



PRACTICAL IDENTITY MANAGEMENT WITH MIDPOINT

BY RADOVAN SEMANČÍK ET AL.

VERSION 1.1
APRIL 2018

Practical Identity Management with midPoint

Radovan Semančík et al.

Evolveum

Book revision: 1.1

Publication date: April 2018

Corresponding midPoint version: 3.7.1

© 2015-2018 Radovan Semančík and Evolveum, s.r.o. All rights reserved.

This work is licensed under a [Creative Commons Attribution-NonCommercial-NoDerivatives 4.0 International License](https://creativecommons.org/licenses/by-nc-nd/4.0/).

Major sponsoring for this book was provided by:



Table of Contents

Chapter 1: Introduction.....	1
Chapter 2: Understanding Identity and Access Management.....	5
Directory Services and Other User Databases.....	5
Directory Servers are Databases.....	8
Single Directory Server Myth.....	8
Access Management.....	9
Web Single Sign-On.....	10
Authorization in Access Management.....	12
SAML and OpenID Connect.....	13
Kerberos, Enterprise SSO and Friends.....	16
Access Management and the Data.....	18
Advantages and Disadvantages of Access Management Systems.....	19
Single Access Management Myth.....	19
Practical Access Management.....	20
Identity Management.....	21
First Generation.....	22
Second Generation.....	23
What is This Identity Management, Anyway?.....	24
How Does Identity Management Technology Work?.....	31
Identity Management Connectors.....	31
Identity Provisioning.....	33
Synchronization and Reconciliation.....	34
Identity Management and Role-Based Access Control.....	35
Identity Management and Authorizations.....	37
Organizational Structure, Roles, Services and Other Wildlife.....	39
Everybody Needs Identity Management.....	41
Identity Governance and Compliance.....	42
Complete Identity and Access Management Solution.....	42
IAM and Security.....	44
Building Identity and Access Management Solution.....	46
Chapter 3: MidPoint Overview.....	48
How MidPoint Works.....	49
Case Study.....	52
Connectors and Resources.....	53
User and Accounts.....	55
Initial Import.....	58
Assignments and Projections.....	61
Roles.....	65
There Is Much More.....	67
What MidPoint Is Not.....	68
Chapter 4: Installation and Configuration Principles.....	70
Requirements.....	70
MidPoint Installation.....	71
MidPoint User Interface.....	71
User Interface Areas.....	72
User Interface Concepts.....	73

Object Details Page.....	75
MidPoint Configuration Basics.....	77
Configuration Objects.....	77
XML, JSON and YAML.....	79
Maintaining MidPoint Configuration.....	80
Looking Around The Installation.....	81
Logging.....	82
Chapter 5: Resources and Mappings.....	83
Resource Definitions.....	83
Connectors.....	85
Bundled and Deployed Connectors.....	86
Connector Configuration Properties.....	87
Testing the Resource.....	89
Resource Schema Basics.....	90
Hub and Spoke.....	91
Schema Handling.....	95
Attribute Handling.....	96
Mappings.....	98
Expressions.....	102
Script Expressions.....	103
Activation.....	104
Credentials.....	105
Complete Provisioning Example.....	106
Shadows.....	112
Chapter 6: Synchronization.....	115
Synchronization in MidPoint.....	116
Sources, Targets And Other Creatures.....	118
Inbound and Outbound Mappings.....	118
Correlation.....	121
Synchronization Situations and Reactions.....	122
Synchronization Tasks.....	126
Synchronization Example: HR Feed.....	128
HR Feed Recommendations.....	135
Synchronization and Provisioning.....	137
Mapping and Expression Tips and Tricks.....	139
Expression Functions.....	141
Resource Capabilities.....	141
Synchronization Example: LDAP Account Correlation.....	143
Reconciliation.....	151
Deltas.....	152
Live Synchronization.....	155
Conclusion.....	157
Chapter 7: MidPoint Development, Maintenance and Support.....	158
Professional Development.....	158
Open Source.....	158
MidPoint Release Cycle.....	159
MidPoint Subscriptions.....	159
MidPoint Community.....	160
Chapter 8: To Be Continued.....	162

Chapter 9: Additional Information.....167
MidPoint Wiki.....167
Samples.....167
Book Samples.....167
Story Tests.....168
MidPoint Mailing List.....168
Chapter 10: Conclusion.....170

Chapter 1: Introduction

*It's a dangerous business, Frodo, going out your door.
You step onto the road, and if you don't keep your feet,
there's no knowing where you might be swept off to.*

*– Bilbo Baggins
(The Lord of the Rings by J.R.R. Tolkien)*

Several years ago we started a project. Because we had to. Back then we didn't think too much about markets and business and things like that. We just focused on the technology. And the project simply went on. It had its ups and downs – but all the time there was pure engineering passion. The effort brought fruits and now there is a product like no other: midPoint.

MidPoint is an identity management and governance system. We built it from scratch. It is a comprehensive and feature-rich system. MidPoint can handle complete identity lifecycle, management, and parts of identity governance and compliance. It can speed up the process that creates accounts for new employee. It automatically disables accounts of an employee after his contract has expired. It manages assignment of roles and privileges to employees, partners, agents, contractors, customers, students – almost any kind of user. MidPoint keeps an eye that the policies are continually maintained. It governs the processes of access reviews. It provides auditing and reporting based on the identity data. And recent midPoint versions are heading towards the field of compliance: management of new and changed policies, evaluation how compliant is your setup according to those policies and governing smooth adoption of the policies.

MidPoint is such a comprehensive system that there are only few products in existence that can compete with midPoint. But midPoint has one critical advantage over the competing products: it is completely open source. Open source is the fundamental philosophy of midPoint. Open source is a critical aspect in the development of modern quality software. But open source principle is absolutely essential for the midPoint community: partners, contributors supporters and in fact all the engineers that work with midPoint. Open source strategy means that any engineer can completely understand midPoint. It also means that midPoint can be modified as needed, that issues can be fixed quickly and especially to ensure the continuity of midPoint development. After all these years with midPoint we simply cannot imagine using any identity technology which is not open source.

Significant part of the team that have built midPoint have been dealing with identity management deployments since early 2000s. The term “Identity and Access Management” was not even invented at that time. We have seen a lot of IAM solutions during our careers. The IDM system was the core of vast majority of these solutions. Whether it is

given by our point of view or whether that is the generic rule we do not know for sure. We leave that decision to the reader. All we know is that midPoint is a really useful tool. When it is used by the right hands it can do miracles. And this is exactly what this book is all about: the right use of midPoint to build a practical Identity Management solution. This book will tell you *how* to deploy a practical IDM solution. But it will also tell you *why* to do it in the first place. The book will explain not just the features and configuration options. It will also describe the motivation and the underlying principles. Understanding the principles is as at least as important as knowing the mechanics of an IDM product. The book usually describes how the things work when they work. But we also try to describe the limitations, drawbacks and pitfalls. The limitations are often much more important than the features, especially when designing a new solution on a green field.

First chapter is an introduction to the basic concepts of Identity and Access Management (IAM). It is very general and does not deal with midPoint at all. Therefore if you are familiar with Identity and Access Management feel free to skip the first chapter. If you are impatient and want to start directly with midPoint then do the same (you would do that anyway, right?). But in that case please find the time to return to the first chapter later. That chapter contains important information to put midPoint in broader context. You will need that information to build a complete IAM solution.

Second chapter describes the midPoint big picture. It shows how midPoint looks like from the outside. It describes how the midPoint is usually used and how it behaves. The purpose of this chapter is to familiarize the reader with midPoint workings and basic principles. It describes how midPoint is *used*.

Third chapter describes the basic concepts of midPoint deployment and configuration. It guides the reader through midPoint installation. It describes how midPoint is *customized* to suit the needs of a particular deployment. However midPoint customization is a very complex matter. This chapter describes just the basic principles. It will take most of the book to fill in the details.

