Straw Man OSIdM4HE Report

Executive Summary

As we expected following the Chicago meeting, we continue to believe that provisioning is both a critical element of an IDM for use within higher ed, and a major area of shortcoming in existing implementations, both commercial and open source. Existing commercial implementations exhibit limitations around some key features of most higher ed environments, and while some good work is underway toward developing open standards for provisioning protocols and data models, existing implementations are as yet very limited. We believe that building a minimal provisioning facility capable of performing the most basic "CrUD" operations via one or more standardized interfaces based on information gleaned from one or more loosely-coupled data registries is a worthwhile initial goal, with an eye toward eventually extending the facility with additional capabilities. We envision significant initial effort in three primary areas – definition and development of an interface between the OSIdM4HE registry or registries and the provisioning facility, development of a basic provisioning engine, and definition and development of one or more standard target interfaces.

OSIdM4HE Overall Architecture – Provisioning in Context

In Chicago, we identified at least three key components we expect will be a part of any successful IdM solution "stack" within higher ed: a "registry" (where among other things, identity information is collected, maintained, and stored), a provisioning facility (responsible for providing registry information to relying systems), and an authorization or privileging mechanism (responsible for managing and storing access control information pertinent to both the IdM stack and relying systems). At a very rough approximation, we choose to model the IdM stack as a traditional "layer cake", with the provisioning layer falling between and acting as "glue" between the other two components and various reliant systems which depend on them for identity information. In very broad strokes, the model we envision might be diagramed as:

