Journal of Information Systems Engineering and Management
2025, 10(62s)
e-ISSN: 2468-4376
https://jisem-journal.com/

Research Article

The Enduring Relevance of SOAP in Enterprise Integration Architecture

Venugopal Reddy Depa
CRH Americas Materials Inc. USA

ARTICLE INFO

Received: 06 Oct 2025

Revised: 25 Nov 2025

Accepted: 05 Dec 2025

ABSTRACT

The Simple Object Access Protocol maintains a strong position in enterprise integration despite REST's widespread adoption. Mission-critical systems in financial services, healthcare, and government rely on SOAP's comprehensive security frameworks. WS-Security leverages features such as message-level encryption and digital signatures that meet regulatory requirements and, therefore, are regarded as better security than transport-layer security on its own. In addition, WS-ReliableMessaging and WS-AtomicTransaction provide assurances regarding message delivery and coordination of distributed transactions. These features become more necessary when situations arise where data loss or any difference in state cannot be tolerated. Web Services Description Language enables formal contracts for centralized governance in complex organizations. Strongly-typed contracts prevent integration errors and enable automated code generation. Legacy systems in mainframes and ERP platforms have deep SOAP infrastructure investments. The choice between SOAP and REST depends on Quality of Service requirements, not technology trends. Consumer applications benefit from REST's simplicity. Core enterprise systems need the reliability and security that SOAP provides through decades of standardization.

Keywords: SOAP Protocol, Enterprise Integration Architecture, Ws-Security Standards, Distributed Transactions, Service-Oriented Governance

I. Introduction

SOAP versus REST debates persist in enterprise technology circles. Developers typically prefer REST for its simplicity. But this view misses critical enterprise needs. SOAP handles mission-critical systems that demand strict governance and reliability.

REST became popular through its lightweight design. It uses standard HTTP methods. Implementation is straightforward for web services. The protocol works well for consumer-facing applications where speed matters. REST payloads use compact JSON formats. Mobile applications and public APIs benefit most from this approach. Modern developers find REST intuitive. The learning curve is gentler than SOAP's XML-heavy structure.

SOAP delivers capabilities that REST cannot match easily. The protocol was built for enterprise distributed computing from the start. Security, reliability, and transaction management are core features, not add-ons. Large organizations invested heavily in SOAP infrastructure over twenty years. Legacy systems depend on these established patterns. The comparison reveals fundamental design philosophy differences [1].

Enterprise environments face unique challenges. Regulatory compliance requires provable message-level security. Financial transactions need guaranteed delivery across multiple systems. Healthcare data exchanges must maintain audit trails and non-repudiation. SOAP's native capabilities address these requirements directly. Medical institutions need HIPAA-compliant channels for patient information. Message-level security protects data through transmission and storage [2].

Journal of Information Systems Engineering and Management
2025, 10(62s)
e-ISSN: 2468-4376
https://jisem-journal.com/

Research Article

The WS-* standards family gives SOAP architectural maturity. WS-Security handles message-level encryption and digital signatures. WS-ReliableMessaging guarantees delivery and sequencing. WS-AtomicTransaction coordinates distributed transactions across different systems. These standards create a comprehensive integration framework. SOAP's extensibility lets organizations add capabilities without infrastructure redesign.

Government agencies and banks mandate SOAP for critical integrations. Regulatory frameworks reference WS-Security explicitly. Migrating to REST means recreating complex functionality at the application level. This introduces risk and raises development costs. SOAP's proven track record justifies continued use. Banks process billions daily through SOAP systems. These platforms demonstrate reliability under regulatory scrutiny.

This article examines SOAP's persistence despite REST's popularity. Three areas show SOAP's distinct advantages. WS-Security delivers compliance-grade security mechanisms. Transactional protocols ensure data integrity in complex workflows. WSDL contracts enable governance in large-scale environments. Technology choice depends on operational requirements, not trends. Each protocol serves specific purposes in modern integration.

II. WS-Security Standards and Compliance

WS-Security forms the cornerstone of SOAP's enterprise value. The standard defines message-level security within SOAP envelopes. This addresses requirements that transport-layer security cannot satisfy. Organizations under regulatory oversight benefit particularly from these mechanisms. The framework provides end-to-end security beyond network transmission.