Fourth chapter describes the concepts of *resource* and *mappings*. This is the bread-and-butter of an identity management. This chapter will tell you how to create very basic midPoint deployment, how to connect target systems and how to map and transform the data.

Fifth chapter is all about *synchronization*. Primary purpose of synchronization is to get the data from the resources such as HR system to midPoint. But midPoint synchronization is much more powerful than that. This chapter also expands the explanation of underlying midPoint principles such as mappings and deltas.

Sixth chapter provides overview of midPoint development process and overall approach. It is also explained how midPoint development is funded and how midPoint subscriptions work.

The following chapters are not written yet. The description of role-based access control, policies and all the other advanced topics is missing. This is still an early version of the book. Similarly to midPoint itself this book is written in an incremental and iterative way. Writing a good book is a huge task in itself and it takes a lot of time. But obviously a book like this is needed for midPoint community. Therefore we have decided not to wait until the book is complete. We have decided to be continuously publishing those chapters that are reasonably well finished. Something is usually better than nothing. Please be patient. The whole book will be finished eventually. And as always – your support, contributions and sponsoring may considerably speed up things here.

We would like to thank all the midPoint developers, contributors and supporters. There was a lot of people involved in midPoint during all these years. All these people pushed midPoint forward. But most of all we would like to thank the people that were there when the midPoint project was young and that are still there until this day. We would like to thank Katka Stanovská, Katka Valalíková, Igor Farinič, Ivan Noris, Vilo Repán, Pavol Mederly and Radovan Semančík. Those were the people that were and still are the force that drives midPoint into the future.

Anything that is stated in this book are the opinions of the authors. We work for Evolveum – a company that is also an IDM vendor. We have tried really hard to remain objective. However as hard as we might try some points of view are difficult to change. Therefore our opinions may be slightly biased. We have honestly tried to avoid any biases and follow proper engineering practices. And you are the judge and the jury in this matter. You, the reader, will decide whether we have succeeded or not. You have free access to all the necessary information to do that: this book is freely available as is all the midPoint documentation and the source code. We are not hiding anything. Unlike many other vendors we do not want or need to hide anything.

This work is licensed under a [Creative Commons Attribution-NonCommercial-NoDerivatives 4.0 International License](https://creativecommons.org/licenses/by-nc-nd/4.0/). This essentially means that you can freely use this book for a personal use. You can retrieve and distribute it at no cost. However you are not allowed to sell it, modify it or use any parts of this book in commercial projects. There is no direct profit that we make from this book. The primary reason for writing this book is to spread knowledge about midPoint. However even open source projects such as midPoint need funding. If you use midPoint in a commercial project that is a source of profit we think it is only fair if you share part of that profit with midPoint authors. Therefore we have chosen the CC BY-NC-ND license for this book. You can use this book freely to learn about midPoint. However this license does not give you right to take parts of this book and include it in your project documentation. You can point to this book by URL, but you are not allowed to pass this book to the customer as a part of product documentation in a commercial project. You are not allowed to use this book as material during commercial training. You are not allowed use the book in any way that generates profit. If you need to use this book in such a way please contact Evolveum and you can obtain special license to do this. The license fees collected in this way will be used to improve

midPoint and especially midPoint documentation. You know as well as we do that this is needed.

Following people have worked on the words and pictures that make up this book:

- Radovan Semančík (author and maintainer)
- Veronika Kolpaščíková (illustrations, corrections)

Yet there is much more people whose work was needed to make this work happen: midPoint developers, contributors, analysts and deployment engineers, specialists and generalists, theoretical scientists and practical engineers, technical staff and business people, people of Evolveum and the people that work for our partners, our families, friends and all the engineers and scientists for generations and generations past. We indeed stand on the shoulders of giants.

Chapter 2: Understanding Identity and Access Management

The beginning of knowledge is the discovery of something we do not understand.

– Frank Herbert

What is identity and access management? Answer to that question is both easy and very complex. The easy part is: Identity and access management (IAM) is a set of information technologies that deal with identities in the cyberspace. The complex part of the answer takes the rest of this book.

This book deals mostly with *Enterprise Identity and Access Management*. That is identity and access management applied to larger organizations such as enterprises, financial institutions, government agencies, universities, health care, etc. The focus is on managing employees, contractors, customers, partners, students and other people that cooperate with the organization. However many of the mechanisms and principles can also be applied to non-enterprise environments.

The story of identity and access management starts with information security. The security requirements dictate the need for authentication and authorization of the users.

Authentication is a mechanism by which the computer checks that the user is really the one he pretends to be. And authorization is a related mechanism by which the computer determines whether to allow or deny specific action to a user. Almost every computer system has some means of authentication and authorization.

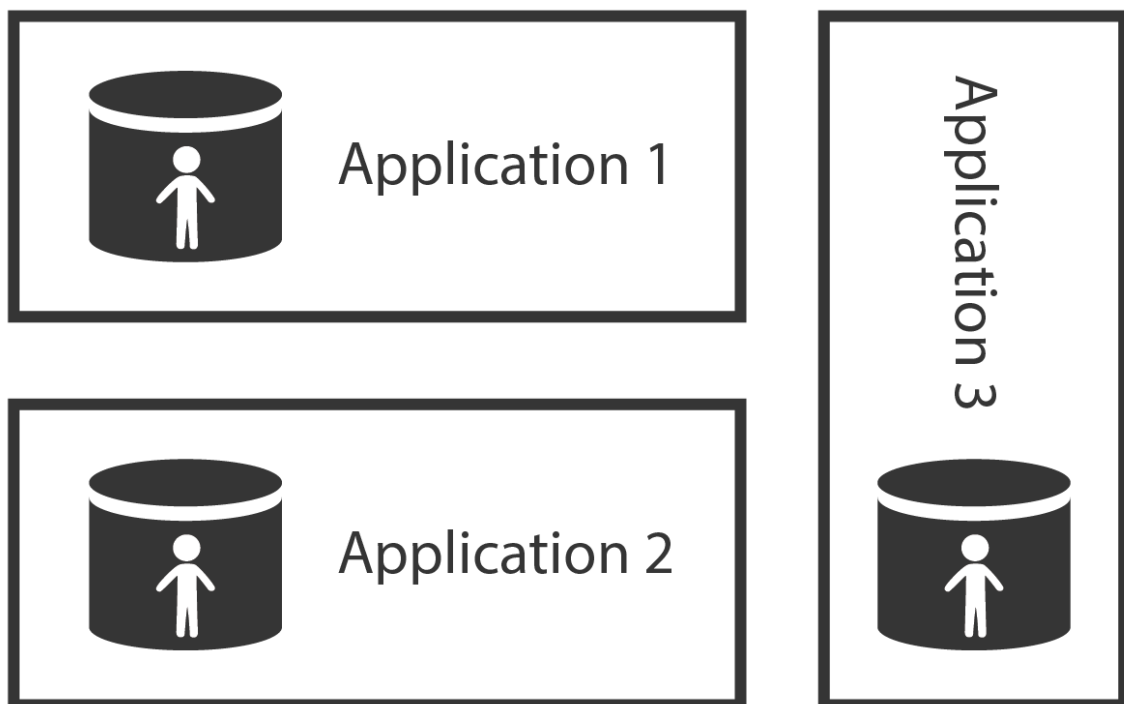
Perhaps the most widespread form of authentication is a password-based “log in” procedure. The user presents an identifier and a password. The computer checks whether the password is valid. For this procedure to work the computer needs an access to the database of all valid users and passwords. Early stand-alone information systems had their own databases that were isolated from the rest of the cyberspace. The data were maintained manually. But the advent of computer networking changed everything. Users were able to access many systems and the systems themselves were connected to each other. Maintaining an isolated user database in each system no longer made much sense. And that's where the real story of digital identity begins.

