SIMPLE IDENTITY MANAGER

A Paper
Submitted to the Graduate Faculty
of the
North Dakota State University
of Agriculture and Applied Science

By

Sunil Kumar Kolluru

In Partial Fulfillment
for the Degree of
MASTER OF SCIENCE

Major Department:
Computer Science

May 2011

Fargo, North Dakota

North Dakota State University
Graduate School

Title

SIMPLE IDENTITY MANAGER

By

Sunil Kumar Kolluru

The Supervisory Committee certifies that this *disquisition* complies with North Dakota State University's regulations and meets the accepted standards for the degree of

MASTER OF SCIENCE

SUPERVISORY COMMITTEE:

Dr.Kendall Nygard
Chair

Dr.Anne Denton

Dr.Wei Jin

Dr.Cristinel Ababei

Approved:

07/11/2014
Date

Dr.Brian Slator
Department Chair

ABSTRACT

Identity management is a key area for any enterprise. Maintaining user profiles, including e-mail, passwords, and other personal information, and providing a way to manage them is ubiquitous for any company. Specifically, one could not rule out the necessity of basic operations, such as creating a user profile, modifying it, changing the password, finding/resetting a forgotten password, finding a forgotten userID, and enabling/disabling a user account. Identity management is observed on any website that needs credentials to login. Companies are investing millions of dollars in identity management products because, the corporations are aware that it is inevitable.

This paper presents a Simple Identity Manager (SIM) that caters to the need for such system, that can simply be plugged into any web infrastructure and begin availing its services. SIM is Java based and, hence, is platform independent. It is designed to run on any J2EE server.

ACKNOWLEDGMENTS

I wish to express my appreciation and gratitude to my advisor, Dr. Kendall Nygard, for his unending support, encouragement, teaching, and invaluable guidance throughout this paper; without his help, this work would not have come to fruition. I would also like to thank my academic committee members, Dr. Anne Denton, Dr. Wei Jin, and Dr. Cristinel Ababei, for the support they gave me from the beginning until the last minute of my oral exam.

I would like to thank the Lord Almighty who got me to the place where I am at now. I would like to express my deepest gratitude to my beloved parents, Mrs. K. Esther Rani and Mr. K. Sudarsanam (1950-2012), for their support and encouragement throughout the course of my studies. I would like to thank my uncles, Mr. M. Ruben and Mr. G. Aseervadam, because of whom I was able to come to this great nation for my master's study and have earned a decent life. I also want to thank my wife, Mrs. K. Malini Shilpa, for her love, patience, continuous support, and encouragement.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF SCHEMES

1. INTRODUCTION

Since the advent of the Internet, its users and utilization have evolved phenomenally. Starting from being an easy way to send and receive data, the Internet is now the means of business. Internet applications are inevitable, integral parts of the banking and share market. The World Wide Web (or the simply web) gives access to business customers to perform common tasks, such as viewing their personal/business information and making business transactions. These are the days of e-commerce.

Even the intranet (a network limited to a given organization) has its own share of applications that mimic the activity observed on the Internet. Below are some generalized scenarios where network-based access is applied:

- To access business data (examples – information such as a bank statement, credit-card activity, insurance info, investments, and 401k).
- To generate reports or analysis (examples – market-data analysis, customer-activity analysis, and data mining).
- To perform complex calculations (examples – performance index, foreign exchange, derivatives, and mortgages).
- To send data to a remote destination (examples – file transfers, movie uploads, streaming, maps, and emails).
- To perform a transaction and leave the trail of activity (examples – purchases, shipping, and wires) and to remotely perform actions as part of the workflow.
- To view publicized information (examples – company profile, events, news, weather, maps, timetables, and flight/train status).

While these services are desirable, they come with some challenges, such as feasibility, compatibility, and security.

- Feasibility – The Internet and computers are no longer technically sophisticated terms. Common man has activity in almost all scenarios mentioned. Care must be taken to build applications with the target audience in mind.
- Compatibility – Applications should be designed to be portable and platform independent to stretch the scope of target users.
- Security – Because the enterprise involves users, money, and competition, importance must be given to user privacy, accountability, and network/data security. Information/data must be protected from unauthorized access.

1.1. Security in an Enterprise

We would like to shed some light on security in enterprise systems and common practices. There are various flavors of security. While some are for redundant purposes, some, as a pack, form a reliable system from a security perspective. This pack consists of data security (encryption using certificates/smart cards and masking), network security (firewall, SSL, TLS, and dedicated channel), access-control based security (requiring prior successful identification of the user or device requesting access), and application-level security (making use of programs to identify attacks to steal data, e.g., injections, CSS attacks, and worms).

1.1.1. Data Security [1]

Data security is achieved via disk encryption, data masking, and other similar approaches. Disk encryption refers to the technology that encrypts data on a hard-disk drive. Typically, disk encryption is either in the form of software or hardware. Data

masking of structured data is the process of obscuring (masking) specific data within a database table or cell to ensure that data security is maintained and that sensitive information is not exposed to unauthorized personnel.

1.1.2. Network Security

Network security is achieved via network devices, encrypting communication channels, and isolating communication channels. Network devices, such as routers, switches, and firewalls, can be set up in such a way to block communication from untrusted machines and other network-enabled devices.

1.1.3. Application-Level Security

As a general practice, application code must handle vulnerabilities, injections, and CSS attacks. There are readily available filters with fixes for historical issues.

1.1.4. Access-Control Based Security [2]

There is a need for a reliable and accountable way of uniquely identifying (authenticating) a user before providing the services, to which he/she is authorized (e.g., requiring a user to provide a username and password, and validating before allowing access, asking for a digital certificate before trusting the user and allowing access).

A decision-making system is used to identify the allowed services after uniquely identifying the user. Such a mechanism is called access management. A system that provides such services is commonly known as an access manager or a web-access manager in the web world.

To ensure adequate security, most enterprises that offer online web services implement security on two service levels:

- Authentication – To verify that you are who you claim to be.

- Authorization – To confirm that you have the privileges to use the resources you want to access.

An access manager enables centralized authentication and policy-evaluation services as well as dynamic access-management control. In addition, enhanced manageability is achieved through a common user interface.

The most common deployment scenario for securing Web applications with an access manager is through a policy agent-based architecture. With this architecture, you install a policy agent on the container that hosts the application, which acts as a policy enforcement point (PEP). This architecture is illustrated in Figure 1.1 below.

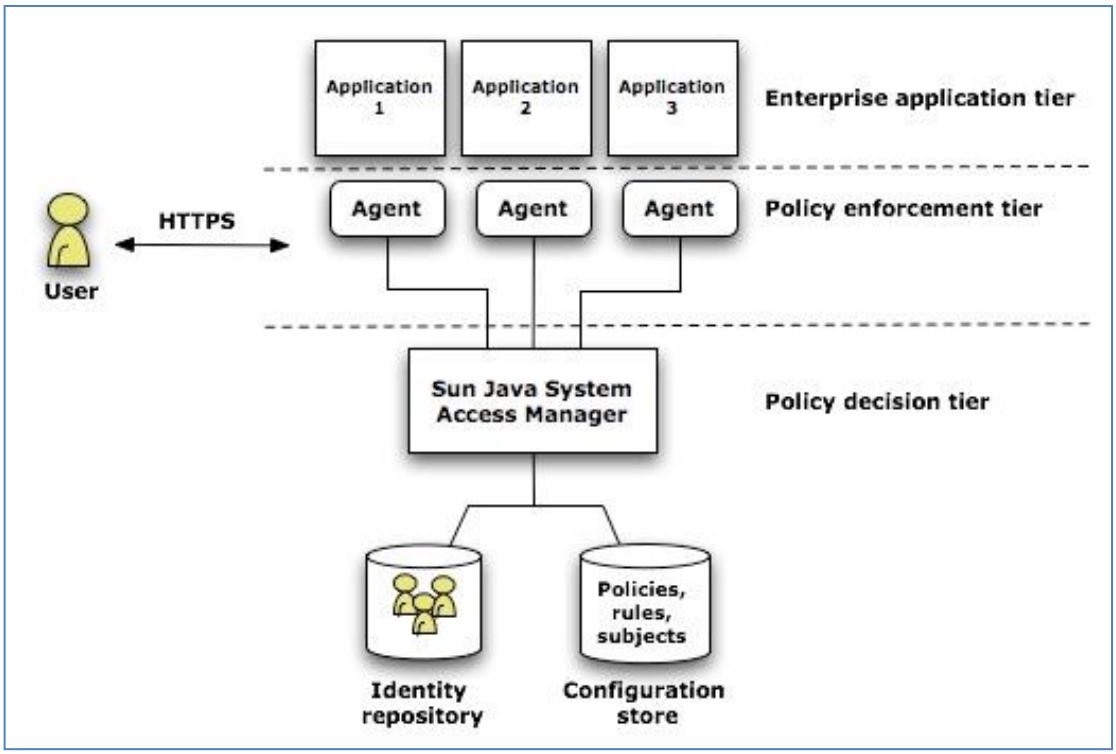

Figure 1.1. Sun One Directory-Server Architecture.

1.2. Identity Management

User identity is crucial for any access-management system. The identity manager enables modification of a user's identity information which is used with the access manager to authenticate and authorize people. A Department of Motor Vehicles (DMV) office is a practical example of an identity management system. To procure a state ID at a DMV, one would have to establish himself/herself as the one that he/she claims to be. In order to do so, one must present 6-point identification. Once the DMV grants a state ID, the person can use it as one's identity for access. Security personnel at gates requiring a check of government-issued photo ID are a good example of an access-management system. Modification of the user's identity information may be a done by the corresponding user (self-service) or by the administrator of such information. While web administrators have privileges to modify a user's identity information, they can also modify the user's privileges to access resources over the web and can enable/disable certain user accounts.

Maintaining profiles with ease and allowing users to manage their individual profiles defines the quality of business for users. Customers would not like to do business with companies that have access issues, and customers demand an easy interface to manage their profiles. Identity management constitutes handling user attributes, such as username; password; and other individual information, including first name, last name, and phone number. Companies face severe business loss if a user runs into access issues, depending on the kind of activity the user does. At the same time, businesses would like to activate and deactivate user accounts for policy and compliance reasons. Hence, identity management has become very crucial to handle these requirements.

An identity management system alleviates the load of operational costs and manpower, thereby boosting overall business performance. According to an article from Computer World [3], when a company with around 15,000 user accounts rolled out a self-service password application, it cut help-desk calls from more than 6,683 to 534 a year. If a mere self-service password could bring down operational costs so much, huge corporations with thousands of employees and hundreds of thousands of customers would benefit with an identity management system that provides a self-service way to fix user accounts. Hence, there is a niche for identity management as a business. According to Nancy Davis Kho [4], "A February 2008 report from Forrester titled 'Identity Management Market Forecast: 2007 To 2014' predicts that the identity management market will grow from nearly $2.6 billion in 2006 to more than $12.3 billion in 2014."

Below are the possible identity management system activities at a higher level:

- **Register a User – User Self Registration or the Administrator Registers a User**

  The user will have to choose a username, password, first name, last name, security questions and answers that only user would know, and other required information about the user as deemed necessary by the business.

- **The User Forgets the Key for Self-Identity and Needs to Retrieve it for Successful Access**

  e.g., Forgot Password – The user forgot his/her password and wants to reset it or wants to be reminded of the password.

- **The User or Administrator Modifies the User's Key to Self-Identity**

  e.g., Change Password – The user wants to change the password to something he/she can remember or something that is more secure.

- **The User or Administrator Modifies the User's Personal Information**

  e.g., Change my phone number – The user wants to change his/her profile information, such as phone number, emails, etc.

- **The Administrator or the Identity Management System Enables or Disables the User**

  Enabling/disabling a user account is an administrative task to disable violators or user accounts that are prone to misuse. While there are tools to audit user activity and network traffic, there should be one common way to enable/disable a user account.

While one would want to perform these simple operations on an identity record, it opens the door to standards and procedures. For security/audit and tracking purposes, one's authenticity is verified before letting his/her profile information be altered.

1.3. Objective

In this paper, we discuss these standards and define the specifications and design for such a system when the user information is stored in a repository that complies with LDAP standards and abides with the following specifications:

- The system must be platform independent.
- Clients that access the system must be platform independent.
- Make use of JNDI (Section 2.3) to interface with LDAP.
- Must be independent from choice of other peer components, such as access manager.
- The end product must be readily usable with mere configuration and client-system development.
- Provide an easier way for the client system to validate a user/administrator.

- Provide a framework to filter CSS attacks with better data validation.
- Changes to the system must be dynamically picked without requiring a restart.
- The end product must be open to enhancements while not impacting the entire application.
- Application code as a whole or in part must be reusable to develop similar applications wherever possible.
- Facilitate application developers with a web services gateway (Section 2.2) in order to perform all the possible operations of the identity management system following a proper authentication in the client system.
- Using the above-mentioned web services gateway, the developer should be able to build customized tasks to accommodate the organization's requirements which could be different classifications as shown in Section 1.2.

2. LITERATURE REVIEW

2.1. Lightweight Directory Access Protocol (LDAP) [5]

A directory refers to a well-organized repository of information to quickly look up a specific entity like a telephone directory, which is a repository of names and phone numbers for people and organizations, where, given the name of the entity, the corresponding phone number can easily be found.

LDAP originated from the X.500 series of the International Telecommunication Union's (ITU) recommendations. ITU is an international standards body, and X.500 is a set of recommendations about directories. Because of this relationship, the structure of the X.500 and LDAP directories is similar. LDAP directory implementations are often X.500 compliant, and gateways between the two directories are also plentiful. LDAP was pioneered at the University of Michigan, and there is still a free implementation available from the website, along with documentation, source code, and other resources.

LDAP is defined by a set of published Internet standards that are commonly referenced by their Request for Comment (RFC) number as published on the Internet Engineering Task Force (IETF) website – http://www.ietf.org. The IETF helps manage a rigorous proposal process where ideas, such as LDAP, are reviewed in drafts until they are ready to be published as an Internet standard. LDAP version 3 (v3) is defined by nine RFC documents. RFCs 2251 through 2256 gave the core details and were later followed by others, including RFC 2829 and RFC 2830.

2.1.1. LDAP Components

LDAP is mainly comprised of the following components:

2.1.1.1. Namespace

Namespace refers to how objects within LDAP are named and organized for a quick lookup or reference. Namespace helps uniquely refer to an atomic entity in LDAP. This convention also helps identify the location of an object in a given organizational structure. In other words, given the entity's name, the entity's location in the organizational structure is also known. This unique name is called Distinguished Name (DN).

An entity within a portion (container) of the LDAP repository can be referred to by a Relative Distinguished Name (RDN). The RDN of record (otherwise called an LDAP entry) is an attribute and its value. An RDN may not be unique across multiple containers but is unique within a given container. Care is taken to make sure the attribute that forms an RDN of an entry in a given container is unique, and duplicates are not allowed.
A DN is a combination of two RDNs separated by a comma. That is, the entity's RDN and the container's RDN are separated by a comma. Figure 2.1 below illustrates how DNs and RDNs form unique references for entries and containers within LDAP.

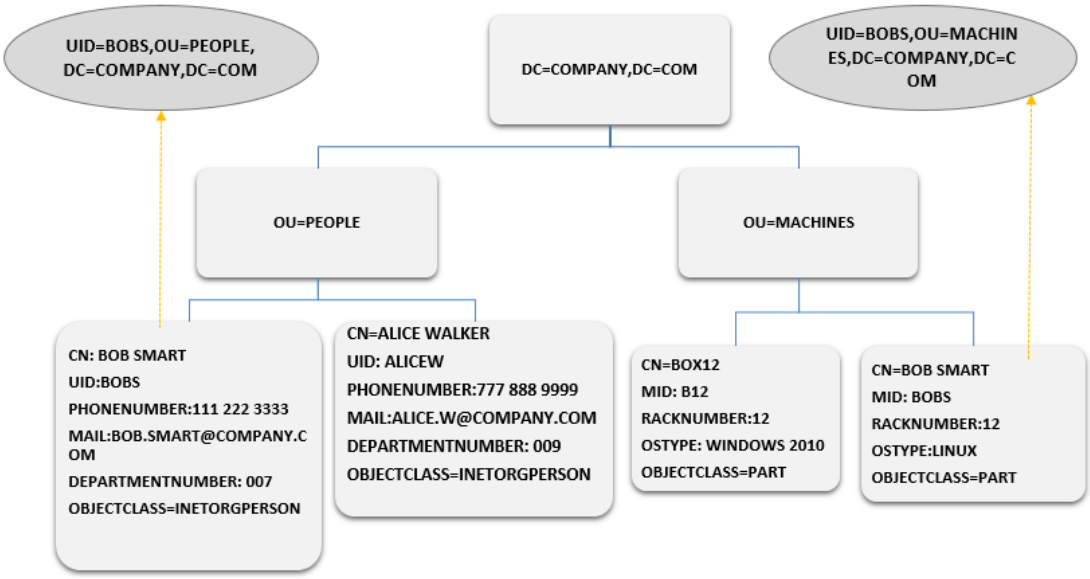

Figure 1.2. Unique Reference by Using DNs and RDNs.

2.1.1.2. Clients and Operations

Client systems that connect to a LDAP server follow RFC 2251. A client will use a Transmission Control Protocol(TCP) connection and perform a bind operation with a principal account (a LDAP administrator/user account that is allowed to perform a bind). LDAP returns a connection handle called the context after successfully authenticating the principal credentials. Using this context, the client can perform other operations like Connection Operations (Bind, unbind, and abandon), Query Operations (search and compare), and Modification Operations (add, modify, modifyRDN, and delete).

- bind – After the bind above is done to make an initial connection to LDAP, the same context can be used to authenticate users by making additional bind operations. A successful bind indicates valid credentials.
- unbind – This operation is performed to disconnect the connection made by the bind operation.
- abandon – This operation gives the client an opportunity to request abandoning a previously requested operation.
- search – The LDAP search operation can be used to identify entries in the directory server that match a given set of criteria. The operation may return zero or more entries, and also zero or more referrals.
- compare – The LDAP compare operation can be used to determine whether a specified entry contains a given attribute value.
- add – The LDAP add operation can be used to create an entry in the directory server.

- modify – The LDAP modify operation can be used to alter an existing entry in the directory server. One or more attributes for an entry can be modified using this operation.
- modifyRDN – This operation can be used to rename an entry and change its physical location in the directory.
- delete – This operation is performed to delete an entry.

2.1.1.3. Schema

The set of rules that define what types of entries can be in the directory is known as the schema. If a particular object class is not in the schema, it is not possible to create an entry with that object class. It is possible to extend the schema to include a new object class or to allow new optional attributes for an existing object class. A schema further defines pertinent rules, such as what type of value can be placed in an attribute and what operators are valid for those attributes. The operators are what the directory uses to compare one attribute's data value to another value. Greater than, less than, and equality are examples of common data operators.

- **Schema Checking**

  The addition of any new entry in a directory is subject to a schema-checking process. Should any of the data not meet the applicable definitions, the addition of the entire entry fails. The schema is not something one can ignore. Some LDAP implementations allow turning off schema checking, which is not advisable. The data would lose their uniformity and order.

- **Default Schema**

  The minimum set of schema objects required by the LDAP standard, as listed in RFCs 2252 and 2256, will form a functional directory. The minimum

LDAP schema is largely formed from the set of X.500-defined schema objects and follows the basic rules for the X.500 schema. And, this is the key reason why so many LDAP products can also be X.500 compliant. Directory vendors take care of implementing this minimum set of schema objects, so you only need to be familiar with what these schema objects are and how you might use them. Most software vendors that leverage a directory find this minimum set insufficient for their purposes and further extend the schema with their own definitions.

- **Extending the Schema**

  Although the schema is arcane because of its syntax format, it is also the source of most flexibility for the LDAP. A directory can implement schema extensions to include whatever data types the company deems necessary. The schema also allows defining new ways to interact with the directory and new ways to work with the data. LDAP publishes the directory schema so that any client can determine what definitions and rules the server employs. The location where the schema is published is stored on every entry, and this information tells you where to look for the schema. The location is called the root Directory Systems Agent-Specific Entry (DSE) container. Most LDAP directories have a single schema that applies to the entire directory, so the location is the same for all entries. Some LDAP servers may allow the definition of unique schemas for different parts of the directory. Here is a sample schematic definition for the person-object class:

Scheme 1. LDAP Schema Example.

```
person OBJECT-CLASS ::= { SUBCLASS OF { top } KIND
abstract MUST CONTAIN { sn, | cn} MAY CONTAIN {
```

(continues)

Scheme 1. LDAP Schema Example. (continued)

```
userPassword | telephoneNumber |seeAlso | description }
ID 2.5.6.6}
```

2.2. Web Services and Simple Object Access Protocol (SOAP)

SOAP [6], originally defined as the Simple Object Access Protocol, is a specification for exchanging structured information to implement web services in computer networks. SOAP relies on Extensible Markup Language (XML) for its message format and usually relies on other application-layer protocols, most notably Remote Procedure Call (RPC) and Hypertext Transfer Protocol (HTTP), for message negotiation and transmission. SOAP can form the foundation layer of a web services protocol stack, providing a basic messaging framework upon which web services can be built. This XML-based protocol consists of three parts – an envelope, which defines what is in the message and how to process it; a set of encoding rules for expressing instances of application-defined data types; and a convention for representing procedure calls and responses.