Message-level encryption protects data throughout its lifecycle. HTTPS only secures point-to-point transmission. WS-Security encrypts the payload itself. Messages stay protected in intermediate queues and databases. This model aligns with HIPAA and PCI-DSS compliance frameworks. Financial institutions processing sensitive transactions require this protection level. Healthcare providers must keep patient data encrypted during all processing. The medical histories of patients are extremely confidential.

By using digital signatures, the processes of user authentication and non-repudiation become possible. WS-Security supports XML Signature standards for cryptographic signing. Recipients verify sender identity and detect tampering. This creates immutable audit trails for regulated industries. Healthcare organizations prove data provenance under federal regulations. Electronic health records need mechanisms to prevent unauthorized modifications. Non-repudiation ensures senders cannot deny transmitting messages [3]. This legal certainty matters for medical documentation and insurance claims.

Furthermore, token-based authentication works seamlessly with enterprise, identity-based systems. WS-Security works with standards such as SAML, Kerberos, and X.509 certificates. By implementing authentication policies, organizations are enabling a secure environment across distributed systems. This enables single sign-on architectures in complex environments. Legacy applications join modern identity federation without extensive modifications. SAML providers facilitate enterprise SSO across heterogeneous platforms [4]. Employees access multiple systems using a single credentials. Security improves while user experience gets better.

The specification allows selective message part encryption. Sensitive elements get encrypted while metadata stays accessible for routing. This granular control optimizes performance while maintaining security. Enterprise Service Buses route messages without decrypting entire payloads. The approach balances security with operational efficiency. Routing headers remain plaintext for quick processing. Only business payloads receive encryption.

Compliance auditors recognize WS-Security as an industry standard. Banking and healthcare regulatory frameworks reference these protocols explicitly. Demonstrating compliance becomes straightforward with established standards. REST implementations must recreate capabilities at the

Journal of Information Systems Engineering and Management
2025, 10(62s)
e-ISSN: 2468-4376
https://jisem-journal.com/

Research Article

application layer. Custom security faces greater scrutiny during audits. Auditors familiar with WS-Security verify implementation quickly. This reduces compliance verification time and cost.

Government systems rely heavily on WS-Security for interagency communications. Federal mandates require specific data exchange security protocols. The Department of Defense standardized on WS-Security years ago. Their infrastructure investments reflect long-term protocol commitments. REST APIs struggle to meet these requirements without significant custom development. Classified systems demand proven security frameworks. WS-Security provides the necessary guarantees for national security applications.

WS-SecurityPolicy framework enables declarative security configurations. Organizations define security requirements in policy documents, not code. This separation simplifies governance and reduces errors. Automated tools validate implementations against policy requirements. Enterprise architects enforce security standards across projects using these policies. Policy files specify encryption algorithms, signature methods, and token types. Developers implement services that automatically comply with organizational policies.

Security Component	Enterprise Implementation	Regulatory Alignment
Message-Level Encryption	End-to-end payload protection across queues and databases	HIPAA and PCI-DSS compliance frameworks
Digital Signatures	XML Signature standards for cryptographic authentication	Federal healthcare data provenance requirements
Token-Based Authentication	SAML, Kerberos, and X.509 certificate integration	Enterprise single sign-on architectures
WS-SecurityPolicy Framework	Declarative security configurations in policy documents	Automated organizational standard enforcement

Table 1: WS-Security Framework Components and Compliance Applications [2][3]

III. Transactional Integrity and Reliability

WS-ReliableMessaging handles guaranteed delivery for enterprise systems. The protocol ensures messages reach destinations exactly once in correct order. This is fundamental for financial transactions and supply chain operations. REST's stateless nature makes similar guarantees significantly more complex. Banking systems cannot tolerate lost or duplicate transactions. Every message must be accounted for.

The protocol establishes reliable sessions between communicating parties. Each message gets a unique sequence number within the session context. Receiving systems acknowledge successful message delivery. Without acknowledgement, senders automatically retransmit messages. This mechanism operates independently of the transport layer. Network interruptions don't cause lost messages. The protocol handles retransmission without application intervention.

