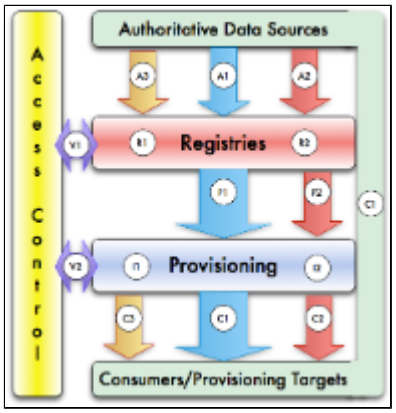
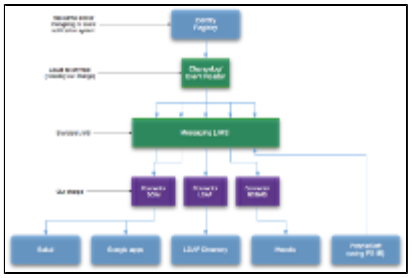
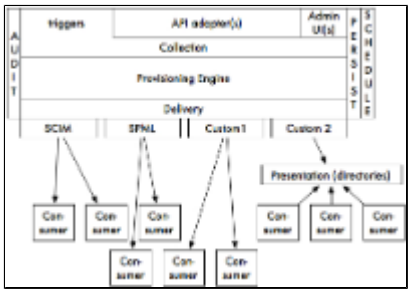
Rough Provisioning Architectural Diagrams

Diagram	Description	Functional Walk-Thrus
	• Authoritative Data Sources - Agreed upon sources for entity information. In the case of people, these may be central systems for institutionally-mastered data like payroll or enrollment information, distributed systems for departmentally-mastered data like academic appointments and job assignments, or individual people for self-asserted information like preferences and personal contact information. Some entities may have authoritative information provided by multiple authoritative data sources. Some data sources may overlap with other data sources, providing the same attributes about the same or different entities. Authoritative data sources must provide some mechanism for interacting with registries in order to have their data included in institutional identity construction. • Registries - Intelligent repositories for centrally-maintained identity information. Entity data from authoritative sources is transformed into unique, persistent identities within the registries. Typically, registries assign unique identifiers (both persistent and transient), and may enhance authoritative source data through attribute deconfliction and promotion and through derivation and compositing of attributes. Registries must interact with authoritative sources as consumers of data, and must expose the results of their operations in some useful fashion to provisioning systems. • Provisioning Systems - Engines responsible for exposing and conveying the results of registry activities to the systems reliant on central identity information. In a classical view, provisioning engines are responsible for maintaining the lifecycle of identities and identity information in consumer or target systems, ensuring that identities minted in registries are properly reflected in, maintained in, and ultimately deactivated in or removed from participating consumer or target systems. As identity data in registries evolves, provisioning engines are responsible for translating registry changes into appropriate actions relative to their consumers. This may include applying consumer-specific business logic or implementing certain access control requirements. Provisioning systems must interact with registries to consume identity information and identity information changes, and must interact with consumer systems to create, update, and delete identity information there. In some scenarios, provisioning systems may interact with other data delivery systems, such as directories, to provide consumers with realtime access to data they (the consumers) do not store locally. • Access Control Systems - Much of the identity data managed by these systems and services is of a profoundly sensitive nature. Access control systems are responsible for facilitating the expression, persistence, exposure and in some cases, enforcement of data access policies both between the other components of the IDM stack and within individual components of the IDM stack. In the diagram at the left, access control systems are depicted interacting only with the registry and provisioning systems, but in some scenarios, access control systems may bypass provisioning systems and provision access control rules and permissions directly into target systems. In some models, Access Control systems may provide repositories for permissions and access policies that are in turn consumed in situ by target sytems, rather than being provisioned locally to those systems.	**Arrows filled in blue represent "push" data flows.** **Arrows filled in red represent "pull" data flows.** **Arrows filled in orange represent "batch" data flows.** **Arrows filled in purple represent bidirectional data flows.** • **A1 -** Some authoritative sources may implement "live push" mechanisms for sending data updates to registries. Registries should have the ability to accept such live updates. • **A2 -** Other authoritative sources may implement mechanisms for retrieving updates from source change logs, or may expose their entire databases for periodic difference processing. Registries should have the ability to consume source data from such systems via "pull" mechanisms. • **A3 -** Some authoritative sources may only implement batch extract/import mechanisms for providing data to registries. In the interest of consuming complete authoritative data, registries need to support these batch export mechanisms, as well. • **R1 -** Once data has been imported through one of the A1 - A3 mechanisms above, logic within the registry normalizes it, matches it with existing normalized data to verify uniqueness before either using it to create a new identity (if the data are unique) or update an existing identity (if the data refer to an existing identity), and ensures data persistence. • **R2 -** The registry also applies business logic to deconflict attribute values that may be provided differently by multiple authoritative sources, and, and may enhance authoritative data with derived or composited attributes. • **P1 -** The registry