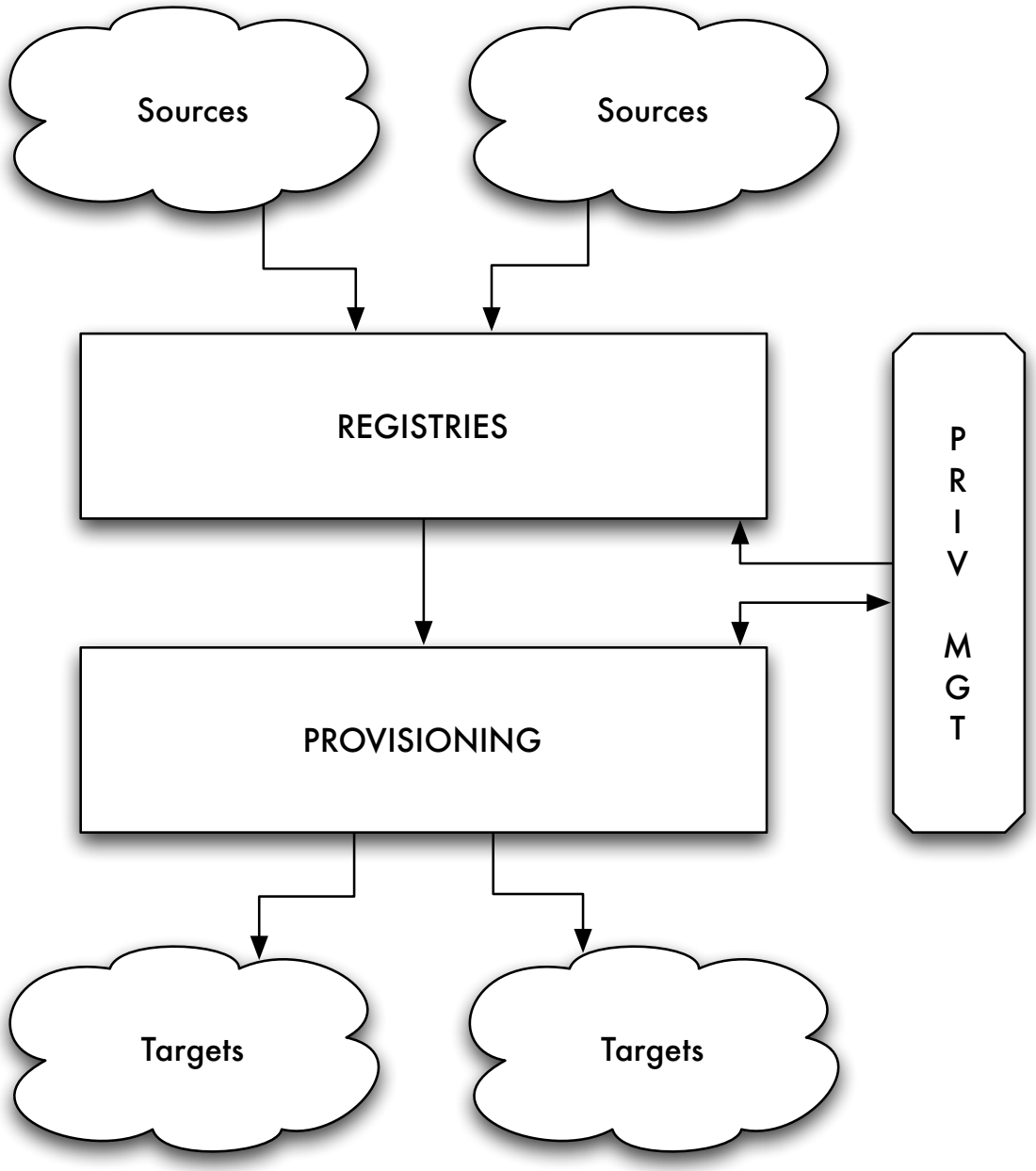

Figure 1

In this high-level model, we envision one or more authoritative data sources providing information to a registry layer, from which a provisioning facility consumes information which it presents to one or more relying or target systems, potentially including and/or subject to constraints expressed through a privilege management facility. In this sense, the provisioning facility may be construed as being responsible for "enacting" or "actualizing" information collected and managed by registries in the context of various target systems.

Provisioning: Definitions

Boiled down to a single sentence, we might propose to define provisioning as "the mechanism or mechanisms by which an IdM stack achieves, maintains, and in some cases enforces consistency between identities collected and/or minted in one or more registries and their corresponding representations within various relying or dependent systems". Note that the use of "consistency" rather than "synchronization" in this definition is intentional – we view synchronization as one of many possible means to achieving and maintaining data consistency, and believe that the term (as commonly used) implies a strict interpretation of attribute value mapping that we consider neither necessary nor, typically, desirable. Note also that the definition implies the potential for more than one registry to exist within a given IdM stack, as well as the potential for many different target systems to exist.

Scoping and Categorizing Provisioning Approaches

We believe it is appropriate to take a comparatively broad view of the the scope of provisioning, interpreting it to encompass three very high-level classes of scenarios:

- **Enterprise Provisioning Scenarios**, in which target systems are typically under the direct control of the same entities with responsibility for and control over the associated registries,
- **Federated Provisioning Scenarios**, in which target systems may be under the control of other entities, but in which strong trust relationships exist between the participants, and
- **Cloud Provisioning Scenarios**, in which target systems may be under the control of other entities with whom there is no significant mutual trust outside that provided implicitly through contractual obligation.

These classes of scenarios differ most significantly in the extent to which a provisioning facility can be expected to have access to and insight into the state(s) of objects within target systems. In the enterprise scenario, a provisioning facility can reasonably expect to have access to all or nearly all target system state, while in the federated scenario, only a subset of the target system's state may be available, and in cloud provisioning scenarios, there may be no expectation of state exposure from target systems to the provisioning facility. We expect that scenarios in all three classes are or will be of importance within higher ed.

In a broad sense, regardless of the scenario, we consider as "in-scope for provisioning" those tasks necessary to achieve and maintain consistency between data (and for the foreseeable future, specifically, identity-related data) in registries throughout the lifecycle of those identities. This clearly includes support for the creation of new identities in target systems, the deletion and/or deactivation of identities in target systems, and the addition, removal, and modification of identity attributes and their respective values within target systems. We consider

provisioning to also encompass the application of business logic, which may be applied ubiquitously or differentially depending on the characteristics of specific target systems, both for the purpose of defining and limiting which attributes and attribute values are eligible to be kept consistent between registries and target systems and for purposes of mapping both attribute identifiers and attribute values between registries and target systems. For example, a provisioning facility might be called upon to ensure that telephone number information is provided to one target system, but that it is not made available to another target system (limiting the attributes presented to a different target systems), or might be required to populate telephone numbers in "10+" format (eg. "+1 815 555 5555) in one target system while populating a collection of distinct component attributes with the same data in another (eg., "countryCode=1, areaCode=815, phoneExchange=555, phoneExtension=5555).

