Model Driven Architecture based Agile Modelled Layered Security Architecture for Web Services Extended to Cloud, Big Data and IOT

Dr.D.Shravani

Rayalaseema University, Kurnool, A.P, India

Abstract *This research paper consists of Model Driven Architecture based Agile Security Architecture for Web Services extended to Cloud, Big Data and IOT.*

Keywords - *Security Engineering, Security Architectures, Web Services, Cloud Computing, Big Data, IOT*

I. INTRODUCTION

In this paper, our methodology of Security design of Model Driven Architecture based Agile Modeled Layered Security Architecture is given, for Web Services Security Design with appropriate Web Services Case Studies design using Class Diagrams and Sequence Diagrams design.

3.1 MDA BASED AGILE MODELED LAYERED SECURITY ARCHITECTURE:

Because of several vulnerabilities in software products and high amount of damage caused by them, software developers are enforced to produce more secure systems. Software grows up through its life cycle, so software development methodologies should pay special attention to security aspects of the product. Using this approach method engineer of the research implementation can enhance their agile software development process with security features to increase product's trustworthiness.

A secure system is one that is protected against specific undesired outcomes. Delivering a secure system, and particularly, a secure web application, is not easy. Integrating general-purpose information systems development systems with security development activities could be a useful means to support these difficulties. Agile processes, such as Extreme programming, are of increasing interest in software development. Most significantly for web applications, agile processes encourage and embrace requirements change, which is a desirable characteristic for web application development. Agile methods include Feature Driven Development (FDD) and mature security methods, namely risk analysis, and integrate them to address the development of secure web applications. This approach key features includes: a process capable of dealing with the key challenges of applications development like decreasing life-cycle times and frequently changing requirements and an iterative approach to risk analysis that integrates security design throughout the development process.

Class diagram Design for MDA authentication using Executable UML

The Figure 3.1 (Class Diagram Design) provides the MDA authentication using Executable UML. User enters username and password to access information. Authenticator checks the username and associated password to know whether the user is really he or she claims to be. Authenticator allows the user depending on the check result. Authorizer checks this user type (for example, administrator) and associated access rights. Authorizer restricts the user to access the information. The entered username and password by any user will be transformed in an encrypted format so that any other user who is correctly logged in cannot recognize it. Therefore security class provides a key and algorithm used to encrypt the data. Its implementation Sequence Diagram works as: User enters the username and password which are encrypted and transferred to authenticator to verify correctness. Then access rights for specified user are checked to allow for accessing of information.

Class diagram Design of Agile Methodologies with Security Activities

The Figure 3.2. (Class diagram design) the Agile Methodologies with Security Activities. Agile methodologies for security activities include applying agility measurement and applying an efficient agility reduction tolerance. First Security Activities are extracted from existing processes and guidelines from SecurityActivity class. The activities are named as "Security Activities" and these are used as basis for next steps. Classification of activities is done by understanding them in life cycle. Agility degree of activities is defined to measure their nimbleness. Agility degree for each activity is defined as its agile behavior. It represents level of activity's compatibility with agile methodologies. Grades between 0 and 5 are assigned in agility degree vector (ADVect).Then integration issues of agile and security activities are handled. By analyzing agile methodologies and identifying their

core engine activities integration is done. Activity integration compatibility matrix (AICM) is generated with binary values. An algorithm to integrate security activities with organization's agile process is introduced in Algorithm class. Follows all steps activity by activity recursively. Finally agility reduction tolerance parameter and its optimization value are discussed in ART class. Tuning ART parameter is SMET's (Secure Method Engineer Team) art to keep a balance between security and weight of the software development process.

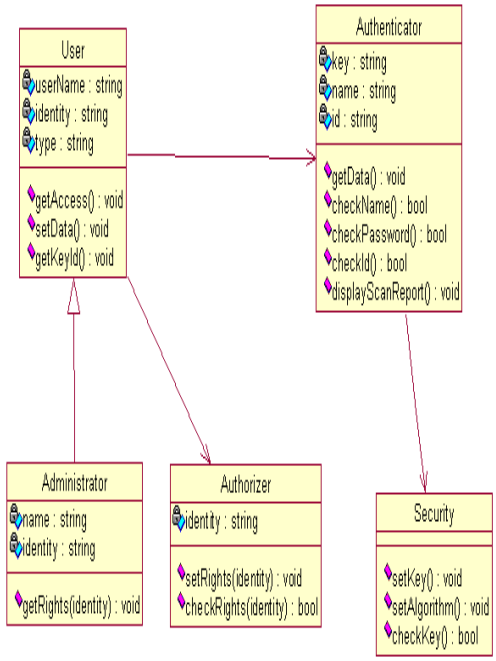


Figure 3.1 Class diagram Design for MDA authentication using Executable UML

Sequence diagram of Methodology for Predecessor activities of an Agile Methodology. The Figure 3.3. provides predecessor activities of an Agile Methodology as an Sequence diagram methodology design.

Sequence diagram of Methodology for Agile Implementation. The Figure 3.4 provides implementation details of Agile Security Implementation as a Sequence diagram methodology design.