provides updates to the provisioning facility through one or more mechanisms (RPCs, triggers, change log entries, etc.) in real-time. Updates may be provided in the form of simple pointers (indicating that a particular entity has been updated and should be inspected to determine how the provisioning facility should respond), complex references (indicating **h ow** a particular entity has been updated as well as **that** t he entity has been updated) or mandates (indicating precisely what operations were performed on what entities). • **P2 -** The provisioning facility may retrieve information from the registry for a variety of reasons. In scenarios in which changes are presented to the provisioning facility as simple or complex references, the provisioning facility may be responsible for retrieving the data values relevant to any given change. In any scenarios, the provisioning engine may require additional information stored in the registry but not part of the change itself (and perhaps related to another, unchanged entity) in order to complete its computations. Registries need to expose interfaces to allow provisioning services to retrieve data as necessary to complete their operations. • **I1 -** The provisioning facility, having been notified or having detected in some fashion that a change has occurred in the registry, may consult various internal configuration repositories to determine how to proceed. In some cases, a change may need to be delivered to multiple consumer systems; in others, a change may not result in any action on the part of the provisioning engine. In still others, a change may result in a cascade of operations being performed involving one or more consumer systems. • **I2 -** The provisioning facility may apply business logic to map changes presented by or detected in the registry into operations suitable for consumption by a particular consumer. Such logic may include masking or transformation of data as well as mapping of attributes and operations. • **C1** The bulk of provisioning operations are expected to be performed in real- or near real-time through one or more "push"-oriented connectors targeting specific groups of pre-identified consumers. Such operations should take advantage of standards-based interfaces wherever possible, but in cases in which a standards-based provisioning interface is not available or cannot be used effectively, custom code may be required. Note that the set of authoritative source systems and the set of consumer systems may not be entirely disjoint from one another – some consumers may also be authoritative sources, especially when an authoritative source is only considered authoritative for one or a small number of specific attributes. Special care must be taken in these situations to avoid triggering tail-recursion during real or near-real-time update processing. • **C2 -** A class of consumers likely exists which will prefer to use some form of directory or other presentation mechanism to access identity information. While this "pull" mechanism may take the form of direct queries against registries in some special cases, the most common case will likely involve direct provisioning of identity information into some form of directory by the provisioning facility, followed by consumers' querying the directory in real-time as they require identity information.

		• **C3 -** A small class of primarily legacy consumers may be unable to provide real- or near-real-time interfaces to the provisioning facility, and may be unable to consume data in real-time from any form of directory. In these cases, the provisioning facility needs to exhibit the ability to produce periodic data extracts based on the content of registries, and to provide those in some fashion to consuming systems. Such extracts may consist of complete data extracts for subsets of identities interesting to specific consumers, or may consist of incremental change reports. • **V1 -** By virtue of their holding sensitive and sometimes confidential information about persons and possibly other entities, it is critical that access to registries and registry information be consistently managed. Interaction between registries and the access control facility may be of at least two forms – the access control facility may need to directly assign permissions within registries based on its access rules and privilege configuration, and it may also need to consume data from registries in order to make access management decisions. • **V2 -** While the provisioning facility is not expected to maintain permanent stores of sensitive (nor common) information, it will require the support of the access control facility to ensure that provisioning tasks are not used to bypass the data access rules enforced in the registries. It may also act as the active engine for provisioning access control information into consuming systems, where appropriate. Simultaneously, the access control facility may itself become a consumer of provisioning data in some scenarios (eg., rather than directly retrieving information from the registry, an access control facility may be designed to consume that information via the provisioning facility).