Ordered delivery prevents race conditions in multi-step processes. Banking systems process multiple transactions per account. Operations must execute in the exact initiation sequence. WS-ReliableMessaging guarantees ordering without application-level intervention. The protocol handles

sequencing at the infrastructure level. Deposits must be processed before subsequent withdrawals. Out-of-order processing creates incorrect account balances.

Message buffering handles temporary outages gracefully. Messages queue automatically when receiving systems become unavailable. The protocol retries delivery per configurable policies. This resilience is essential for systems that cannot afford data loss. Manufacturing systems coordinating production lines depend on this reliability. Financial institutions implement strict transaction audit controls. Compliance mandates complete transaction histories without gaps. Guaranteed delivery ensures regulatory reporting accuracy [5].

WS-AtomicTransaction extends SOAP with distributed transaction capabilities. The protocol implements two-phase commit semantics across services. All participating systems must complete successfully, or everything rolls back. This all-or-nothing guarantee prevents inconsistent distributed database state. Financial transfer systems require this transactional integrity level. Money must move atomically between accounts. Partial completion creates accounting discrepancies.

The coordination protocol manages complex multi-enterprise workflows. Supply chain scenarios involve order processing, inventory management, and shipping systems. WS-AtomicTransaction ensures atomic updates across all three systems. Any component failure triggers a clean rollback of the entire operation. This prevents inventory decrements without shipping initiation. Multi-party transactions require sophisticated coordination mechanisms. The protocol provides standardized coordination without custom logic.

Transaction context propagates automatically through message flow. Applications participate in distributed transactions without explicit coordination code. The protocol handles complex two-phase commit state management. This abstraction reduces development complexity and eliminates common errors. Developers focus on business logic instead of transaction coordination. Infrastructure manages, prepares, and commits phases transparently. Two-phase commit protocols in microservices demonstrate the continued relevance of distributed coordination [6].

Enterprise Service Buses orchestrate complex integrations using these protocols. Long-running business processes span multiple systems over extended timeframes. WS-ReliableMessaging ensures no execution steps are lost. WS-AtomicTransaction guarantees consistency when system updates occur. These capabilities enable sophisticated workflows that REST architectures struggle to implement reliably. Order fulfillment involves inventory, payment, and shipping systems. All components must complete successfully for valid orders.

Compensating transactions handle scenarios where full atomicity is impractical. WS-BusinessActivity provides mechanisms for long-running compensating actions. This pattern suits business processes spanning hours or days. The protocol coordinates compensation logic when transactions must unwind after partial completion. Travel booking systems use compensating transactions extensively. Flight and hotel reservations are reversed if payment fails. Event meshes in distributed systems require duplicate prevention alongside delivery guarantees [7]. Modern integration platforms build on these foundational SOAP concepts.

Journal of Information Systems Engineering and Management
2025, 10(62s)
e-ISSN: 2468-4376
https://jisem-journal.com/

Research Article

Protocol Feature	Operational Mechanism	Business Application
Reliable Sessions	Unique sequence numbers with acknowledgment retry	Banking transaction processing and account operations
Ordered Delivery	Infrastructure-level message sequencing	Multi-step financial workflows preventing race conditions
Message Buffering	Automatic queuing during system unavailability	Manufacturing production line coordination
Two-Phase Commit	Distributed atomic updates with rollback capability	Supply chain inventory and shipping synchronization

Table 2: Transactional Protocol Capabilities in Distributed Systems [7][9]

IV. WSDL Contracts and Enterprise Governance

WSDL provides formal service contracts enabling centralized governance. The specification defines service interfaces using strongly-typed XML schemas. Developers favoring REST flexibility often criticize this formality. Enterprise environments benefit significantly from WSDL's rigorous contract-first approach. Formal contracts prevent integration failures before deployment.

Strongly-typed contracts prevent production integration errors. WSDL explicitly defines input parameters, output types, and fault conditions. Development tools automatically validate messages against contracts. Build processes detect incompatible service interface changes. Early error detection reduces costly production failures. Integration tests verify contract compliance during continuous integration. Type mismatches are caught immediately, not in production.