Directory Services and Other User Databases

The central concept of identity management is a data record that contains information about a person. This concept has many names: user profile, persona, user record, digital identity and many more. The most common name in the context of identity management is *user account*. Accounts usually hold the information that describes the real-world person using a set of attributes such as given name and family name. But probably the most

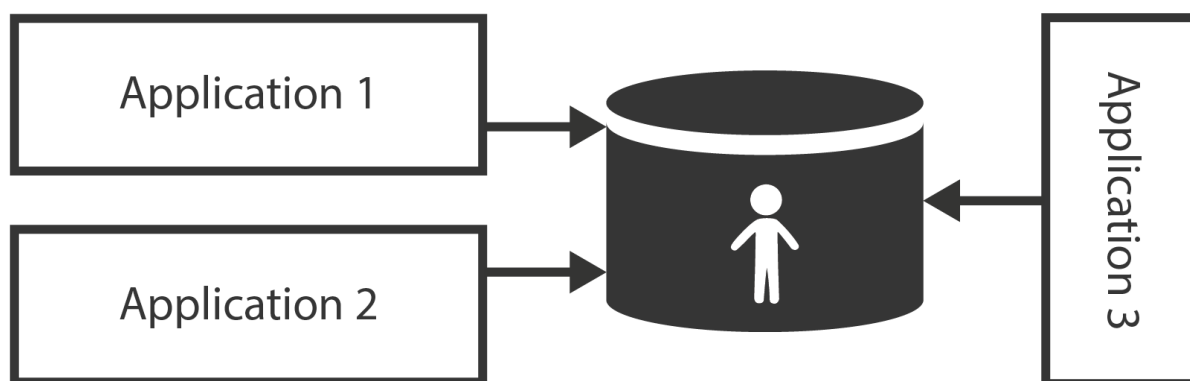
important part is the technical information that relates to operation of an information system for which the account is created. This includes operational parameters such as location of users home directory, wide variety of permission information such as group and role membership, system resource limits and so on. User accounts are resented in a wide variety of forms ranging from relational database records through structured data files to semi-structured text files . But regardless of the specific method used to store and process the records the *account* is undoubtedly one of the most important concepts of IAM field. And so are the databases where the accounts are stored. Because the accounts, being data records, have to be stored somewhere.

The account databases are as varied as are the user records. Most user databases in the past were implemented as an integral part of the monolithic information system using the same database technology as the system itself used. This is an obvious choice and it remains very popular even today. Therefore many accounts are stored in relational database tables and similar application data stores.



Application data stores are usually tightly bound to the application. Therefore accounts stored in such databases are difficult to share with other applications. However, sharing account data across the organization is more than desirable. It makes very little sense to maintain account data in each database separately – especially if most of them are the same in each application. Therefore there is a strong motivation to deploy account databases that can be shared by many applications.

Shared data storage is the purpose of directory servers. While application databases usually use their own proprietary protocol, the directory servers implement standardized protocols. While databases are built for completely custom data model, directory servers usually extend standardized data model which improves interoperability. While databases are often heavyweight and expensive to scale, directory servers are lightweight and designed for massive scalability. That makes directory servers ideal candidates for shared account database.



Shared identity store is making user management easier. An account needs to be created and managed in one place only. Authentication happens in each application separately. But as the applications use the same credentials from the shared store, the user may use the same password for all the connected applications.

Lightweight Directory Access Protocol (LDAP) is a standard protocol for directory service access. It is an old protocol by Internet standards with roots going as far back as 1980s. But it is far from being obsolete. Quite the contrary. It is a very efficient binary protocol that was designed to support massively distributed shared databases. It has small set of well defined simple operations. The operations and the data model implied by the protocol allow very efficient data replication and horizontal scalability of directory servers. This allows low latencies and extreme throughput for read operations. The horizontal scalability and relative autonomy of directory server instances increase the availability of the directory system. These benefits usually come at the expense of slow write operations. But as identity data are often read but seldom modified this is more than an acceptable trade-off. Therefore LDAP-based directory servers were and still remain the most popular databases for identity data.

LDAP is one of the precious few established standards in the IAM field. And almost all organizations store identities in LDAP-enabled data stores. Therefore we will be getting back to the LDAP protocol many times in this book.

Identity management solutions based on shared directory servers are simple and quite cost-efficient. We have been giving the same advice for many years: if you can connect all

your applications to an LDAP server, do not think too much about it and do it. The problem is that this usually works only for very simple systems.

Directory Servers are Databases

Directory servers are just databases that store information. Nothing more. The protocols and APIs used to access directory servers are designed as database interfaces. It means that they are excellent for storing, searching and retrieving data. While the data in account may contain entitlement information (permissions, groups, roles, etc.), identity stores are not well suited to evaluate them. I.e. directory server can provide information what permissions an account has but it is not designed to make a *decision* whether to allow or deny a specific operation. And there are other issues. Directory servers do not contain data about user *sessions*. It means that directory servers do not know whether user is currently logged in or not. Many directory servers are used for basic authentication and even authorization. But the directories were not designed to do it and therefore provide only the very basic capabilities. There are plug-ins and extensions that provide partial capabilities to support authentication and authorization. But that does not change the fundamental design principles. Directory servers are databases, not authentication or authorization servers.

Single Directory Server Myth

Shared directory server makes user management easier. But this is not a complete solution and there are serious limitations to this approach. The heterogeneity of information systems makes it nearly impossible to put all required data into a single directory system.

The obvious problem is the lack of a single, coherent source of information. There are usually several sources of information for a single user. For example a HR system is authoritative for the existence of a user in the enterprise and for assignment of employee identifier. The Management Information System is responsible for determination of user's roles (e.g. in project-oriented organizational structure). The inventory management system is responsible for assigning telephone number to the user. The groupware system is authoritative source of the user's e-mail address and other electronic contact data. There are usually 2 to 20 systems that provide authoritative information for a single user. Therefore there is no simple way how to feed and maintain the data in the directory system.

Almost all complex applications need local user database. They must store the copies of user records in their own databases to operate efficiently. For example, large billing systems cannot work efficiently with external data (e.g. because of relational database *join*). Therefore even if directory server is deployed these applications still need to maintain a local copy. Keeping the copy synchronized with the directory data may seem like a simple task, but it is not. And there are legacy systems which usually cannot access the external data at all (e.g. they do not support LDAP protocol).

Some services need to keep even more state than a database record. For example file servers usually create home directories for users. While the automation of state creation can usually be done on-demand (e.g. create user directory at first user log-on), the modification and deletion of state is much more difficult. Directory server will not do that.

