# Chapter 1 Introduction

**1.1 Introduction to the Ningbo E-government Cloud**

Cloud computing is a model for enabling convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction.

The Ningbo E-government cloud computing center is designed to play the role of the support platform for cloud applications of the smart city and electronic government project, both of which have swept the world. Based on centralized management aiming at optimizing configurations of public infrastructure applications, it implements a nice loose coupling layout between the applications and middleware. Using the idea of the bus, not only does the cloud improve the interagency collaboration between businesses across different departments, but also create a positive environment of application interaction.

**1.2 Introduction to the PaaS Cloud Architecture**

Platform as a service (PaaS) is a category of cloud computing services that provides a platform allowing customers to develop, run, and manage applications without the complexity of building and maintaining the infrastructure typically associated with developing and launching an app.

According to characteristics of the network in Ningbo E-government cloud computing center, my project team decided to deploy two sets of PaaS cloud architecture in the internal and external network. The applications on the left PaaS cloud will provide a collection of smart city web services for the individuals and enterprises. At the same time, the right PaaS cloud will run governmental applications sternly. It is worth mentioning that applications running on the exclusive right one mostly migrate from original government services platform for the sake of rapid development and reusability of business logic. Indistinguishably and expectantly, the PaaS cloud architecture makes a contribution to the convenience and efficiency of construction, deployment and management in software engineering field because of the unified planning and management of the resources, the elastic expansion of resources and the fabulously rapid application deployment in a timely manner.

Each PaaS Cloud Architecture consists of five participants (the structure is shown as figure 1.2). In addition to third-party smart city applications outside, they are, respectively: the underlying IaaS layer which adopts the latest OpenStack version called *Mitaka,* the enterprise services bus (ESB) which is in charge of managing and monitoring public or restricted services, the CloudFoundary Web(cfWeb) which is responsible for managing and monitoring various complex software running platforms such as java or python environment and their dependent middleware ActiveMQ as an illustration and the last but crucially important one on the SaaS cloud domestic consumers’ point of view, the Unify Identity Authentication and Access Control System (UIAACS) which reconstructs the three significant services in E-government cloud environment: federated SSO, tenant management and access control system.



**Figure 1.2 Ningbo E-government PaaS Cloud** **Architecture**

* + 1. **OpenStack Mitaka**

OpenStack is a free and open-source software platform for cloud computing, mostly deployed as an infrastructure-as-a-service (IaaS). The software platform consists of interrelated components that control hardware pools of processing, storage, and networking resources throughout a data center. Users either manage it through a web-based dashboard, through command-line tools, or through a RESTful API. Mitaka brings plenty of benefits to the government information management department: cost savings on hardware and infrastructure, capacity management, disaster recovery/business continuity, cost savings on IT staffing and administration and the ability to access new skills and capabilities.

**1.2.2 Unify Identity Authentication and Access Control System (UIAACS)**

The UIAACS in PaaS Cloud plays a symphonic work. All the other applications or service providers could join the cloud by using the single sign on mechanism of UIAACS and giving up the user management including the following mentioned cfWeb and ESB. Facing some inalienable identity providers (IdP) actively, the UIAACS choose to docks these the existing authentication systems offered by IdPs with little configuration on applications. Anyone who logs in UIAACS could be authorized to log in any application or experience services on the PaaS Cloud magically and trustworthily. Instead, the UIAACS will take the responsibility for tenant management aiming at compatibility with any potential user management model in E-government environment. We design and implement a security solution to protect resources access from wanton destruction on purpose or not.

### 1.2.3 CloudFoundary Web (cfWeb)

The cfWeb is one of the nation’s leading CloudFoundary-based independent research and development project supporting some wildly popular language, typical development frameworks and universal back-end services. The team develop a friendly responsive web-based interface and message system for administrators along with enterprise users. Because of some outstanding features that the CloudFoundary project has given us such as collaborative development, continuous integration, rapid deployment, visual supervisory, efficient management and particularly elastic expansion, developers on cfWeb could focus on applications itself and the commercial plans behind, minimizing the time cost.