With automated code generation, the process of developing a heterogeneous environment can be shortened considerably. WSDL equips tools with the means to auto-generate server skeletons and client stubs, thus developers can use any programming language they like, with less XML and more abstraction. This abstraction raises the developers' output and lessens the number of manual coding mistakes. Organizations standardize WSDL tooling across multiple technology stacks. Java, .NET, and Python developers all work from identical service contracts. Generated code ensures consistent cross-platform implementation.

Version management becomes systematic through WSDL namespacing. Service providers publish new contract versions with distinct namespaces. Consumers migrate to new versions on controlled timelines. Multiple versions coexist during transition periods without conflict. This governance model suits large organizations where team coordination is challenging. Legacy consumers continue using old versions while new applications adopt updates. The migration path is gradual and controllable.

Enterprise Service Buses rely on WSDL for service registry and discovery. Central repositories catalog available services and their contracts. Integration architects query registries when designing new workflows. Formal contracts enable automated integration pattern validation. Centralized governance reduces duplicate implementations and ensures consistency. Service catalogs provide searchable organizational capability inventories. Teams discover existing services before building redundant functionality.

Journal of Information Systems Engineering and Management
2025, 10(62s)
e-ISSN: 2468-4376
https://jisem-journal.com/

Research Article

Legacy system integration benefits particularly from WSDL's structured approach. Mainframe applications and SAP ERP systems expose SOAP interfaces defined by WSDL. These systems represent decades of business logic and data. Existing SOAP interfaces cannot be replaced easily without a massive investment. WSDL ensures new applications integrate reliably with these systems. Core banking platforms and insurance policy systems rely on SOAP connectivity. Modern applications must interoperate with these established systems.

Contract validation at the ESB layer enforces organizational standards. Messages must conform to published contracts before backend system routing. This validation prevents malformed requests from reaching critical infrastructure. The approach provides a control point for security and compliance checks. REST's flexibility makes implementing similar validation more complex. Schema validation catches data quality issues at integration boundaries. Backend systems receive only properly formatted messages.

Schema evolution strategies maintain backward compatibility over time. WSDL supports optional elements and schema extensibility points. Service providers add new capabilities without breaking existing consumers. This evolutionary approach suits enterprise systems with long operational lifetimes. Organizations maintain service compatibility over decades, not just years. Financial institutions operate systems installed in the previous century. These platforms must integrate with modern applications through stable contracts. Financial services require secure communication protocols ensuring message integrity [8]. Banking networks process trillions through SOAP-based infrastructure.

Governance Aspect	Technical Implementation	Organizational Benefit
Strongly-Typed Contracts	Explicit parameter and output type definitions	Early detection of integration errors during builds
Automated Code Generation	Client stub and server skeleton creation	Cross-platform consistency in Java, .NET, and Python
Version Management	Distinct namespace publication for contract versions	Controlled migration with multiple version coexistence
Schema Evolution	Optional elements and extensibility points	Backward compatibility over decades of operation

Table 3: WSDL Contract Governance in Enterprise Integration [8]

V. Strategic Considerations

SOAP's persistence reflects strategic decisions, not technical inertia. Organizations evaluate integration technologies based on operational requirements and risk tolerance. SOAP's comprehensive standards address specific needs that REST cannot easily satisfy. Protocol choice depends fundamentally on the required Quality of Service attributes. Technology decisions should be in line with business objectives and regulatory requirements.

Journal of Information Systems Engineering and Management
2025, 10(62s)
e-ISSN: 2468-4376
https://jisem-journal.com/

Research Article

New consumer-facing applications are mostly using REST for their simplicity and performance. Public APIs benefit from REST's intuitive resource model and stateless operations. Mobile applications require lightweight protocols with minimal overhead. These use cases align perfectly with REST's design philosophy. SOAP introduces unnecessary complexity in these scenarios. Web developers find REST more approachable and faster to implement. The REST tools and frameworks ecosystem is extensive.

Core enterprise systems face dramatically different requirements than consumer applications. Financial transaction processing demands guaranteed delivery and transactional integrity. Healthcare data exchange requires provable security and audit trails. Supply chain coordination needs reliable messaging across organizational boundaries. Mission-critical systems justify SOAP's additional complexity through enhanced guarantees. Failure cost in these domains far exceeds implementation complexity.