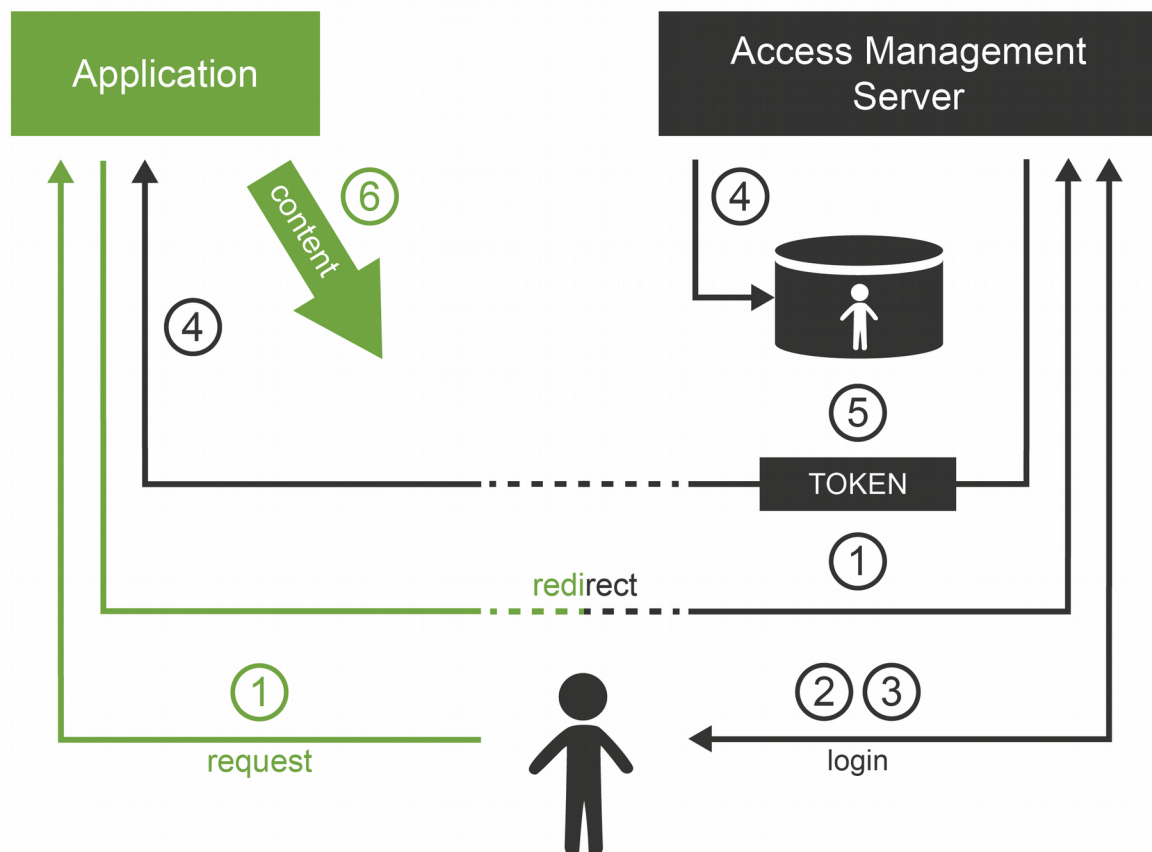
But perhaps the most painful problem is the complexity of access control policies. Role names and access control attributes may not have the same meaning in all systems. Different systems usually have different authorization algorithms that are not mutually compatible. While this issue can be solved with per-application access control attributes, the maintenance of these attributes is seldom trivial. And if every application has its own set of attributes to control access control policies then the centralized directory does provide very little advantage. The attributes may as well reside in the applications themselves. And that's exactly how most deployments end up. Directory servers only contain groups that roughly approximate RBAC roles. The access control policies and fine-grained authorizations are still maintained in the applications.

Single directory approach is feasible only in very simple environments or almost entirely homogeneous environments. In all other cases there is a need to use also other identity management technologies. This does not mean that the directory servers are useless. Quite the contrary. They are very useful when used properly. They just cannot be used alone. More components are needed to build a complete solution.

Access Management

While directory systems are not designed to handle complex authentication the *access management* (AM) systems are built to handle just that. They handle all the flavors of authentication, access and partially also authorization. The principle of all access management systems is basically the same:

1. Access management system gets between the user and the target system. This can be done by a variety of mechanisms but the most common is that the applications themselves redirect the user to the AM system if they do not have existing session.
2. Access management system prompts user for username and password, requests the certificate, creates a challenge and prompts for the response or in any other way initiates the authentication procedure.
3. User enters the credentials.
4. Access management system checks the validity of credentials and evaluates access policies.
5. If access is allowed then the AM system redirects user back to the application. The redirection usually contains an access token: small piece of information that tells the application that the user is authenticated.
6. Application validates the token, creates local session and allows the access.



After that procedure the user works with the application normally. So only the first access goes through the AM server. This is important for AM system performance and sizing.

The applications only need to provide code that integrates with the AM system. Except for that applications do not need to provide any authentication code at all. It is the AM system that prompts for password, not the application. This is a difference when compared to LDAP-based authentication. In the LDAP case it is the application that prompts for password. In the AM case the Access Management server does everything. Many applications do not even care how the user was authenticated. All they need to know is that he was authenticated and that the authentication was good enough. This feature brings a very desirable flexibility to the entire system. The authentication mechanism can be changed at any time without disrupting the applications. We live in an era when passwords are migrated to a stronger authentication mechanisms. The flexibility that the AM-based approach brings may play a key part in that migration.

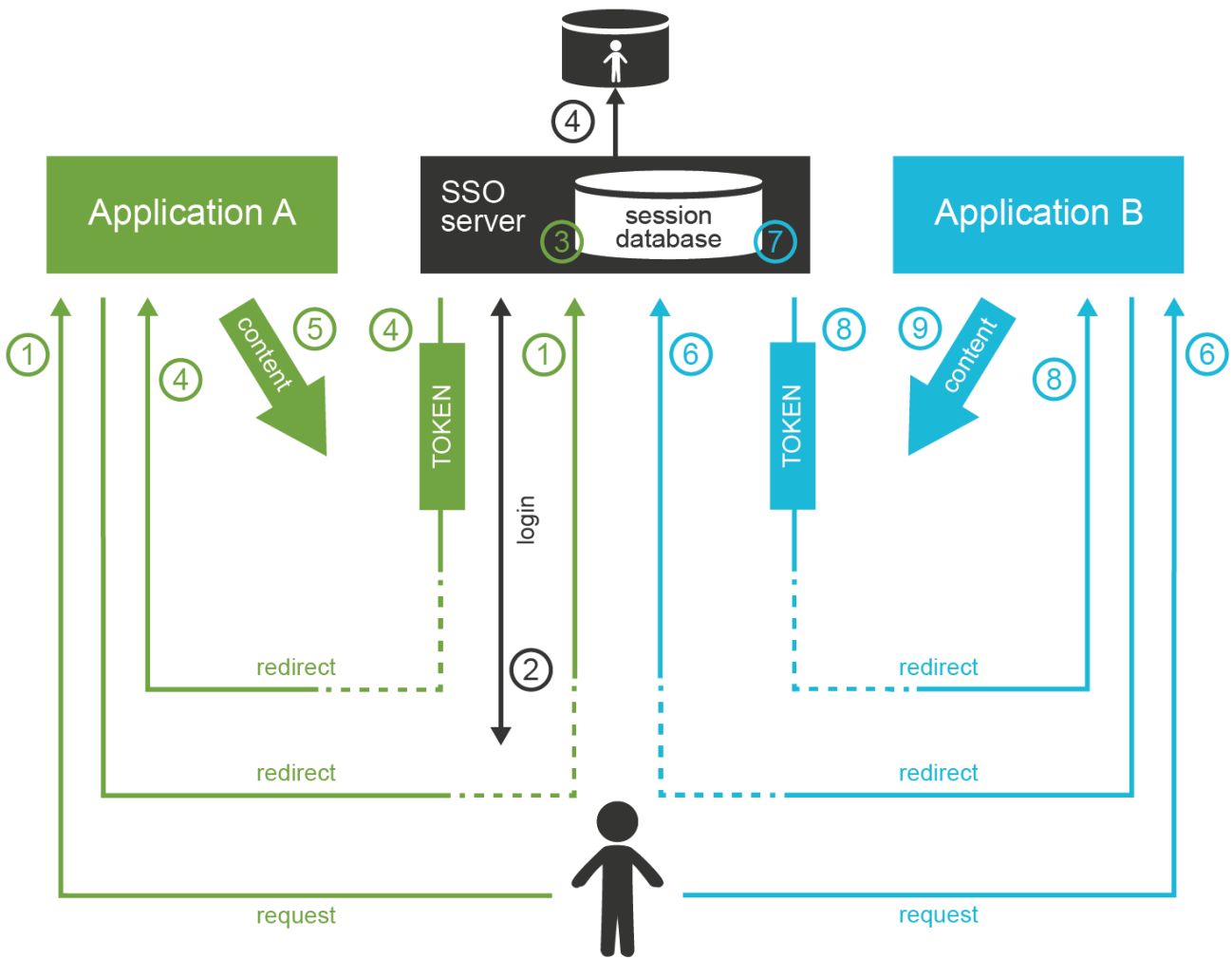
Web Single Sign-On

Single Sign-On (SSO) systems allow user to authenticate once and then to access number of different system. As the authentication usually depends on direct user interaction there are many systems that implement the SSO features using variety of different mechanism.