**1.2.4 Enterprise Service Bus (ESB)**

Enterprise Service Bus is the product of traditional middleware technology and web service technology. The ESB collects all the service in the enterprise internal development, providing a more fine-grained and standard unified control of services including a review and approval process and a supervision service system. ESB is divided into three accessible parts logically which are, in order, the service registry center who receives all the requests from services developers, the service application center who publishes all approved services to all validated applicants and apparently the services monitoring center telling different levels and dimensions of resources usage. To carry out the design principle on the centralism unification management, these three parts in ESB hires the UIAACS for authentication and authorization.

**1.3 Introduction to Existing Work**

This section will present the existing related work of how to organize the unify identity authentication and access control system in traditional enterprise.

### 1.3.1 About single sign on:

The various types of SSO shown in Fig: 1.3.1. This topic falls under different categories, which are: where they are deployed (Intranet, Extranet, Internet); how they are deployed (architecture – Simple, Complex); the credentials they use (token, certificate ..) and the protocols they use (Kerberos, SAML, OpenID..).Following picture shows the types of SSO and their classification:



**Figure 1.3.1 Classification of Single Sign-On**

### 1.3.2 About cross domain user management:

Along with the explosive growth in the number of applications, opening a new account becomes to be a tedious and risky issue. Developers or administrators need to assures consistency for his permissions in applications. It inevitably gets more complicated to modify access permissions as he shifts position or just leaves. Another scary thing is the potential security risks in exchanging user information or credentials among applications. Therefore, the following solutions stand out:

* P2P Identity Copy Mode: establish data conversion and transmission mechanism to copy user information from one application to another;
* Identity Mapping Mode: authentication service component keeps a mapping table that records different users according to a unique identifier;
* Agent-based Unified User Management Mode: install a special user agent in applications to transform various structure user data to unify structure

### 1.3.3 About tenant access control:

After accumulation and sedimentation in a long time since 1960s, the core idea about tenant access control is about the extensible mechanism between user and his role. Bertino came up with the Temporal-RBAC model which imports time parameter into traditional RBAC model. Administrator must consider time constraints when assigning roles to support for temporary permissions’ dependency. Joshi expands Temporal-RBAC model to Generalized Temporal RBAC model which applies time constraints to role abstraction and responsibility. Then more impact factors such as environment, location, and even system status keeps joining the constraint party. Kumar and McDaniel send out the content-sensitive RBAC model finally. These models exist a common weakness, that these models rely on manual role assignment.

**1.4 Main Work and Content Organization**

The whole Ningbo E-government PaaS cloud is a large system. It consists of four main subsystems: OpenStack, cfWeb, ESB and UIACCS. In this paper, I cannot cover all these four topics because of the limit on time and knowledge. So I focus mainly on the unified identity authentication and access control system which is also one of my results during the internship. In the rest of this paper, I introduce the UIACCS architecture, data structures and detailed algorithms, devoting myself to break down contradiction in three requirement-driven fields which are the unify identity, tenant management and centralism access control and then constructively provide our on-line practice. Considering the competition restrictions, no code leakage is allowed.

This paper is organized as follow: in Chapter 1, the PaaS architecture outline and UIAACS background is drawn including related work. Some unmissable relevant concepts and theories are introduced in Chapter 2. Chapter 3 moves to specific requirements. The preliminary and detailed design shows up in Chapter 4. The implementation based on WSO2 identity server working with some pivotal algorithms completes our UIAACS in Chapter 5. Next Chapter is about test and related evaluation of the performance. At last Chapter 6 concludes the dissertation and discusses future work.

# Chapter 2 Concepts and Technology

On the shoulders of predecessor, the UIACCS system carries the weight of some breakthrough concepts and technologies in the most recent decade and this chapter gives these to you for a quick once-over. Note that there is still other knowledge that is worth us pondering earnestly due to principal theme limitations.