	In this diagram, the connectors do not communicate with the identity registry at all, and in a lot of ways, the connectors are dump, i.e., they can only do what they are "told" by the messaging system. So, with that in mind, it goes without saying that the consumers can not ask the connectors for information, i.e., a system can now perform a query against the connector. The flow is from the top of the diagram to the bottom. • Starting with an Identity Registry (this diagram is not at all concerned with how data gets into the registry, nor how it is reconciled), any changes to a person's identity will be reflected in some kind of standard changlog. The changelog could be any number of things, but it should adhere to some standard for how changes are reflected. • The Changelog/Event Reader could either be internal to the Identity Registry or external, but the purpose of the Reader is to read changes from the Identity Registry, and push them onto the messaging layer. Depending on how complex the connector is, these changes could be partial records, full records, consolidated records, or just the ID of the person who's data changed. • The Messaging layer, of course, provides topics/queues that will be read by the Connectors. • The Connectors, again, depending on how complex they are, could either just take the entire message as is and perform the change (add, modify, delete) against the consumers. • The Connectors can either have logic for what exactly the consumers are allowed to consumer, or that logic could be in the Changelog/Even reader which pushes the changes to the Messaging system (meaning that only the changes/data that are available for a given consumer ever make it to the Messaging system).	• **Scenario 1 - Last Name Change by Employee (update push from registry to comsumers), with Changelog/Event Reader doing the bulk of processing**# Sarah Jones' name has changed in the Identity Registry.# The Changelog/Even Reader picks up the change, queries the Identity Registry for the rest of Sara's information, and pushes the entire record to the Messaging system. If the messaging system is down, the Reader will try again later. 1. The Messaging system now has the change and is prepared for the Connectors to get the change. 2. The Connectors pick up the change as is and send a modify request to the consumers. If any of the consumers are down, the message will remain in the Messaging system until the consumer becomes available and the connector can again process the request.* **Scenario 2 - Last Name Change by Employee (update push from registry to comsumers), with Changelog/Event Reader doing the bulk of processing, but only for the data that has changed**# Sarah Jones' name has changed in the Identity Registry. 3. The Changelog/Even Reader picks up the change and pushes only the changes to the Messaging system. If the messaging system is down, the Reader will try again later. 4. The Messaging system now has the change and is prepared for the Connectors to get the change. 5. The Connectors pick up the change as is and send a modify request to the consumers. If any of the consumers are down, the message will remain in the Messaging system until the consumer becomes available and the connector can again process the request.* **Scenario 3 - Last Name Change by Employee (update push from registry to comsumers), with Connector doing the bulk of processing**# Sarah Jones' name has changed in the Identity Registry. 6. The Changelog/Even Reader picks up the change, and pushes Sarah's ID to the Messaging system noting that it was a change. If the messaging system is down, the Reader will try again later. 7. The Messaging system now has the change and is prepared for the Connectors to get the change. 8. The Connectors pick up the change, query the Identity Registry, and/or some other system(s) to create the full record, and send a modify request to the consumers. If any of the consumers are down, the message will remain in the Messaging system until the consumer becomes available and the connector can again process the request.



- Data enters the Provisioning Engine in one of three primary ways – "pushed" into the provisioning interface via a trigger (or other mechanism) fired from the layer above (presumed to be one of the IdM registries), "pushed" into the provisioning interface via an administrative UI (for use by administrators in manually "forcing" provisioning actions to occur), or through a set of adapter API routines designed to allow the provisioning engine itself to retrieve data from the layer above (eg., in the event that completing a provisioning task started by a trigger from an IdM registry requires access to other data in the same or a different registry). It is the responsibility of the "Collection" layer and its associated modules to manage the interface between the registries and the Provisioning Engine, and to marshal and collect the information required by the Provisioning Engine.
- The provisioning engine in turn is responsible for applying business logic, including attribute access controls, attribute transformations, and dependency computations.
- The Delivery layer is then responsible for taking the finalized output of the provisioning engine and, based on pre-arranged subscription information, selecting particular consumers and their associated delivery mechanisms and triggering them to provision data (according to their own configuration and protocol specifications) to consumer systems.
- The Delivery layer relies on a collection of protocol- and/or consumer-specific modules to actually drive its provisioning efforts. Multiple consumers may use the same delivery protocol module (eg., SCIM may be used to provision into more than one consumer) but multiple protocol modules may be used by multiple different consumers.
- A special case collection of consumers are the presentation layer interfaces (eg., directories). Provisioning into those consumers is not typically performed for their own purposes, but in order to allow still other, secondary consumers to access the provisioned information via "pull" methods (eg., via LDAP). Some consumers may also use custom APIs to directly "pull" data directly from the IdM registries (much as the API adapters associated with the provisioning engine's collection layer do) but those are not considered in-scope in this diagram).