As an example of how SOAP procedures can be used, a SOAP message could be sent to a web-service-enabled website, such as a real-estate price database, with the parameters needed for a search. The site would then return an XML-formatted document with the resulting data, e.g., prices, locations, and features. With the data returned in a standardized machine-parseable format, they can then be integrated directly into a third-party website or application.

After SOAP was first introduced, it became the underlying layer of a more complex set of web services that are based on the Web Services Description Language (WSDL) and Universal Description Discovery and Integration (UDDI). These services, especially

UDDI, have proven to be of far less interest, but an appreciation for them gives a more complete understanding of SOAP's expected role compared to how web services have actually evolved.

The SOAP specification defines the messaging framework which consists of:

- The SOAP processing model that defines the rules for processing a SOAP message
- The SOAP extensibility model that defines the concepts of SOAP features and SOAP modules
- The SOAP underlying protocol-binding framework that describes the rules for defining a binding to an underlying protocol that can be used for exchanging SOAP messages between SOAP nodes
- The SOAP message construct that defines the structure of a SOAP message

2.2.1. SOAP Processing Model

The SOAP processing model follows a distributed processing model, its participants, the SOAP nodes, and how a SOAP receiver processes a SOAP message. The following SOAP nodes are defined:

- SOAP sender – A SOAP node that transmits a SOAP message.
- SOAP receiver – A SOAP node that accepts a SOAP message.
- SOAP message path – The set of SOAP nodes through which a single SOAP message passes.
- SOAP sender (Originator) – The SOAP sender that originates a SOAP message at the starting point of a SOAP message path.
- SOAP intermediary – A SOAP intermediary is both a SOAP receiver and a SOAP sender, and it is targetable from within a SOAP message. It processes the SOAP

header blocks targeted towards it and acts to forward a SOAP message to the ultimate SOAP receiver.

- Ultimate SOAP receiver – The SOAP receiver that is the final destination of a SOAP message. It is responsible for processing the contents of the SOAP body and any SOAP header blocks targeted towards it. In some circumstances, a SOAP message might not reach an ultimate SOAP receiver, for example, because of a problem at a SOAP intermediary. An ultimate SOAP receiver cannot be a SOAP intermediary for the same SOAP message.

2.2.2. Transport Methods

Both SMTP and HTTP are valid application-layer protocols used as transportation for SOAP, but HTTP has gained wider acceptance because it works well with today's Internet infrastructure; specifically, HTTP works well with network firewalls. SOAP may also be used over HTTPS (which is the same protocol as HTTP at the application level, but uses an encrypted transport protocol underneath) with either simple or mutual authentication; SOAP has a major advantage over other distributed protocols, which are normally filtered by firewalls. SOAP over messaging queue is another possibility that some implementations support.

2.2.3. Message Format

XML was chosen as the standard message format because of its widespread use by major corporations and open-source development efforts. Additionally, a wide variety of freely available tools significantly eases the transition to a SOAP-based implementation. The somewhat lengthy syntax of XML can be both a benefit and a drawback. While it promotes readability for humans, facilitates error detection, and avoids interoperability

problems such as byte-order (Endianness), it can slow processing speed and can be cumbersome. For example, Common Object Request Broker Architecture (CORBA) uses much shorter, binary-message format. On the other hand, hardware appliances are available to accelerate the processing of XML messages. Binary XML is also being explored as a means for streamlining the XML throughput requirements.

2.2.4. Sample SOAP Message

Scheme 2. Example of a SOAP POST.

```
POST /InStock HTTP/1.1
Host: www.example.org
Content-Type: application/soap+xml; charset=utf-8
Content-Length: 299

<?xml version="1.0"?>
<soap:Envelope
xmlns:soap="http://www.w3.org/2003/05/soap-envelope">
  <soap:Header>
  </soap:Header>
  <soap:Body>
    <m:GetStockPrice
xmlns:m="http://www.example.org/stock">
      <m:StockName>IBM</m:StockName>
    </m:GetStockPrice>
  </soap:Body>
</soap:Envelope>
```

2.2.5. Advantages

SOAP is versatile enough to allow the use of different transport protocols. Because the SOAP model tunnels(pass through) fine in the HTTP get/response model, it can easily tunnel over existing firewalls and proxies, without modifications to the SOAP protocol, and can use the existing infrastructure.

2.3. LDAP JNDI [7]

The Java Naming and Directory Interface (JNDI) API, in general, allows Java applications to access a variety of naming and directory services. The LDAP defines a set of operations or requests. (See RFC 2251.) In the JNDI, these map to operations on the DirContext and LdapContext interfaces (which are sub-interfaces of Context). For example, when a caller invokes a DirContext method, the LDAP service provider implements the method by sending LDAP requests to the LDAP server.

2.3.1. Bind

The corresponding way of creating an initial connection to the LDAP server in the JNDI is the creation of an InitialDirContext. When the application creates an initial context, it supplies client-authentication information via environment properties. To change that authentication information for an existing context, use Context.addToEnvironment() and Context.removeFromEnvironment().

2.3.2. Unbind

Context.close() is used to free resources used by a context. It differs from the LDAP "unbind" operation in that, within a given service-provider implementation, resources can be shared among contexts, so closing one context will not free all the resources if those

resources are being shared with another context. Make sure to close all contexts if your intent is to free all resources.

2.3.3. Search

The corresponding JNDI method is the overloading of DirContext.search() that accepts a search filter (RFC 2254).

2.3.4. Modify

The corresponding JNDI method is overloading DirContext.modifyAttributes() that accepts an array of DirContext.ModificationItems.

2.3.5. Add

The corresponding methods in the JNDI are DirContext.bind() and DirContext.createSubcontext(). You can use either to add a new LDAP entry. Using bind(), you can specify not only a set of attributes for the new entry, but also a Java object to be added along with the attributes.

2.3.6. Delete

The corresponding methods in the JNDI are Context.unbind() and Context.destroySubcontext(). You can use either to remove an LDAP entry.

2.3.7. Modify DN/RDN

The corresponding method in the JNDI is Context.rename().

2.3.8. Compare

The corresponding JNDI operation is a suitably constrained DirContext.search().

2.3.9. Abandon

When you close a context, all of its outstanding requests are abandoned. Similarly, when you close a NamingEnumeration, the corresponding LDAP "search" request is abandoned.

3. SIMPLE IDENTITY MANAGER (SIM)

Eetu Heino [8] builds an identity-management system (IDM) to show the financial benefits of identity management systems and tries to establish that the systems do give a return on investment (ROI). He did it by taking a constructive approach where he built an IDM system and evaluated cost savings for a fictional company. While the identity manager system definition varies in the IT industry, requirements gathered from various businesses are often generalized and spin into an identity management system. Some such requirements are automated provisioning [9] and workflow for role assignment to a given user.

- Example for automated provisioning – A user attempts to create a profile for himself/herself with a userID. Before the user can start using a secure system, ManagerA, ManagerB and ManagerC must approve his/her request because they are contacted electronically in order, following successful approvals.
- Example for automated workflow – An employee who gets his/her userID approved based on his/her title must also be tagged with access to Printer1, Phone4, Desktop3, and iPad4.

Today's businesses are so complex that such automation cannot be generalized due to the diversity for the thousands of applications each company has. According to the computer world [3], "Exxon needed to manage identities and provision access based on each user's role and the types of system access required to do the job, but that was difficult with 84,000 employees in 200 countries. Available products could handle a small number of static roles but were not well suited to managing dynamic, attribute-based roles."

We propose a Simple Identity Manager as an enterprise J2EE web application that leaves automated provisioning and role assignment via workflow as a plugin, but acts as a server that responds to SOAP clients with valid requests to modify a user's identity information after validating the requestor. A self-service interface for users to manage their own accounts as well as a management interface for administrators to manage all user accounts in the enterprise can be built by the developer and can avail the SOAP gateway to perform such actions. The server interprets the SOAP requests and alters the respective user data that are stored in an LDAP repository (userstore).

A User's profile, in terms of LDAP, is an entry in the userstore. A user's profile can be created as an LDAP entry using BIND and creating new InitialUserContext (Section 2.3). The user's identity can be determined (authenticated) by accessing the user's RDN and the user's given password following a successful BIND (Section 2.1.1.2). An identified user's profile can be modified with MOD operations. A user's profile can be looked up or printed with the SRCH operation. A user's ID (the attribute that is part of RDN) can be modified using the RENAME RDN method (Section 2.1.1.2).

If a user wants to retrieve access because he/she have forgotten his/her password, SIM will confirm the user's identity by asking for his/her personal email, first name, and last name. If the user is able to provide this information correctly, he/she will be challenged with a security question, and after giving a valid response, the user will be given the chance to reset his/her password.

Each of the above functions is implemented with the Java platform while making sure the functions are reusable and are generalized whenever possible. These functions can be called Java Beans. These Java Beans are separated into two categories:

- AdminTasks – The Java Beans that are designed to be part of the administrator's activity
- UserTasks – The Java Beans that are designed to be part of the user's activity

The following sections have the requirements to achieve these specifications. The requirements are refined and improvised as they are defined. Sometimes, these requirements are only relevant when read in order because, in some cases, one requirement leads to other(s).

3.1. Attribute Names in Various Organizations – The Challenge

Each organization has its own set of attributes, some of which are newly defined via schema extensions (Section 2.1.1.3). This limits the hard-coding possibility for attribute names in the implementation phase. There must be a way to retrieve the attribute names in the user's identity, especially the mandatory ones such as the attribute(s) that form the user's RDN, password, security question(s), name, etc.

For this reason, the organization must be given a provision to feed the SIM with these attributes. SIM will have generic, hard-coded names, and organizations could map their real attributes to these hard-coded names. We call the hard-coded names logical attributes. We call the real attributes in the LDAP physical attributes. Logical and physical attributes are mapped in an XML [10][11].

3.2. LDAP System Settings and Thresholds in Various Organizations

Each organization is unique with respect to the number of user entries in the LDAP repository (volume). The hardware capabilities vary and are generally dependent on the volume being handled by the systems. The ports that the LDAP server listens on are unique. The superuser credentials and other LDAP system-related settings could be

different. Hence, organizations must be given a provision to feed the SIM with LDAP system-related information. This information is vital for the JNDI framework. LDAP system information can be given in the XML format.

3.3. General Settings and Parameters to be Identified

As the system is designed and implemented, the settings and properties that could help tune the system must be identified, generalized, and made available to be modified. This could be an XML.

3.4. AdminUI

While the XML files are the way to store data in the background, the SIM administrator must have a way to feed the XML via a friendly user interface.

3.5. Platform Independence

SIM must be platform independent. While the core system is being built on a Java platform using LDAP JNDI (Section 2.3), the services must be available to any system regardless of the platform. We chose to build web services (Section 2.2) exposed via the web services gateway called the web services end-point. Due to its open standard, web services are callable from any platform, given the end-point. Therefore, the Java Beans that form the AdminTasks and UserTasks are converted into bottom-up web services. Each organization will have to design screens that work on its platform and, in the background, call the web services made available via SIM to perform identity management activities.

3.6. Design

Inspired by the OSI model, like network topology, care has been taken to divide the SIM implementation into the following layers as shown in Figure 3.1 below.

- Presentation Layer – Proxy-servlet based calls to the Web services gateway in SOAP format
- Application Layer/Web Layer – Web services gateway that passes SOAP format requests to the transition layer in the form of Java Bean calls.
- Transition Layer – Converts application-level calls into high-level LDAP operations
- Physical Layer – LDAP operations at a basic level via JNDI. If, in the future, specific JNDI operations are deprecated, only the physical layer will have to be modified accordingly.

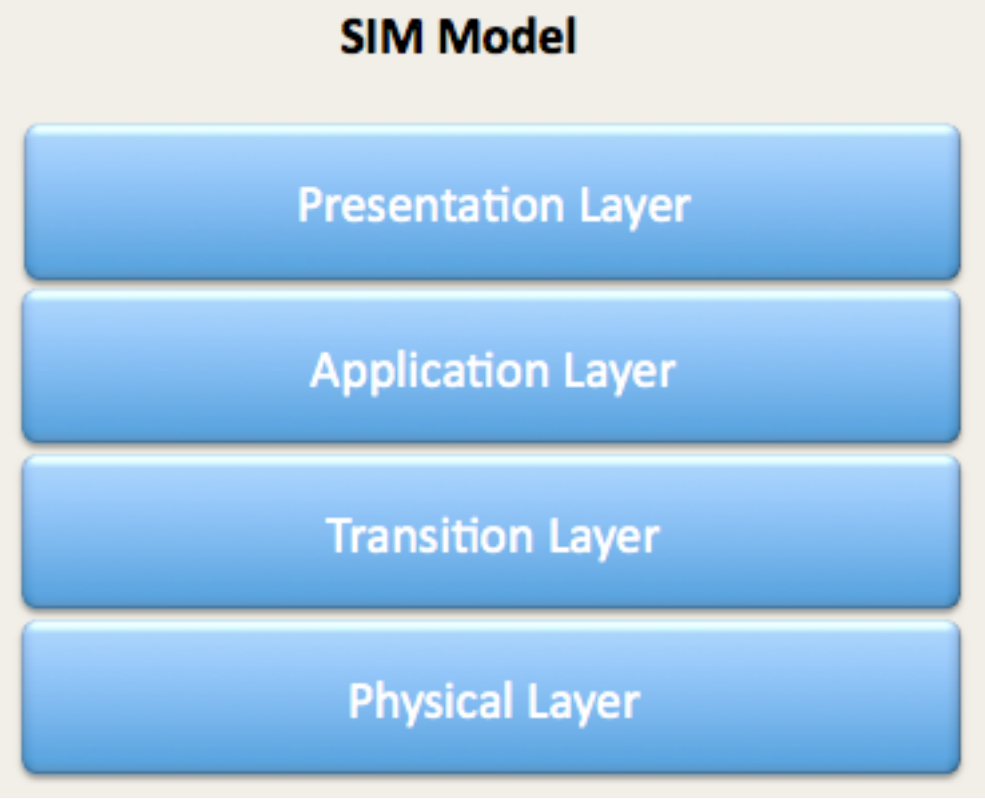

Figure 3.1. SIM's Multi-tier Architecture.

3.7. LDAP Connection Management

Although connection pooling is available via JNDI (Section 2.3), we decided to maintain connections at the physical layer through a custom connection pool. Rules for the custom connection pool:

3.7.1. Pool of Connections

The connection pool implementation must maintain a pool of connections that are reused for various user operations.

3.7.2. One and Only One Connection Pool [12]

For a given SIM environment, there can only be one connection pool, and the pool can only be used for the intended purpose and cannot be modified or destroyed by operational calls from any layer.

3.7.3. Stick to Transition Layer

This connection pool will be an integral part of the transition layer.

3.7.4. Configuration and Tuning Parameters

Configuration and tuning parameters that determine the behavior of this connection pool must be read from an XML file. The SIM Admin UI (Section 4) must have a screen to input these parameter values.

3.8. Handling XML Input Files

So far, we have seen multiple instances where a configuration data feed is expected from the SIM administrator via the Admin UI. All this information has to be stored in XML files.

- Standard mechanisms for reading and writing to an XML file in the background of the Admin UI called XMLReader and XMLWriter will be implemented.
- These configuration parameters must be available to the entire system for reference.
- These parameters are read once from the XML files and, thereafter, read at regular intervals (ConfigDataReloadInterval).
- While the parameters cannot be modified or destroyed, but can only be looked up with the XMLReader.

3.9. LDAP Attributes

We have identified a few attributes that are mandatory. As discussed in Section 3.1, each organization will have its own attribute names. Hence, we only refer to the mandatory attributes in this section. Mandatory attributes are those without which SIM cannot start to operate. They are as follows:

- FirstName – First name of the user
- LastName – Last Name of the user
- UID – User ID, one of the attributes, is unique for a user and is part of the RDN.
- There must be a way to get the user's DN/RDN when the UID is known.
- Password – The user's password is a secret that only the user knows. A combination of the UID and password will let the user authenticate.
- A BIND method must be implemented. It takes the UID and password as input and returns true or false upon a successful BIND or failure, respectively.
- UserType – This attribute defines the user's record type. It must be a multi-valued attribute, and its values must include all the necessary object classes that the organization desires to set for a user record. Missing a value will cause a schema

violation when adding an attribute of a particular type to a user record. (for example, the person objectclass will let a user record have the attribute “mail” which is generally used to hold the user’s email address.)

- Security Questions – This element can be a single-valued or a multi-valued attribute as per the organization’s requirement. The purpose of this attribute is to store a secret question and answer separated by the “^” character. If the organization wants to ask more than one security question when a user forgets his/her password, this attribute must be a multi-valued attribute. Organizations may choose to ask all questions or a few of them. In the implementation phase, we see the necessity to provide a way to identify these multi-valued attributes.
- Email – This email address is for the user. This value is unique for a given user. This value is used in multiple SIM activities. There must be a way to get the UID value when the email value is known.
- UserStatus – When a user is in violation of policies or the user is no longer with the organization, that user record has to be flagged with pertinent information. For this reason, there must be a UserStatus attribute with a value that gives one of DISABLED_USER_LOCK_TEMP, DISABLED_USER_LOCK_PERM, or DISABLED_USER_LOCK_REQPASSWDCHG. A method must be available to the administrators to lock a user with a particular UserStatus value. Accept input for this method in such a way that only valid values are accepted.

3.10. Initial Specifications for SIM

The following subsections are the common tasks that are observed with any identity-management system.

- **Register as a New User**

  Gather all a user's attributes from logical-attribute XML properties. Ask the user to populate the values for each attribute. Each attribute value will have to be validated.

- **Forgotten Password**

  There are two cases for a forgotten password:

  i. Reset and Send

  A password inside the LDAP cannot be read because it is a one-way hash. Hence, set the password to a known random value, and send the password to the user's email address, which is the value of the user's email (Section 3.9) attribute.

  ii. Allow Reset

  Allow the user to reset a new password after answering security questions (Section 3.9) retrieved from his/her LDAP record. There must be a property in the Config XML directory that can let the SIM administrator know how many questions a user should correctly answer and how many questions should be asked.

- **Change Password**

  The user must provide the current password and new password. The current password must be validated. If it is valid, then the user can change to a new password.

- **Modify User Attribute**

  Modify a single attribute, such as phone number and first name, using the JNDI modify operation (Section 2.3).

- **Change UID**

  Changing the UID, which is the RDN, is not the same as changing any other user attribute. Modify a single attribute, such as phone number, etc.

- **Authenticate User**

  Validating a user's identity with a given UID and password combination, by performing a bind operation against LDAP using the UID and password. If the bind is successful, the given credentials are valid, and the user is authenticated; otherwise, the user fails to authenticate.

- **Fetch the UID Given a Unique Attribute**

  Fetch the user's UID or DN by retrieving the user's record with a unique attribute that can be part of the RDN. This is similar to fetching UID for a given email address value.

3.11. Properties Needed at Startup

As discussed in Sections 3.1 through 3.7, there are properties needed as input from the organization's administrator during the SIM startup. These properties must be received as input via a user-friendly GUI (Section 4) and stored in XML format (Section 3.8). A mandatory list of properties needs to be provided by the organization with a list of attributes. From Section 3.9, the list of attributes are as follows:

- UID – can be alphanumeric, is mandatory, and is not multi-valued
- Password – can be any type of ASCII characters, is mandatory, and is not multi-valued
- FirstName – can be alphanumeric, is mandatory, and is not multi-valued
- Lastname – can be alphanumeric, is mandatory, and is not multi-valued

- FullName – can be alphanumeric with spaces, is mandatory, and is not multi-valued
- Email – can only be of the email type, is mandatory, and is not multi-valued
- Entitlements – can be alphanumeric type, are not mandatory, and are multi-valued
- UserType – can be alphanumeric, is mandatory, and is multi-valued
- SecurityQuestion – can be alphanumeric, is mandatory, and may or may not be multi-valued
- UserStatus – can be a specified set of strings only, is mandatory, and is not multi-valued. This attribute value gives the user's status as disabled, need to force a password change, locked for an hour, etc.

3.12. LDAP Configuration Parameters

Another mandatory set of properties that the SIM must have to successfully initialize and ready itself for service. The mandatory properties that must be part of this XML configuration are as follows:

- LDAP Host – Hostname or IP address for the LDAP server
- LDAP Port – Port number on which the LDAP is listening
- QueryBase – The branch or root in an LDAP directory underneath which users or group records are found.
- UserBase – LDAP branch to look up user accounts RELATIVE to the root branch. The lookup excludes the root branch.
- SearchScope – The depth to parse through while looking for a user or group record. As per the JNDI API (Section 2.3), this can be one of
    i. ALL SUBTREES
    ii. UPTO ONELEVEL BELOW

iii. WITHIN THE OBJECT

- ADMIN_GROUP – Must be the DN of an LDAP group. This group will have all the administrator user DNs that can perform administrator tasks.
- LDAPServerURL – LDAP URL with the format,

  ldap://HOST_NAME:PORT_NUMBER

  SECURITY PRINCIPAL & SECURITY_CREDENTIALS
- Credentials for system user who communicates with LDAP; this user should have administrator privileges on QueryBase.
- CONNECTION_TIMEOUT – The interval, in milliseconds, for which a connection is considered active. This property is a tuning parameter that influences the SIM's performance and behavior (Section 3.7.4).
- GROUP_TYPE – Objectclasses that defines a group record
- USER_TYPE – Objectclasses that defines a user record
- DISABLED_USER_LOCK_TEMP, DISABLED_USER_LOCK_PERM & DISABLED_USER_LOCK_REQPASSWDCHG – Acceptable values for the Userstatus (Section 3.11)

3.13. Data Type Definition(DTD) of the XML

From the above discussion on SIM requirements (Sections 3.1-3.11), XML elements (properties) must have following attributes:

- Value – Intended property value
- Description – A comment about the property and its characteristics
- isManadatory – Whether the property is mandatory
- isMultiValued – Whether the property can have multiple values

- valueType – What type of value it is (restricted to one of alphanumeric, alphabet, number, or any ASCII character)

Hence, the DTD looks like Figure 3.2 below.