There also many SSO systems for web applications, however it looks like these systems are all using the same basic principle. The web SSO is based on the general access management principle described above:

1. Application A redirects the user to the access management server (SSO server).
2. The access management server authenticates the user.
3. The access management server establishes session (SSO session) with the user browser. This is the crucial part of the SSO mechanism.
4. User is redirected back to the application A. Application A usually establishes local session with the user.
5. User interacts with application A
6. When user tries to access application B, the application B redirects user to the access management server.
7. The access management server checks for existence of SSO session. As the user authenticated with the access management server before, there is a valid SSO session.
8. Access management server does not need to authenticate the user again and immediately redirects user back to application B.
9. Application B establishes local session with the user and proceeds normally.

The user usually does not even realize that he was redirected when accessing application B. There is no interaction between the redirects and the processing on the access management server is usually very fast. It looks like the user was logged into the application B all the time.



Authorization in Access Management

The request of a user accessing an application is directly or indirectly passed through the access management server. Therefore the access management server can, in theory, analyze the request and evaluate whether the user request is authorized or not. But the situation is much more complicated in practice.

The AM server usually intercepts only the first request to access the application as it would be a performance impact to intercept all the requests. After the first request the application established local sessions and proceeds with the operation without any communication with the AM server. Therefore the AM server can only evaluate authorization during the first request. Which means it can only evaluate a very rough-grained authorization decisions. In practice it usually means that the AM server can make authorization decisions that a particular user can access all parts of a particular application or that he cannot access the application at all. The AM server usually cannot make any finer-grain decisions just by itself.

Some AM systems provide agents that can be deployed to applications and that enforce a finer-grain authorization decisions. Such agents often rely on HTTP communication and they are making decisions based on the URLs that the user is accessing. This approach might work well in 1990s, but it has only very limited applicability in the age of AJAX, single-page web applications and mobile applications.

Sophisticated applications often need to make authorization decisions based on context which is simply not available in the request or user profile at all. E.g. an e-banking application may allow or deny a transaction based on the sum of previous transactions that were made earlier that day. While it is usually possible to synchronize all the authorization information into the user profile, it is usually not desirable. It would be a major burden to keep such information updated and consistent.

AM systems often come with a promise to unify authorization across all the applications and to centralize management of organization-wide security policies. Unfortunately, such promises are almost always false. The AM system can theoretically evaluate and enforce some authorization statements. And this may work well during demonstrations and in very simple deployments. But in complex practical deployments this capability is extremely limited. Vast majority of the authorization decisions is carried out by each individual application and is completely outside of the reach of an AM system.

SAML and OpenID Connect

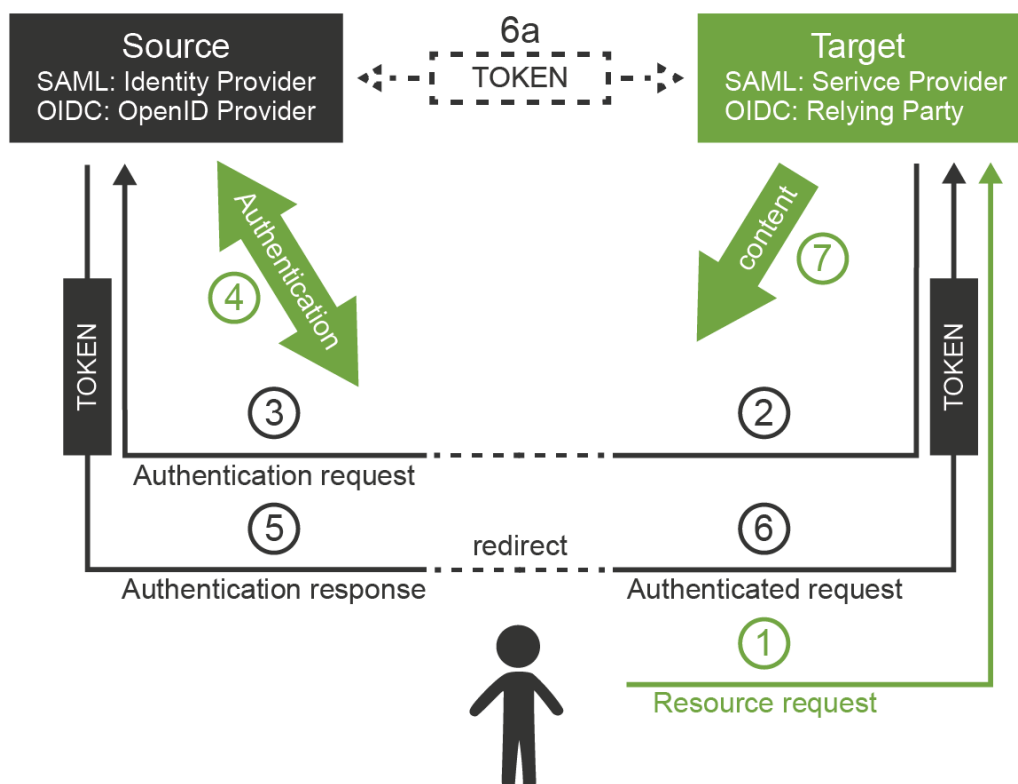
Many access management systems use proprietary protocols to communicate with the applications and agents. This is obviously an interoperability issue – especially when the AM principles are used in the Internet environment. And it is the Internet that motivated standardization in this field.

The first widespread standardized protocol in this field was Security Assertion Markup Language (SAML). The original intent of SAML was to allow cross-domain sign-on and identity data sharing across organizations on the Internet. It is quite complex protocol and security token format heavily based on XML standards. SAML philosophy and design is very close to the SOAP-based web services and in fact it was designed to integrate easily into the web service world. SAML specifications are long, divided into several profiles, there are many optional elements and features and overall SAML is a set of very rich and flexible mechanisms.

Primary purpose of SAML is transfer identity information between organizations. There are big federations with tens or hundreds of organizations based on SAML tokens and protocols. Many e-government solutions are based on SAML, there are big partner networks running on SAML and overall it looks like SAML is a success. But SAML was a victim of its own flexibility and complexity. And the latest fashion trends are not very favorable to SAML. XML and SOAP-based web service mechanisms are getting out of fashion. All of that has probably motivated the inception of other protocols.

The latest fashion favors RESTful services and similar simpler architectural approaches. All of that probably contributed to the development of OpenID Connect protocol (OIDC). OpenID Connect is based on much simpler concepts than SAML, but it is reusing the same basic principles. OpenID connect has a very eventful history. It all started with a bunch of homebrew protocols such as LID or SXIP that are now mostly forgotten. That was followed by the development of OpenID protocol, which was still very simple. OpenID gained some attention especially with providers of Internet services. But despite its simplicity OpenID was not very well engineered and it quickly reached its limits. It was obvious that OpenID needs to be significantly improved. Then there was almost unrelated protocol called OAuth which was designed for management of cross-domain authorizations. That protocol was developed into something that was almost, but not quite, entirely unlike the original OAuth protocol. It is called OAuth2. And in fact it is not really a protocol. It is rather a vaguely-defined framework to build other protocols. OAuth2 framework was used to build a cross-domain authentication and user profile protocol. This new protocol is much more similar to SAML than to the original OpenID, so it was an obvious choice to call it OpenID Connect. Some traditions are just worth maintaining.

Now there are two protocols that are using the same principle and do almost the same thing. The principle is illustrated in the following diagram.