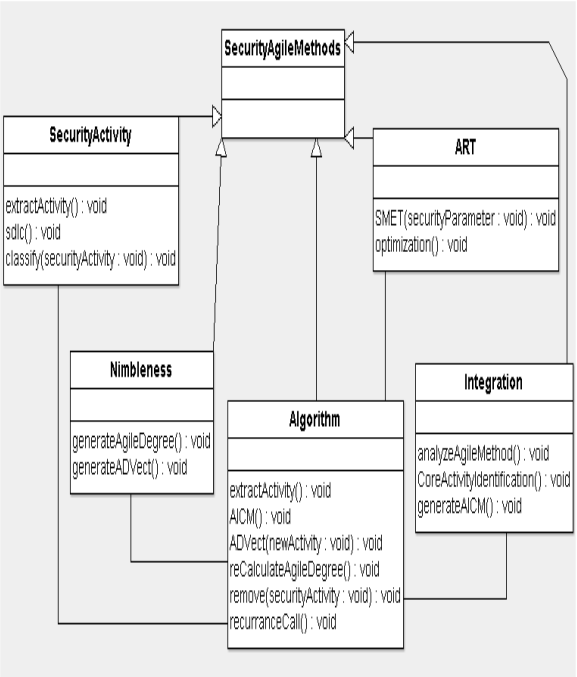


Figure 3.2 Class Diagram Design of Agile Methodologies with Security Activities.

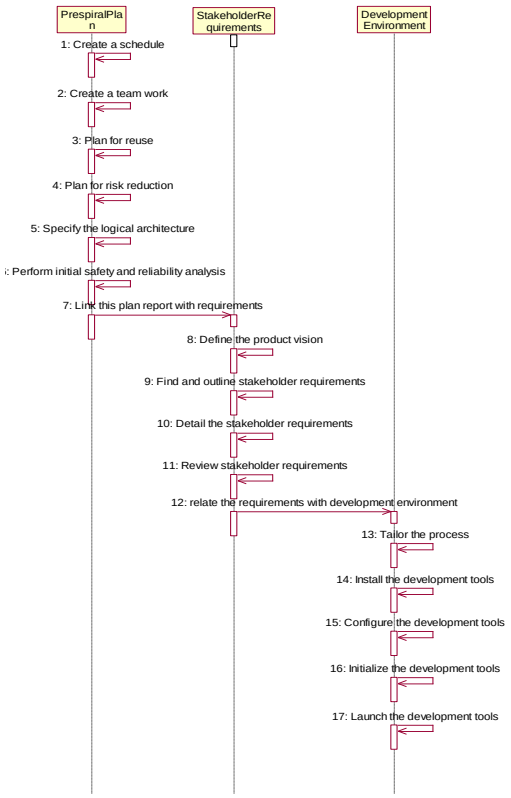


Figure 3.3: Sequence diagram for Predecessor activities of an Agile Methodology.

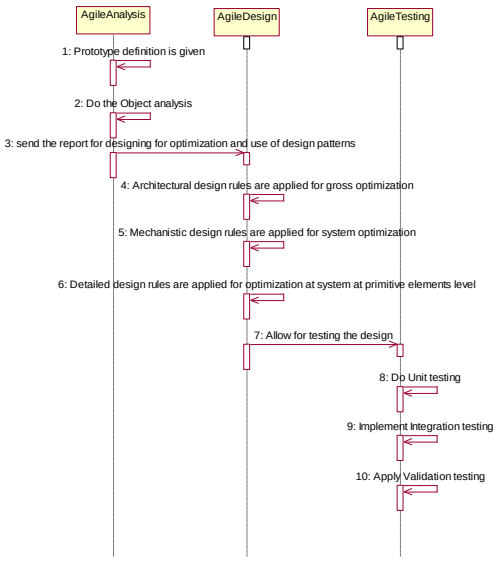


Figure 3.4: Sequence diagram for Agile Implementation

Agile Security Patterns

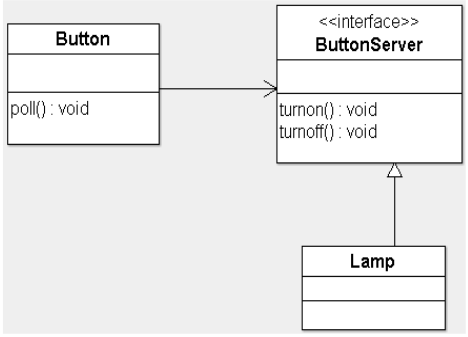
The Dependency-Inversion Principal

Dependency Inversion Policy: Dependency inversion can be applied wherever one class sends a message to another. For example, the case of the Button object and the Lamp object [Bruce Powel Douglass].

The Button object senses the external environment. On receiving the Poll message, the Button object determines whether a user has "pressed" it. It doesn't matter what the sensing mechanism is. It could be a button icon on a GUI, a physical button being pressed by a human finger, or even a motion detector in a home security system. The Button object detects that a user has either activated or deactivated it. The Lamp object affects the external environment. On receiving a TurnOn message, the Lamp object illuminates a light of some kind. On receiving a TurnOff message, it extinguishes that light. The physical mechanism is unimportant. It could be an LED on a computer console, a mercury vapor lamp in a parking lot, or even the laser in a laser printer. The Button object receives Poll message, determines whether the button has been pressed, and then simply sends the TurnOn or TurnOff message to the Lamp.