* 1. **Concepts**

### 2.1.1 Browser/Server Model

Browser/Server Model (B/S) derives from traditional client-server model but without the installations of any particular client application except some ones known as browsers such as Chrome which owns the biggest global market share, normal users could view, create, share information and essentially interact with the database through some Web services and the mechanism feel really good between different platforms. There is an impressive compromise that developers increase the amount of computations on the server side which means much more appetite for expensive hardware and decrease it on the browse as client side by contrast. One of benefits is potential power in extensibility and I regard this compromise as a cloudization of user experience which almost 20 years later comes to sever side. In B/S model the developers just pay enough attention to one side and the B/S model have proved so popular years with the rapid growth of the back-end technologies development and the informatization level. Web application has mushroomed in the last 10 with the rise of the Internet. When things reach their extreme they turn back, the victims need to manage their huge on-line credentials for surfing the huge internet and hackers in the basements always drool in anticipation of the one and only one password set by some sheep. Privacy leak may have serious consequences.

### 2.1.2 Single sign-on

Single sign-on (SSO) is a property of access control of multiple related, but independent software systems. With this property a user logs in with a single ID and password to gain access to a connected system or systems without using different usernames or passwords, or in some configurations seamlessly sign on at each system. This is typically accomplished using the Lightweight Directory Access Protocol (LDAP) and stored LDAP databases on servers also called directory servers.

A simple version of single sign-on can be achieved over IP networks using cookies but only if the sites share a common DNS parent domain. For clarity it is best to refer to systems requiring authentication for each application but using the same credentials from a directory server as Directory Server Authentication and systems where a single authentication provides access to multiple applications by passing the authentication token seamlessly to configured applications as SSO.

Conversely, single sign-off is the property whereby a single action of signing out terminates access to multiple software systems. As different applications and resources support different authentication mechanisms, single sign-on must internally translate and store credentials for the different mechanisms, from the credential used for initial authentication.

Other shared authentication schemes not to be confused with SSO include OAuth, OpenID, OpenID Connect and Facebook Connect, which require the user to enter their login credentials each time they access a different site or application.



**Figure 2.2 Model Migration from Net to Hub**

### 2.1.3 ABAC Model

In many AC systems, logical access control solutions have been based primarily on the identity of a subject requesting execution of an operation (e.g., read) upon an object (e.g., a file). Examples include IBAC or RBAC where access to an object has been individually granted to a locally identified subject, or when access to an object has been granted to locally defined roles that the subject is a member of. This approach to AC is often cumbersome to manage. In this non-ABAC multi-organizational access method example (illustrated below in Figure 1), authenticated access to objects outside of the subject’s originating organization would require the subject’s identity to be pre-provisioned in the target organization and pre- populated on an access list.

Attribute-based access control (ABAC) defines an access control paradigm whereby access rights are granted to users through the use of policies which combine attributes together. The policies can use any type of attributes (user attributes, resource attributes, environment attribute etc.). Attribute values can be set-valued or atomic-valued. Set-valued attributes contain more than one atomic values. Atomic-valued attributes contain only one atomic value. Attributes can be compared to static values or to one another thus enabling relation-based access control.

A method is needed to make AC decisions without previous knowledge of the object by the subject or knowledge of the subject by the object-owner. By relying upon the concepts of subject and object attributes consistently defined between organizations, ABAC avoids the need for explicit authorizations to be directly assigned to individual subjects prior to a request to perform an operation on the object.

Moreover, this model enables flexibility in a large enterprise where management of access control lists or roles and groups would be time consuming and complex. Leveraging consistently defined attributes that span both subjects and objects, authentication and authorization activities can be executed and administered in the same or separate infrastructures, while maintaining appropriate levels of security

**Table 2.3 Performance comparison among access control models**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | RBAC | TBAC | ABAC | UCON | BLP |
| Security |  |  |  |  | √ |
| Confidentiality |  |  |  |  | √ |
| Auth. Flexibility | √ |  |  | √ |  |
| Least privilege | √ | √ |  |  |  |
| Sep. of Resp. | √ | √ |  |  |  |
| Description | √ |  | √ | √ |  |
| Fine-grained |  |  | √ | √ | √ |
| Cloud Attribute |  | √ | √ | √ |  |
| Constraint | √ |  |  |  |  |
| Dynamic Env. |  | √ | √ | √ |  |
| Compatibility |  |  | √ | √ |  |
| Elasticity |  |  | √ | √ |  |
| Manageability | √ |  |  |  |  |
| Easy-Modeling | √ |  |  |  | √ |

### 2.1.4 XACML Model

XACML stands for "eXtensible Access Control Markup Language". The standard defines a declarative fine-grained, attribute-based access control policy language, an architecture, and a processing model describing how to evaluate access requests according to the rules defined in policies.