We also envision provisioning as encompassing not only the management of individual identities in target systems, but also the management of various identity-dependent or identity-related objects – roles, groups, and potentially privileges or permissions. To the extent that we expect registries may exist to contain any or all of these elements, we expect that a provisioning facility will need to support their presentation to relying systems.

Conversely, we consider the operation of interfaces for presenting identity information to systems that do not, themselves, store such information to be in the realm of directory services, and out of scope for provisioning. Such facilities are of great importance, and insofar as they may be considered provisioning targets in which identity information needs to be kept consistent with registry information, they fall in-scope for provisioning, but their actual operation we consider to be the province of other efforts.

Finally, we believe that it is important for any provisioning implementation to follow the traditional maxi of being forgiving in what it accepts from other systems and strict in what it presents to other systems. We expect that the basic pattern of employing standards within the core of the facility and pushing extensions and non-standardized interfaces as far to the periphery of a facility as possible can be effectively employed in designing a provisioning facility by segregating those components responsible for interacting with other systems (registries, on the one hand, and target systems on the other) from those components responsible for operating solely on data "in transit" through the provisioning facility. We remain undecided on the questions of whether the "outward facing" components should be considered the responsibility of a provisioning facility to design and implement, or whether they should be considered a shared responsibility of both the provisioning facility and the systems with which it interacts, but we have high confidence that the use of standards for the interaction between those components and the "core" of the provisioning facility will be important to the success of any implementation.

Provisioning Strategies and Technical Models

A number of technical approaches may be taken when implementing the basic goal of provisioning (maintaining consistency between data in multiple data stores that represent the same identities in different contexts). Some may be more effective in certain scenarios than others, and most imply, or at least lend themselves most naturally to one or more classes of scenario. We observe that technical implementations of provisioning vary along at least three axes – goal-orientation, temporal orientation, and architectural orientation.

Along the goal orientation axis, we recognize two broad classes of provisioning implementation – those that implement:

- **"just in case provisioning"**, in which the goal is to maintain information about identities in target systems regardless of those systems' demonstrated needs at the time. If an identity or a component of an identity may be needed by a target system, the provisioning facility strives to ensure its availability and consistency "just in case" it's needed by the target system
-

and those that implement:

- **"just in time provisioning"**, in which the goal is to maintain only the identity information for which a target has demonstrated a need. If an identity or a component of an identity has not been used or requested by a target system the provisioning facility does not attempt to maintain it in the target system until such time as the data is required.

Similarly, along the temporal orientation axis we recognize two broad classes of implementation – those that implement:

- **real-time provisioning**, in which the provisioning facility responds to changes in registry information in real- or near-real-time (as measured on human-detectable timescales), and those that implement
- **batch provisioning**, in which the provisioning facility may respond to changes in registry information only when prompted by some external trigger or on some fixed schedule.

Both of these implementations result in "on average" consistency between registries and target systems, but they vary in the degree of latency introduced by the provisioning process.

Along the architectural orientation axis, we recognize three broad classes of implementation, loosely focused around three approaches to presenting change information to target systems:

- **"slim" provisioning**, in which the provisioning tool acts primarily as a messaging or notification mechanism informing target systems that some object in a registry has undergone a change, but conveying no additional metadata with that information. In this approach, the expectation is that the recipient of the notification will retrieve data directly from its source (typically, but not necessarily a registry) and bring its representation of the object into a state consistent with the source system's information. The responsibility for ensuring consistency in this case rests with the target system, and the provisioning facility is only responsible for accurately identifying and notifying on changes in registry information.
- **"incremental" provisioning**, in which the provisioning tool delivers to target systems more extensive change metadata than in the "slim" approach, explicitly identifying what component or components the registry has modified and how they have changed. In this approach, the target system may expose an API through which the provisioning facility may directly update its object(s), or may expose an interface for receiving enriched change notifications but reserve the implementation of actual changes to itself. The responsibility for ensuring consistency between registry information and target system information rests more or less equally with the provisioning facility and the target system in this case.
- **"full" provisioning**, in which the provisioning tool delivers fully-specified objects to the target whenever changes occur. In this approach, the target is expected to be entirely "enslaved" to the provisioning facility, and any identity data in the target system is expected to have been placed there explicitly by the provisioning facility. The responsibility for ensuring consistency between the registry and the target system lies almost exclusively with the provisioning facility in this approach – any change results in what amounts to a full resynchronization of the modified object, and all relevant target data is placed in the target by the provisioning facility.

