Early Draft - User Provisioning Detail Design

User Provisioning Detail Design Executive Summary

This document provides a high-level description of a UCTrust-based infrastructure to support user provisioning for inter-campus applications within the University of California. This infrastructure represents an extension to the existing Shibboleth-based UCTrust infrastructure to address use cases, such as those described in User Provisioning Use Cases.

For the purposes of this document, user provisioning is defined to be the processes, both human and automated, that authorize (and de-authorize) people to use application systems, when those processes occur at times other than the start of an online session. This is distinguished from application systems that use a "pure" single sign-on infrastructure (*e.g.*, Shibboleth), authorizing anyone with a defined set of attributes that are provided at the start of a session.

The infrastructure described in this document will support the exchange of identity information from campus Identity and Access Management (IAM) systems to application systems, not the entire set of provisioning processes. The Roles and Responsibilities section below describes where those other provisioning processes should be implemented.

While UCTrust is the first intercampus use of middleware in the University of California, this project is UC's first use of middleware as an application development paradigm. The infrastructure described is specific to the exchange of identity information for user provisioning. It does, however, embody many aspects of a more general-purpose infrastructure for data interchange among arbitrary systems that should be useful in the future.

UP HLD (copied above) can be leveraged after confirmed and updated to reflect additional discovery and detail.
This section is to provide a general description of the software system including its functionality and matters related to the overall system and its design (perhaps including a discussion of the basic design approach).

Principles and Assumptions

- Campus identity and access management systems and the organizations that operate them are authoritative for information about the members of their respective communities. The same campus organization that currently operates Shibboleth IdP (Identity Provider) will be the organization that operates the infrastructure described in this document.
- (Note that much of the IAM's information will likely be aggregated from other systems of record on the campus.; Nevertheless, UCTrust designates the IAM as the authoritative contact for its campus.)
- This framework provides a common mechanism for application systems to obtain identity information from campus IAM systems. Merging the results from multiple IAM systems, however, is left to the application.
- The existing UCTrust agreements, policies, processes, and technology should be leveraged as much as possible. All participating campuses have implemented UCTrust and are operating a recent version of Shibboleth IdP that supports SAML2 (v2.1 or higher).
- Campus IdMS system is capable of capturing changes to identity records. IdMS provides hooks so that provisioning adaptor can retrieve the data.
- The design and implementation must make effective use of University resources. Where possible implementations should be shared and/or reused. Deployment plans should accommodate differing priorities and schedules at different campuses, allowing for inter-campus collaboration and partial implementations at each campus until the entire infrastructure is deployed.
- This effective use of University resources extends beyond this project, in particular by being the first UC-wide deployment of common middleware that can be used by other projects in the future

HLD Principles and assumptions (copied above) can be leveraged after confirmed and updated to reflect additional discovery and detail.
Describe any principles and assumptions regarding the software and its use. These may concern such issues as:

- **Related software or hardware**
- **Operating systems**
- **End-user characteristics**
- **Possible and/or probable changes in functionality**

General Constraints

Describe any global limitations or constraints that have a significant impact on the design of the system's software (and describe the associated impact). Such constraints may be imposed by any of the following (the list is *not* exhaustive):

- The existing UCTrust agreements, policies, processes, and technology *should be leveraged as much as possible*
- Use standard network protocols and formats

- The user provisioning functions supported in this design scope:
  - Snapshot
  - Subscription
  - ChangeLog

- Hardware or software environment
- End-user environment
- Availability or volatility of resources
- ~~Standards compliance~~
- Interoperability requirements
- Interface/protocol requirements
- Data repository and distribution requirements
- ~~Security requirements (or other such regulations)~~
- Memory and other capacity limitations
- Performance requirements

- ~~Network communications~~
- Verification and validation requirements (testing)

Out of Scope

Identity Matching is out of scope for this project and is left to Service Providers to resolve. Please add why.
Anything else out of scope? If so, why?

- When identity information for one user appears in multiple IAM data sets, SPs may opt to unify the identity information using an identity merge process. Currently, there is no universal way to determine if two identity records strongly match, indicating the two records should be merged.

Development Method

Briefly describe the method or approach used for this software design. If one or more formal/published methods were adopted or adapted, then include a reference to a more detailed description of these methods. If several methods were seriously considered, then each such method should be mentioned, along with a brief explanation of why all or part of it was used or not used.

- Project will be divided into phases, with some acceptable overlap between phases. Emphasis is on planning, time schedules, target dates, and implementation of a gradual system over time. Control is maintained over the life of the project via extensive written documentation, design reviews, and approval.
- Software design will follow a "top-down" approach where the overall system is broken down to gain insight into its compositional subsystems. Each subsystem is then refined in yet greater detail until the entire specification is reduced to base elements.

Architectural Strategies

Describe any design decisions and/or strategies that affect the overall organization of the system and its higher-level structures. These strategies should provide insight into the key abstractions and mechanisms used in the system architecture. Describe the reasoning employed for each decision and/or strategy (possibly referring to previously stated design goals and principles) and how any design goals or priorities were balanced or traded-off. Such decisions might concern (but are not limited to) things like the following:

- Use of a particular type of product (programming language, database, library, etc. ...)
- Reuse of existing software components to implement various parts/features of the system
- Future plans for extending or enhancing the software
- User interface paradigms (or system input and output models)
- Hardware and/or software interface paradigms
- Error detection and recovery
- Memory management policies
- External databases and/or data storage management and persistence
- Distributed data or control over a network
- Generalized approaches to control
- Concurrency and synchronization
- Communication mechanisms
- Management of other resources

System Architecture

This section should provide a high-level overview of how the functionality and responsibilities of the system were partitioned and then assigned to subsystems or components without too much detail about the individual components themselves (there is a subsequent section for detailed component descriptions). The main purpose here is to gain a general understanding of how the individual system parts work together to provide the desired functionality.

Message Formats

Note: Includes but not limited to -

- Protocol messages - describe the syntax and semantics of creating request response messages
- Bindings - describe how protocol messages are transmitted over underlying communications/transport protocols
- Profiles - combine protocol messages and bindings to support the three use cases - snapshot, subscription, change log
- Security - how will this be addressed?
- Protocol should be extensible - leave open access point to layer in roles/groups as attributes for future.
- Attribute definition.

Class Diagram/Conceptual Component Diagram

- Conceptual component diagram link could go here once confirmed and additional detail added if needed.
- A list/description of any and all resources that are managed, affected, or needed by this entity. Resources are entities external to the design such as memory, processors, databases, printers, databases, or a software library.

Use Case Description

- Use Cases from HLD could go here once confirmed and additional detail added if needed.
- A description within use case format of precisely how this components goes about performing the duties necessary to fulfill its responsibilities. This should encompass a description of any algorithms used; changes of state; relevant time or space complexity; concurrency; methods of creation, initialization, and cleanup; and handling of exceptional conditions.