Architecture and Design

Project Goals

The primary goal of midPoint project is to create a state-of-the-art open source Identity Management (IDM) system. The essential functionality is identity provisioning, synchronization, reconciliation and application of security and organizational policies such as Advanced Hybrid RBAC.

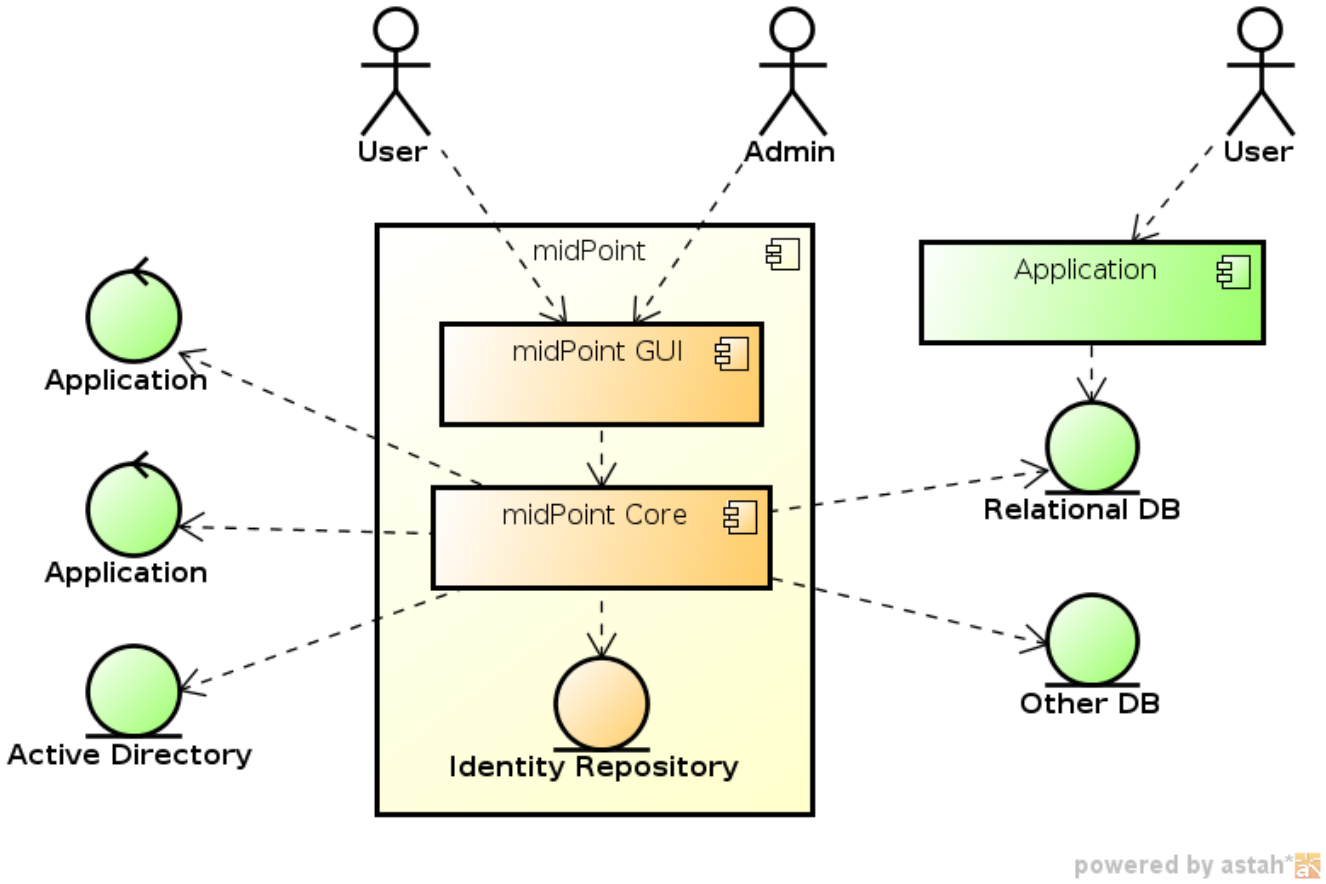
midPoint is designed and built as a modular and extensible system. This is necessary to support plethora of unforeseeable scenarios in the identity management field. The flexibility is not the only goal of midPoint, as even the most flexible system is useless if it cannot be deployed in time and budget. Therefore deployment efficiency and usability is even more important than flexibility. Therefore the primary tools for midPoint customization is configurability. The majority of common IDM scenarios can be *configured* in midPoint without any need for programming (save perhaps for one-line scripting expressions in attribute mappings).