As a published standard specification, one of the goals of XACML is to promote common terminology and interoperability between access control implementations by multiple vendors. XACML is primarily an Attribute Based Access Control system (ABAC), where attributes (bits of data) associated with a user or action or resource are inputs into the decision of whether a given user may access a given resource in a particular way. Role-based access control (RBAC) can also be implemented in XACML as a specialization of ABAC.

The XACML model supports and encourages the separation of the access decision from the point of use. When access decisions are baked into client applications (or based on local machine userids and Access Control Lists (ACLs)), it is very difficult to update the decision criteria when the governing policy changes. When the client is decoupled from the access decision, authorization policies can be updated on the fly and affect all clients immediately.



**Figure 2.4 XACML Language Decision Model**

### 2.1.5 WS-\* style and RESTful Web Service

A Web service is a service offered by an electronic device to another electronic device, communicating with each other via the World Wide Web. In a Web service, Web technology such as the HTTP, originally designed for human-to-machine communication, is utilized for machine-to-machine communication, more specifically for transferring machine readable file formats such as XML and JSON. In practice, the Web service typically provides an object-oriented Web-based interface to a database server, utilized for example by another Web server, or by a mobile application, that provides a user interface to the end user. Another common application offered to the end user may be a mashup, where a Web server consumes several Web services at different machines, and compiles the content into one user interface.

A WS-\* style Web service has an interface described in a machine-processable format (specifically WSDL). Other systems interact with the Web service in a manner prescribed by its description using SOAP (Simple Object Access Protocol) messages, typically conveyed using HTTP with an XML serialization in conjunction with other Web-related standards.

In computing, representational state transfer (REST) is an architectural style consisting of a coordinated set of components, connectors, and data elements within a distributed hypermedia system, where the focus is on component roles and a specific set of interactions between data elements rather than implementation details. Its purpose is to induce performance, scalability, simplicity, modifiability, visibility, portability, and reliability. REST is the software architectural style of the World Wide Web

REST-compliant Web services, in which the primary purpose of the service is to manipulate representations of Web resources using a uniform set of stateless operations.



**Figure 2.5 Two different Style Web-Services**

To the extent that systems conform to the constraints of REST they can be called RESTful. RESTful systems typically, but not always, communicate over Hypertext Transfer Protocol (HTTP) with the same HTTP verbs (GET, POST, PUT, DELETE, etc.) that web browsers use to retrieve web pages and to send data to remote servers.[1] REST systems interface with external systems as web resources identified by Uniform Resource Identifiers (URIs), for example /people/tom, which can be operated upon using standard verbs such as GET /people/tom.

The name "Representational State Transfer" is intended to evoke an image of how a well-designed Web application behaves: a network of web pages (a virtual state-machine), where the user progresses through the application by selecting links (state transitions), resulting in the next page (representing the next state of the application) being transferred to the user and rendered for their use.

* 1. **Technologies**

### 2.2.1 WSO2 Identity Server

WSO2 Identity Server is an identity and entitlement management server that facilitates security while connecting and managing multiple identities across different applications. It enables enterprise architects and developers to improve customer experience through a secure single sign-on environment.

As the industry’s first enterprise identity bus (EIB), WSO2 Identity Server is the central backbone that connects and manages multiple identities across applications, APIs, the cloud, mobile, and Internet of Things devices, regardless of the standards on which they are based. The multi-tenant WSO2 Identity Server can be deployed directly on servers or in the cloud, and has the ability to propagate identities across geographical and enterprise borders in a connected business environment.

### 2.2.2 SOAP and Apache axis

SOAP is a lightweight protocol for exchanging structured information in a decentralized, distributed environment. It is an XML based protocol that consists of three parts: an envelope that defines a framework for describing what is in a message and how to process it, a set of encoding rules for expressing instances of application-defined datatypes, and a convention for representing remote procedure calls and responses.