The Button class depends directly on the Lamp class. This dependency implies that Button will be affected by changes to Lamp. This violates DIP. The high level policy of the application has not been separated from the low level implementation. High-level policy is the abstraction that underlies the application, the truths that do not vary when the details are changed. It is the system inside the system, it is the metaphor. Here, the Button now holds an association to a ButtonServer, which provides the interfaces that Button can use to turn something on or off. Lamp implements the ButtonServer interface. Thus, Lamp is now doing the depending rather than being depended on. This allows a Button to control any device that is willing to implement the ButtonServer interface. This provides a great deal of flexibility. (The Figure 3.5 provides the class diagram for this principle.)

Figure 3.5: Class Diagram for Dependency Inversion Principal.



Interface Segregation Principle

Interface Pollution: Consider a security system in which Door objects can be locked and unlocked and know whether they are open or closed. This Door is coded as an interface so that clients can use objects that conform to the Door interface without having to depend on particular implementations of Door. Let us consider a TimedDoor which needs to sound an alarm when the door has been left open for too long. In order to do this, the TimedDoor object communicates with another object called a Timer. When an object wishes to be informed about a timeout, it calls the Register function of the Timer. Force Door, and therefore TimedDoor, to inherit from TimerClient. This ensures that TimerClient can register itself with the Timer and receive the TimeOut message. The problem with this solution is that the Door class now depends on TimerClient. Not all varieties of Door need timing. The applications that use those derivatives will have to import the definition of the TimerClient class, even though it is not used. This causes complexity and redundancy. Separate Client Means Separate Interfaces: Door and TimerClient represent interfaces that are used by completely different clients. Timer uses TimerClient, and classes that manipulate doors use Door. Since the clients are separate, the interfaces should be separate too, because clients exert forces on their server interfaces. Include a unique timeOutId code in each timeout registration and repeat that code in the TimeOut call to the TimerClient. (The Figure 3.6 provides the class diagram for this principle.)

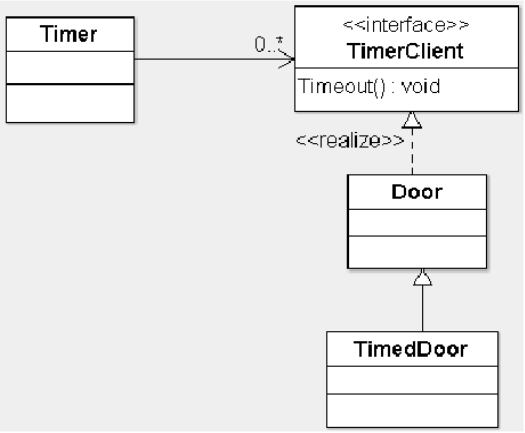


Figure 3.6. Class Diagram for Interface Pollution

Separation through Delegation: One solution to ISP is to create an object that derives from TimerClient and delegates to the TimedDoor. When it wants to register a timeout request with the Timer, the TimedDoor creates a DoorTimerAdapter and registers it with the Timer. When the Timer sends the TimeOut message to the DoorTimerAdapter, the DoorTimerAdapter delegates the message back to the TimedDoor. This solution conforms to ISP and prevents the coupling of Door clients to Timer. Even if the change to Timer were to be made, none of the users of Door would be affected. Moreover, TimedDoor does not have to have the exact same interface as TimerClient. The DoorTimerAdapter can translate the TimerClient interface into the TimedDoor interface. Thus this is a very general purpose solution. But in this solution, the delegation requires a very small amount of runtime and memory. (The Figure 3.7 provides the class diagram for this principle.)

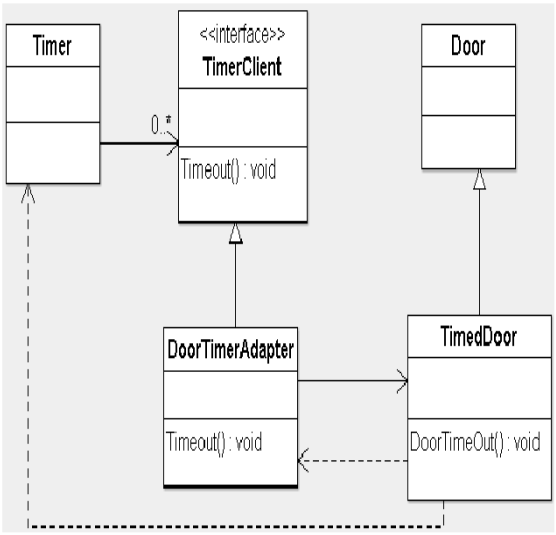
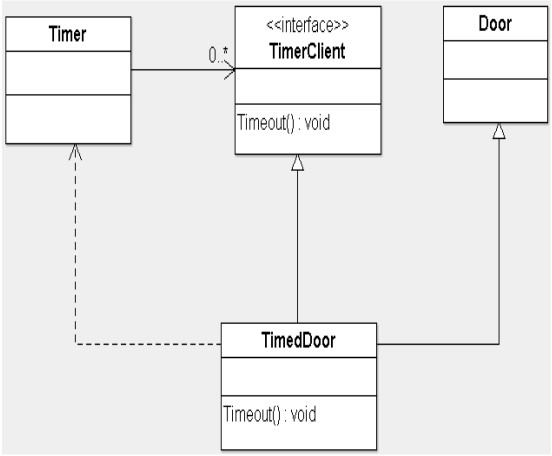


Figure 3.7 Class Diagram for Separation through delegation.