- All layers of the provisioning stack are responsible for interacting with an audit logging interface, whose responsibility is to build and maintain audit trails suitable for debugging as well as reporting on the history of data passed through the provisioning stack.
- The stack includes persistence and scheduling mechanisms in support of time-dependent provisioning operations (eg. to support a rule of the form "all terminated employees' "employee" affiliations should be removed at the time of termination, and their electronic mailboxes and Kerberos principals should be disabled 14 days following their termination). The same persistence mechanism might also be used to support "retry-on-failure" options (eg., if a triggered event cannot be processed immediately due to a failure in the consumer system or in the network between the provisioning system and the consumer, the event might be queued and retried on some fixed schedule some number of times before being completely dropped).

- **Scenario 1 - Last Name Change by Employee (update push from registry to consumers with persistence)**# Sarah Jones marries and changes her legal last name as recorded by the SSA from "Jones" to "Morgenstern". Her HR representative is notified, and after properly validating her new legal documentation, records the last name change in the HR ERP.# That change makes its way from the HR ERP into a campus person registry, where it results in an update to the "SN" attribute in the person registry for Sarah's entry.# That update causes the registry to call out to an "update" trigger in the provisioning module, passing Sarah's unique identifier, a "change SN" operation tag, and "prior value" and "new value" qualifiers of "Jones" and "Morgenstern" as arguments.

1. The update routine in the data collection layer of the provisioning module in turn passes the information to the engine layer, which consults subscription data and business logic definitions to determine that three consumer systems are subscribed to SN changes for Sarah's entry, and that the SN value is to be passed to all three of them unmodified.
2. The engine layer in turn hands off the update to two connector modules – the SCIM module (which passes the update via a SCIM "push" request to two of the three subscribed consumers) and a custom LDAP update module (which passes the update directly into a campus-wide LDAP directory used for white pages searches).
3. The SCIM updates both succeed, but due to the LDAP consumer being in maintenance mode at the time of the update, the LDAP update fails, and the custom connector module reports an unsuccessful return code back to the engine layer. The Engine layer sends the update to its persistence module, where the update is archived for later resubmission to the custom module.
4. Some 20 minutes later, the scheduling module triggers a scan of the persistence repository for failed updates, and re-queues the update of Sarah's SN attribute to the custom LDAP module that failed it previously. The LDAP consumer is now in an operative state, and accepts the update. The connector reports successful completion of the operation back to the engine, which records the update as successful via a call to the audit logging module and discards it.* **Scenario 2: Side Effects of Affiliation Changes (and additional data collection)**# John Wesley Harding has been a full-time employee in Accounts Payable for three years, taking a part-time course load in an effort to complete a Masters degree in Finance. He decides to complete his last two semesters of course work as a full-time student, and resigns his staff position to become a full-time student.
5. The employee ERP system registers John's termination as en employee, while the student ERP system registers his transition from part-time to full-time student status, and notifies the person registry of these changes. The person registry recomputes John's affiliation and primary affiliation attributes, changing his records to reflect that he is now primarily a full-time student.
6. As an employee, John's preferred name has been restricted to matching his official first name, but as a student, he reported in his enrollment paperwork that his preferred first name is "Wes". As a student, John has the right to assert FERPA protections over some of his student records information, and he has requested anonymity under FERPA (out of concern that his boss might think ill of him were he to find out about his part-time enrollment in the Finance graduate program).
7. When the person registry calls the provisioning module's "update primary affiliation" trigger to change his affiliation from "staff" to "student", the provisioning engine detects that four consumer systems are subscribed to changes in John's affiliation. It also detects that three of those four consumer systems need to implement FERPA restrictions, while the fourth (an internal system used for student registration) is authorized to maintain student records information despite FERPA restrictions.
8. The provisioning engine calls out to an API interface in the collection layer to retrieve John's FERPA preferences and his "student preferred first name" attribute from the person registry. After applying the associated business logic, the engine passes affiliation updates to four separate consumers via the SCIM, SPML, and two custom delivery adapters, and passes a "givenName" update to the student registration application via SPML (replacing John's official employee-derived first name of "John" with his preferred student first name of "Wes"). All updates succeed, are recorded via the audit logging layer, and subsequently discarded.