DTD

```
<!-- DTD for properties -->

<!ELEMENT properties ( comment?, entry* ) >

<!ATTLIST properties version CDATA #FIXED "1.0">

<!ELEMENT comment (#PCDATA) >

<!ELEMENT entry (#PCDATA) >

<!ATTLIST entry key CDATA #REQUIRED>

<!ATTLIST entry type (ALPHA|ALPHANUMBERIC|NUMBERIC|ANY) 'ALPHA'>

<!ATTLIST entry multivalued (Y|N|y|n) "N">

<!ATTLIST entry mandatory (Y|N|y|n) "Y">
```

Figure 3.2. DTD for Configuration Settings in XML File Format.

3.14. Transaction Management System for Web Services

All the functions and their derivatives (Section 3.10) are opened for use as web services. While most of them involve a request and response, some need a continued dialog. The user or administrator requests the SIM to perform a particular task, where the SIM will need to ask for additional information.

For example,

User: Forgot password

SIM: What is your Email Address, FirstName and LastName

User: (responds with Email Address, FirstName and LastName)

SIM: (fetches the user's security questions) Answer this security Question

User: (responds with answer)

SIM: Choose a new password

User: Enters a new password

SIM: (changes the password and confirms)

SIM is an enterprise application that is accessed by vast number of users at a given instance. So, there must be a way to identify and isolate each user request and the user's status (the step in the sequence of dialogs back and forth as shown in the above example) for the transaction. We devised such mechanism by which this unique identification of users is made possible. We call this mechanism as TransMan (short for transaction manager).

The pseudo code for implementing a task while leveraging the TransMan implementation is as follows:

Scheme 3. The Pseudo Code for Implementing a Task While Leveraging the TransMan Task.

```
Let TOTAL be all the input values required from the user
during all dialogs in a Task.
Function multipleDialogTask(String TID,
arguments[1….TOTAL]){
     If(TID is NULL){
Return: TransactionDatabase.createNewTransaction.getID;
     }
```

(continues)

Scheme 3. The Pseudo Code for Implementing a Task While Leveraging the TransMan Task. (continued)

```
stage=TransactionDatabase.getStagefor(TID);
     if(stage.equals("1")){
TransactionDatabase.pushToNextStage(TID);



       Return: "logic to handle step1. Reading
argument1 :";
}
     else if(stage.equals("2")){
TransactionDatabase.pushToNextStage(TID);
Return: "logic to handle step2.Reading argumentset 2”;
}
…
…
     else if(stage.equals(“FinalStage”)){
          PerformLogic
          TransactionDatabase.deleteTransaction(TID)
          Return: FinalResult
}
     Return: InvalidTaskRequest
}
```

TransMan mechanism requires two entities:

- Transaction ID (TID) – This unique ID is assigned to each transaction. It is negotiated back and forth as part of a SOAP request. Because TID is part of a SOAP request which gets posted to the SIM gateway, it is not necessary to expose this transaction ID to the end user. Instead, another web-session ID can be mapped to this transaction ID.
- Transaction Database – This is simply, a database to store all transactions, their statuses, and other transaction-related information. A small, lightweight database, such as SQLite or Derby, is ideal to store the session information. We chose Derby over SQLite for the reasons given in Appendix 1. There are two advantages of storing session information in a database instead of using cookies or memory. The session becomes persistent while being failsafe. Because the state of the transaction is maintained, intruders cannot bypass the system and perform tasks without proper validation. The transaction database will simply contain a single table called SIM with the following columns:

  TID, STAGE, UID, FIRSTNAME, LASTNAME, EMAIL, SECURITYQUESTION, SESSIONSTARTINMILLISEC

  Below are the SQL queries that need to be run when Derby is configured for the first time as a Transaction Manager database (SIM DB):

Scheme 4. SQL for Creating a SIM DB.

```
drop table SIM; (to drop any previously existing tables
with same name)
create table SIM (TID varchar(50) primary key, STAGE
```

(continues)

Scheme 4. SQL for Creating a SIM DB. (continued)

```
integer not null, UID varchar(20), FN varchar(20), LN
varchar(20), EMAIL varchar(50), SecQuestion
VARCHAR(120), SESSIONSTARTINMILLISEC BIGINT NOT NULL );
select * from SIM;describe SIM;
```

Transactions in the SIM table which are older than a session-timeout value must be deleted from time to time. The query is as follows:

Scheme 5. Deleting Older Transactions from TransDB.

```
delete from SIM where TRANSMANTIMEOUT <
CURRENTTIMEINMILLISEC - SESSIONSTARTINMILLISEC;
```

For administrator tasks, a separate web services endpoint, DB/table, will be created. Those tasks will additionally verify to see if the accessing UID is a valid admin ID, by checking for the "Entitlement" logical attribute in LogicalAttrs.xml to see if it has value as a member of the ADMIN_GROUP set in DirectoryConfig.xml.

3.15. ObjectPool for LDAP Connections and Operations

LDAP connections (LDAP contexts via bind) need to be limited in number and must be reused. Instead of having pool of LDAP contexts, we prefer a pool of operation objects called simldapops. An operation object consists of LDAP context, a flag that indicates whether the context is in use and performing an operation against LDAP, the LDAP connection information, and the DN or LDAP branch under which the operation is performed. The ObjectPool design is dependent on the valid implementation of the validateConnection method in the application layer.

Scheme 6. Validating a Connection in the Pool.

```
simldapops.validate(){
   If (this.free=true)
     Return true;
}
when performing an ldap operation:
simldapop.setEngaged() → this.free=false;
try{
simldapop.performoperation
}catch(){
}finally{
     simldapop.setFree() → this.free=true;
}
```

The LDAP operations pool is based on the following ObjectPool design pattern [13]:

Two data tables, LOCKED and UNLOCKED, are used to track objects that are in use and objects that are free, respectively. EXPIRATION_TIME is the time after which an object in use (in the LOCKED table) is freed (placed in UNLOCKED table). This value is retrieved from DirectoryConfig XML properties file. POOL_SIZE is the maximum number of objects that can exist.

Scheme 7. Checking Out Connections from a Pool.

```
Function CheckOut {
    if (UNLOCKED.getSize() > 0) {
```

(continues)

Scheme 7. Checking Out Connections from a Pool. (continued)

```
            For( Object Obj in UNLOCKED table) {
              if (Obj reached EXPIRATION_TIME) {
               move Obj from UNLOCKED to LOCKED TABLE
               return Obj;
              }
              else {
                if (isFree(Obj)) {
                     move Obj from UNLOCKED to LOCKED TABLE
                          return Obj
                }
                else {
                     // object failed validation
                     remove Obj from UNLOCKED TABLE
                     expire(Obj);
                      Place Obj in LOCKED TABLE
                      return Obj
                }
              }
            }
          }
          // no Objects in free pool.
```

(continues)

Scheme 7. Checking Out Connections from a Pool. (continued)

```
    //so, 1. check if one of engaged Object is beyond
expiration time, if so expire the Object and reassign it
to current requester
    //    2. check if total engaged Object is less than
50, if so create an Object and assign it to the
requester
    else {
     if(LOCKED.getSize() <= POOL_SIZE){
           For( Object Obj in LOCKED table) {
            if ( (Obj reached EXPIRATION_TIME) OR
 isFree(Obj) ){
                  // object has expired
                   expire(Obj);
                        remove Obj from LOCKED TABLE
                        expire(Obj);
                        Place Obj in UNLOCKED TABLE
                        return checkOut();//recursion,
this time Obj will be returned
            }
     }
     if(LOCKED.getSize() < POOL_SIZE){
          Obj = create();
```

(continues)

Scheme 7. Checking Out Connections from a Pool. (continued)

```
                Place Obj in LOCKED table
                return Obj;
             }
            }
            Thread.sleep(1000);
            return checkOut();
             //this forms a recursive loop until an Object
        //becomes available and
            //the requester is automatically put on wait while
        an Object becomes available
          }
          Function checkIn(Object Obj) {
             remove Obj from LOCKED TABLE
            Place Obj in UNLOCKED TABLE with
        currentSystemTimeStamp on it.
          }
```

3.16. Reload Thread Mechanism

There must be a way to reload XML properties at an acceptable interval. A daemon/thread must be running in parallel on the SIM to populate data at runtime. This mechanism must be part of every XML property object in the SIM. The reload interval must be a system property available as part of the System Properties XML. The System Properties XML is the only properties file that is read-only-once, meaning that changes to this file will not be applicable until the system is restarted.

3.17. Input Data Validation

Throughout SIM, various data objects must be identified. These data objects' expected values must be formatted into regular expressions. These regular expressions are encapsulated into a data-validation mechanism.

This data validation logic must be reusable and open. This mechanism will be used to validate input values that are expected from the user.

3.18. Infrastructure

Below is typical infrastructure illustrating where SIM will fit in an enterprise web environment. Identity Manager stands in parallel with Access Manager. The web server forms the front-end facing the Internet user. The SIM UserTasks UI is the view part of the SIM model that calls SIM web services in the background to fulfill user requests. The SIM AdminTasks UI is not open to Internet users. This interface is strictly for identity management administrators. Although Figure 3.3 below, shows one instance of the SIM infrastructure, multiple, parallel instances can exist for high availability.

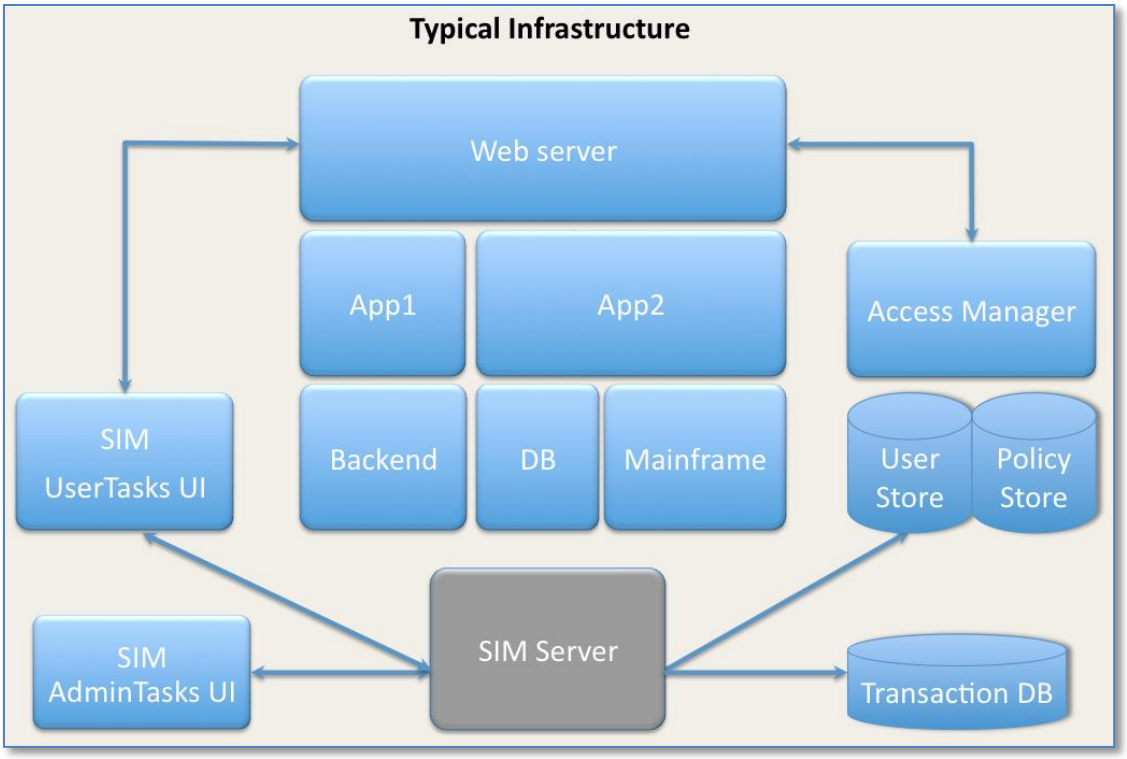

Figure 3.3. Typical Enterprise Security System Architecture that Uses SIM for IDM.

3.18.1. SIM Black Box

In Figure 3.3 above, the SIM Server depicted in gray is essentially a black box. This black box has been magnified in Figure 3.4 below, depicting the contents of SIM. The UserTasks gateway and AdminTasks gateway are two dynamic web applications deployed on a J2EE-compliant application server. These gateways,

- make use of the SIM framework to communicate with the LDAP.
- make use of the simldapops ObjectPool (Section 3.14) to make a connection and perform LDAP operations.
- use the TransMan to track transactions (Section 3.13).

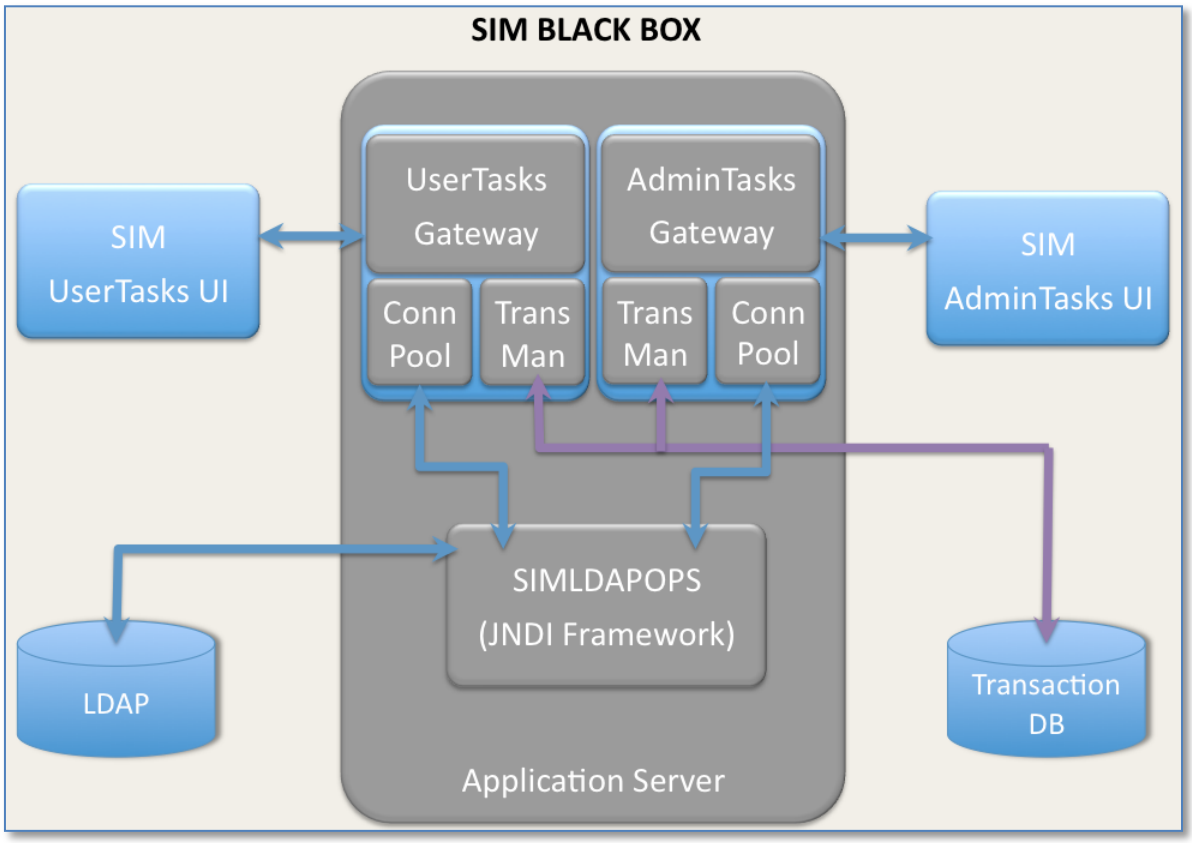

Figure 3.4. SIM System Architecture.

Once the SIM is configured, it is up to the developer to build the AdminTasks UI or the UserTasks UI in either Java or .NET. The developer would need the web services definition language(WSDL) endpoints (Gateway URLs) to write the UI. The Developer Guide (Section 6) discusses UI development in detail.

3.18.2. High Availability and Failover

Figure 3.5 illustrates an example where SIM gets high availability and failover capabilities. The transaction database forms a centralized database and empowers the SIM's high availability environment with persistence.

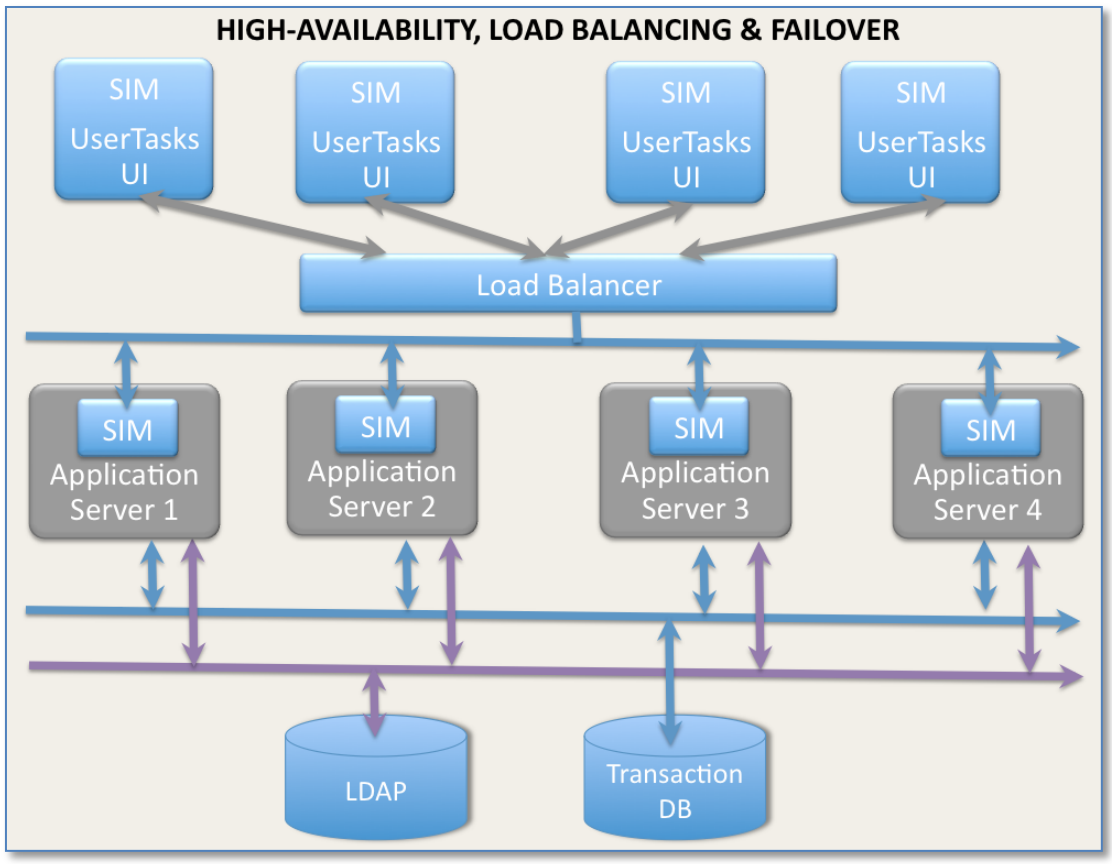

Figure 3.5. SIM Architecture – High Availability and Fail Over.

SIM User Tasks UI, developed for an organization by leveraging the UserTasks gateway, can be a clustered web application deployed on multiple application servers or web servers that support servlets. Developers will have to point their web services at the load balancer, which will distribute the incoming requests equally/proportionately to all SIM black boxes (Section 3.18.1). If any of the SIM Black Boxes are unavailable to serve requests, for reasons such as machine shutdown, failure, and network errors, the load balancer will automatically distribute requests among the available SIM Black Boxes.

This model suits organizations where Quality of Service (QOS) is highly regarded. In such companies, high availability, load balancing, and failover are mandatory.

3.19. Virtualization

Virtualization is a widely adapted technology in the enterprise world. Virtualization is simply mimicking a machine in software format while being able to do everything one can do with a physical machine. The hard disks are flat files. The system architecture and resources, are embedded into a single configuration file. The virtualization-software platform interprets these files and runs as a physical machine.

This concept can be used with a SIM by deploying one SIM Black Box on a virtual machine. Multiple copies of this machine can then be replicated for high availability.