Separation through Multiple Inheritances: TimedDoor inherits from both Door and TimerClient. Although clients of both base classes can make use of TimedDoor, neither depends on the TimedDoor class. Thus, they use the same object through separate interfaces. (The Figure 3.8 provides the class diagram for this principle.)



. Figure 3.8.Class Diagram for separation through Multiple Inheritance.

3.2 MDA BASED AGILE MODELED LAYERED SECURITY DESIGN FOR WEB SERVICES:

3.2.1 Architecting Secure Web Services Architecture

Service-Oriented Architectures (SOA) represents a new evolving model for building distributed applications. Services are distributed components that provide well-defines interfaces that process and deliver XML messages. A service-based approach makes sense for building solutions that cross organizational, departmental, and corporate domain boundaries. A business with multiple systems and applications on different platforms can use SOA to build a loosely coupled integration solution that implements unified workflows. Security in an SOA environment involves verifying several elements and maintaining confidence as the environment evolves. Organizations deploying SOA implementations should identify practical strategies for security verification of individual elements, but should be aware that establishing the security characteristics of composites and applications using services is an active research. Organizations should also identify the deployment strategies for the SOA infrastructure, services, composites, and applications because different deployment strategies can entail different security verification practices. Finally, all elements should be verified in their operational contexts [Coppolino L].

Web Services are the most popular implementation approach for SOA. The elements of

a Web Service from a security perspective are the service interface, service implementation, message payload, and service level agreement (SLA). All of these elements are visible to participating parties except for the service implementation, which is usually hidden and known only to the service provider. Table 3.1. presents Web Services Security Threat Framework

Web Services Layer	**Attacks and Threats**
Layer 1: Web Services in Transit	• In transit Sniffing or Spoofing • WS-Routing security concern • Replay attacks
Lauer 2: Web Services Engine	• Buffer Overflow • XML parsing attacks • Spoiling Schema • Complex or Recursive structure as payload • Denial of Services • Large payload
Layer 3: Web Services Deployment	• Fault Code Leaks • Permissions and Access issues • Poor Policies • Customized error leakage • Authentication and Certification
Layer 4: Web Services User Code	• Parameter tampering • WSDL probing • SQL/LDAP/XPATH/OS command injection • Virus/Spyware/Malware injection • Brute force • Data type mismatch • Content spoofing • Session tampering • Format string • Information Leakage, Authorization

TABLE 3.2: WEB SERVICES SECURITY PATTERNS

Category	**Pattern**
Authentication	Brokered Authentication Brokered Authentication: Kerberos Brokered Authentication: X509 PKI Brokered Authentication: STS Direct Authentication
Authorization	Trusted Subsystem
Exception Management	Exception Shielding
Message Encryption	Data Confidentiality
Message Replay Detection	Message Replay Detection
Message Signing	Data Origin Authentication
Message Validation	Message Validator
Deployment	Perimeter Service Router

Design Patterns for Web Services

Design Patterns for Building Message-Oriented Web Services

There are six steps involved in building message-oriented Web services, which is simply a Web service that exchanges XML schema-based input and output messages rather than simple parameter-oriented values. The steps are described in the following sections.

Step 1: Design the Messages and Data Types

Step 2: Build the XSD Schema File for the Data Types

Step 3: Create a Class File of Interface Definitions for the Messages and Data Types. Optional step 3A: Generate the WSDL Document Manually

Step 4: Implement the Interface in the Web Service Code-Behind File

Step 5: Generate a Proxy Class File for Clients Based on the WSDL Document

Step 6: Implement a Web Service Client Using a Proxy Class File

Design Patterns for Building Service-Oriented Web Services

Message-oriented web services are the building blocks for service-oriented applications. There are six steps involved in building a message –oriented web service that is compatible with SOA.

Step 1: Create a dedicated type definition Assembly

Step 2: Create a Dedicated Business Assembly

Step 3: Create the Web Service Based on the Type Definition Assembly

Step 4: Implement the Business Interface in the Web Service

Step 5: Generate a Web Service Proxy Class File Based on the WSDL Document

Step 6: Create a Web Service Client

Architecting Secure Web Services Architectures

Web as a media and Web Services as a technology is emerging as a mode of business-to-business and e-commerce transactions. Most of these transactions will carry business-critical and sensitive information that must be secured. Like any other technology domain, secure Web Services is complex and possibly overwhelming. Addressing a breach-in that includes cost of liability, public relations, and loss of business could be more expensive than implementing security measures in advance. Also, security should be enforced throughout the infrastructure. Research issues include Web Services technology, its vulnerabilities, enforcing security in this media, emerging security standards incorporating into Web Services applications. Secure SOA Web Services with WS_Security – A Case Study introduction

Companies have started the adoption of Web Service technology and the WS-Security specification as an approach to ensure the integrity of transmitted messages and data. The WS-Security specification is a joint effort by Microsoft, IBM, and VeriSign to address this most important issue. The WS-Security specification is designed to provide an extensible security implementation that will evolve as Web Services technology becomes more sophisticated. Both WS-Security and WSE 3.0 plays an important role when building Microsoft .NET-based Web Services or Web Services consumers. WS-Security integrates a set of popular security technologies, including digital signing and encryption based on security tokens, including X.509 certificates. It is flexible and is designed to be used as the basis for the construction of a wide variety of security models, including PKI, Kerberos and SSL. Particularly WS-Security provides support for multiple security tokens, multiple trust domains, multiple signature formats, and multiple encryption technologies. Table 3.3 provides Security concepts and security patterns in development phases.