We have taken an approach that does not create a single monolithic product but rather a set of maintainable and extensible components. The components are wired together to create a complete product, but it is still possible to rewire and modify the components to get extreme customizability of midPoint. We use lightweight Java frameworks as a basic infrastructure to partially reach that goal. The rest of the flexibility is achieved with open interfaces, plug-ins and especially by the open-source nature of the product.

This page provides an overview of midPoint architecture. The goal of this page is to explain both *how* is the system structured and *why* it is structured in that way. It is not going into all the details; it describes just the basic ideas to get the big picture. Details regarding individual subsystems are provided on separate sub-pages.

Big Picture

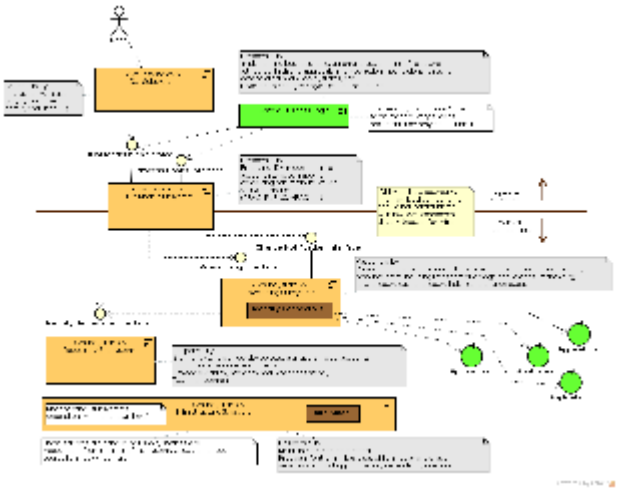
MidPoint is an provisioning system designed to fit in an existing enterprise environment and even into a telco or cloud solution. The goal of midPoint is to synchronize and manage many identity repositories. It can manage systems like Active Directory, applications based on relational databases, applications that expose services for user management and many other system types. midPoint is extensible and new connector types can be added as needed. This approach is illustrated in the following diagram.



MidPoint is designed with an assumption that the data in the various identity stores **will not be consistent**. E.g. a single user may have account `smith` in one system, `jsmith` in another and `js432543` in yet another one. User's name could be stored in attribute `cn` in one system and could be allowed to contain national characters, while it could be stored in attribute `GCOS` in another system, restricted to US ASCII character set and obligatorily supplemented with user's organizational unit name. The goal of midPoint is to merge these data and keep them as consistent as practically possible. midPoint has many controls, rules, expressions and pre-built identity management logic to assist in solving this difficult problem.

MidPoint Subsystems

The architecture consists of several subsystems that are connected by the Spring Framework. Each of the subsystems has a specific purpose in the system and can be replaced by custom solutions if needed. The high-level architecture is illustrated in the following diagram (please click on it to englarge it).



System core consists of repository, provisioning and model subsystems. Repository Subsystem is storing authoritative identity data, pointers to identity objects in other systems (such as accounts), definition of roles, expressions, synchronization policies, etc. Provisioning Subsystem can talk to other systems, can read data and modify them. The IDM Model Subsystem is gluing everything together. It is a component through which all activities pass, therefore it is an ideal place to enforce policies, fill-in missing data, map attributes and properties govern processes and do all the identity management logic.

The structure of the system core (low-level components) is mostly fixed. Some customization is possible, but this is mostly achieved by changing component configuration, not their structure. For example the customization may be to add new attribute expression, change definition of a role, introduce new attribute in the schema, etc.

On the other hand there are components that change a lot, both with respect to their behavior and structure. User interface is a component well-know for a need to adapt to any whim a customer may have. A similar example may be the business logic: approvals, integration with trouble-ticket systems, sending notification messages, and so on. It is expected that the structure and behavior of high-level components will change a lot. midPoint provides a "stock" web-based graphical user and administration interface. However, we expect that this will not always be the only interface that the system provides. New interfaces may be added to the system or existing interfaces may be customized.