4. SIM ADMIN UI

An Admin UI has been built to provide an interface for the SIM administrator to feed various property values identified throughout Section 3. The Admin UI has been divided into four screen bases on the classifications of the properties into the following:

- System Config – Properties that are only read once and are vital for the SIM process to be available to service are part of this screen. SIM must be restarted to initialize/invoke the changes to these properties
- Directory Config – All LDAP-related properties are part of this screen. These include attributes that are vital for the SIM to initialize and function.
- LogicalAttrs Config – The logical to physical directory mapping (Sections 3.1 and 3.9) is specified on this screen.
- Transaction Management Config – To handle user/admin tasks that need back-and-forth dialogs, the Transaction Manager needs some predefined properties to function. Such attributes are set on this screen.

The SIM Admin UI is protected by SIM's own authentication and authorization system, avoiding dependency on the access manager and leveraging the SIM implementation. The SIM Admin UI will only allow an administrator to configure the system with a fresh installation. Once configured, authentication and authorization are mandatory.

4.1. The SIM Admin Console for System Configuration

Figure 4.1 below, illustrates the system screen that loads all the properties which are used by the system configuration. Here, the user can add new attributes by clicking the

Add Row button. Changes on this screen will not affect the configuration XML file directly. The user has to explicitly click the Submit button to take the new values.

- Add Row – Adds a new attribute.
- Delete Row – Deletes a new attribute
- Submit – Submits the new changes to the configuration XML file. The file for system configuration is SystemConfig.xml.

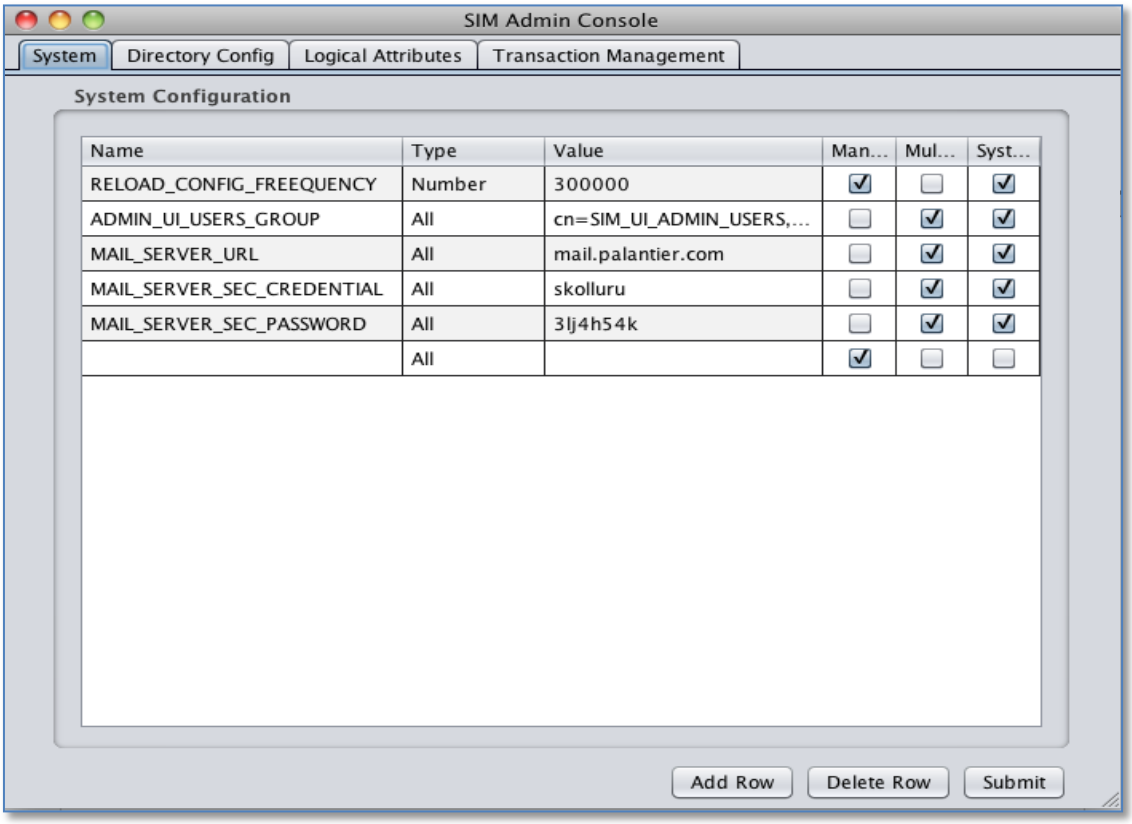

Figure 4.1. SIM Admin UI Console, Showing System Configuration Tab.

4.2. Directory Configuration Screen

Figure 4.2 below, illustrates the screen that loads all the properties which are used by the directory configuration. For more screenshots and use cases for the Admin UI, please refer to Section 8.

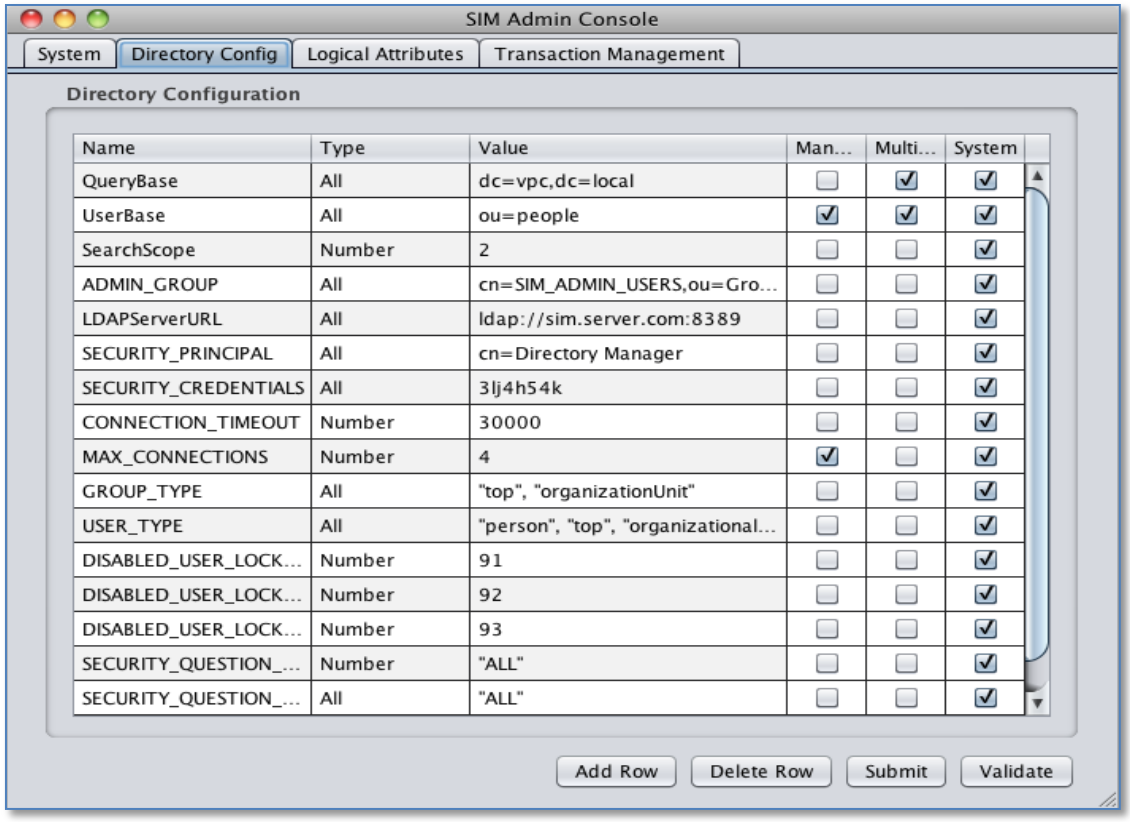

Figure 4.2. SIM Admin UI Console – Directory Configuration.

5. SIM PERFORMANCE TUNING

There are quite a few tunable parameters within the SIM. To avoid throttling, perform load tests to determine the optimal ObjectPool size for simldapops. Our study (Section 10) shows that, for every 10,000 concurrent user transactions, the SIM needs simldapops pool size of 4. However, we suggest that a load test be conducted once the SIM is configured in order to tune this parameter. The SIM Black Box runs on a J2EE application server.

Enable the use of native i/o instead of green treads. This will make the Application server rely on the operating system's threading mechanism, instead of the Java threading mechanism. Enable a higher cache size for the LDAP server. This value must suit the system capacity.

SIM application per user takes at an average, a footprint of 2KB. So, along with the number of target users per SIM Black Box, the application server heap size must be the minimum required to serve. The SIM is built to reuse existing objects in JVM and, hence, puts minimum stress on garbage collection(GC). However, we recommend – XX:MarkParallelSweepGC as the JVM runtime argument.

6. DEVELOPER GUIDE TO GENERATE A USER INTERFACE FOR TASKS

The SIM User Tasks UI, developed by an organization by leveraging the UserTasks gateway, can be a clustered web application that is deployed on multiple application servers or web servers that support servlets in the Java world or ASP in the .NET world. Here, we give some guidelines for leveraging the web services gateway.

6.1. Different Kinds of Web Service Calls

There are two kinds of web service calls that can be made to SIM for AdminTasks or UserTasks.

- **Transaction Based**

  These web services involve back-and-forth dialog and need negotiation for a unique ID called the TransactionID. In order to get this ID, the first call with required inputs is sent with the TransactionID value as "initial." Such call will signal the Web services gateway to generate a new TransactionID and send it back as a response. The rest of the dialogs must use this TransactionID. Below are the tasks that are transaction based:

  i. ForgotPassword

  ii. ForgotUID

- **Single Request**

  The majority of the Web service calls are of this type. A Web service call will receive a response to complete the transaction.

Below are the tasks that are transaction based:

i. ChangePassword
ii. ForcePasswordReset
iii. getUserStatus
iv. isAdminUser
v. lockUser
vi. ModifyProfile
vii. RegisterUser
viii. unLockUser
ix. userBelongsToGroup
x. validateAdminCredentials
xi. validateCredentials
xii. ChangeUID
xiii. printUserData

6.2. Implementation of Tasks UI

The Implementation of Tasks UI involves retrieving the WSDL file. A WSDL file is generally available at

i. http://sim.myserver.com/shudwork/services/MyPasswordBeans?wsdl (User Tasks)
ii. http://sim.myserver.com/adminws/services/AdminTasks?wsdl (Admin Tasks),

where, sim.myserver.com is a server on which the SIM Black Box is hosted.

6.2.1. Java Web-Based UI

Create bottom-up web services using eclipse with one of the above URLs as input, resulting in generation of proxy Java classes. Use the proxy Java classes to make calls to the web services end-point while getting user input via servlets. Here are some sample web pages that invoke AdminTasks via the UI. Figure 6.1 below, illustrates a sample Java Server Page(JSP) which prints user information. Here, input is given for the UID field, and when the method is invoked, the result displays the user's information.

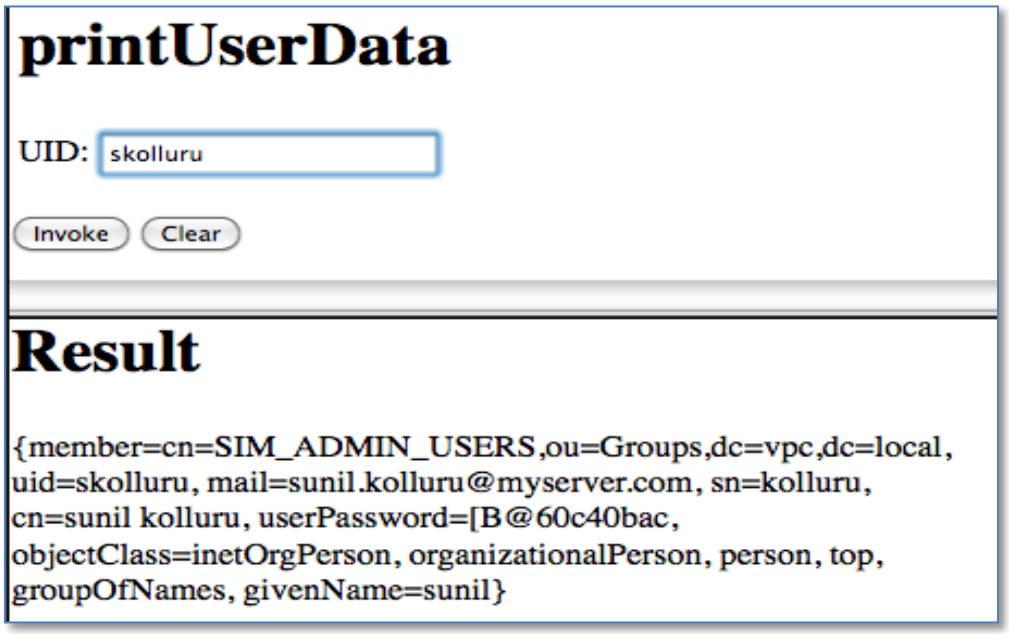

Figure 6.1. Sample JSP – Print User Information.

Figure 6.2 below illustrates a sample JSP which creates a Transaction ID when the ForgotPasswordReset method is invoked. Here, inputs are given for TID, UID, email, FN, and SN. When the method is invoked, it creates a Transaction ID for the user.

Note – TID values are set to "intial" the first time a client intiates a connection to perform the forgotPasswordReset task.

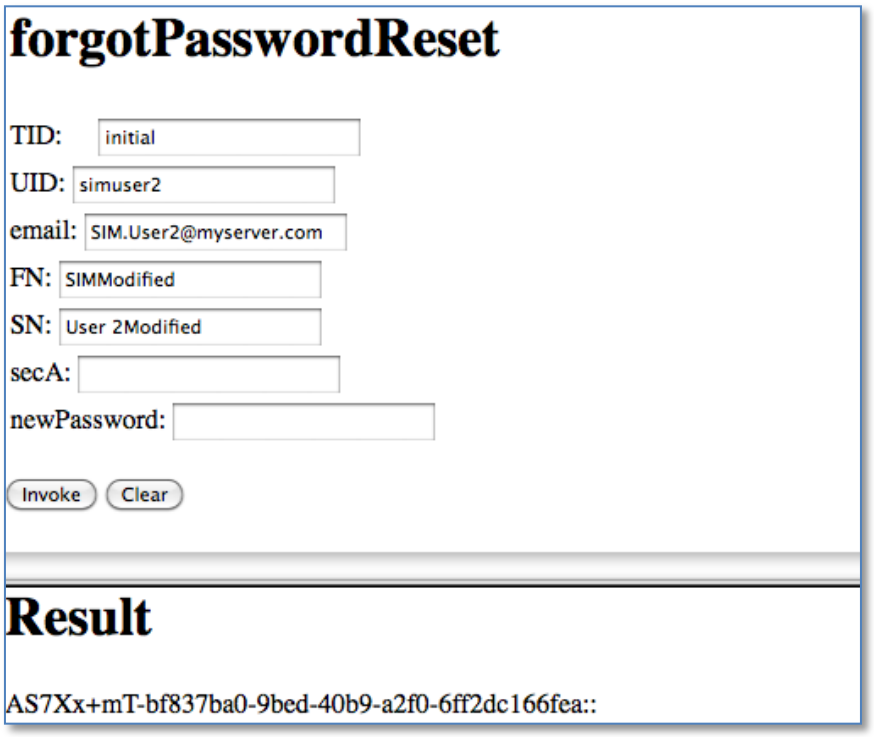

Figure 6.2. Sample JSP – Forgot Password Reset, Create Transaction ID.

Figure 6.3 below, displays a sample JSP which gets a security question when the ForgotPasswordReset method is invoked by inputting the Transaction ID.

Note – Once the Transaction ID is generated, it is used during the back-and-forth communication for the ForgotpasswordReset task.

Figure 6.3. Sample JSP for Forgot Password Reset – Get Security Question.

Figure 6.4 below, displays a sample JSP asking for a new password to be set by the user. The answer for the security question is set as the parameter in secA(in Figure 6.4 below) and verifies. After verification of the security question, the new password to set is displayed to the user.

Figure 6.4. Sample JSP – Forgot Password Reset – Choose A New Password To Set.

Figure 6.5 below, displays a sample JSP where a new password is successfully set for the ForgotPasswordReset method. When the user enters a new password and it is set, the result page displays “success,” meaning that the password reset is successful.

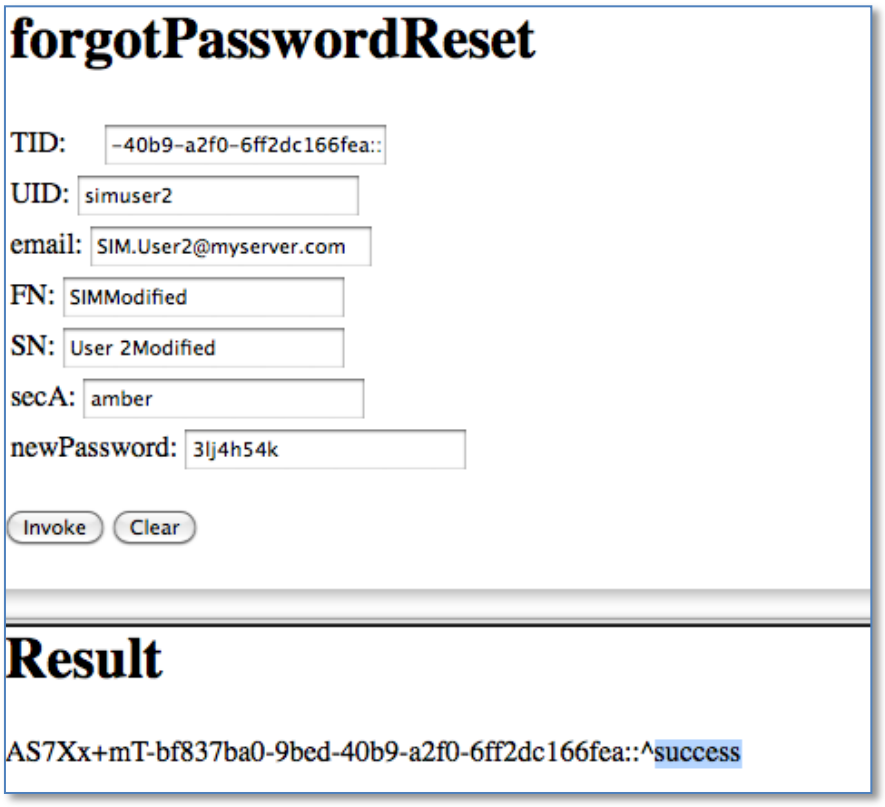

Figure 6.5. Sample JSP – ForgotPasswordReset – Password Reset Successfully.

6.2.2. .NET Web Based UI [14]:

This work is automatically done by Visual Studio .NET when referring to a web service that has been added. Here are the steps to follow:

- Create a proxy for the Web service to be consumed. The proxy is created using the WSDL utility supplied with the .NET software development kit(SDK). This utility extracts information from the Web service and creates a proxy. Thus, the created proxy is only valid for a particular Web service. If you need to consume other Web services, you have to create a proxy for that service as well. Visual Studio(VS) .NET automatically creates a proxy for you when the reference for the Web service is added. Create a proxy for the Web service using the WSDL utility supplied with

the .NET SDK, resulting in creation of AdminTasks.cs in the current directory. We need to compile it to create FirstService.dll (proxy) for the Web service.

c:> WSDL http://sim.myserver.com/adminws/services/AdminTasks?wsdl

c:> csc /t:library AdminTasks.cs

- Put the compiled proxy in the bin directory of the virtual directory for the Web service (e.g., c:\AdminTasksUI\bin). The Internet Information Services(IIS) server looks for the proxy in this directory.
- Create the service consumer, which we have already done. This proxy takes care of interacting with the service.
- Type the consumer's URL in Internet Explorer to test it (for example, http://sim.myserver.com/AdminTasksUI/WebApp.aspx).

7. TEST CASES

Following are some test cases run on the SIM system.

7.1. RegisterUser

A call to this task, registers the new user with the given parameters in the SIM application. Mandatory attributes are known from the DirectoryConfig.xml file. Check for the mandatory "Y/N." If the user is successfully registered with the SIM application, then the service will return “true.” To check the user’s details, use the printUserData method.

- **Input**

  Figure 7.1 below, displays the RegisterUser page where the administrator details are given by the admin user while registering for the first time. Admin user details are given for the UID, firstname, lastname, fullname, entitlement, security question, phone, and password fields.

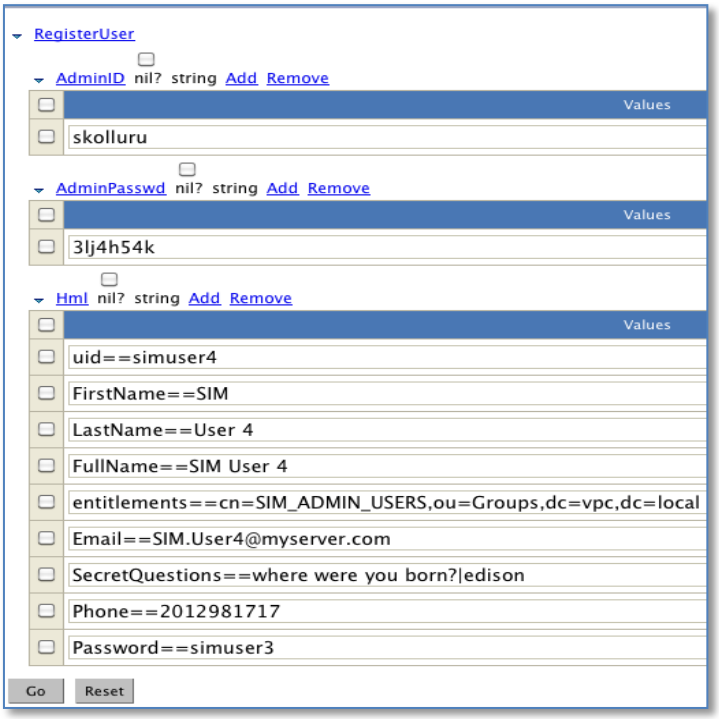

Figure 7.1. Test Case Input – Register New User.