Design Patterns for Web Services

Design Patterns for Building Message-Oriented Web Services

There are six steps involved in building message-oriented Web services, which is simply a Web service that exchanges XML schema-based input and output messages rather than simple parameter-oriented values. The steps are described in the following sections.

Step 1: Design the Messages and Data Types

Step 2: Build the XSD Schema File for the Data Types

Step 3: Create a Class File of Interface Definitions for the Messages and Data

Types. Optional step 3A: Generate the WSDL Document Manually

Step 4: Implement the Interface in the Web Service Code-Behind File

Step 5: Generate a Proxy Class File for Clients Based on the WSDL Document

Step 6: Implement a Web Service Client Using a Proxy Class File

Design Patterns for Building Service-Oriented Web Services

Message-oriented web services are the building blocks for service-oriented applications. There are six steps involved in building a message –oriented web service that is compatible with SOA.

Step 1: Create a dedicated type definition Assembly

Step 2: Create a Dedicated Business Assembly

Step 3: Create the Web Service Based on the Type Definition Assembly

Step 4: Implement the Business Interface in the Web Service

Step 5: Generate a Web Service Proxy Class File Based on the WSDL Document

Step 6: Create a Web Service Client

Architecting Secure Web Services Architectures

Web as a media and Web Services as a technology is emerging as a mode of business-to-business and e-commerce transactions. Most of these transactions will carry business-critical and sensitive information that must be secured. Like any other technology domain, secure Web Services is complex and possibly overwhelming. Addressing a breach-in that includes cost of liability, public relations, and loss of business could be more expensive than implementing security measures in advance. Also, security should be enforced throughout the infrastructure. Research issues include Web Services technology, its vulnerabilities, enforcing security in this media, emerging security standards incorporating into Web Services applications. Secure SOA Web Services with WS_Security – A Case Study introduction

Companies have started the adoption of Web Service technology and the WS-Security specification as an approach to ensure the integrity of transmitted messages and data. The WS-Security specification is a joint effort by Microsoft, IBM, and VeriSign to address this most important issue. The WS-Security specification is designed to provide an extensible security implementation that will evolve as Web Services technology becomes more sophisticated. Both WS-Security and WSE 3.0 plays an important role when building Microsoft .NET-based Web Services or Web Services consumers. WS-Security integrates a set of popular security technologies, including digital signing and

encryption based on security tokens, including X.509 certificates. It is flexible and is designed to be used as the basis for the construction of a wide variety of security models, including PKI, Kerberos and SSL. Particularly WS-Security provides support for multiple security tokens, multiple trust domains, multiple signature formats, and multiple encryption technologies. Table 3.3 provides Security concepts and security patterns in development phases.

Table 3.3 : SECURITY CONCEPTS AND SECURITY PATTERNS IN DEVELOPMENT PHASES

Concept / Phase	**Architecture and Design Phase**
Countermeasure	Feasibility ++
Risk	Estimated
Threat	Feasibility
Attack	Feasibility
Attacker	Feasibility
Vulnerability	Feasibility
Specification 1. Asset	Designed with Security +
Specification 2. Stakeholder	Reviews
Specification 3. Objective	Reviewed +

Case Study

We had implemented a case study, a simple example that secures the StockTrader application. We implemented the UsernameForCertificate assertion that secures the WSE Security Settings wizard and created a custom username token manager. Finally we authorized users using either code or a policy file.

Brokered Authentication:

The client and service do not attempt to authenticate each other directly. They use an intermediary that validates the client's identity and then provides a security token as proof of successful authentication. The client attaches this token to the request and the service uses this token to authenticate the client. There are some authentication brokers such as VeriSign, Windows Active Directory exists.

Implementation and Validation of this case study

The Figure 3.9 consists of class diagram design for Place trade before UserNameToken. Client requests the web page for placing the trade; Stock Trader sends the respond as web page along with the request to enter "accNo., symbol, share, price, tradeType" values; Client enters the values and invokes the page; Trader sends the respond as an xml page acceptance.No security involves in this approach.

The Figure 3.10 consists of class diagram design for Place trade after UserNameToken. Client requests the web page for placing the trade; Stock Trader sends the respond as web page along with the request to enter "accNo., symbol, share, price, tradeType" values; Client enters the values and invokes the page; Trader requests for security checkup; StockTraderSecure checks the usernametoken value for specified client and generates reply to Trader; Trader sends the respond as an xml page**.** Security is involved as UserNameToken value.