The IDM Model Subsystem is the real heart of the system. A major system boundary is crossing the model subsystem:

- The components *below* the model are expected to change little in the code and structure but change a lot in configuration. This is the domain of an IDM administrator. New resources can be added, schema can be modified, passwords changed, expressions adjusted. But no real heavyweight code is changed here. There is little need to change the code, as the low-level components implement a logic that is common to vast majority of IDM deployments.
- The components *above* model are expected to change in code and structure. This is the domain of an IDM engineer customizing the system. New components can be added here, including components written in high-level languages such as BPMN. The system places little constraints on engineer's creativity in this domain.

The following sections describe the components that are part of stock midPoint system.

User Interface Subsystem

User interface subsystem implements web-based end-user interface and administration interface. This subsystem does not contain any business logic, however it contains relatively sophisticated presentation logic. It interacts with IDM Model subsystem using the IDM Model Interface. In some rare cases it also interacts with other subsystems but this is mostly a matter of technical "wiring", application start-up and shutdown and similar cases.

The interfaces is constructed as a Web user interfaces with dynamic interaction elements. The user interface is written using Apache Wicket framework. The "look and feel" is implemented by utilizing Bootstrap Framework.

IDM Model Subsystem

Main page: IDM Model Subsystem

Every identity management solution has some kind of model that it needs to be built around. It is usually the Role Based Access Control (RBAC) model or some of its variants. midPoint provides a hybrid Role-Based Access Control (RBAC) model. Such model uses mappings and expressions to extend definition of roles, so less roles can handle more situations. The model also supports ad-hoc provisioning (model exceptions) by the use of assignments, supports scripting expressions, temporal conditions and other advanced features.

IDM Model subsystem implements the most important interface in the system: IDM Model Interface. This interface is used by user interface and business logic, and may be also used by other systems.

IDM Model is theoretically interchangeable. The whole component can be replaced with a component that implements a different model. However, that may not be easy and most of the high-level components may need to be adapted.

Provisioning Subsystem

Main page: Provisioning Subsystem

Provisioning, deprovisioning and reprovisioning are the basic functions of a provisioning system. This means managing accounts, account attributes, groups, organizational units and similar objects on the target systems.

There are numerous strategies on how to provision accounts on target systems. The accounts can be provisioned directly and synchronously, while the acting user is waiting for the result. Or the request can be placed in the queue and processed later. The provisioning system may maintain only the account identified and fetch all the attributes from the target system when needed. Or the provisioning system may create a copy of all resource objects and synchronize them. Specific strategies are good for specific situations and in practice all of them are needed. Therefore the provisioning system must be flexible enough to support most of them, sometimes even combine them in a single deployment.

Currently the provisioning system maintains shadows of the real resource objects such as accounts and groups. The shadows are used for loose but reliable linking between the midPoint concepts (such as *User*) and the resource objects (such as *Account*). The provisioning subsystem provides transparent access to the resource objects attributes. The attributes are fetched right when they are needed by the model logic. Later on the provisioning subsystem will include (persistent) caching of data on the target systems. The provisioning subsystem is using the repository subsystem for data storage.

Provisioning is currently based on ConnId framework which is a continuation of Identity Connectors Framework (ICF) created by Sun Microsystems. Identity connectors connect to the target systems and manage identity-related objects such as accounts. Although the identity connectors are somehow limited in functionality (e.g. they provide no support for asynchronous provisioning) and the design of identity connector interface is far from ideal, it is a practical and working framework. Yet we see that the ICF cannot provide sufficient advanced functionality and therefore we plan to evolve or replace it in the future.

Repository Subsystem

Main page: Repository Subsystem

The repository subsystem takes care of storing the identity data. It takes XML data objects used by other subsystems and converts them to any format appropriate for the data store. The primary purpose of the repository subsystem is to store a well-known set of IDM objects such as User, Account, Resource, Role, etc. There is an (extensible) XML schema provided for these data types to improve consistency of implementations and interoperability. Storing such "standard" objects is quite convenient; schema checking helps to detect problems early in the development cycle and therefore speed-up the development and customization.