- **Output**

  Figure 7.2 below, displays the page that results after a user's successful registration.

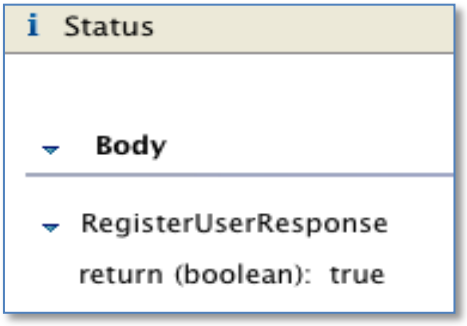

Figure 7.2. Test Case Output – Register New User.

7.2. PrintUserData

A call to this task displays the information about the registered application user.

The output of this service contains user information in the following format:

{<attribute_name>=<attribute_value>,…}.

If a given user identifier is not given, then the service will throw an exception.

- **Input:**

  Figure 7.3 below, illustrates sending the user ID as input for the printUserData task.

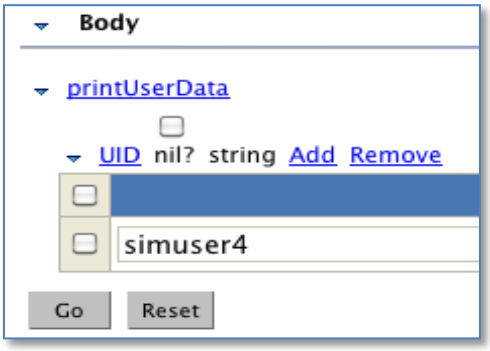

Figure 7.3. Test Case Input – Print User Information.

- **Output**

Figure 7.4 below, illustrates the output for the printUserData task.

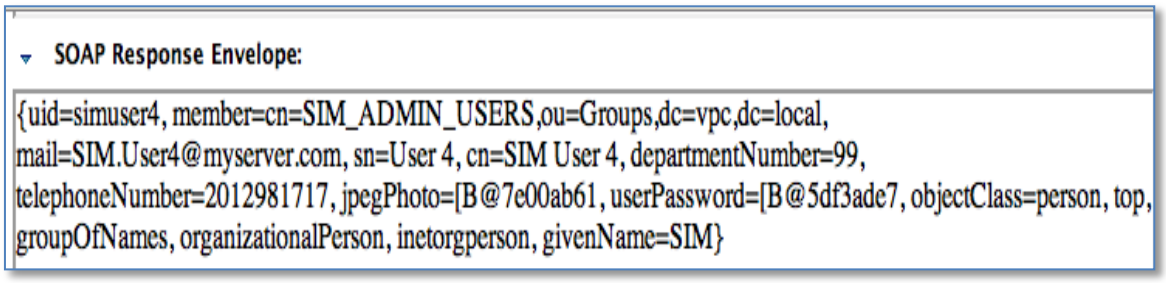

Figure 7.4. Test Case Output – Print User Information.

7.3. ValidateCredentials

A call to this task validates the given user credentials. If the given credentials (as shown in Figure 7.5 below) are successful, then the service returns "true" (as shown in Figure 7.6 further below); otherwise, the result is "false."

- **Input**

  Figure 7.5 below, illustrates the input for validating credentials.

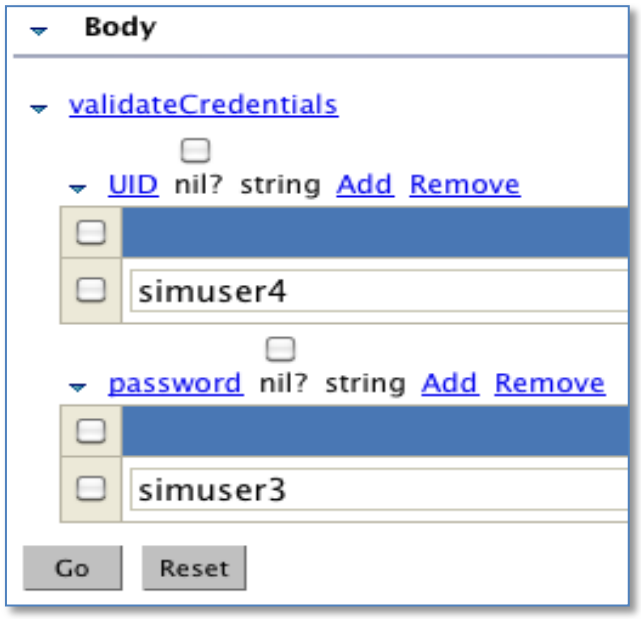

Figure 7.5. Test Case Input – Validate Credentials.

- **Output**

  Figure 7.6 below, illustrates the output for validate credentials.

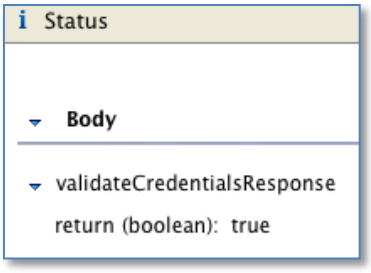

Figure 7.6. Test Case Input – Validate Credentials.

7.4. IsAdminUser

A call to this task verifies if the given user is the administrator. The administrator user group is configured in the DirectoryConfig.xml in the SIM_HOME/Config directory. Returns "true" if the user is an administrator; otherwise, the result is "false."

7.4.1. Test Case 1 – Give the User, Who is an Administrator.

- **Input**

Figure 7.7 below, illustrates the input to check whether the given user is an Admin User.

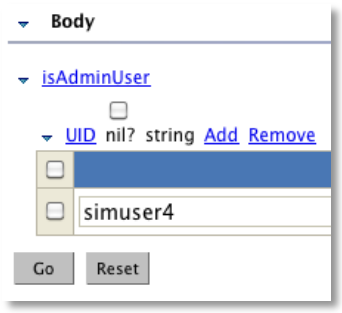

Figure 7.7. Test Case Input – Check If Given User is an Admin User.

- **Output**

Figure 7.8 below, illustrates the output to check if the user is an admin user.

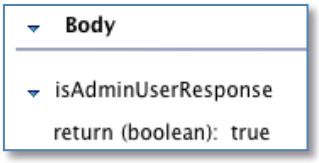

Figure 7.8. Test Case Output – Check If Given User is an Admin User.

7.4.2. Test Case 2 – Give the User, Who is Not an Administrator.

- **Input**

  Figure 7.9 below, illustrates the input to check whether the given user is an Admin User.

Figure 7.9. Test Case Input – Given Non Admin User – Case – is an Admin User.

- **Output**

  Figure 7.10 below, illustrates the output to check if the user is an Admin user.

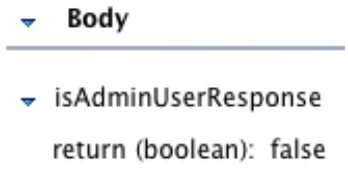

Figure 7.10. Test Case Output – isAdminUser when Given Non Admin User.

7.5. ValidateAdminCredentials

A call to this task validates if the given credentials belong to the administrator. First, authenticates the given credentials and then authorizes the given user as an administrator. If the user is authenticated and authorized, then the service returns “true”; otherwise, the result is “false.”

7.5.1. Test Case 1 – Valid User Credentials

- **Input**

  Figure 7.11 below, illustrates the input to validate a valid administrator’s credentials.

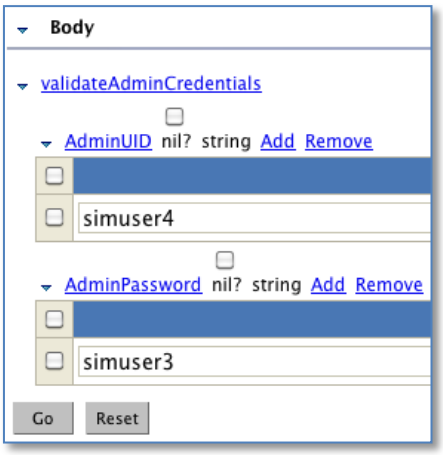

Figure 7.11. Test Case Input – Validate Admin Credentials.

- **Output**

  Figure 7.12 below, illustrates the output to validate a valid administrator’s credentials.

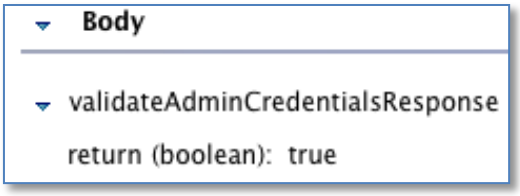

Figure 7.12. Test Case Output – Validate Admin Credentials.

7.5.2. Test Case 2 – Invalid User Credentials

- **Input**

  Figure 7.13 below, illustrates the input to validate invalid user credentials.

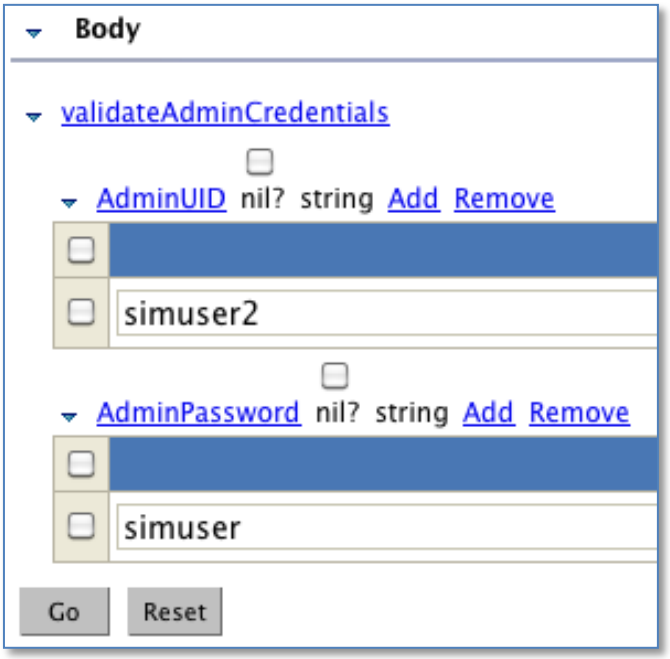

Figure 7.13. Test Case Input – Validate Invalid Admin Credentials.

- **Output**

  Figure 7.14 below, illustrates the output to validate invalid user credentials.

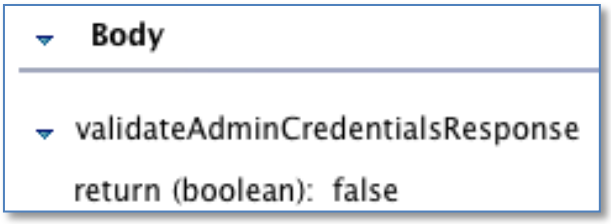

Figure 7.14. Test Case Output – Validate Invalid Admin Credentials.

7.6. ModifyProfile

A call to this task modifies the attributes of a given registered user. The administrator's and user's credentials are mandatory to modify the user's attribute. The attributes are modified for a given user identifier. If the attributes are modified successfully, then the service method returns "true"; otherwise, the result is "false." The attribute format is as follows: <LogicalAttribute_Name>==<value>

- **Input**

  Figure 7.15 illustrates the input to modify the existing user profile using the Modify User task.

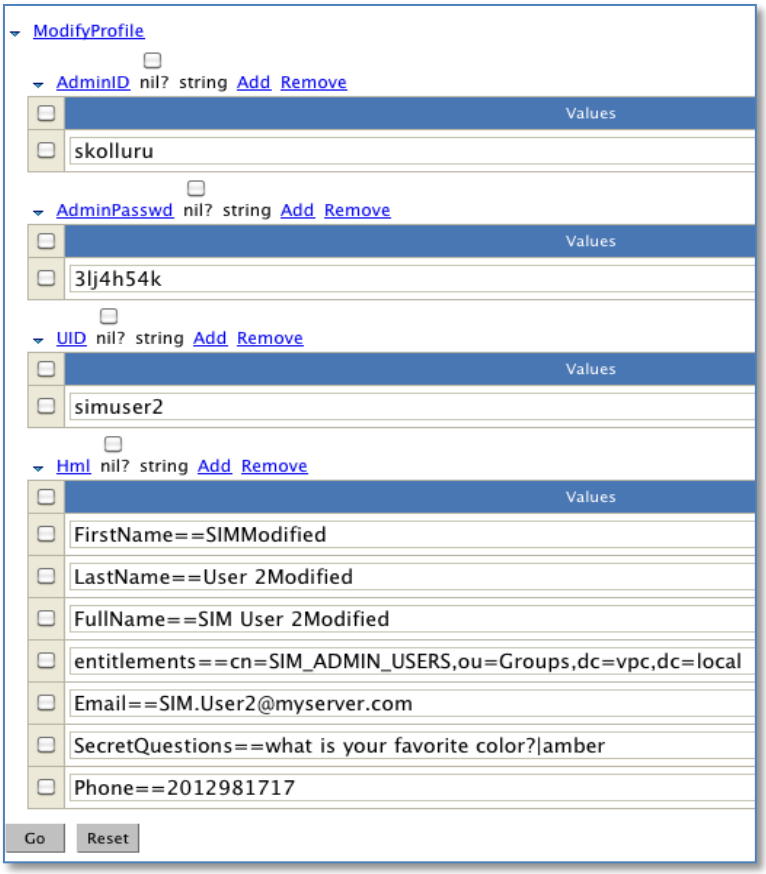

Figure 7.15. Test Case Input – Modify User Information.

- **Output**

Figure 7.16 below, illustrates the output for the Modify User task.

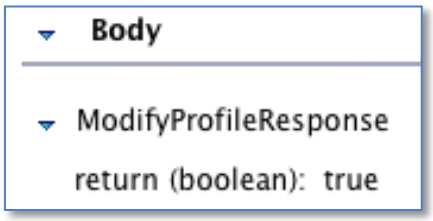

Figure 7.16. Test Case Output – Modify User Information.

- **Assertion for this Output from PrintUserData**

Figure 7.17 displays the modified user information that is invoked by the Modify User task.

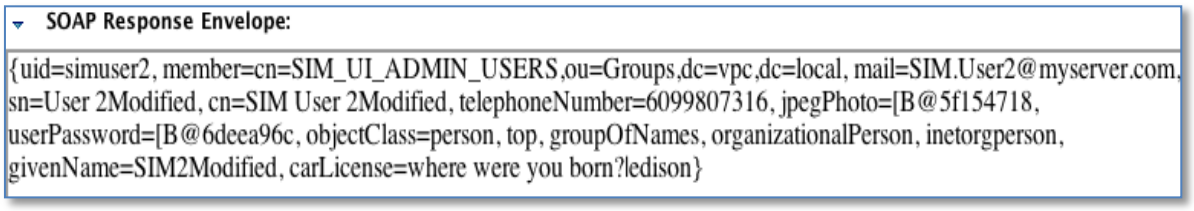

Figure 7.17. Test Case Assert Using Service printUserData – Modify User Information.

7.7. UserBelongsToGroup

A call to this task verifies whether the user belongs to a given group. The service method returns "true" if the user belongs to the given group; otherwise, the result is "false."

7.7.1. Test Case 1 – Give Proper User Identifier and his Group, Returns True

Figure 7.19 below, illustrates the validation being successful when a valid user and groups are given as input in Figure 7.18 below.

- **Input**

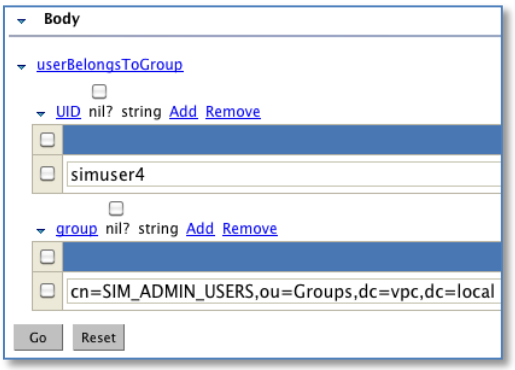

Figure 7.18. Test Case Input – Verify that a User Belongs to a Given Group.

- **Output**

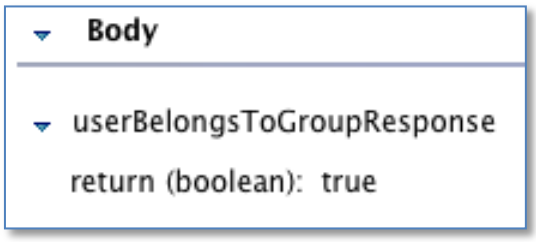

Figure 7.19. Test Case Output – Verify User Belongs To Given Group.

7.7.2. Test Case 2 – Change the group from SIM_ADMIN_USERS to SIM_ADMIN_USERS_1

Figure 7.21 below, illustrates validation being unsuccessful when a valid user and groups are given as input in Figure 7.20 below.

- **Input**

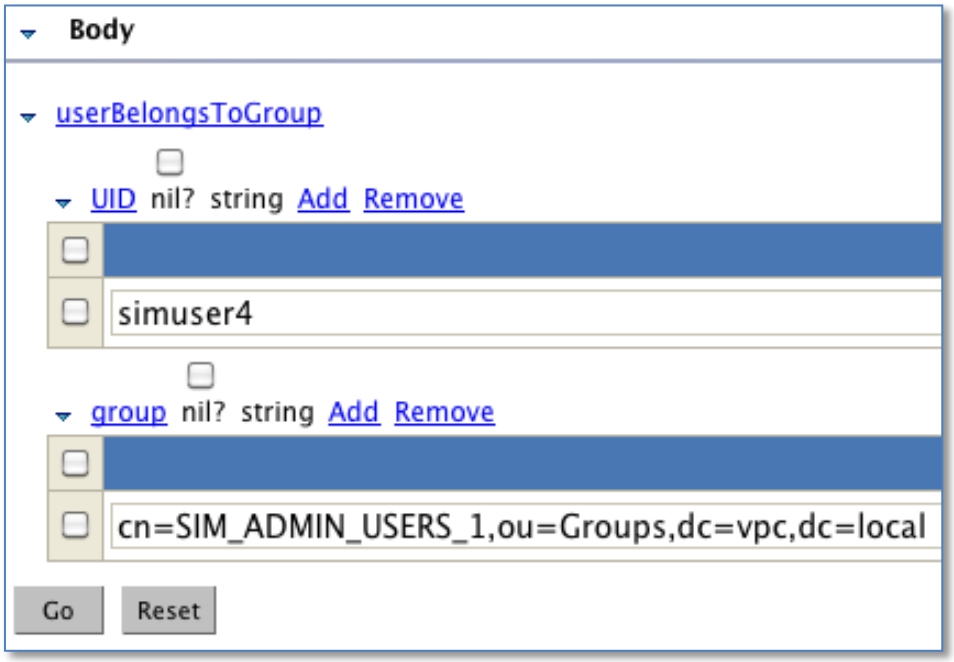

Figure 7.20. Test Case Input – Verify User Doesn't Belong To Given Group.

- **Output**

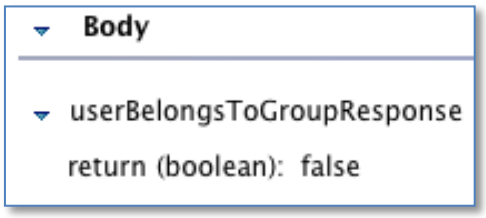

Figure 7.21. Test Case Output – Verify User Doesn't Belong To Given Group.

7.8. LockUser

A call to this task locks the given user. Internally changes the userStatus attribute value of the user to 99. Only administrators can perform this operation. Administrator credentials are mandatory to perform this operation. Use the getUserStatus service to retrieve the user's status.

Figure 7.23 below, illustrates the successful locking of a user account when valid user and administrator credentials are given as input in Figure 7.22 below.

- **Input**

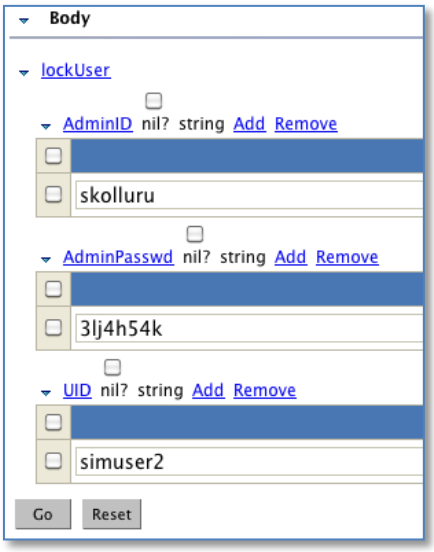

Figure 7.22. Test Case Input – Lock User.

- **Output**

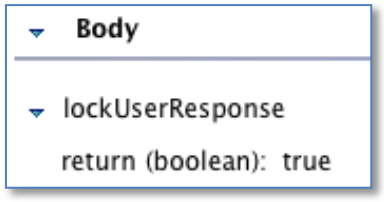

Figure 7.23. Test Case Output – Lock User.