Institutional knowledge represents a significant persistence factor. Large organizations employ specialists with deep SOAP and WS-* expertise. ESB platforms embody years of configuration and integration patterns. Migration to REST architectures requires staff retraining and infrastructure rebuilding. The business case for such transitions often fails the cost-benefit analysis. Experienced integration architects understand SOAP nuances and troubleshooting techniques. This expertise cannot be discarded without substantial retraining investment.

Regulatory compliance frameworks evolve slowly and reference established standards. Financial regulators understand WS-Security and recognize its implementations during audits. Healthcare compliance officers accept SOAP-based integrations meeting federal requirements. Government procurement processes specify WS-* protocols in technical requirements. Changing these institutional frameworks would require years of industry coordination. Regulatory bodies trust proven technologies over newer alternatives. SOAP's track record in regulated industries provides confidence.

Hybrid architectures represent the pragmatic approach for most enterprises. REST APIs serve external consumers and modern web applications. SOAP integrations continue for core system interconnections and B2B transactions. This dual-protocol strategy optimizes each integration point for specific requirements. Organizations leverage both architectural styles' strengths appropriately. Payment gateways expose REST APIs to merchants while using SOAP internally. Integration layers translate between protocols as needed.

SOAP's future lies in maintenance rather than expansion. Few organizations initiate new SOAP projects except when integration requirements mandate it. Existing SOAP infrastructure will remain operational for decades. The installed base of mission-critical systems ensures continued relevance. Organizations will maintain SOAP expertise and tooling supporting these essential integrations. Cloud platforms in distributed environments increasingly use multi-microservice architectures with distributed transaction protocols [9]. These modern implementations build on the concepts that SOAP pioneered.

Technology decisions should be based on business situations instead of industry trends. SOAP is equipped with the features that are required for cases where security, reliability, and transactional integrity are the most important. The protocol's age and standardization are the advantages that very cautious enterprise environments get. REST excels in scenarios prioritizing simplicity, performance, and rapid development. Understanding when each protocol delivers optimal value enables sound architectural decisions. SOAP's enduring relevance demonstrates that enterprise architecture requires diverse technological approaches matched to specific operational demands.

Compensating transactions represent an important pattern for long-running business processes. Full ACID transactions become impractical in certain scenarios. Compensating actions provide eventual consistency. The pattern allows systems to undo previously completed work when subsequent steps fail. Travel booking and order processing systems commonly employ compensating transactions [10]. Each step records compensation logic that can reverse its effects. This approach balances consistency requirements with system autonomy.

Journal of Information Systems Engineering and Management
2025, 10(62s)
e-ISSN: 2468-4376
https://jisem-journal.com/

Research Article

Evaluation Factor	SOAP Optimization	REST Optimization
Target Application Type	Core enterprise systems and B2B transactions	Consumer-facing APIs and mobile applications
Security Requirements	Message-level encryption with non-repudiation	Transport-layer security for public endpoints
Transaction Model	Distributed two-phase commit with compensation	Stateless operations with eventual consistency
Institutional Considerations	Regulatory compliance and legacy system integration	Rapid development and developer experience

Table 4: Protocol Selection Criteria for Integration Architecture [1][10]

Conclusion

One of the important benefits of SOAP, going against the general narrative, is that its features are still needed, and thus it can't just be replaced by REST. The decision-making process of organizations on technology should be based on their operational requirements rather than popularity contests. Some of the very vital capabilities that are critical for the mission-critical systems and cannot be easily done by REST without considerable custom development are delivered by SOAP.

WS-Security offers security measures at the compliance level that are not only secure but also recognized by regulatory authorities across different industries. Banks rely on these standard protocols to exhibit their compliance with stringent data protection directives.

Healthcare providers are taking advantage of the security features in message-level encryption and digital signatures to be in line with the federal rules that require keeping the user access log.

Government agencies are setting the standards for implementation of WS-* protocols in the specification sections that deal with communication between different agencies and the exchange of classified data. These institutional requirements ensure SOAP maintains its enterprise architecture position.

Transactional integrity protocols deliver guarantees essential for distributed business processes spanning multiple systems and organizational boundaries. WS-ReliableMessaging prevents data loss where guaranteed delivery is non-negotiable. WS-AtomicTransaction coordinates distributed transactions with two-phase commit semantics, maintaining consistency across heterogeneous databases. Supply chain operations coordinating inventory management, order processing, and logistics systems require this transactional control level. Manufacturing environments where production line coordination demands exact sequencing cannot tolerate eventual consistency models common in REST architectures. Banking systems processing millions of daily transactions depend on SOAP's reliability guarantees.