The interaction goes like this:

1. User is accessing a resource. This can be web page or web application on the target site.
2. Target site does not have a valid session for the user. Therefore it redirects user browser to the source site. It will add authentication request into that redirect.
3. Browser follows the redirect to the source site. The source site gets the authentication request and parses it.
4. If the user is not already authenticated with the source site then the authentication happens now. The source site prompts for username, password, certificate, one-time password or whatever credential that is required by the policy. With a bit of luck the authentication succeeds.
5. The source site redirects the browser back to the target site. The source site adds authentication response to the redirect. The most important part of the response is a token. The token directly or indirectly asserts user's identity.
6. The target site will parse the authentication response and process the token. The token may be just a reference to the real token (SAML artifact) or it may be access key to another service that provides the identity (OIDC UserInfo). In that case the target site makes another request (6a). This request is usually a direct one and does not use browser redirects. One way or another, the target site now has claims about user identity.
7. Target site evaluates the identity, processes authorizations and so on. Local session with the user is usually established at this point to skip the authentication redirects on the next request. The target site finally provides the content.

Following table compares the terminology and technologies used in SAML and OIDC worlds.

	SAML	OpenID Connect
Source site	Identity Provider (IdP)	Identity Provider (IDP) or OpenID Provider (OP)
Target site	Service Provider (SP)	Relying Party (RP)
Token	SAML Assertion (or artifact)	ID token, access token
Intended for	Web applications, web services (SOAP)	Web applications, mobile applications, REST services
Based on	N/A	OAuth2
Data representation	XML	JSON
Cryptography framework	XMLenc, XMLdsig	JOSE
Token format	SAML	JWT

Careful reader will notice the similarity with the web-based access management mechanisms. That's right. This is the same wheel reinvented over and over again. However, to be completely honest we have limited the description to the flows for web browser. Both SAML and OIDC has broader applicability and in these cases the differences are more obvious. But the web browser case nicely illustrates the principles and similarities of SAML, OpenID Connect and also the simple web-SSO systems.

Maybe the most important differences between SAML, OIDC and web-SSO systems is the intended use:

- SAML was designed for the web applications and web services world. It will handle centralized (single-IDP) scenarios very well, but it can also work in decentralized federations. Go for SAML if you are using SOAP and WS-Security or if you plan to build big decentralized federation.
- OpenID Connect was designed mostly for use with social network and similar Internet services. Its philosophy is still somehow centralized. It will work well if there is one strong identity provider and many relying parties. Technologically it will fit into RESTful world much better than SAML. Current fashion trends are favorable to OIDC.
- Web-SSO systems are designed to be used inside a single organization. This is ideal to implement SSO between several customer-facing applications so the customers will have no idea that they interact with many applications and not just one. The web-SSO systems are not designed to work across organizational boundaries.

Although SAML and OIDC are designed primarily for cross-domain use, it is no big surprise to see them inside a single organization. There is a clear benefit in using an open standardized protocol instead of a proprietary mechanism. However, it has to be expected that the SSO system based on SAML or OIDC will have slightly more complicated setup than a simple Web-SSO system.

Kerberos, Enterprise SSO and Friends

Many of us would like to think that everything is based on web technologies today and that non-web mechanisms are a thing of yesterday. But there are still cases that are not web-based and where web-based SSO and AM mechanisms will not really work. There is still a lot of legacy applications especially in the enterprise environment. Applications based on rich clients or even character-based terminal interactions are not that difficult to find. And then there are network operating systems such as Windows and numerous UNIX variants, there are network access technologies such as VPN or 802.1X and so on. There are still many cases where web-based access management and SSO simply won't work.

These technologies usually pre-date the web. And honestly, the centralized authentication and SSO are not entirely new ideas. Therefore it is no big surprise that there are authentication and SSO solutions even for non-web applications.

The classic classroom example of enterprise SSO system is Kerberos. The protocol originated at MIT in 1980s. It is a single-sign on protocol for operating systems and rich clients based on symmetric cryptography. Even though it is a cryptographic protocol it is not too complicated and it definitely stood the test of time. It has been used to this day especially for authentication and SSO of network operating systems. It is a part of Windows network and it is often the preferred solution for authentication of UNIX servers. The most serious limitation of Kerberos is given by its use of symmetric cryptography which makes key management very difficult when the Kerberos realm gets very big. And it is not very realistic to use Kerberos in cross-domain scenarios. However inside a closed organization Kerberos is still very useful solution.

The major drawback in using Kerberos is that every application and client needs to be “kerberized”. In other words everybody that wants to take part in Kerberos authentication needs to have Kerberos support in one's software. There are kerberized versions of many network utilities so this is usually not a problem for UNIX-based networks. But it is a problem for generic applications. There is some support for Kerberos in common web browsers which is often referred to as “SPNEGO”. However this support is usually limited to interoperability with the Windows domains. Therefore even though Kerberos is still useful for operating system SSO it is not a generic solution for all applications.

Many network devices use RADIUS protocol for what they call Authentication, Authorization and Accounting (AAA). However RADIUS is a back-end protocol. It does not take care of client interactions. The goal of RADIUS is that the network device (e.g. WiFi access point, router or VPN gateway) can validate user credentials that it has received as part of other protocol. The client connecting to VPN or WiFi network does not know anything about RADIUS. Therefore RADIUS is similar to the LDAP protocol and it is not really an access management technology.

Obviously there is no simple and elegant solution that can provide SSO for all enterprise applications. However one technology called “Enterprise Single Sign-On” (ESSO) appeared in 1990s and early 2000s. The ESSO approach was to use agents installed on every client device. The agent will detect when login dialog appears on the screen, fills in the username and password and submits the dialog. If the agent is fast enough the user does not even notice the dialog and this creates the impression of Single Sign-On. However, there are obvious drawbacks. The agents need to know all the passwords in a cleartext form. There are ESSO variations with passwords randomly generated or even single-user passwords which partially alleviates this problem. But the drawback is that the ESSO also needs to be integrated with password management of all the applications, which is not entirely easy. However the most serious drawback of ESSO are the agents. These only work on workstations that are strictly controlled by the enterprise. Yet the world is changing, enterprise perimeter has efficiently disappeared and the enterprise cannot really control all the client devices. Therefore also ESSO is now mostly a thing of the past.

Access Management and the Data

Access Management servers and Identity Providers need the data about users to work properly. But it is complicated. The purpose of access management systems is to manage access of users to the applications. Which usually means processing authentication, authorization (partially), auditing the access and so on. For this to work the AM system needs access to the database where the data about users are stored. It needs access to usernames, passwords and other credentials, authorization policies, attributes and so on. The AM systems usually do not store these data themselves. They rely on external databases. And in vast majority of cases the databases are stored in the directory services. This is obvious choice: directory services are lightweight, highly available and extremely scalable. The AM system usually need just a simple attributes, therefore the limited capabilities of directory data models are not a limiting factor here. Access management and directory services is an obvious and very smart match.

However, there is one critical issue – especially if the AM system is also used as a single sign-on server. The data in the directory service and the data in the applications must be unified and absolutely consistent. E.g. it is a huge problem if one user has different usernames in several applications. Which username should he use to log in? Which username should be sent to the applications? There are ways how to handle such situations, but this is usually very cumbersome and expensive. It is much easier to unify the data before the AM system is deployed.

Even though the “M” in AM stands for “management”, typical AM system has only a very limited data management capabilities. The AM systems usually assume that the underlying directory system is already properly managed. E.g. a typical AM system has only a very minimalistic user interface to create, modify and delete user records. Some AM systems may have self-service functionality (such as password reset), but that functionality is usually also very limited. Even though the AM relies on the fact that the data in the AM directory service and the data in applications are consistent there is usually no way how to fully synchronize the data by using the AM system itself. There may be methods for on-demand or opportunistic data updates, e.g. creating user record in the database when the user logs in for the first time. But there are usually absolutely no solutions for deleting the records or for updating the records of inactive users.

Therefore the AM systems usually cannot be deployed alone. The underlying directory service is almost always a hard requirement for even the most humble AM functionality. But for the AM system to really work properly it needs something to manage and synchronize the data. Identity Management (IDM) system is usually used for that purpose. And in fact, it is usually strongly recommended to deploy the directory and the IDM system before the AM system. The AM system cannot work without the data. And if it works on data that are not maintained properly it will not take a long time until it fails.