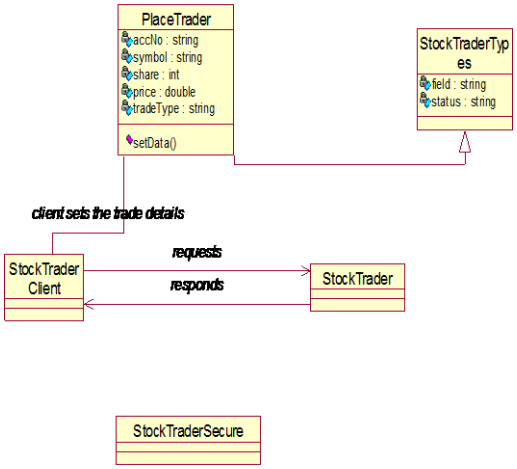


Figure 3.9. Class diagram for Place trade before UserNameToken.

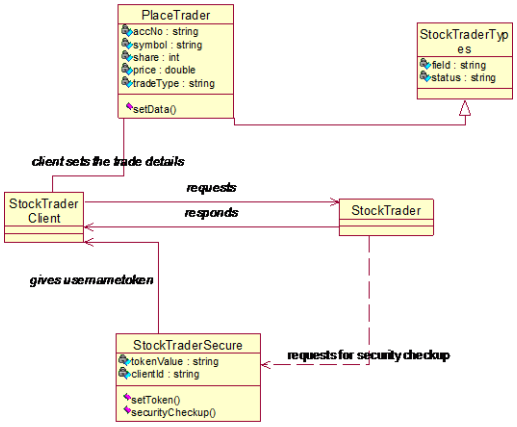


Figure 3.10. Class diagram for Place trade after UserNameToken**.** The Figure 3.11 consists of class diagram for *RequestQuote.* Client requests for RequestQuote web page; Trader replies with page by asking the client to enter "*symbol, tradeType"* values; Client enters the values and invokes; Trader makes a security checkup with StockTraderSecure and sends the reply; Reply consists of all the trade values of particular symbol.

An Active Directory Kerberos ticket has a default of ten hours duration. Client need to request the token once during the session. Brokered Authentication can be implemented in using WSE 3.0 in: Kerberos; X.509 certificates; Custom security token. Brokered Authentication using Mutual Certificate using X.509

certificate option is given as below. (The Figure 3.12 Class Diagram for Mutual Certificate assertion message flow)

The steps involved are given as: Attach X.509 certificate to the message at client side; Sign the message using the client's private key; Encrypt the message using the service's public key; Validate the client certificate; Decrypt the message at service side using private key of service; Validate the signature by decrypting it using public key of client. Brokered Authentication using Kerberos Protocol option is as follows: When user logs in, client encrypts the password using a symmetric key and sends a request to the KDC (Key Distribution Center) for a Ticket Granting Ticket (TGT). If key matches the value stored in Active Directory the KDC sends the TGT and session key. This session key is encrypted by KDC using user's long term key. The TGT is encrypted using KDC secret key. The client sends a request to KDC. The KDC decrypts the TGT with long term key, and decrypts the authenticator using session key. KDC validates and creates new session key. The server receives the request that has the Kerberos security token attached to it. Server will use session key to decrypt the authenticator. The Figure 3.13 provides the execution screen shot of the StockTrader_Web_Service_Home_Page.htm output

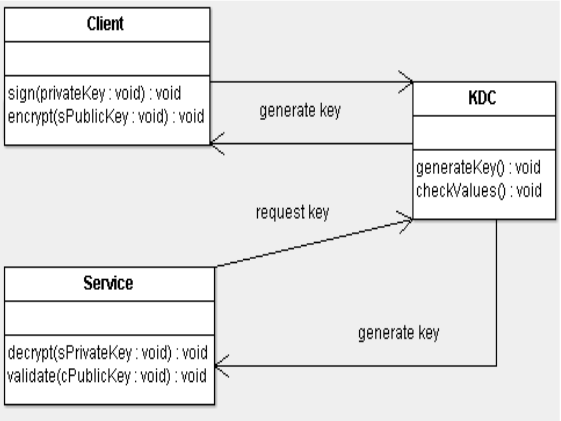


Figure 3.12. Class Diagram for Mutual Certificate assertion message flow.