The three architectural approaches are not mutually exclusive, nor are the two temporal strategies nor the two goal-orientations – one provisioning facility may employ, for example, just-in-case, full batch provisioning of one target (perhaps because the target does not provide real-time APIs of any sort), and just-in-time, incremental real-time provisioning of another target (which perhaps exposes a rich set of APIs and depends sensitively on registry data for its own real-time privileging decisions).

Note that these different approaches may be applied not only to the interaction between a provisioning facility and its target systems but also to the interactions between components within a provisioning facility. This leads us, conceptually, to the topic of componentizing provisioning facilities.

Provisioning Componentry

In the broadest possible sense, we envisioning provisioning facilities as encompassing three "layers" of operational components, distinguished by their interactions with other facilities:

(1) An "input" layer, through which registries communicate state information and state transitions to the provisioning facility. This layer may be implemented in various fashions – as a change log consumer for real-time "pull" of state information from a registry, as a synchronization mechanism for on-demand communication of state information to the provisioning facility, or using some for of "event trigger" for real-time "push" of changes into the provisioning facility. Some registries may present more than one of these interfaces to the provisioning facility, and different registries may implement different interfaces. All registries need to provide at least one such mechanism in order to exchange their information with the provisioning facility, however.

(2) An "engine" layer wherein business logic is applied. This logic may entail various aspects, including:
   a. Selection of appropriate target consumers for changes based on target metadata and change characteristics. For example, a provisioning engine might be required to only deliver SSN information (including registry-recorded changes to SSN values) to specific systems in order to protect individual privacy).
   b. Mapping of source attribute identifiers onto target attribute identifiers. For example, a provisioning engine may be instructed to ensure consistency between the "Last Name" attribute recorded in a person registry and the "sn" attribute stored in an LDAP directory.
   c. Mapping of source attribute values onto target attribute values. For example, a person registry might encode FERPA preferences using single-letter values of "Y" (for attributes that may be published under FERPA constraints) and "N" (for attributes that are not publishable under FERPA constraints), while a target system may require that those same flags be presented to it as "public" and "private", respectively, or may require that they be delivered in the form of access control statements actionable by the target system.

(3) An "output" layer, where target-specific protocols and APIs are employed to actually effect state transitions as dictated by (1) and (2) above. For example, when a user's last name changes in a person registry and the provisioning engine determines that the change should be mapped to the "sn" attribute value for the matching entry in a white pages LDAP repository, the output layer might be responsible for actually binding to the LDAP directory as an administrative user and performing the requisite MOD operation to effect the change in that attribute.

We believe that "provisioning" encompasses all three of these layers, but we also believe that, properly designed, the layers could be very loosely coupled to one another, allowing for separate development of code to implement each layer. Depending on the specifics of the implementations, we expect that some of the responsibilities outlined above could shift between layers (for example, attribute value mapping might be implemented in the "output" layer rather than in the "engine" layer without significantly varying from the model). We expect, however, that the layers need to be clearly delineated, and that there needs to be some separation both in the logical architecture of the provisioning facility and in the actual implementation of the architecture.

Gap and Needs Analysis and Recommendations for OSIdM4HE

With that as background, we would make the following observations and recommendations to the committee of the whole:

(1) As we tentatively inferred in Chicago, we continue to believe that provisioning is a key and necessary components of any effective IdM stack for higher ed use.
(2) Existing efforts toward solving the provisioning problem bear close tracking by the group, especially as some efforts are nearing readiness for production implementation. There remain many more standardization efforts in this space, however, than delivered implementations, and many more partial implementations of provisioning that full implementations. Some key efforts to continue to track include:
    a. SCIM – the Simple Cloud Identity Management effort. SCIM is of particular interest with respect to the cloud provisioning scenario, as its name would imply, but its componentization of the provisioning problem space and its proposed language for describing changes (what SCIM refers to as a "patch" mechanism) is sufficiently novel to bear further study. SCIM does not at this time appear to be near the point of being fully implementable, however, and is probably not a short-term option for our purposes.
    b. SPML – the Simple Provisioning Markup Language. SPML is of interest as a generalized provisioning model, but while real-world implementations exist, we question both its viability in the face of decreased developer interest and its applicability to some federated and particularly cloud provisioning scenarios.
    c. SAML attribute provisioning seems perhaps the most promising of the primary contenders in this space, with a nascent implementation and near certain agreement to a standard definition with OASIS. It may not lend itself to all possible architectural models for provisioning, though, and to that extent may be somewhat limiting to the scope of a generic provisioning effort.