Advantages and Disadvantages of Access Management Systems

Access management and web SSO systems have significant advantages. Most of the characteristics are given by the AM principle of centralized authentication. As the authentication is carried out by a central access management server it can be easily controlled and audited. Such centralization can be used to consistently apply authentication policies and to easily change them. It also allows better utilization of an investment into authentication technologies. E.g. multi-factor or adaptive authentication can be quite expensive if it has to be implemented by every application. But when it is implemented in the AM server then it is re-used by all the applications without additional investment.

However there are also drawbacks. As the access management is centralized it is obviously a single point of failure. In case that the AM server fails nobody would be able to log in, which usually means major impact on system functionality. Therefore AM servers need to be highly available which is not always an easy task. The AM servers need a very careful sizing as they may easily become a performance bottlenecks. But perhaps the most severe drawback is the total cost. The cost of the AM server itself is usually not a major issue. But the server needs to be integrated with every application. Even though there are some standards in this field the integration is usually quite painful. The support for AM standards and protocols in the applications is far from being perfect. Especially older enterprise applications need to be modified to switch their authentication subsystem to the AM server. This is often so costly that the adoption of AM technologies is usually limited just to few enterprise applications.

Even though many enterprises are planning deployment of an AM system as their first step in the IAM project, this step seldom succeeds. The project usually plans to integrate 50-80% applications into the AM solution. But the reality usually is that only a handful of applications is integrated with the AM system. The rest of the applications is integrated using an IDM system that is hastily added to the project. Therefore it is better to plan ahead: analyze the AM integration effort and make a realistic plan for the AM solution. Make sure that the AM can really bring the promised benefits. Starting with IDM and adding AM part later is often much more reasonable strategy.

Single Access Management Myth

The redirection approach of Access Management systems assumes that the user has something that can display any authentication prompts and carry out user interaction. Which is usually a web browser. Therefore this approach applies mostly to conventional web-based applications. Variations of this approach are also applicable to single-page web applications. However this approach is usually not directly applicable for applications that use rich clients, operating system authentication and mobile applications as in these cases the browser is not the primary environment that can be used to carry out the authentication. There are some solutions that usually rely on embedded browser, however that does not change the basic fact that the AM technologies are not entirely suitable for

these applications. These applications usually rely on Kerberos as an SSO system or do not integrate with any SSO system at all.

As typical enterprise environment is composed of a wild mix of technologies and not all of them are entirely web-based. Therefore it is unlikely that a single AM system can apply to everything that is deployed in the enterprise. Authentication is very tightly bound to the user interaction, therefore it depends on the method how the user interacts with the application. As the user is using different technologies to interact with the web application, mobile application and operating system then it is obvious that also authentication and SSO methods for these systems will be different.

Therefore it has to be expected that there will be several AM or SSO systems in the enterprise, each one serving its own technological island. And each island needs to be managed.

Practical Access Management

Unifying access management system, Single Sign-On, cross-domain identity federation, universally-applicable 2-factor authentication – there are the things that people usually want when they think about Identity and Access Management (IAM). And these are all perfectly valid requirements. However, everything has its cost. And especially in the access management field the cost is very difficult to estimate because majority of the cost is not in the AM software. Huge part of the total cost is hidden inside existing applications, services and clients. All of this has to be considered when planning an access management project.

Even though the AM is what people usually want, it is usually wise **not** to start with AM as the first step. AM deployment has many dependencies: unified user database, managed and continually synchronized data, applications that are flexible enough to be integrated and so on. Unless your IT infrastructure is extremely homogeneous and simple it is very unlikely that these dependencies are satisfied. Therefore it is almost sure that the AM project attempted at the beginning of the IAM program will not reach its goals or fail entirely. However if the AM project is properly scoped and planned and has realistic goals there is high chance of success and it can bring substantial value.

Perhaps the best way to evaluate an AM project is to ask several questions:

- Do I really need access management for all applications? Do I need 100% coverage? Can I afford all the costs? Maybe it is enough to integrate just a couple of applications that are source of the worst pain. Do I know which applications are these? Do I know what my users really use during they workday? Do I know what they need?
- Do I fully realize that the effect of AM and SSO is mostly user convenience? What are the real security benefits of AM deployment? Will I be disabling the native authentication to the applications? Even for system administrators? What will I do in case of administration emergencies (e.g. system recovery)? Would system

administrators still be able to circumvent the AM system? If yes then what is the real security benefit? If not then what will be the recovery procedure in case the AM system fails?

- Do I really need SSO for older and rarely used applications? What is the real problem here? Is the problem that users are entering the password several times per day? Or is the real problem that they have to enter different username or password to different applications and they keep forgetting the credentials? Maybe simple data cleanup and password management will solve the worst problems and I can save a huge amount of money on AM project?

The AM technologies are the most visible part of the IAM program. But it is also the most expensive part and the most difficult to set up and maintain. Therefore do not underestimate other IAM technologies and do not try to solve every problem with AM golden hammer. Using the right tool for the job is a good approach in every situation. But in IAM program it is absolutely critical for success.

Identity Management

Identity management (IDM) is maybe the most overlooked and underestimated technology in the whole identity and access management (IAM) field. Yet IDM is a crucial part of almost every IAM solution. And it is IDM that can bring substantial benefits to almost any organization. So, what that mysterious IDM really is?

Identity management is exactly what the name says: it is all about managing identities. It is about the processes to create Active Directory accounts and mailboxes for a new employee. IDM makes it possible to immediately disable all access to a suspicious user during a security incident. IDM takes care of adding new privileges and removing old privileges of users during reorganization. IDM makes sure that all the accounts are properly disabled when the employee leaves the company. IDM records access privileges of temporary workers, partners, support engineers and all the third-party identities that are not maintained in your human resources (HR) system. IDM automates the processes of role request and approval. IDM records every change in user privileges in the audit trail. IDM governs the annual reviews of roles and access privileges. IDM makes sure the copies of user data that are kept in the applications are synchronized and properly managed. And IDM does many other things that are absolutely essential for every organization to operate in an efficient and secure manner.

It looks like IDM is the best thing since the sliced bread. So where's the catch? Oh yes, there is a catch. Or it is perhaps better to say that there was a catch. The IDM systems used to be expensive. Very expensive. The IDM systems used to be so expensive that it was very difficult to justify the cost even with such substantial and clear benefits. But that time is over now.

Note: Terminology. The term *identity management* is often used for the whole identity and access management (IAM) field. This is somehow confusing because technologies such

as single sign-on or access management do not really manage the identities. Such technologies manage the access to the applications. Even directory servers do not exactly *manage* the identities. Directory servers store the identities and provide access to them. There is in fact one whole branch of technologies that manage identities. Those systems are responsible for creating identities and maintaining them. Those are sometimes referred to as *identity provisioning*, *identity lifecycle management* or *identity administration* systems. But given the current state of the technology such names are indeed an understatement. Those systems can do much more than just provisioning or management of identity lifecycle. We will refer to these systems simply as *identity management* (IDM) systems. When we refer to the entire field that contains access management, directory services, identity management and governance we will use the term *identity and access management* (IAM).

Let's start at the beginning. In 1990s there was no technology that would be clearly identified as "identity management". Of course, all the problems above had existed almost since the beginning of modern computing and there had always been some solutions for them. But most of that solutions were based on paperwork and scripting. The situation was not critical until the big system integration wave spread through the industry in 1990s and 2000s. As data and processes in individual applications got integrated the IDM problems became much more pronounced. The manual paper-based processes were just too slow for the age of information superhighways. The scripts were too difficult to maintain in the world where new application is deployed every couple of weeks. The identity integration effort naturally started with the state-of-the-art identity technology of the day: directory services. But as we have already shown the directories were not ideal for the job. The directories were not entirely suitable for environment where people thought that LDAP is some kind of exotic disease, where usernames and identifiers were assigned quite randomly by application administration teams and where every application insisted that the only authoritative data are those stored in its own database.