Apache Axis2 is a core engine for Web services. It is a complete re-design and re-write of the widely used Apache Axis SOAP stack. Axis2 provides the capability to add Web services interfaces to Web applications. It can also function as a standalone server application.

A new architecture for Axis2 was introduced during the August 2004 Axis2 Summit in Colombo, Sri Lanka. The new architecture on which Axis2 is based is more flexible, efficient and configurable in comparison to Axis1.x architecture. Some well-established concepts from Axis 1.x, like handlers etc., have been preserved in the new architecture.

### 2.2.3 Spring Framework

The Spring Framework is an application framework and inversion of control container for the Java platform. The framework's core features can be used by any Java application, but there are extensions for building web applications on top of the Java EE platform. Although the framework does not impose any specific programming model, it has become popular in the Java community as an alternative to, replacement for, or even addition to the Enterprise JavaBeans (EJB) model. The Spring Framework is open source.

### 2.2.4 Spring MVC

Model–view–controller (MVC) is a software architectural pattern for implementing user interfaces on computers. It divides a given software application into three interconnected parts, so as to separate internal representations of information from the ways that information is presented to or accepted from the user.

The Spring web MVC framework provides model-view-controller architecture and ready components that can be used to develop flexible and loosely coupled web applications. The MVC pattern results in separating the different aspects of the application (input logic, business logic, and UI logic), while providing a loose coupling between these elements.



**Figure 2.7 Spring MVC Basic Framework**

### 2.2.5 jQuery and Bootstrap

Bootstrap is a free and open-source front-end library for creating websites and web applications. It contains HTML- and CSS-based design templates for typography, forms, buttons, navigation and other interface components, as well as optional JavaScript extensions. It aims to ease the development of dynamic websites and web applications.

Over the last few years, Bootstrap has become an increasingly popular front-end development framework. More and more developers and designers continue to get on board. Evidently, there’s a reason for that. The framework allows for rapid, responsive development that is consistent and well supported by the development and design community.

### 2.2.6 MyBatis and MySQL

MyBatis is a Java persistence framework that couples objects with stored procedures or SQL statements using an XML descriptor or annotations. Unlike ORM frameworks, MyBatis does not map Java objects to database tables but Java methods to SQL statements.

MyBatis lets you use all your database functionality like stored procedures, views, queries of any complexity and vendor proprietary features. It is often a good choice for legacy or de-normalized databases or to obtain full control of SQL execution. It simplifies coding compared to JDBC. SQL statements are executed with a single line. MyBatis provides a mapping engine that maps SQL results to object trees in a declarative way. SQL statements can be built dynamically by using a built-in language with XML-like syntax or with Apache Velocity using the Velocity integration plugin. MyBatis integrates with Spring Framework and Google Guice. This feature allows one to build business code free of dependencies.

MySQL is the world's most popular open source database. Whether you are a fast growing web property, technology ISV or large enterprise, MySQL can cost-effectively help you deliver high performance, scalable database applications.

* 1. **Others**

In development process of the UIACCS system, the following components are put into our team toolkit:

* Tomcat as Java Servlet Container;
* Hibernate Validator as server side validator;
* Ehcache as cache Framework;
* SLF4J and log4j as log management framework;
* Apache Commons and fastjson as language extensions;
* JUnit as test framework.

# Chapter 3 System Analysis

System analysis is a kind of research strategy to find out the feasible solution to the problem through a comprehensive analysis of the system elements in an uncertainty case. The main task of the system analysis is to focus on the information obtained in the detailed investigation of the system together, and analyze the overall management of the organization and information processing process. The purpose of the system analysis is to determine the requirements of users and their solutions.

### 3.1 User structure

Today’s apps, even those that are federated, need a local account for user identity management. The challenge is managing data about users, especially routine changes like password resets and account registrations. With cloud-enabling user management, every app performs user management differently and usually does it internal to the application; user management APIs are neither con­sistent nor standardized.