The repository also stores generic data objects for whose the schema in not known during compile-time. This type of objects can be used by custom business logic or custom extensions of the IDM system. Support for generic objects provides almost unconstrained flexibility, but it might be a bit uncomfortable to use and it is not able to detect data inconsistencies (e.g. schema violation).

Current repository is based on a simple relational database schema. Different implementations may be provided later, but we expect that these will be more efficient and therefore more complex to understand and use. We also plan to be able to store a subgroup of the object types (like user or roles) in an LDAP-compliant server.

Infrastructure Subsystem

The infrastructure subsystem contains stateless components and utilities that are used by the rest of the system. Logging, tracing, spring framework and similar libraries belong there.

The libraries in infrastructure subsystem are stateless. That means they cannot rely on persistent configuration, use repository or maintain persistent state any other way. They must either be able to be initialized just by themselves (zero-configuration) or must be initialized from the upper layers (usually from the utility subsystem).

Infrastructure subsystem is also a place where the Data Model is materialized. It is present there both in form of XML schema (XSD) and Java classes.

Data Model

Main page: Data Model

Data design is an important part of the system architecture. Although we aim to encapsulate the data into subsystems and components as much as possible, the flow of data through interfaces forms a kind of system-wide data model. The data model describes basic form of an object, object types for common identity management objects as user or account and also some auxiliary types. The data model is internally segregated into (shearing) layers. Please see the Data Model page for more details.

Common Principles

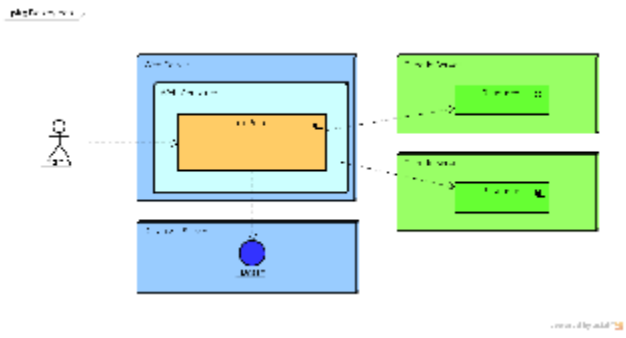
Main page: Common Mechanisms

There are several principles that apply to the system design as a whole. These form kind of "philosophy" for the system. They are described on separate pages:

- Form Follows Purpose
- IDM Means Integration
- Separation of Concerns
- Objects, XML, JSON and others
- Consistency Model

Deployment

midPoint is currently a Java web application and it needs a Java web container to run. It also needs a database for identity repository and also the systems that it is supposed to control (resources). This deployment is illustrated in the following diagram (click to enlarge).



However, midPoint is using a component-based architecture and future versions may provide more deployment options (e.g. OSGi). As switch to a different container or deployment model should not be difficult; we even expect that a specific midPoint solutions delivered by our partners that need more flexibility will introduce new custom ways of midPoint deployment. The architecture of midPoint takes this flexibility into account.

Future

While midPoint is initially designed to be an identity integration system, the future plans are significantly more ambitious. We believe that it is not enough to maintain users *within* a specific organization. The enterprises are *extending* and it is a frequent sight that partners, contractors and other types of *external* users participate in day-to-day business. The long-term goal of midPoint is to reach *out* of a single organization and provide integration with external entities.

Please note that we do not consider "cloud" systems to be really external. The could can be addressed with a traditional enterprise IDM and there is actually not much difference between cloud and traditional systems. This does not change the game much. The real long-term plan for midPoint is to really work *outside*, to put together teams from different companies, to allow real inter-company cooperation.

Documentation Dynamics

We will try to keep this page updated as we progress with the development. But expect some degree of "motion blur" all the time, as the system is evolving continually. The architecture and design are always a bit ahead of the development and these pages provide both description of what *is* and what is *planned* to be. There will be "TODO" labels around some pages. This is normal for a open source product`s Wiki. Therefore if you have the information that should replace the "TODO" label, feel free to edit the page and provide the information. (See also Feedback page).

See Also

- System Interactions
- Data Model
- Design Notes

External links

- What is midPoint Open Source Identity & Access Management
- Evolveum - Team of IAM professionals who developed midPoint