WSDL's formal contract model facilitates governance in large organizations managing hundreds of services across diverse technology stacks. Strongly-typed contracts catch integration errors during development rather than production deployment. Automated tooling generates client code and validates messages against published schemas without manual intervention. Enterprise Service Buses leverage WSDL contracts, maintaining service registries and enforcing organizational integration standards. Legacy systems representing decades of business logic investment expose SOAP interfaces that newer applications must consume reliably. Migrating these core systems to REST-based

architectures would require massive investment with questionable return, given their proven stability and regulatory acceptance.

Institutional knowledge surrounding SOAP implementation and troubleshooting represents significant organizational capital. This cannot be discarded without substantial retraining costs. The pragmatic architectural strategy employs both protocols based on specific integration requirements. REST serves external consumers, mobile applications, and scenarios prioritizing developer experience and rapid iteration. SOAP continues handling core system integration where reliability, security, and transactional guarantees justify additional complexity. This hybrid model optimizes technology selection for each use case rather than forcing uniform single-protocol adoption. Organizations recognize different integration scenarios and demand different technological solutions.

SOAP's persistence demonstrates that enterprise architecture requires diverse technological approaches matched to operational demands rather than trend adherence. Modern distributed systems incorporate lessons learned from decades of SOAP deployment. Microservice architectures face similar challenges around distributed transactions and reliable messaging. Cloud-native platforms have increasingly implemented SOAP standardized patterns years ago. The fundamental problems SOAP addresses remain relevant regardless of underlying technology trends. Understanding when to apply SOAP versus REST enables architects to make informed decisions serving business objectives effectively.

References

1. Ankur Ghosh, et al., "SOAP vs ReST API: A Comparative Analysis," DBSync, 2024. [Online]. Available: https://www.mydbsync.com/blogs/soap-vs-rest-api-a-comparative-analysis
2. Anand K, et al., "HIPAA Compliant Text Messaging For Healthcare Providers," LeadSquared, 2025. [Online]. Available: https://www.leadsquared.com/us/industries/healthcare/hipaa-compliant-texting-for-medical-professionals/
3. Mohamed Sharaf, et al., "Non-repudiation and privacy-preserving sharing of electronic health records," Cogent Engineering, 2022. [Online]. Available: https://www.tandfonline.com/doi/full/10.1080/23311916.2022.2034374#abstract
4. Logto, "SAML identity provider for enterprise SSO authentication." [Online]. Available: https://logto.io/products/saml-app
5. Dana Lawrence et al., "Compliance audits in financial services: Techniques and tools for success," Wolters Kluwer, 2024. [Online]. Available: https://www.wolterskluwer.com/en/expert-insights/compliance-audits-financial-services-techniques-tools-for-success
6. Holistic Security, "Applying WS-Security Policy Framework to Webservices (WSO2 ESB) and Dataservices (WSO2 DSS)," 2015. [Online]. Available: https://holisticsecurity.wordpress.com/2015/04/19/applying-ws-security-policy-framework-to-wso2esb-wso2dss/
7. Zoe McCarthy, et al., "Secure Communication Protocols for Financial Institutions," Cyber Finance Guard, 2024. [Online]. Available: https://www.cyberfinanceguard.com/secure-communication-protocols-financial-institutions/
8. Tom Fairbairn, "How to Guarantee Delivery While Preventing Duplicates in your Event Mesh," Solace, 2022. [Online]. Available: https://solace.com/blog/guarantee-delivery-prevent-duplicates-event-mesh/
9. Pan Fan, et al., "2PC*: a distributed transaction concurrency control protocol of multi-microservice based on cloud computing platform," Journal of Cloud Computing, 2020. [Online]. Available: https://journalofcloudcomputing.springeropen.com/articles/10.1186/s13677-020-00183-w
10. Microsoft AZURE, "Compensating Transaction pattern." [Online]. Available: https://learn.microsoft.com/en-us/azure/architecture/patterns/compensating-transaction