The users of the PAAS cloud are the general public including part of government staffers and so the UIACCS’s are. Literally, users can be broadly classified into five categories: administrator, department manager, service user, service provider and identity provider.

The administrator is the top level manager who could control all over the UIACCS system and this group of people are the IT guys in Ningbo Cloud Computing Center. The department managers are mapping from the assigned person(s) from different departments in the government such as education bureau and tour bureau. They make decision in their own realm. The service provider deploys their applications into the cfWeb system and registers his services in the ESB system. The service provider and the identity provider need to configuration on the UIACCS for whatever the unified authentication and selective user synchronization. Two types of providers would be under surveillance by the corresponding managers. Regardless of registration or not, the service user enjoys the service from providers and if registered, the system keeps basic data.



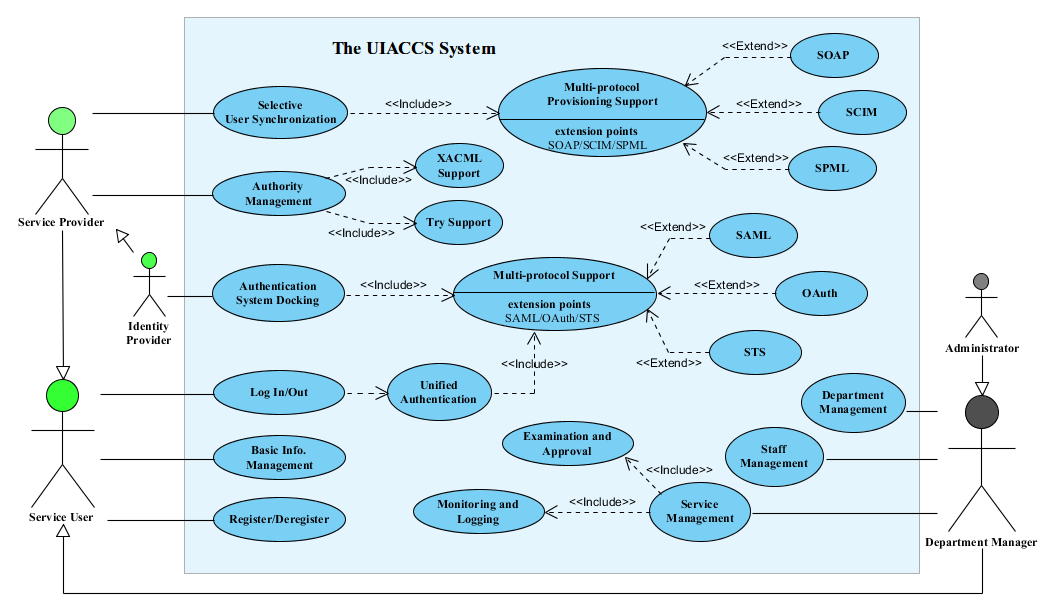
**Figure 3.1 the user structure of UIACCS**

### 3.2 Functional Requirements

#### 3.2.1 Use Case Diagram

A use case diagram at its simplest is a representation of a user's interaction with the system that shows the relationship between the user and the different use cases in which the user is involved. A use case diagram can identify the different types of users of a system and the different use cases and will often be accompanied by other types of diagrams as well.

Here is the use case diagram of the UIACCS system and this covers both five types of user with their use cases needed and foundational relationships among them. For more details, please check the next section.



**Figure 3.2 use case diagram of the UIACCS system**

#### 3.2.2 Use Case Description

**About relationship definitions**

* **Association between actor and use case**

An actor must be associated with at least one use case. An actor can be associated with multiple use cases. Multiple actors can be associated with a single use case.

* **Generalization of an actor**

Generalization of an actor means that one actor can inherit the role of another actor. The descendant inherits all the use cases of the ancestor. The descendant has one or more use cases that are specific to that role. Let’s expand the previous use case diagram to show the generalization of an actor.

* **Extend between two use cases**

The extending use case is dependent on the extended (base) use case. The extending use case is usually optional and can be triggered conditionally. The extended (base) use case must be meaningful on its own. This means it should be independent and must not rely on the behavior of the extending use case.