7.9. UnlockUser

A call to this task locks the given user. Internally changes the userStatus attribute value for the user to 100. Only administrators can perform this operation. Administrator credentials are mandatory to perform this operation. Use the getUserStatus service to retrieve the user's status.

Figure 7.25 illustrates the successful unlocking of a locked user account when valid user and administrator credentials are given as input in Figure 7.24 below.

- **Input**

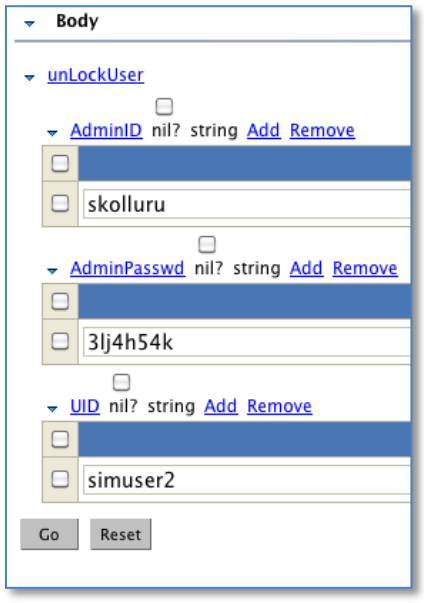

Figure 7.24. Test Case Input – Unlock User.

- **Output**

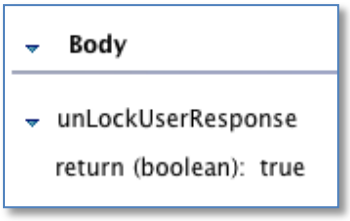

Figure 7.25. Test Case Output – Unlock User.

7.10. GetUserStatus

A call to this task gets the user-status attribute for a given user. If the user status is 99 - Locked User or 100 - Unlocked user. Only administrators can perform this operation. Administrator credentials are mandatory to perform this operation.

7.10.1. Test Case 1 – Unlocked User

Figure 7.27 below, illustrates the successful status for an unlocked user when valid user and administrator credentials are given as input in Figure 7.26 below.

- **Input**

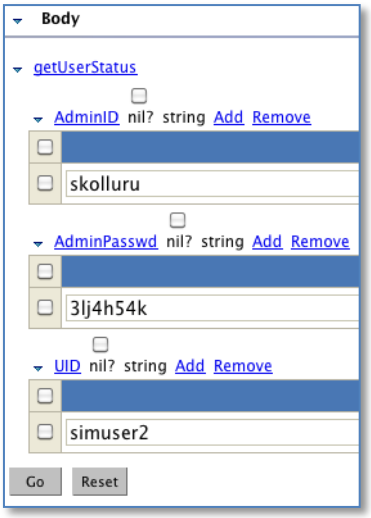

Figure 7.26. Test Case Input – Get User Status for Unlocked User.

- **Output**

Figure 7.27. Test Case Output – Get User Status for Unlocked User.

7.10.2. Test Case 2 – Locked User

Figure 7.29 below, illustrates successful status for a locked user when valid user and administrator credentials are given as input in Figure 7.28 below.

- **Input**

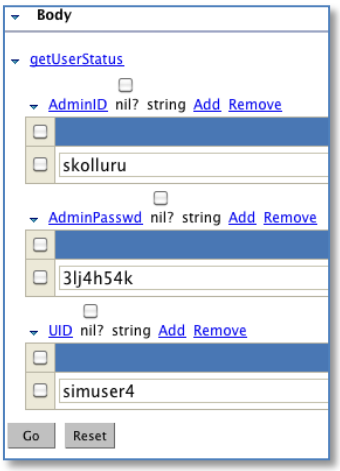

Figure 7.28. Test Case Input – Get User Status for Locked User.

- **Output**

Figure 7.29. Test Case Output – Get User Status for Locked User.

7.11. ChangePassword

A call to this task changes the password for the given user. If the user knows the old password, then he/she can use this method to change the password. The service returns "true" if the password is changed successfully; otherwise, the result is "false." The password verification can be done using the validateCredentials service for further verification.

7.11.1. Test Case 1 – With Proper UserID and Password Information

Figure 7.31 below, illustrates a successful password change when the old and new passwords are given as inputs in Figure 7.30 below.

- **Input**

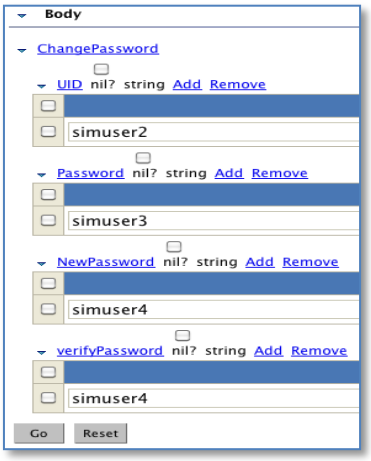

Figure 7.30. Test Case Input – Change Password with Valid Credentials.

- **Output**

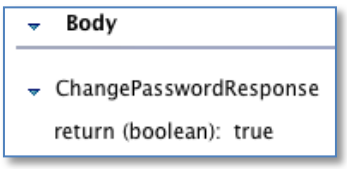

Figure 7.31. Test Case Output – Change Password with Valid Credentials.

7.11.2. Test Case 2 – With Different Old Password

Figure 7.33 below, illustrates an unsuccessful password change when invalid old and new passwords are given as inputs in Figure 7.32 below.

- **Input**

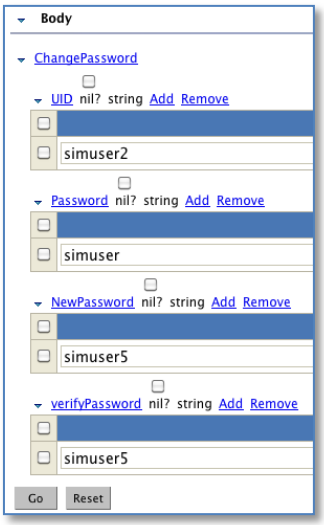

Figure 7.32. Test Case Input – Change Password with Invalid Credentials.

- **Output**

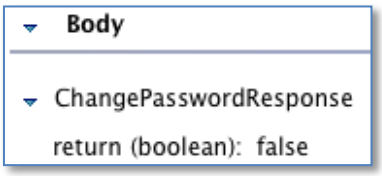

Figure 7.33. Test Case Output – Change Password with Invalid Credentials.

7.12. ForgotPasswordReset

A call to this task resets the password for the given user. The service has four steps to perform the complete operation. The steps are given in the following subsections.

7.12.1. Step 1 – Generate the Transaction ID

Figure 7.35 below, shows the transaction identifier generated for the ForgotPasswordReset task for the inputs given in Figure 7.34 below.

- **Input**

  Transaction ID (TID needs to be sent as “initial”) and give all the user’s information

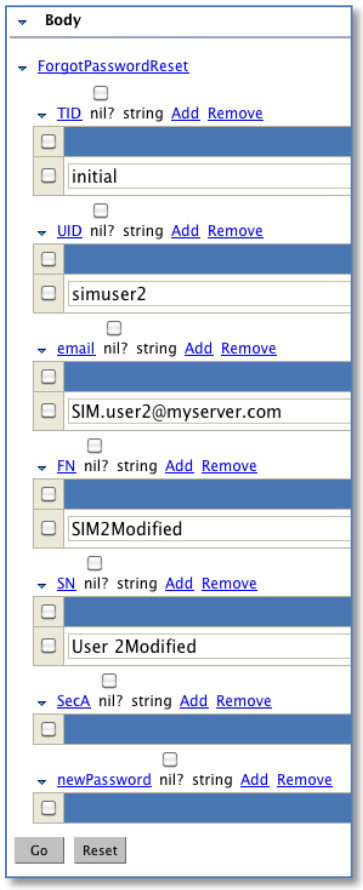

Figure 7.34. Test Case Input – Forgot Password Reset, Generate Transaction.

- **Output**

  Creates a Transaction ID if all the information is properly given

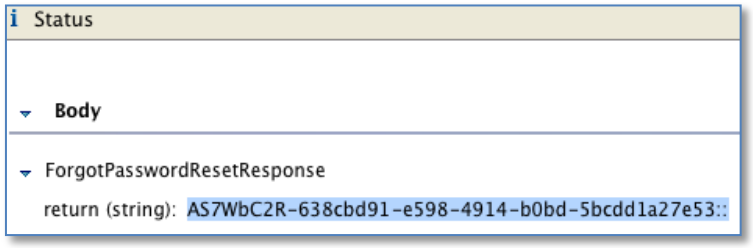

Figure 7.35. Test Case Output – Forgot Password Reset, Generate Transaction.

7.12.2. Step 2 – Get the Security Question

Figure 7.37 below, shows the security question generated for the ForgotPasswordReset – Step 2 task by giving the transaction identifier generated in Step 1 and shown in Figure 7.36 below.

- **Input**

  Get the transaction ID, which is generated in Step 1. Give this Transaction ID, and call the same service again.

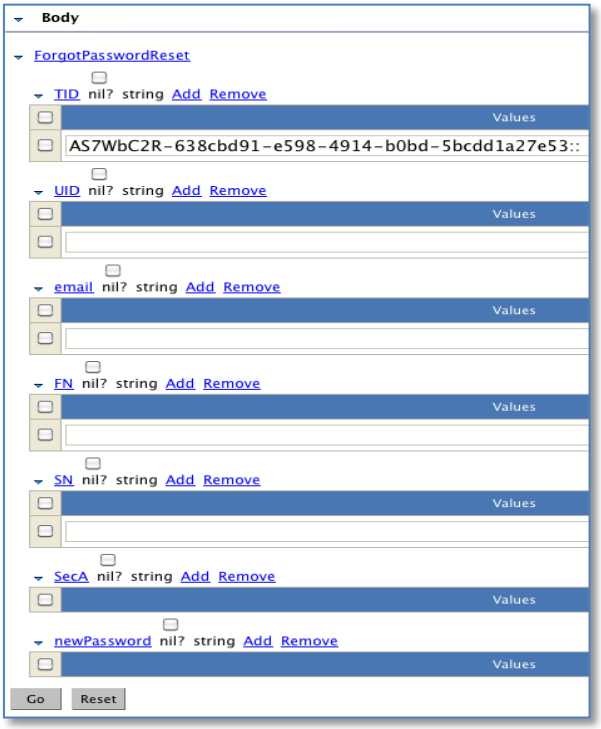

Figure 7.36. Test Case Input – Forgot Password Reset, Get Security Question.

- **Output**

  Gives the Security Question for the User with the Transaction ID

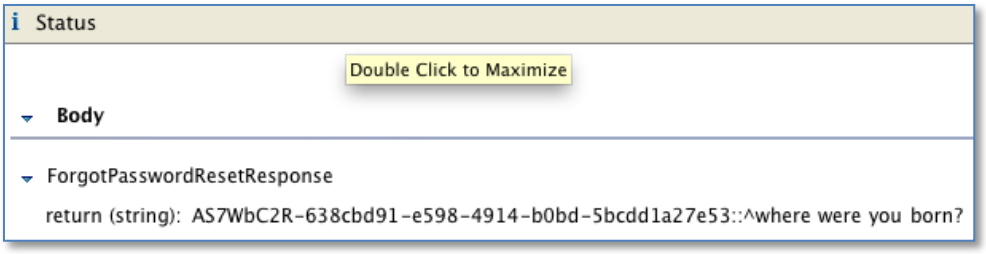

Figure 7.37. Test Case Output – Forgot Password Reset, Get Security Question.

7.12.3. Step 3 – Verify the Security Question

Figure 7.39 shows acceptance of a new password for the ForgotPasswordReset – Step 3 task by giving the transaction identifier generated in Step 2 and the secret-question answer given in Figure 7.38 below.

- **Input**

  Give the Transaction ID and security-question answer

Figure 7.38. Test Case Input – Forgot Password Reset, Verify Security Question.

- **Output**

    If the security-question answer is proper, then the response will ask the developer to choose a new password.

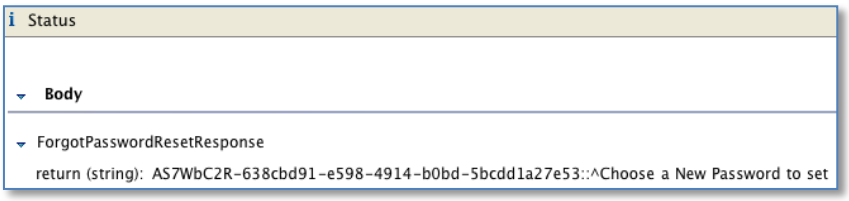

Figure 7.39. Test Case Output – Forgot Password Reset, Verify Security Question.

7.12.5. Step 4 – Change the Password

Figure 7.41 below, illustrates the reset of a new password for the ForgotPasswordReset – Step 4 task by giving the transaction identifier generated in Step 3 and the new password shown in Figure 7.40 below.

- **Input**

    Give the Transaction ID and the new password.

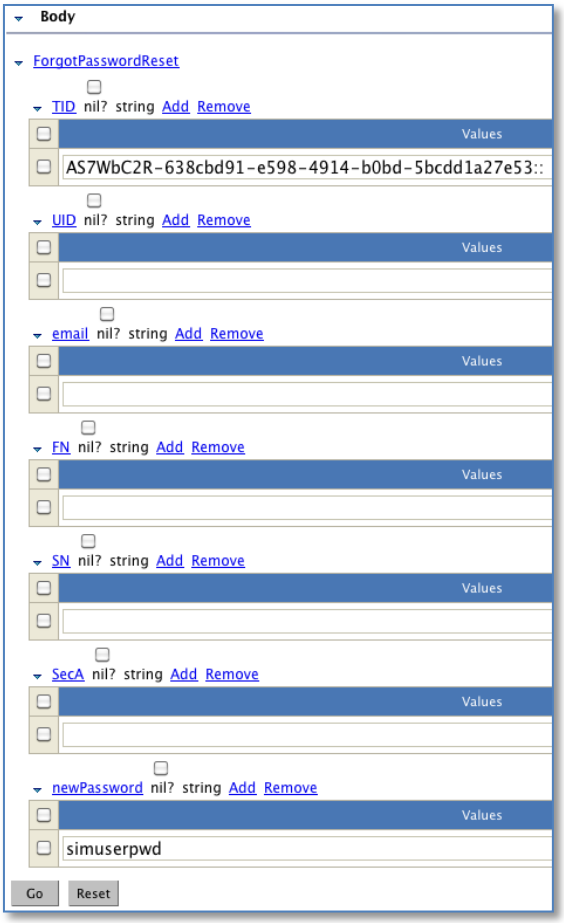

Figure 7.40. Test Case Input – Forgot Password Reset, Change Password.

- **Output**

The “Password is changed successfully” message will be sent to the user.

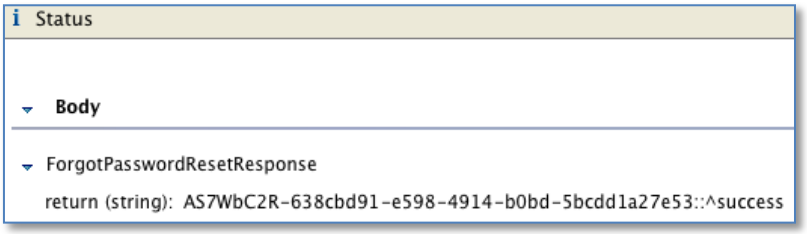

Figure 7.41. Test Case Output – Forgot Password Reset, Change Password.

7.13. Use Service ValidateCredentials to Check if the Password is Modified or Not

Figure 7.43 below, illustrates the successful validation of the credential input given in Figure 7.42 below. The input credentials are taken from the ForgotPasswordReset task that was executed previously.

- **Input**

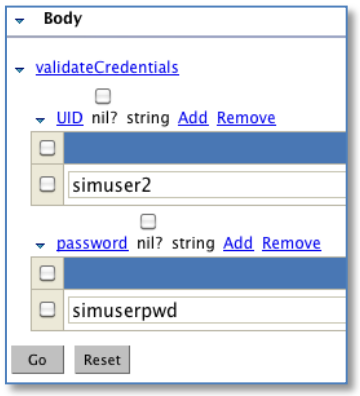

Figure 7.42. Test Case Input – Validate User Credentials.

- **Output**

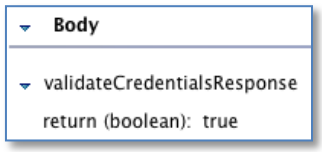

Figure 7.43. Test Case Output – Validate User Credentials.

7.14. ForgotUID

A call to this task gets the user identifier if the user has forgotten his identifier. The service has three steps to perform the complete operation. The steps are given in the following subsections.

7.14.1. Step 1 – Generate Transaction ID

Figure 7.45 below, illustrates the generation of the Transaction ID for the ForgotUserID task by giving the transaction identifier as “initial” with the required inputs shown in Figure 7.44 below.

- **Input**

  Transaction ID (TID needs to be sent as initial) and give all the information of the user.

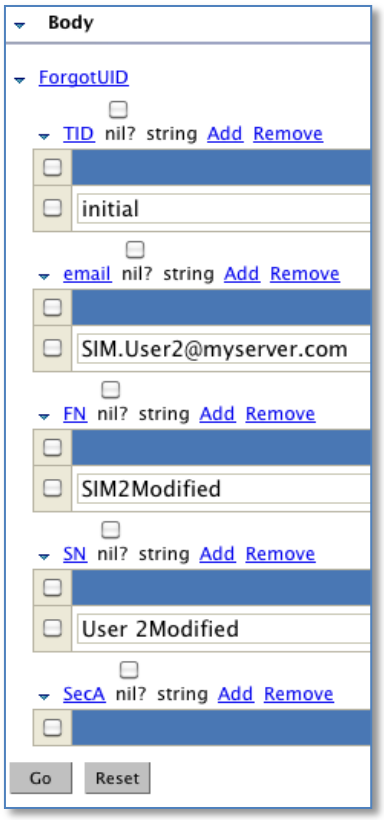

Figure 7.44. Test Case Input – Forgot User, Generate Transaction ID.

- **Output**

  Creates a Transaction ID if all the information is given properly

Figure 7.45. Test Case Input – Forgot User ID, Generate Transaction ID.

7.14.2. Step 2 – Get the security question

Figure 7.47 below, illustrates the retrieval of the security question for the ForgotUserID – Step 2 task by giving the transaction identifier generated in Step 1 as the input shown in Figure 7.46 below.

- **Input**

  Get the transaction ID which is generated in Step 1. Give this transaction Id, and call the same service again.

Figure 7.46. Test Case Input – Forgot User ID, Get Security Question.

- **Output**

  Gives the security question for the given user with the transaction id.

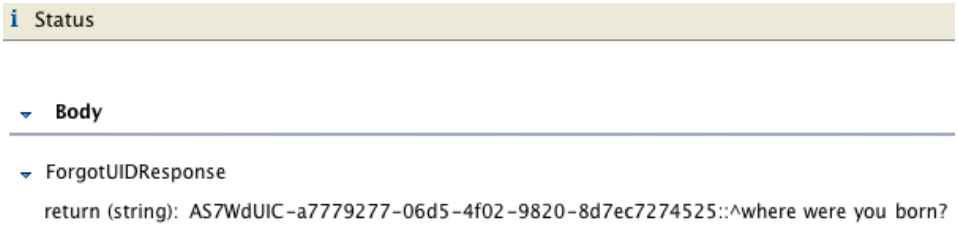

Figure 7.47. Test Case Output – Forgot User ID, Get Security Question.

7.14.3. Step 3 – Get the UserID

Figure 7.49 illustrates the retrieval of the user identifier for the ForgotUserID – Step 3 task by giving the transaction identifier and security-question answer generated in Step 2 as the inputs specified in Figure 7.48 below.

- **Input**

  Give the transaction ID and security question answer.

Figure 7.48. Test Case Input – Forgot User ID, Get User ID.

- **Output**

  If a valid answer to the security question is submitted, then the response from the service will the user identifier.

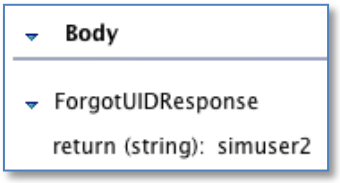

Figure 7.49. Test Case Output – Forgot User ID, Get User ID.

7.15. ForgotPassword

A call to this task resets the password for a given user and sends the password to him/her. Figure 7.51 below, illustrates the successful sending of the new password by verifying the valid user identifier and email as the inputs given in Figure 7.50 below.

- **Input**

  Entering UID and verifyEmail values and submitting returns true if UID and email were validated.

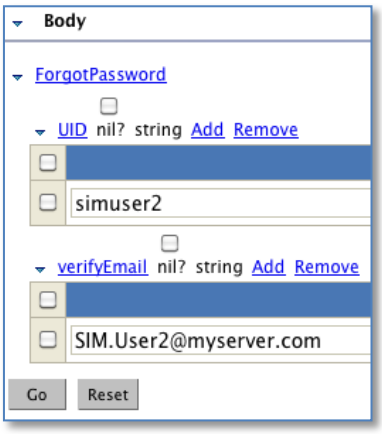

Figure 7.50. Test Case Input – Forgot Password.

- **Output**

  Figure 7.51 below shows that the service returns true.