d. Commercial offerings from Oracle, Novell, Fischer, and others exist that cover some of the provisioning space, and may be highly effective in certain environments, but they tend to exhibit either rigidity around the implementation of business logic (which may not be as well standardized between higher ed institutions as between commercial organizations) or a lack of support for key features (such as multi-valued attributes, or privacy enhancements), that make their wide deployment within higher ed as much a development effort as a deployment effort, and denude their value as "out of the box" solutions.

(3) We believe that higher ed organizations likely have minimal requirements that can be met by a provisioning system that provides a clearly-specified change description mechanism (most likely based on one of the emerging provisioning standards noted above), well defined APIs for "input" and "output" processing, and a well-defined mechanism for specifying mapping logic between attribute identifiers and attribute values, along with one or more standards-based interfaces against which target system adapters may be developed. Development of an interface between the OSIdM4HE registry or registries will be critical to the success of an integrated provisioning solution, but development of specific target support may be less so.

(4) We believe that it may be feasible to approach key target developers and enlist their assistance in providing standards-based interfaces that could reduce or eliminate the need for target-specific development as part of the production of a provisioning facility for OSIdM4HE. Where vendors have a wide footprint within higher ed but no interest or willingness to provide standard provisioning interfaces, it may be crucial for the project to allot resources to develop a few key target interfaces, however.

Prioritization

We expect that an OSIdM4HE provisioning facility will need to initially meet at least a few key requirements:

- Some form of change communication mechanism between the OSIdM4HE registry or registries and the provisioning facility. This constitutes the basis of the "input" layer, and may be construed as the responsibility of either the provisioning facility or the registry sub-project, but clearly needs to be agreed upon by both development teams.
- A basic provisioning engine capable of consuming change information provided through that mechanism performing minimal attribute identifier mapping between registry identifiers and target identifiers, and delivering those changes to a standardized "output" layer. The engine should provide the flexibility for later implementing additional business logic (beyond simple identifier mapping), but need not initially provide value mapping logic, and should provide the flexibility to encode

provisioning decision logic, but may initially present all changes to all target systems with which it interacts.
- An "output" layer that exposes a standards-compliant interface for targets to consume change information. Initially , this interface may be singular, but the engine should have the ability to support multiple output layer implementations, in case some targets are better or worse-suited to particular modes of operation than others.

We believe a few additional features will become important soon after an initial offering is released, but may not be required in the initial release:

- Native support for integration with one or more message persistence mechanisms (in support of guaranteed delivery of change information to targets that may be temporarily unavailable)
- Support for triggered or explicitly invoked "full resynchronization" of target systems, either through some form of historical replay of incremental change records or, more likely, through a separate synchronization process.
- Support for richer provisioning logic in the provisioning engine – more than one-to-one identifier mappings, extended rule mechanisms in support of defining target provisioning profiles (eg., what attributes should be presented to which target systems under what circumstances).
- Audit logging.

Further enhancements not necessarily required at any foreseeable point but likely to be desirable might include:

- Point in time auditing based on provisioning facility audit logs
- Support for post-provisioning operations, for example, executing code in a target system whenever a change of type X is delivered to the target. This is likely to be of interest in, for example, cases involving provisioning resources that require custom pre-creation of artifacts (such as Exchange mailbox provisioning in Exchange 2010).
- A built-in message persistence mechanism (so that deploying sites need not invest in a separate persistence system if they do not already employ one).
- Friendly user interfaces.
- Canned "output" connectors supporting various common target systems, and possibly also canned "input" connectors supporting common IdM solutions from other vendors (to facilitate data migration).