime?
  - Functional testing: Can we generate the test cases/scripts for inter-organization services?
  - Coverage analysis: What kind of coverage can we anticipate? What would be good enough?
  - Test, evaluation and monitoring: how can we collect and evaluate test results including security and scalability test results?

- Can we develop reliability models for web services?
Managing Web Services Vulnerabilities

- A well-defined service level agreement (SLA) between web services provider and client
- Specific clauses defining reliability, availability, QoS, escalation, etc.
- Recommend a ‘test drive’ stage in the relationship before entering long-term agreements
- Recommend a ‘trust but verify’ clause as part of the agreement, to ensure honesty and transparency on all sides
- Potentially have a financial stake in the service provider to ensure its ‘good behavior’ – the Japanese ‘keiretsu’ model
- Have backup plans to handle emergencies and disasters – e.g. backup data centers helped financial institutions to resume operations within a week of 9/11
Essential Issues:

- Are we really in a new computing paradigm?
- If we are really in a new computing paradigm, what kinds of new technologies will be needed in the new paradigm?
- Are the existing technologies good enough for the new service computing?
- Which of the old technologies need to be drastically changed?
Observations on DoD Architecture Plans
DoD Architecture and UML

- DoD started several architecture initiatives such as DODAF (numerous views such as OV) and in some degree GIG can be considered as an architecture initiative.
- DoD is a heavy user of UML in specifying system behaviors (sequence diagrams, class diagrams, use cases, collaboration diagrams) and architecture.
  - Behavior diagrams
    - A type of diagram that depicts behavioral features of a system or business process.
    - This includes activity, state machine, and use case diagrams as well as the four interaction diagrams.
  - Interaction diagrams
    - A subset of behavior diagrams which emphasize object interactions.
    - This includes communication, interaction overview, sequence, and timing diagrams.
  - Structure diagrams
    - A type of diagram that depicts the elements of a specification that are irrespective of time.
    - This includes class, composite structure, component, deployment, object, and package diagrams.
These are excellent, however…

- These architecture initiatives provide some innovation for DoD, they help to ensure system quality in numerous aspects such as requirement specification, design, implementation, and testing.

- As an universal and common language, UML provides a common vocabulary between different stakeholders such as program managers (PMs), system engineers, QA, and system architect.

- And yet
  - They are not sufficient for modern 21st century agile warfighting, and
  - Leave significant unmet needs
Assurance Observations

- Recent trends in Network-Centric Warfare (NCW) have significant effect:
  - Service-Oriented Architecture (SOA)
    - Network Centric Enterprise Services (NCES), GIG Enterprise Services (GES), JBMC2, FCS, JBI, and Composable FORCEnet
    - Dynamic publication, runtime selection and discovery, dynamic composition
    - Distributed services and agents
  - These systems must be dynamic and keep on changing even at runtime.
  - These architectures are not static but dynamic and evolving
    - in other words, modern DoD systems need to respond to change at runtime and in real time.
- Key question
  - Are the existing approaches to architecture powerful enough to handle these dynamic structure and mechanisms, which are key to building systems for Network-Centric Warfare?
Rapid and adaptive acquisition and deployment for NCW
- Agile and adaptive acquisition (Income-Tax model for adaptive and incremental acquisition)
- Agile warfighting with *dynamic* changing tactics
- *Dynamic* system architecture composed at runtime
- *Dynamic* system reconfiguration or re-composition at runtime even during warfighting
- Rapid secure and dependable system engineering
- High assurance for C2 applications
- *Dynamic* and real-time system interoperability between two systems not knowing each other before

Key question
- *Is existing technology good enough to address the NCW issues identified above?*
SOA-based System has its own architecture

Contrary to popular belief, however, the system has three architectures:

- **The Application Architecture**. This application architecture will be built on top of an SOA.
- **The Service Architecture**. This is the commonly known SOA architecture.
- **The Component Architecture**. This is the sub-SOA architecture that describes the various elements that support the implementation of services.
Network Centric Enterprise Services (NCES)

Support real-time & near-real-time warrior needs and business users

Levels of Services above core level

Community-of-Interest (COI) Capabilities

Comms Backbone

Core Enterprise Services (CES)

Users

Dynamically Created COIs
Weapon Systems
Sensors
Finance
Etc.

C2 Intel
Logistics
Personnel

ESM Messaging
Mediation
Security/IA
User Asst

Discovery Collaboration Storage App
GIG Enterprise Services

SOADR Supports real-time & near-real-time warrior needs

DoD (Title 10)

Business Domains

Users

Warfighter Domains

DoD (Title 10)

Business Domains

DoD (Title 10)

Warfighter Domains

IC (Title 50)

National Intelligence Domain

Domain/COI Capabilities
Levels of Services Above Core Level

Technical Infrastructure Domain

Net-Centric Enterprise Services (NCLs)

Transformational Communications (TC) & Computing Infrastructure
A-Billion-Dollar Question

“Is UML good enough for the DoD Architecture Plan for the new network-centric service-oriented operation?”

Understanding that UML has served its time and served us well in the past in object-oriented computing.
Drawbacks of UML for Network Centricity

UML is an excellent methodology for OO programming

However

- It has evolved into an undifferentiated collection of models and techniques
- The different diagrams in UML are inter-connected but not fully integrated in the sense that transformations between models are difficult to automate
- Each individual model requires lots of effort to create and maintain.
  - If one model is changed, the other models must be updated, and often manually.
  - The effort to maintain consistency among the models is significant
Drawbacks of UML (contd.)