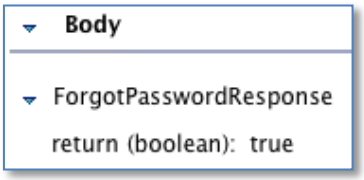

Figure 7.51. Test Case Output – Forgot Password.

8. ADMIN UI USE CASES AND SCREENSHOTS

In the SIM Admin UI Console, there are various use case. Some of those uses cases are explained below:

8.1. Selecting the SIM Home Directory

The SIM_HOME directory can be given as a system attribute. If the user input is invalid for the SIM_HOME directory, the system will show a screen prompting the user to provide a valid SIM_HOME location. Figure 8.1 below, illustrates the selection of the SIM Home Directory.

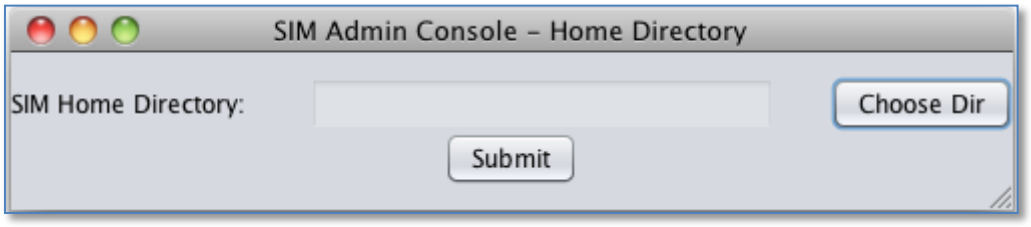

Figure 8.1. SIM Admin UI Console – Select SIM Home Directory.

Figure 8.2 below, shows the error message for the invalid selection of the SIM Home Directory.

Figure 8.2. SIM Admin UI Console – Error for Improper Selection of the SIM Home Directory.

8.2. Login Screen

Once a valid SIM_HOME directory is selected, the user will be taken to the admin login screen. Figure 8.54 below, shows the login screen for giving the LDAP administrator credentials.

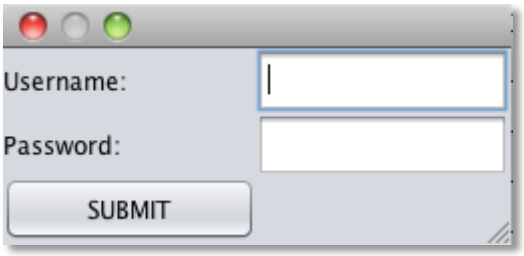

Figure 8.3. SIM Admin UI Console – Login Screen to Collect LDAP Admin Credentials.

If the credentials are not valid, then the system displays the error message "incorrect login or password." Figure 8.4 below, shows the error message for the invalid LDAP administrator's credentials.

Figure 8.4. SIM Admin UI Console – Invalid Admin Credentials.

If the login is successful, then the user is taken to the admin console.

8.3. The SIM Admin Console for System Configuration

Figure 8.5 below, shows the SIM administration console to edit properties for the system configuration file.

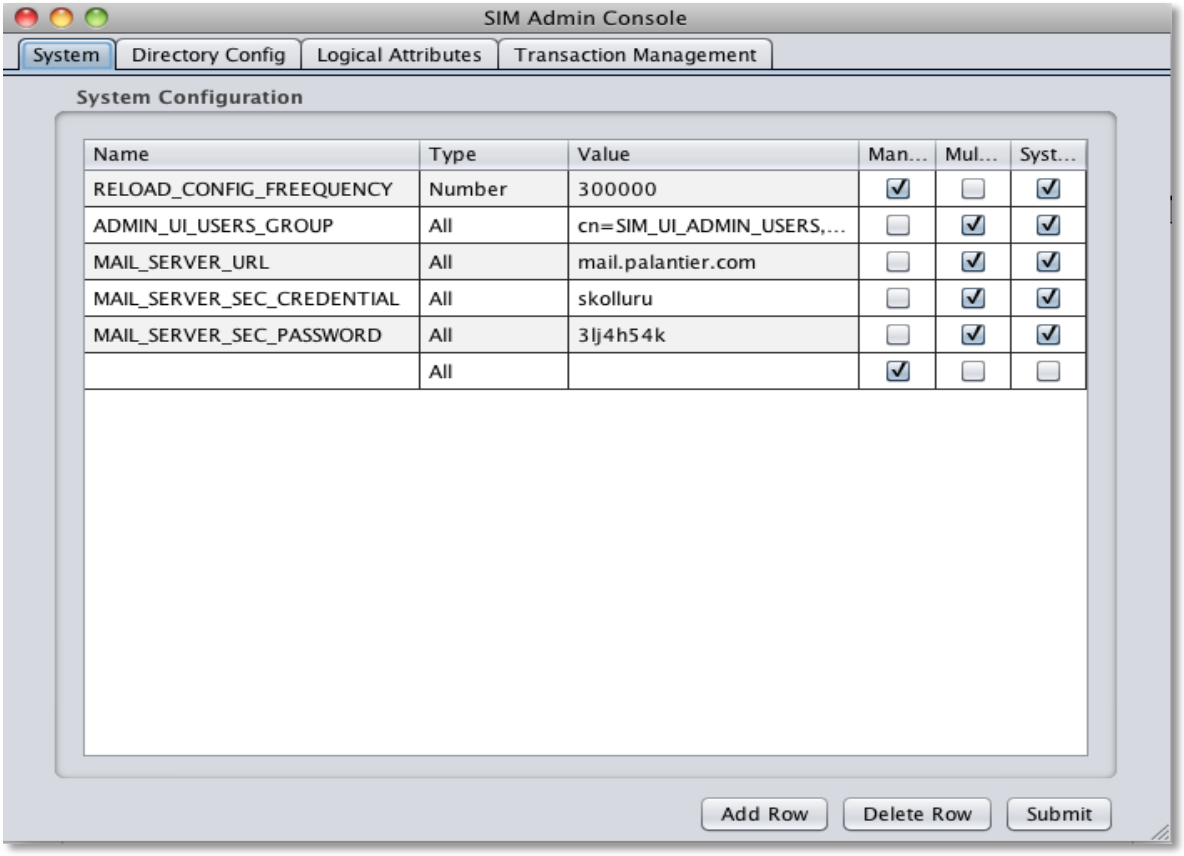

Figure 8.5. SIM Admin UI Console – System Configuration.

8.4. Directory Configuration Screen

Figure 8.6 below, shows the SIM administration console to edit properties for the system configuration file. This screen loads all the properties which are used by the system configuration. Here, the user can add new attributes by clicking the Add Row button. This action will not affect the configuration XML file directly. The user has to click the Submit button to take the new values.

- Add Row – Adds a new attribute.

- Delete Row – Deletes an attribute
- Submit – Submits the new changes to the configuration XML file. The file for system configuration is SystemConfig.xml.

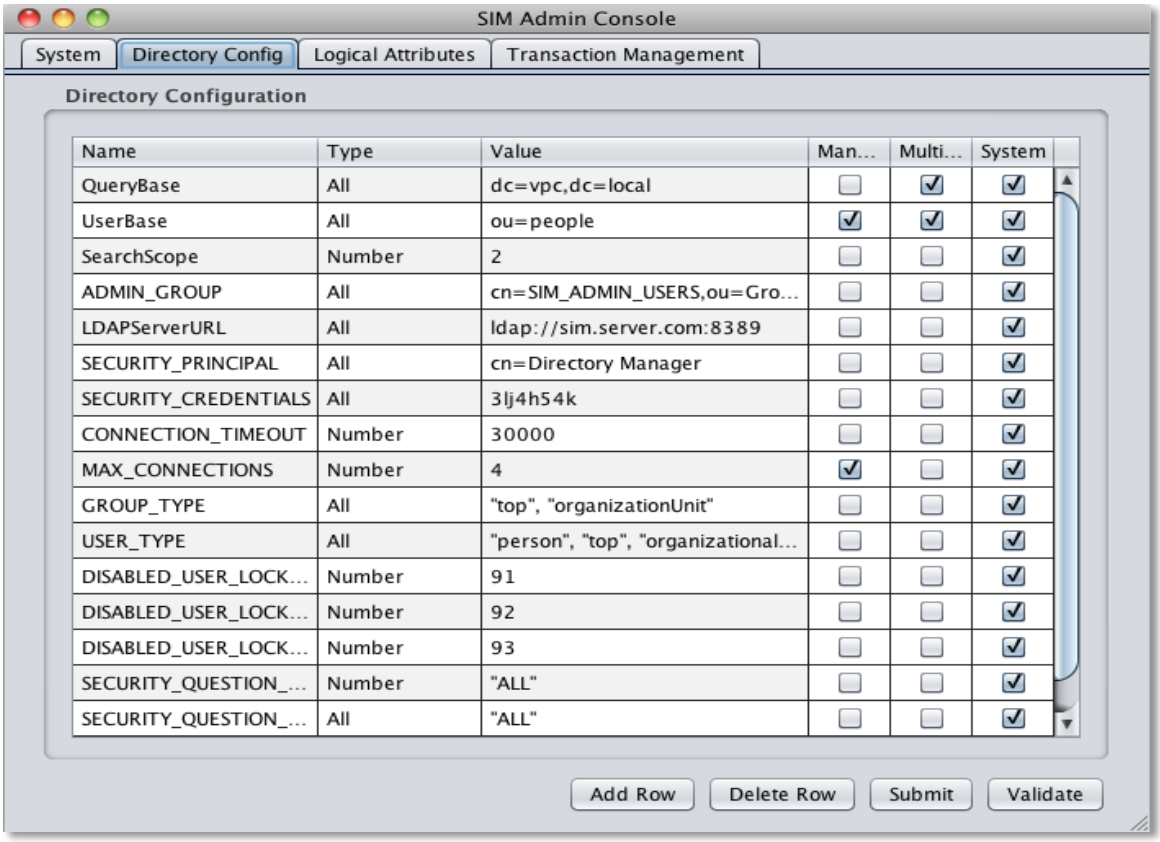

Figure 8.6. SIM Admin UI Console – Directory Configuration.

The above screen loads all the properties which are used by the directory configuration. Here, the user can add attributes by clicking the Add Row button. This action will not affect the configuration XML file directly. The user has to click the Submit button to apply the new values/changes.

- Add Row – Adds a new attribute
- Delete Row – Deletes an attribute

- Submit – Submits the changes to the configuration XML file. The file for system configuration is DirectoryConfig.xml.
- Validate – Validates the LDAP credentials. If the credentials are valid, it shows the message.

Figure 8.7 below, shows the successful validation message if the given LDAP URL and credentials are valid on the directory configuration screen.

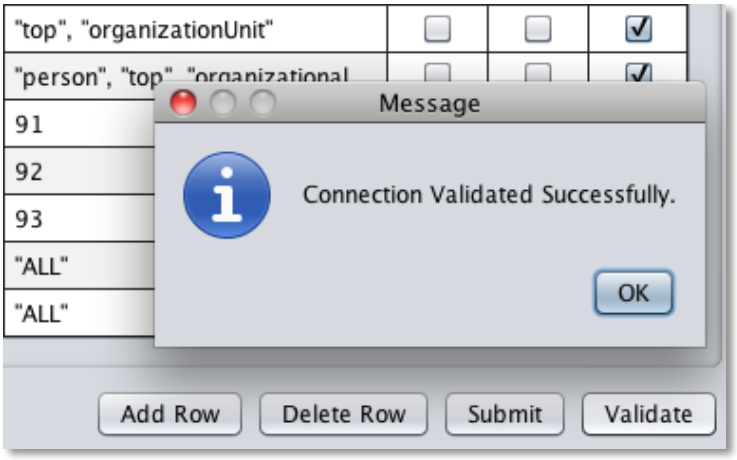

Figure 8.7. SIM Admin UI Console – Successful LDAP Credential Validation Message.

8.5. Logical Attributes Configuration Screen

Figure 8.8 below, shows the SIM administration console to view and edit the Logical Attributes file.

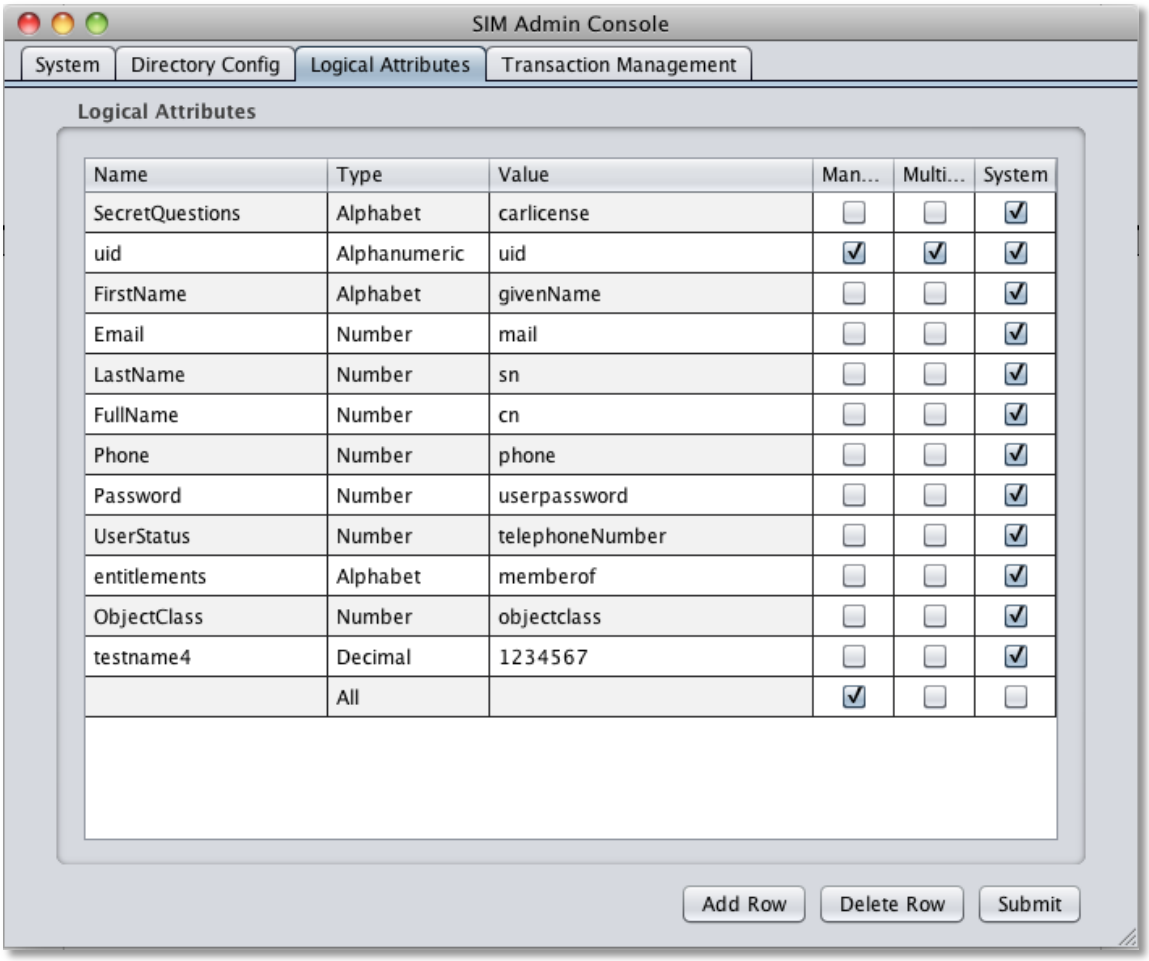

Figure 8.8. SIM Admin UI Console – Logical Attribute Configuration.

The above screen loads all the properties which are used by the logical attributes configuration. Here, the user can add new attributes by clicking the Add Row button. It will not affect the configuration XML file directly. The user has to click the Submit button to insert new values.

- Add Row – Adds a new attribute
- Delete Row – Deletes an attribute
- Submit – Submits the changes to the configuration XML file. The file for system configuration is LogicalAttributes.xml

8.6. Transaction Management Configuration Screen

Figure 8.9 below, shows the SIM administration console to view and edit the Transaction Management Attributes file.

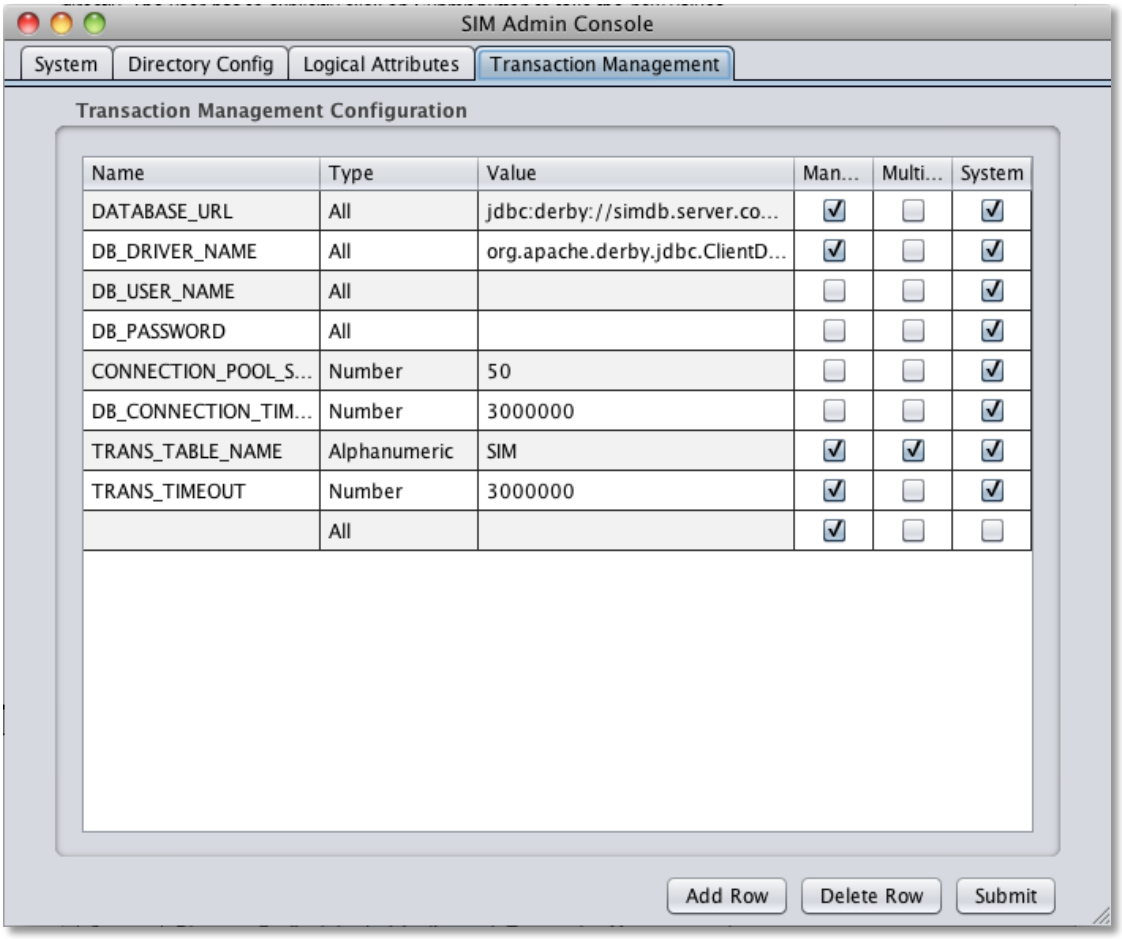

Figure 8.9. SIM Admin UI Console – Transaction Management Configuration.

The above screen loads all the properties which are used by the Transaction Management configuration. It contains all the database connections and supported attributes. Here, the user can add attributes by clicking the Add Row button. Such action will not affect the configuration XML file directly. The user has to click the Submit button to take the new values.

- **Adding a Row**

Adds a new attribute. Not all new attributes are system attributes. Figure 8.10 below, highlights the Add Row button for creating a new configuration entry.

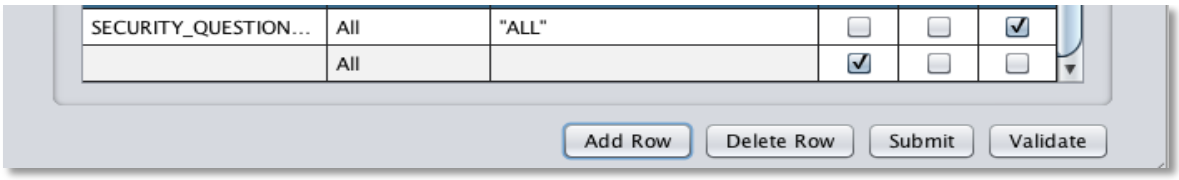

Figure 8.10. SIM Admin UI Console – Add Row.

The user can specify the attribute value type. Figure 8.11 below, shows the allowed value types for the configuration entry.

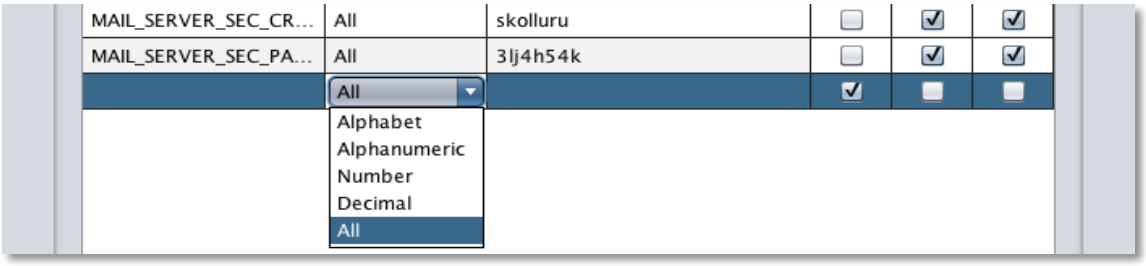

Figure 8.11. SIM Admin UI Console – Attribute Value Type.