* **Include between two use cases**

Include relationship show that the behavior of the included use case is part of the including (base) use case. The main reason for this is to reuse the common actions across multiple use cases. In some situations, this is done to simplify complex behaviors. Few things to consider when using the <<include>> relationship. The base use case is incomplete without the included use case. The included use case is mandatory and not optional.

**About Service User Actor**

* **Log In/Out**

The actor could single sign on the UIACCS system functionality. This mechanism allows user to log in at one and only one time in any one of web applications provided by PaaS cloud and thenceforth there is no longer username or password input when he accesses other mutual trust web applications on the same browser without discarding cookie. Log out is in the same situation.

* + **Unified Authentication**

The UIACCS system should offer a uniform identity management solution called unified authorization component and decouple user authentication module from various applications. All associated applications reuse this component and no longer for cross-domain access.

* + - **Multi-protocol Support**

To meet the requirements of diversification of authentication, the actor could log in or out unified authentication with help of multiple protocols including SAML, STS and Oauth.

1. Security Assertion Markup Language (SAML).

Security Assertion Markup Language is an XML-based, open-standard data format for exchanging authentication and authorization data between parties, in particular, between an identity provider and a service provider.

1. Open Standard for authorization (Oauth)

OAuth is an open standard for authorization, commonly used as a way for Internet users to log in to third party websites using their Microsoft, Google, Facebook, Twitter, One Network etc. accounts without exposing their password. Generally, OAuth provides to clients a "secure delegated access" to server resources on behalf of a resource owner. It specifies a process for resource owners to authorize third-party access to their server resources without sharing their credentials.

1. Security Token Service (STS)

A Security Token Service (STS) is a software based identity provider responsible for issuing security tokens, especially software tokens, as part of a claims-based identity system. The token is creating a chain of trust between the STS and the software application consuming the claims.

* **Register/Deregister**

The actor could register an unduplicated real-name account in the UIACCS system as identity credential on which different services on PaaS cloud could recognize thousands of users. The actor also could deregister an account and left related data archived. Both of two cases are alternative.

* **Basic Information Management**

The actor could add, modify and remove his own basic data such as contact information as well as service-related data from applications such as privacy setting or favorite pushing time. All the changes must be tracked by supervision system.

**About Service Provider Actor**

The service provider actor who may be the government development team or a hired team is derived from service user actor and naturally inherits the latter’s use cases.

* **Selective User Synchronization**

This use case is a little special because of non-universality. On the demand of the confidentiality or privacy policy, the actor cannot share user data with the Unified Authentication completely and he chooses to synchronize limited data of users. The data flow is bi-directional optionally on the support of multi-protocol provisioning.

* + **Multi-protocol Provisioning Support**

1. Simple Object Access Protocol (SOAP)

SOAP, originally an acronym for Simple Object Access Protocol, is a protocol specification for exchanging structured information in the implementation of web services in computer networks. It uses XML Information Set for its message format.

1. Service Capability Interaction Manager (SCIM)

In computer networks, a service capability interaction manager (or SCIM) orchestrates service delivery among application server platforms within the IP multimedia subsystem architecture. A key requirement for the Service Broker functionality is to be able to bridge between the existing legacy networks, and the next-generation networks.

1. Service Provisioning Markup Language (SPML)

Service Provisioning Markup Language (SPML) is an XML-based framework developed by OASIS for exchanging user, resource and service provisioning information between cooperating organizations. The Service Provisioning Markup Language is the open standard for the integration and interoperation of service provisioning requests.

* **Authority Management**

The actor could set some policies between his predefined users and business intrinsic logic in application. Note that the need of authority management here is an access control runtime engine for service development enhancement and security in cloud environment. The actor could implement his own control model based on the engine such as RBAC, T-RBAC and etc.

* + **XACML Support**

XACML describes both an access control policy language, request/response language and reference architecture. The policy language is used to express access control policies (who can do what when). The request/response language expresses queries about whether a particular access should be allowed (requests) and describes answers to those queries(responses).