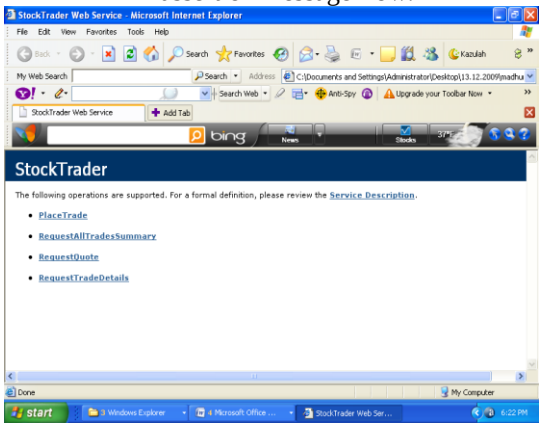


Figure 3.13 : StockTrader_Web_Service_Home_Page.htm output

3.3 IMPLEMENTATIONS AND VALIDATIONS:

3.3.1 Basic Secure Web Services Design using Agile Modeling

SERVICE-ORIENTED computing (SOC) is an emerging paradigm for designing distributed applications [A Mohammed]. SOC applications are obtained by suitably composing and coordinating (that is, orchestrating) available services. Services are stand-alone computational units distributed over a network and are made available through standard interaction mechanisms. Composition of services may require peculiar mechanisms to handle complex interaction patterns (for example, to implement transactions) while enforcing nonfunctional requirements on the system behavior, for example, security, availability, performance, transactional, quality of service, etc. From a methodological perspective, Software Engineering should facilitate the shift from traditional approaches to the emerging service-oriented solutions. Along these lines, one of the goals of this work is to strengthen the adoption of formal techniques for modeling, designing, and verifying SOC applications. In particular, we propose a SOC modeling framework supporting history-based security and call by contract [Constance L Heitmeyer].

The execution of a program may involve accessing security-critical resources and these actions are logged into histories. The security mechanism may inspect these histories and forbid those executions that would violate the prescribed policies. Service composition heavily depends on which information about a service is made public, on how those services that match the user's requirements can be chosen, and on their actual runtime behavior. Security makes service composition even harder. Services may be offered by different providers which only partially trust each other. On the one hand, providers have to guarantee that the delivered service respects a given security policy in any interaction with the operational environment, regardless of who actually called the service. On the other hand, clients may want to protect their sensitive data from the services invoked [Elisa Betrino].

This case study methodology for designing and composing services is to create new services, and to sell it by a package base through a secured media. In particular, we are concerned with Safety properties of service behavior. Services can enforce security policies locally and can invoke other services that respect given security contracts. This call-by-contract mechanism offers a significant set of opportunities, each driving secure ways to compose services. We discuss how we can correctly plan service compositions in

several relevant classes of services and security properties This case study formalism features dynamic and static approach, thus allowing for formal reasoning about systems. Static analysis and model checking techniques provide the designer with useful information to assess and fix possible vulnerabilities.

Several approaches have been developed to support the verification of service-oriented systems. For example, dynamic bisimulation-based techniques have been adopted to analyze the consistency between orchestration and choreography of services whereas state-space analysis has been exploited to check the correctness of service orchestration. This case study approach approach allows for synthesizing and checking the correctness of the orchestration statically [Kearsten Sohr].

In proposed system, we introduced a UML-like graphical language for designing and verifying the security policies of service oriented applications. Another feature offered by this case study framework is that of mapping high-level service descriptions into more concrete programs. This can be done with the help of simple model transformation tools. Such model-driven transformation would require very little user intervention. Here one new framework is introduced called Service Component Architecture (SCA). This framework aims at simplifying implementations by allowing designers to focus only on the business logic while complying with existing standards. This case study approach complements the SCA view, providing a full-fledged mathematical framework for designing and verifying properties of service assemblies. It would be interesting to develop a (model-transformation) mapping from this case study approach for formal framework to SCA. The Figure 3.14 provides the class diagram for Web Services Design Application [Massimo Barloletti].

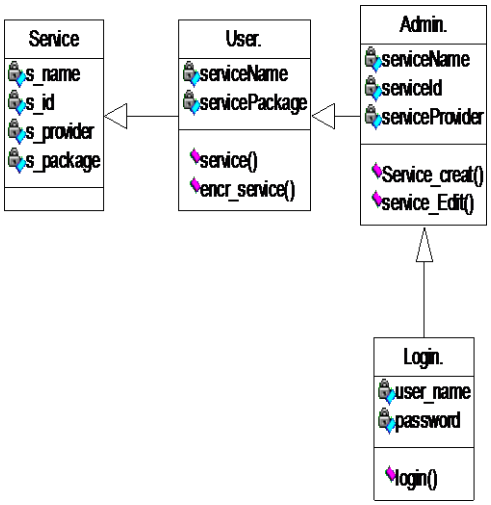


Fig. 3.14 Class Diagram for Web Services Application Agile Design

Role Based Access Control for Web Services Security Policies Design

In the computerized world all the data are saved on electronically. It also contains more sensitive data. In computer systems security, role-based access control is an approach to restricting system access to authorized users [Michele Barletta]. It is a newer alternative approach to mandatory access control and discretionary access control. Security critical business processes are mapped to their digital governments. It needs different security requirements, such as healthcare industry, digital government, and financial service institute. So the authorization and authentication play a vital role. Authorization constraints help the policy architect design and express higher level organizational rules. Access is the ability to do something with a computer resource (e.g., use, change, or view). Access control is the means by which the ability is explicitly enabled or restricted in some way (usually through physical and system-based controls). Computer- based access controls can prescribe not only who or what process may have access to a specific system resource, but also the type of access that is permitted. These controls may be implemented in the computer system or in external devices. The Figure 3.15 and the Figure 3.16 which provides respectively class diagram and sequence diagram for Role-based access control for Web Services policies [Mohammed A].