- UML was designed to facilitate the evolution from the procedure programming paradigm to the OO programming paradigm. Thus, it has weak support for:
  - Dynamic architecture with run-time composition / re-composition / reconfiguration
  - Dynamic Software verification & validation
  - Dynamic System engineering, especially in system where dependability, interoperability, safety, and security are essential
  - SOA and service-oriented computing
- Finally
  - UML does not address a number of key requirements of Network Centricity, e.g. interoperability and service integration
“By and large the dozen or so diagrams UML specifies aren't widely used. That's why we chose to err on the side of notation that's helpful and underpins metamodels. It's easy to relate such models to real development artifacts that we could sync precisely. And we've seen what happened when the industry tried to implement a standard that was not precisely specified, that has difficulty mapping between different development tasks, and for which no standard notation is defined. We didn't want to repeat that.” said Keith Short, Microsoft
Why UML was dropped

“IT's not an anti-UML story; it's a pro-model-driven development story. We think we're better off focusing on how to help developers do complicated things that could be made simple by the use of designers, and which require precise, focused languages with precise, well thought-out transforms between them.” Keith Short, an architect at Microsoft.
Lessons Learned

- Microsoft simply confirmed what we have thought for a long time, i.e.,
  - UML is essentially **old technology** for an **old problem** (OO development).
  - UML is no longer capable of solving the rapid SOA-based applications any more
  - UML simply gets too complicated, most of its models are no longer used in practice, and the effort needed to keep these models consistent with the code has been so high that it becomes counter-productive.
  - Microsoft kept **very few** models of UML, and essentially **abandoned most of UML** completely for **service-oriented computing**.
Lessons Learned (continued)

- It is necessary to develop new technology for the new problems, old technology will prove to be more counter-productive than productive.
- Rapid system development will be the key, and SOA will be an important enabler. Furthermore, OO is considered as old technology now, and only parts of it will survive in the next IT revolution.
A New Modeling Approach is Needed

- If UML is not the most suitable solution, then another modeling language will be needed.
- Furthermore, the new modeling language must support dynamic Service-Oriented System Engineering (SOSE)
  - Runtime reliability engineering
  - Runtime security analysis
  - Runtime safety analysis
  - Runtime verification and validation
  - Runtime system composition and re-composition
Dynamic System Engineering

- When a new application is being considered in a service-oriented environment, we need
  - Dynamic requirement specification by model composition
  - Dynamic specification and model analysis (performance analysis, simulation, model checking, completeness and consistency checking, reliability analysis, security analysis)
  - Dynamic design by composition and discovery
  - Runtime code generation and composition
  - Dynamic verification and validation

- We’ve not begun to investigate these technologies. Old technologies must be updated because they cannot be applied directly now.
Service-Oriented System Engineering

- **Dynamic** service-oriented requirement engineering (model-based, architecture-based, reuse-oriented, framework-oriented analysis, simulation-based analysis with formal analysis)
- **Dynamic** service-oriented architecture and design (enterprise computing, dynamic collaboration, system composition, dynamic system analysis)
- Service-oriented programming languages (model-based development, support automated code generation
- **Dynamic** service-oriented implementation (by dynamic discovery, composition, and model-based architecture, and automated code generation)
- **Dynamic** testing, verification, evaluation, simulation, reliability analysis of services
- **Dynamic** policy construction, verification, simulation, enforcement of security and other policies using formal policy languages
- **Dynamic** System maintenance and update will be via service re-composition and possibly architectural reconfiguration
From Object-Oriented Paradigm to Service-Oriented Paradigm

**OO Languages**
- Object-Oriented Concept & Architecture
  - Simula
  - Smalltalk
  - Objective C
  - C++
  - Java

**OO Modeling Languages & IDE**
- UML
- CORBA
- MS .Net
- JDK
- GCC

**OO Technology & Framework**
- OO system engineering (OOSE)
- OO testing
- OO maintenance
- OO application frameworks
- OO databases (OODB)
- OO lifecycle (XP? MDA?)

**SO Standards**
- Service-Oriented Concept & Architecture
  - XML
  - UDDI
  - ebXML
  - WSDL
  - SOAP
  - OWL

**SO Modeling Languages & IDE**
- BPEL
- WSFL
- XLANG
- MS. Net
- WebSphere

**SO Technology & Framework**
- SO System Engineering (SOSE?)
- SO testing (WebStrar?)
- SO maintenance (re-composition?)
- SO frameworks (FERA? SOI?)
- SO databases
  - (Ontology DB, SODB?)
- SO Lifecycle
  - (MDA, [re-]composition)
Service-Oriented System Engineering

DSCA for Reasoning and Control

- Testing
- C&C
  - Model checking
- Simulation
  - Reliability modeling

Deployment

Policy enforcement

Application

Data collection
Data mining
Conclusion

Computing will become a utility per the GIG and software a service. There will be a profound change in assurance for software engineering and information technology disciplines, industry, government, business, and economics.

The research and development of assurance, security, trust and dependability standards for SOA (service oriented architecture) and SOC (service oriented computing) is a critical component to their effective functionality and ultimate success.
New Paradigm of System Engineering

- Yes, we do need a new paradigm for dynamic system engineering.
- We can no longer use the old paradigm of system engineering in the new service-oriented system engineering.
- Welcome to the new world of dynamic integration and high-assurance service oriented system engineering in the near future.
Thank you for your Attention!

Ray (New System Engineering Paradigm Evangelist) Paul