* + **Try Support**

The actor could use try it tool to test his policies easily, without creating and sending authorization requests to on-line UIACCS. It should be a UI tool through which authorization requests can be created and evaluated against available policies in the system. The actor can create simple authorization requests using the web UI of the Try-It feature. It could be possible to compose more complex authorization requests.

**About Identity Provider Actor**

The identity provider actor who may be the police department or the Social Networking Services such as WeChat and Weibo is also derived from service provider actor and naturally inherits the ancestral use cases.

* **Authentication System Docking**

The actor need to make their own identity provider configuration in UIACCS. This is a favorable alternative to explicitly authenticating a user within a security realm. The responsibility of the identity provider configuration is to represent external identity providers. If the actor wants to authenticate users against these identity providers, then he must associate one or more federated authenticators with the UIACCS. For example, if you want to authenticate users against Weibo, then Weibo in this case must associate the SAML 2.0 authenticator with the Weibo identity provider.

**About Department Manager Actor**

The department manager actor who comes from the IT related team in government department is also derived from service user actor and naturally inherits the ancestral use cases.

* **Department Management**

In this use case, the actor could add, delete and modify his sub-department information. The departments in UIACCS are mapped from real world government department experiently. All the changes must be tracked by supervision system.

* **Stuff Management**

The stuff management use case is a bit complicate for the sake of authority differences at levels of officials and roles of officials.



**Figure 3.2.2 mapping from real world to conceptual world**

In this use case the actor could operate his corresponding operations on stuff such as deletion and blocking someone over a period of time. The privileges on this actor are given by his superior and all the changes must be tracked by supervision system.

* **Service Management**

The actor could manage services provided by service providers and identity provide. It must be put forward that this use case doesn’t involve the management of service instance for example stop and restart the service which belongs to the cfWeb system. The actor in this use case could review all the configuration made by providers and check all the logs from other use cases.

* + **Examination and Approval**

Within the scope of responsibilities, the actor could review all the configurations made in the authority management use case and the authentication system docking. Positively the actor could say no to any suspicious setting.

* + **Monitoring and Logging**

Within the scope of responsibilities, the actor could check all the logs generated by the all above use cases and report it to his superior. It’s notice that the running status of the UIACCS system self doesn’t appear in this case.

**About Administrator Actor**

The Administrator manager actor who represents Ningbo cloud computing center is also derived from department manager actor and naturally inherits the ancestral use cases.

## 3.3 Non-functional Requirements

In systems engineering and requirements engineering, a non-functional requirement is a requirement that specifies criteria that can be used to judge the operation of a system, rather than specific behaviors. They are contrasted with functional requirements that define specific behavior or functions.

* Runtime Environment

The UIACCS system should run on the Linux kernel above version 2.6 with java version 1.6.24 or later.

* Performance Requirements
  + low delay: optimize the responsive time as much as possible <=500ms
  + high concurrency: support 1500 reqs/s throughput
* Security Requirements

There cannot be any mistakes on predefined manager level access control. Any deliberate leakage about department or stuff information cannot be allowed. Password cannot be easy to crack.

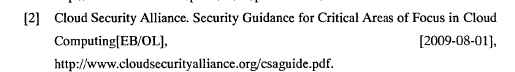
* Quality Demand

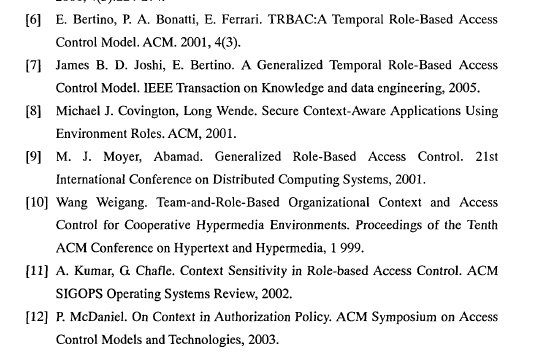
All functions in UIACCS system can be used normally. All links in web page keep alive and no dead link. System upgrade should be scheduled at the minimum visit time.

* Other Requirements

The team should provide well-formed documents for different users.

Reference





云计算访问控制技术研究综述\_王于丁

http://www.w3.org/TR/2004/NOTE-ws-gloss-20040211/#webservice