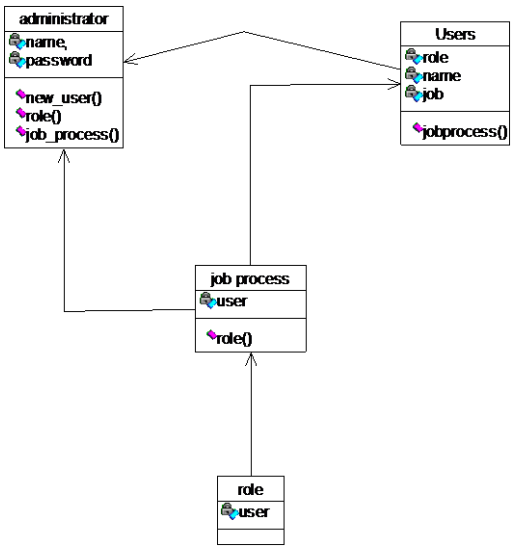


Figure. 3.15 Class Diagram for RBAC Web Services Security Policies

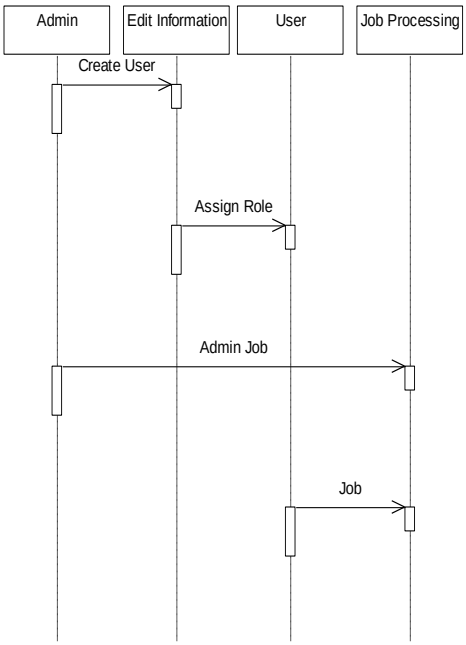


Figure. 3.16 Sequence Diagram for RBAC Web Services Security policies

CONCLUSION

In this paper we developed Agile Modeling for Security Architectures. Later on we developed Agile Modeling for Secure Web Services Architecture design, with simple case study and implementations. Finally we developed Basic Secure Web Services Design using Agile Modeling.

REFERENCES

[1] Frank Innerhofer-Oberperfler, Markus Mitterer, et al [2009], "Security Analysis of Service Oriented Systems – A Methodical Approach and Case Study", IGI Global, Information Science Reference, DOI:10.4018/978-1-60566-950-2.ch002, pp. 33 – 56.

[2] George Spanoudakis and Andrea Zisman [2010] ,"Discovering Services during Service-Based System Design Using UML", IEEE Transactions on Software Engineering, Vol 36, No.3, May/June 2010, PP 371 – 389.

[3] Giorgia Lodi, Leonardo Querzoni, Roberto Beraldi, Roberto Baldoni [2008], "Combining Service-Oriented and Event-driven architectures for Designing Dependable systems", pp. 1 – 13.

[4] Gunnar Peterson, LLC, 2007 "Security Architecture Blueprint", Arctec Group, a White Paper, pp. 1 - 6.

[5] G.Rayana Gouds, M.Sriivasa Rao and Akhilesh Soni [2009], "Semantic Firewall: An approach towards Autonomous Web Security in Service Oriented Environments", International Journal of Recent Trends in Engineering, Vol. 1, No. 1,ACEEE Academy Publishers pp. May 2009 454— 458.

[6] Halvard Skogsrud [2009], "Modeling Trust Negotiation for Web Services", IEEE, February 2009, pp. 1 – 6.

[7] Heiko Tillwick and Martin S Olivier [2004], "A Layered Security Architecture: Design Issues", in Proceedings of the Fourth Annual Information Security South Africa Conference (ISSA2004) July 2004, pp. 1 – 4.

[8] Hiren Bhatt, Arup Dasgupta [2011],"The Disruptive Cloud", Geo Spatial World, May 2011 pp. 20 – 28

[9] Hohn S, Lowis L, Jurjens J, Accorsi R,, [2009], "Identification of vulnerabilities in Web Services using Model-based architecture", IGI Global publishers, 2009, pp. 1 -32.

[10] Hossein Keramati, Seyed-Hassan Mirian-Hosseinabadi [2008], "Integrating software development security activities with agile methodologies," aiccsa, IEEE/ACS International Conference on Computer Systems and Applications pp.749-754.

[11] I. Lazar, B. Parv, S. Motogna, I.-G. Czibula, C.-L. Lazar [2007], "An Agile MDA approach for Executable UML Structured Activities", Studia Univ. Babes-bolyai, Informatics, vol. LII, No. 2, , pp.111-114

[12] Jameela Al-Jaroodi, Alyaziyah Al-dhaheri [2011] "Security issues of Service-oriented middleware", In International Journal of Computer Science and Network Security Vol 2 No 1, pp. 153 – 160.

[13] James S.Tiller [2011], "Adaptive Security management Architecture", Auerbach Publications, pp. 1 – 14.

[14] Jeremy Epstein, Scott Matsumotto and Gary McGraw [2006], "Software Security and SOA: Danger, Will Robinson", IEEE Security and Privacy, January/February 2006, pp. 80–83.