If the attribute value does not match the proper value type, the user cannot leave the screen until he/she hits the escape button.

Figure 8.12 below, shows invalid input for the value given for a new configuration entry.

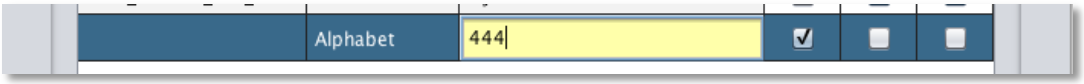

Figure 8.12. SIM Admin UI Console – Validate Input.

Only the values are editable, and all other fields are non-editable. Figure 8.13 below, shows valid input for the value given for the existing configuration entry.

Figure 8.13. SIM Admin UI Console – System Attribute Edit Value.

- **Deleting a Row**

  Deletes a new attribute. The system attributes row cannot be deleted. The button does not have any effect. If the operation is successful, then it shows the message that the row has successfully been deleted.

  Figure 8.14 shows the message after deleting the existing configuration entry.

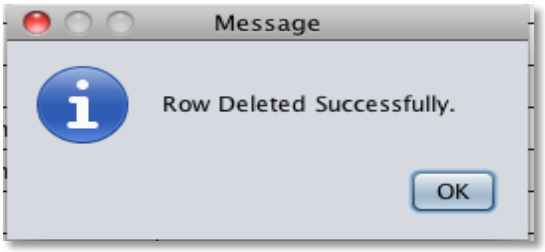

Figure 8.14. SIM Admin UI Console – Delete Row Confirmation Message.

- **Attributes Submit**

  This action writes all the attributes and their values to the corresponding XML file. Figure 8.15 below shows the successful message after the user clicks the Submit button on the configuration screen.

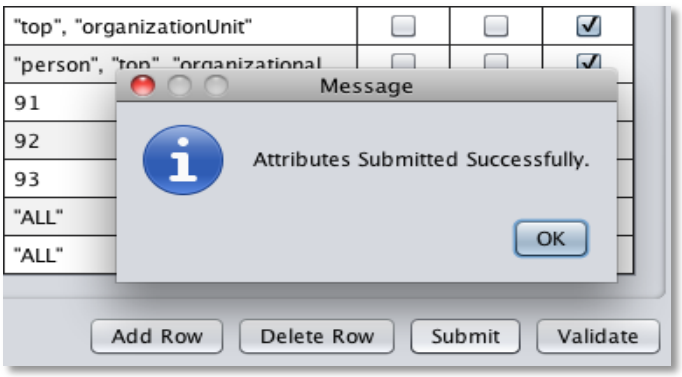

Figure 8.15. SIM Admin UI Console – Submit.

9. SUMMARY

9.1. Conclusion

While SIM is not an invention and there are commercial identity management products available, SIM forms the simplest and lightweight implementation while preserving the core requirements with the goal of managing a user's identity.
This paper forms a successful study of identity management in a case where user information is stored in any LDAP repository conforming to V3 (RFC 2251) while making use of JNDI (RFC 2713) to interface with the LDAP.

We reached our goal of developing an identity management system that can be a black box to the target audience and can make use of the features by following the documentation. The SIM can be readily available in the form of a virtual machine. SIM web services can be leveraged to create customized identity management tasks. The SIM's validation framework makes the system intrusion proof despite lacking a concrete validation check for the presentation layer. Changes to the SIM are dynamic and do not require a restart.

SIM fits well with any business or organization that stores users' identity information in an LDAP compliant repository. This is at least two-thirds of the world organizations. SIM makes enterprise web applications dynamic and platform independent for organizations. Migrating to other platforms and switching access-management systems will still fit with the SIM.

In conclusion, after studying identity manager specification, sticking to LDAP v3 compliant backend, making decisions carefully around logical implementation, and making use of good design patterns, we are able to provide a platform-independent, lightweight identity manager with reusable components and APIs that open doors for more utilities and applications in the enterprise world, that operate against LDAP v3 compliant data repository.

9.2. Future Enhancements

We see few future possible enhancements. They are listed below

9.2.1. SIM as a Cloud Service (SaaS)

Because of the segregation of the SIM system into various levels and its virtualization attributes, SIM can be a SaaS system [15]. As described in Section 3.19, the SIM system can be quickly configured on multiple virtual machines in the cloud. According to an article by Ellen Messmer [16], “Gartner is predicting the cloud-based security services market, which includes secure email or web gateways, identity and access management (IAM), remote vulnerability assessment, security information and event management to hit $4.13 billion by 2017.”

Gartner, Inc. is an American information-technology research and advisory firm that is headquartered in Stamford, Connecticut.

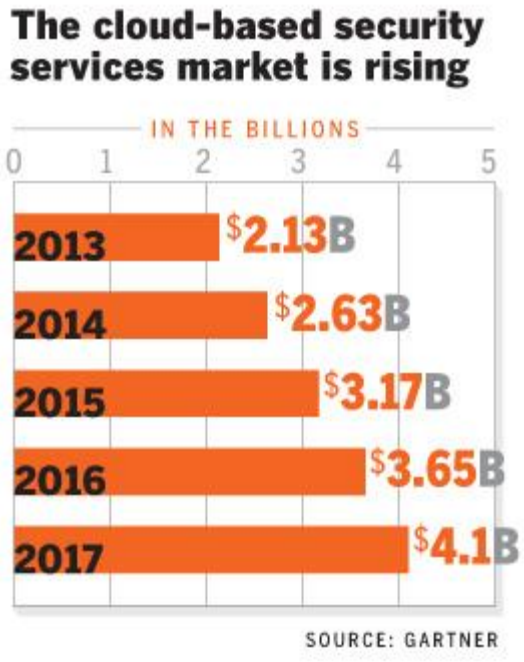

Figure 9.1. Gartner Report Depicting the Possible Growth in the Cloud-based Security Market.

Figure 9.2 below illustrates the SIM SaaS system in the cloud.

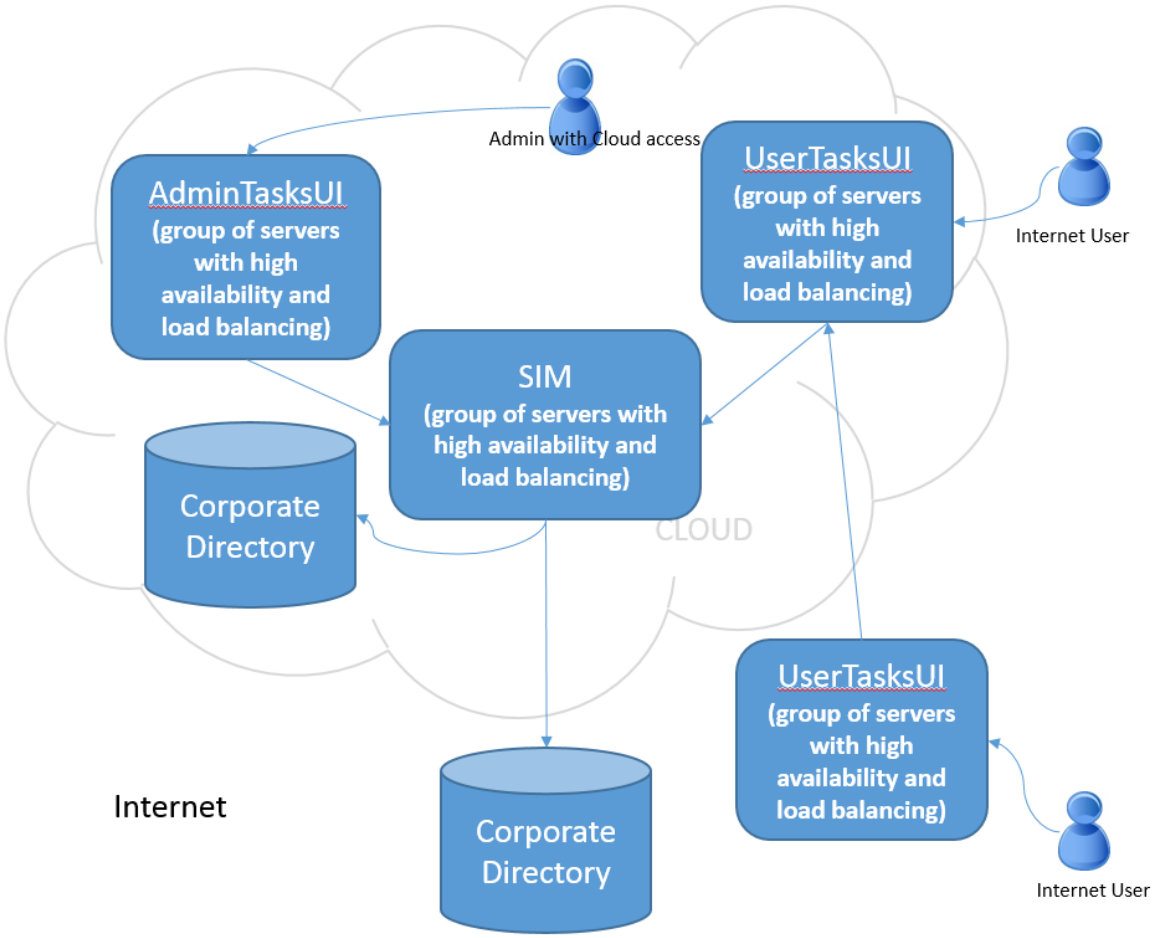

Figure 9.2. SIM SaaS System in the Cloud.

9.2.3. SIM Command Console

The SIM framework-based applications are not limited to the web. The framework can be used in the desktop-support arena. Here is an idea. (There are similar command-line administration utilities for application servers such as Weblogic and Websphere.) The user connects to the SIM application from the desktop via the SIM console and runs commands just like in a Unix shell. Figure 9.3 below, illustrates such a utility.

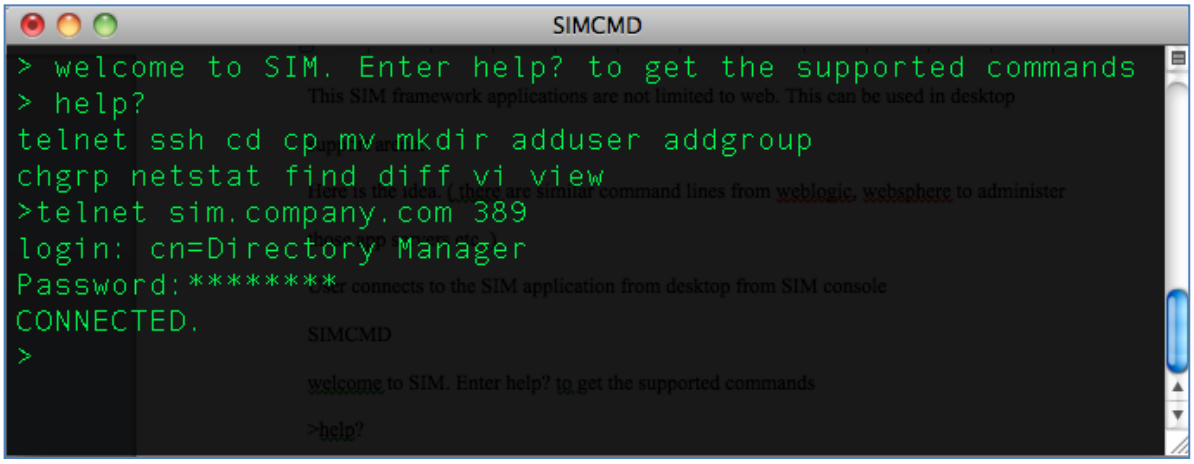

Figure 9.3. SIM Command Console.

9.2.4. Supporting Multiple Attempts to Answer the Security Question

In Section 3.9, organizations may want to allow multiple attempts to answer security questions. In this case, it has to be taken care of at the presentation layer by remembering the firstname, lastname, and email address and then calling forgotPassword again on the user's behalf.

9.2.5. SSL Support

SSL is not supported. It can be a future enhancement. It requires modifications in the physical layer only. Specifically, changes to the simldapops class that acquires the LDAP connection are required.

9.2.6. Transaction Integrity and Persistence

A separate thread to clean up transactions in the database was in mind. At startup, a thread will be initiated to look at all the transactions. If a transaction is in the final stage and in "commit" mode, those methods will be executed, and the transaction will be cleaned up. This thread will remove transactions that are not in the commit mode. This approach gives the transaction persistence and makes the database a persistent store.

9.2.7. Dynamic USER_TYPES

SIM only reads the LogicalDirectory once in the case of USER_TYPE because the USER_TYPE in the Transition Layer is “ENUM.” USER_TYPE can be implemented as a regular Java class to load the USER_TYPE dynamically, upon changes.

10. REFERENCES

[1] Wikipedia authors. (2012, Feb). *Data Security* [Online]. Available: http://en.wikipedia.org/wiki/Data_security

[2] Robert Skoczylas and Marina Sum. (2008, Jan 3). *Developing Secure Applications with Sun Java System Access Manager (2nd ed.)* [Online]. Available: http://www.oracle.com/technetwork/systems/articles/secureapps-136823.html

[3] Robert L. Mitchell. (2006, Nov 20). *Stepping into Identity Management* [Online]. Available: http://www.computerworld.com/s/article/272689/Stepping_Into_Identity_Management

[4] Nancy Davis Kho. (2009, Mar 27). *The Changing Face of Identity Management* [Online]. Available: http://www.econtentmag.com/Articles/Editorial/Feature/The-Changing-Face-of-Identity-Management-53162.htm

[5] Brian Arkills. (2003). *LDAP Directories Explained: An Introduction and Analysis*. Addison-Wesley Professional.

[6] Wikipedia authors. (2012, June). *SOAP* [Online]. Available: http://en.wikipedia.org/wiki/SOAP

[7] David Flanagan et al. (1999). "Chapter 6: JNDI" in *Java™ Enterprise in a Nutshell: A Desktop Quick Reference.* O'Reilly & Associates.

[8] Eetu Heino. (2011). "Evaluating Financial benefits of an Identity Management Solution – CASE Logica," M.S. thesis, Department of Business Technology, Aalto University., Greater Helsinki, Finland.

[9] Frank Ramdoelare Tewari. (2005). "IDENTITY MANAGEMENT DEFINED," M.S. thesis, Department of Informatics and Economics. Erasmus University, Rotterdam., Netherlands.

[10] Computer Associates. *CA Identity Manager: Programming guide for Java* [Online]. Available: https://supportcontent.ca.com/cadocs/0/CA%20Identity%20Manager%20r12%205%20SP6-ENU/Bookshelf_Files/PDF/im_dev_enu.pdf

[11] IBM. *Identity Management Advanced Design for IBM Tivoli Identity Manager* [Online]. Available: http://www.redbooks.ibm.com/redbooks/pdfs/sg247242.pdf

[12] Unknown. *Singleton Design Pattern* [Online]. Available: http://sourcemaking.com/design_patterns/singleton

[13] Unknown. *ObjectPool Design Pattern* [Online]. Available: http://sourcemaking.com/design_patterns/object_pool

[14] Shiv Pal Singh. (2002, Dec 17). .*NET Web Services Tutorial* [Online]. Available: http://www.codeguru.com/csharp/csharp/cs_webservices/tutorials/article.php/c5477

[15] Software & Information Industry Association. (2001, Feb). *Software as a Service: Strategic Backgrounder* [Online]. Available: http://www.siia.net/estore/pubs/SSB-01.pdf

[16] Ellen Messmer. (2013, Oct 31). *Gartner: Cloud-Based Security as a Service Set to Take off* [Online]. Available: http://www.networkworld.com/article/2171424/data-breach/gartner-cloud-based-security-as-a-service-set-to-take-off.html

[17] Unknown. *SQLite vs Apache Derby* [Online]. Available: http://www.sqlite.org/cvstrac/wiki?p=SqliteVersusDerby

APPENDIX. WHICH LIGHTWEIGHT DATABASE TO CHOOSE FOR THE TRANSDB

We chose Derby over SQLite because Derby has replication and failover capabilities as well as other desired features that are not plugged into SQLite by default and need programming implementation.

Below is a comparison [17]:

- **Overall**

  Both SQLite and Derby operate directly from a disk. Only parts of the database file(s) that are needed to carry out the requested operations are read.

- **Zero-Administration**

  Both SQLite and Derby offer zero-administration, embeddable SQL database engines. SQLite stores all the data in a single cross-platform disk file. Derby spreads its data across multiple disk files.

- **Host Language Support**

  SQLite is written in ANSI-C. It supports bindings with dozens of languages, including Java. You cannot use Derby with languages that do not use the JAVA Virtual Machine(VM or JVM). Derby is written in Java and is, thus, usable only by Java and scripting languages that run on the Java VM (Jython, JRuby, Jacl, etc.) and is currently only exposed via a JDBC driver. (There is an ODBC driver for Derby, but it is no longer maintained.) However, this JDBC driver is 100% based and, hence (with occasional glitches), runs cross-platform on any JAVA VM with a single binary distribution. (SQLite is very portable as well, but you would have to maintain multiple binaries if shipping a cross-platform product.)

- **SQL Language Support**

  Derby supports all of SQL92 and most of SQL99. SQLite only supports a subset of SQL92, although the supported subset is large and covers the most commonly used parts. Some differences are pointed out below. One distinct difference is that Derby supports RIGHT JOINS and FULL OUTER JOINS while SQLite does not.

- **Memory Utilization**

  The code footprint of SQLite is less than 250 KB. The code footprint for Derby is about 2000 KB when compressed and is, thus, more than 8 times larger. However, a large amount of this difference is due to Derby's extensive localization and collation support for multiple built-in languages. In general, the memory utilization of Derby is considerably higher than SQLite, occupying several megabytes of memory.

- **Concurrency**

  SQLite allows multiple simultaneous readers and a single writer. Multiple processes can have the database open simultaneously.

  Derby only allows a single process to have the database open at a time in its embedded mode. However, Derby also offers a full client/server mode.

  In client/server mode, Derby supports giving multiple processes access to the database with row-level locking. The client/server mode, of course, requires that there be a thread or process available to act as a server and is less productive than embedded mode.

- **Roles, Security, and Schemas**

  Derby supports full encryption (see below). In addition, it supports multiple databases and full SQL role granting. Derby supports the SQL schemas for separation of data in a single database and full user authorization.

  SQLite is largely a single database at a time engine. The ATTACH DATABASE command can be used to partially ameliorate this. Because of this design, neither SQL roles nor schemas are implemented. Typically, access to the SQLite disk file grants full access to the caller. This is not a defect but is done by design.

- **Callable Procedures**

  Derby has support for this built in. SQLite allows you to "fake" these but has no comparable feature.

- **Typing/Keys**

  SQLite only supports basic types; it is mostly a typeless system which can be very nice in some cases and annoying in others. Derby supports a wide variety of data types, including XML. The foreign key and referential integrity support is also complete.

- **Built-in Utilities**

  Derby has built-in, online backup/restore and database consistency check utilities. SQLite has a basic database consistency check utility, but no corresponding online backup/restore. You must close connections to the file to get a consistent backup. But, in case of SQLite, concurrent usage by multiple programs is not usually a design goal.

- **Encryption/Compression**

  Derby has built-in support for encryption and compression. SQLite has some optional add-ins, but they are not part of the standard library.

- **Collation Support**

  Both the products support custom collation functions. Derby comes with many multilingual collations and localizations built in; these features have to be manually added to the core SQLite package by the programmer.

- **Case-Sensitive LIKE**

  Derby has a case-sensitive LIKE operator while SQLite does not. Derby supports custom collation and indices, like SQLite, but doesn't ship with a built-in case-insensitive option.

- **Pagination**

  Derby now fully supports pagination. SQLite fully supports the non-standard-but-extremely useful LIMIT and OFFSET commands that Postgres and MySQL have adopted.

- **Replication/Failover**

  Derby offers a basic master-slave replication system. SQLite does not have any such mechanism. (Again, this replication feature is rarely part of the design specification for usage of SQLite.)

  There are JDBC clustering drivers available for Derby that allow failover to another Derby server.

- **Crash-Resistance**

  Both Derby and SQLite are ACID compliant in their default configurations. Their databases will survive a program crash or even a power failure.

- **Database File Size**

  No data are about the relative sizes of the database files for SQLite and Derby are currently available. Both SQLite and Derby support the compression of database files; however, SQLITE's VACUUM command makes the database inaccessible during its run; But, Derby's analagous procedure can be run online.

- **Full Text/Virtual Table**

  Derby has no similar functions to compare to SQLite.

- **Encryption**

  The ability to encrypt databases is built into Derby. For SQLite, stubs are left for the implementation, but the implementation itself is an extra-cost feature.

- **Speed**

  No data are currently available for the relative speed of SQLite and Derby database engines. Their query operation is similar in function. Relative speed of different queries depends on cache-utilization, query plan optimization, and implementation. However, one should be prepared for an unpredictable speed penalty when using Derby under different VMs even with same hardware/operating system.