First Generation

The integration problems motivated the inception of IDM technologies in early 2000s. Early IDM systems were just data synchronization engines that were somehow hard-coded to operate with users and accounts. Some simple Role-Based Access Control (RBAC) engines and administration interfaces were added a bit later. During mid-2000s there were several more-or-less complete IDM systems. This was the first generation of the IDM systems. These systems were able to synchronize identity data between applications and also provide some basic management capabilities. But even such a simple functionality was a huge success. The IDM systems could synchronize the data without any major modification of the applications, therefore they brought the integration cost to a reasonable level. The problem was that the cost of the IDM systems themselves was quite high. And these systems were still somehow crude, therefore the configuration and customization required a very specialized class of engineers. This made the deployment of IDM solutions prohibitively expensive for many mid-size and smaller organizations. And even big

organizations often deployed IDM solution with quite limited features to make the cost acceptable. These IDM systems later evolved and improved. And there were companion products for governance and compliance that augmented the functionality. But it often was almost impossible to change the original architecture or a product. Therefore many first-generation products still struggle with limitations that originated in the early product design.

All the first-generation IDM systems were commercial closed-source software. Many of these products are still available on the market and they are even considered to be leaders. However, the closed-source character of the IDM products is itself a huge problem. Every IDM solution has to be more-or-less customized. It has to be the IDM system that adapts and not the applications. Requiring each application to adapt to a standardized IDM interface means a lot of changes in a lot of different places, platforms and languages. The total cost adds up to a huge number. This is being tried from time to time but it almost always fails. It is not a practical approach. While there are many applications in the IT infrastructure, there is just one IDM system. If the IDM system adapts to applications and business processes, the changes are usually smaller and they are all in one place and implemented in a single platform. So the IDM system must be able to adapt. And it has to adapt a great deal and adapt easily. Closed-source software is notoriously bad at adapting to requirements that are difficult to predict. Which in practice means that the IDM projects based on first-generation products were hard and expensive. Also the closed-source software is very prone to vendor lock-in. Once the IDM system is deployed and integrated it is extremely difficult to replace with a competing system. As the closed-source vendor is the only entity that can modify the system and the system cannot be efficiently replaced the end customer is not in a position to negotiate. Which means high maintenance costs. Therefore the first generation of IDM system was a commercial success. But only for the vendors. This generation of IDM solutions really worked only for some customers and even there the real long-term benefits were often questionable.

Second Generation

We can only speculate what was the reason, but the fact is that around the years 2009-2011 several very interesting new IDM products appeared on the market. One interesting thing is that all of them are more-or-less open source. But there is much more. Although all of the products are targeting the IDM space they are significantly different. They differ in their philosophy and governing principles. One product is just a simple framework that can be programmed and extended ad nauseam while another product is a full-fledged IDM product with support for governance and compliance features. Architecture of one product is quite heavyweight, based on Service-Oriented Architecture (SOA) principles. And that product actually contains complete Enterprise Service Bus (ESB) embedded inside the system. On the other hand, architecture of another product is a loosely coupled set of components that can be glued together with JavaScript. Couple of other products are mostly based on proven and efficient lightweight Java architecture. While some products are following good open source standards and they are building user communities, other products are open source only by the source code license and they are obviously not so

much community-oriented. So, there is a lot of variability. Which means that are plenty options to choose from.

To be completely honest, not all second-generation IDM products are entirely ready for prime time. It takes some time for a product to mature. But there are few products that can handle all common IDM scenarios without any problems. And at least one product is a comprehensive and feature-complete IDM system. Most second-generation products are in a very active development phase, so we can expect great improvements in near future.

The most significant advantage of the second-generation IDM products is their open source character. This is a characteristic that is easy to overlook, but it is almost impossible to overstate. As every single IDM engineer knows, understanding of the IDM product and the ability to adapt the product are two critical aspects of any IDM project. Open source is the best way to support both understanding and flexibility. And there is also third important advantage: it is almost impossible to create a vendor lock-in situation with an open source product. All of the open source products are backed by companies that offer professional support services that are equivalent to the services offered by commercial IDM products. Therefore there are no disadvantages originating from the open source character of these projects. Yet there are huge advantages that make this approach really revolutionary.

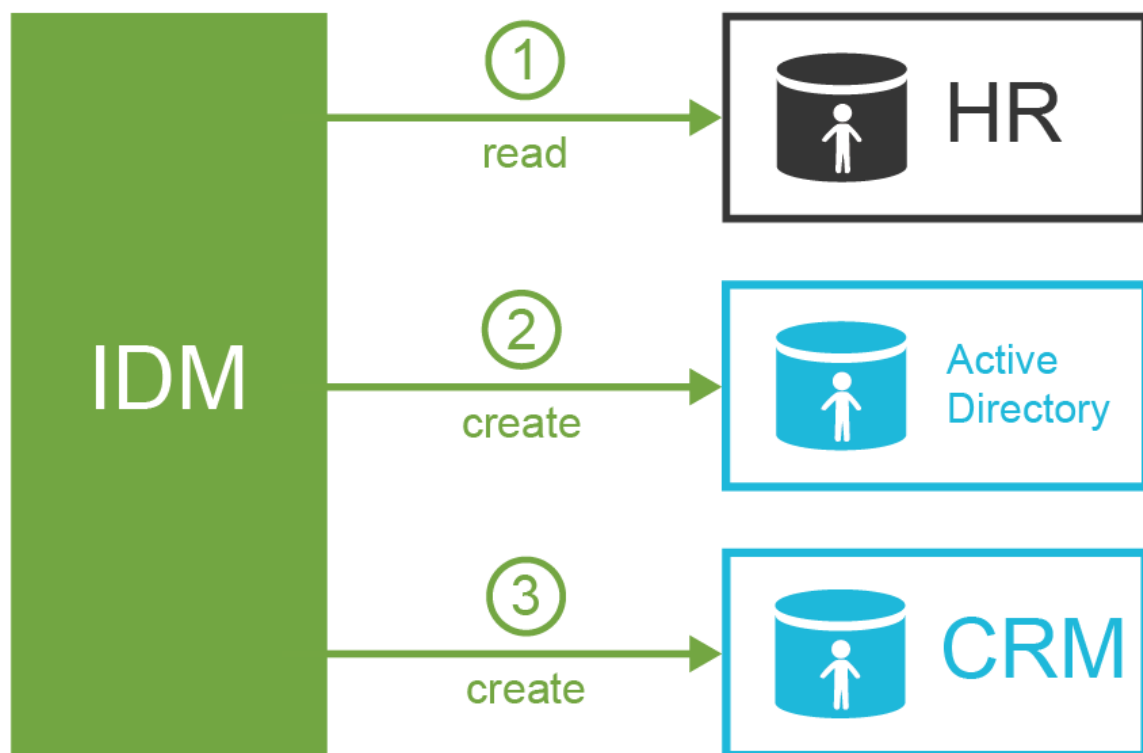
The second-generation IDM products are obvious choice for new IDM deployment projects. The advantages of these products are quite clear and there is a great potential for future development. However, as always, you must choose wisely.

What is This Identity Management, Anyway?

Identity management is a simple term which encompasses a very rich and comprehensive functionality. It contains identity provisioning (and reprovisioning and deprovisioning), synchronization, organizational structure management, role-based access control, data consistency, workflow, auditing and few dozens of other features. All of that is thoroughly blended and mixed with a pinch of scripting and other seasoning until there is a smooth IDM solution. Therefore it is quite difficult to tell what identity management is just by using a dictionary-like definition. We would rather describe what identity management is by using a couple of typical usage scenarios. Let's have a fictional company called ExAmPLE, Inc. This company has few thousand employees, decent partner network, customers and suppliers and all the other things as real-world companies have. And ExAmPLE company has an IDM system running in its IT infrastructure.

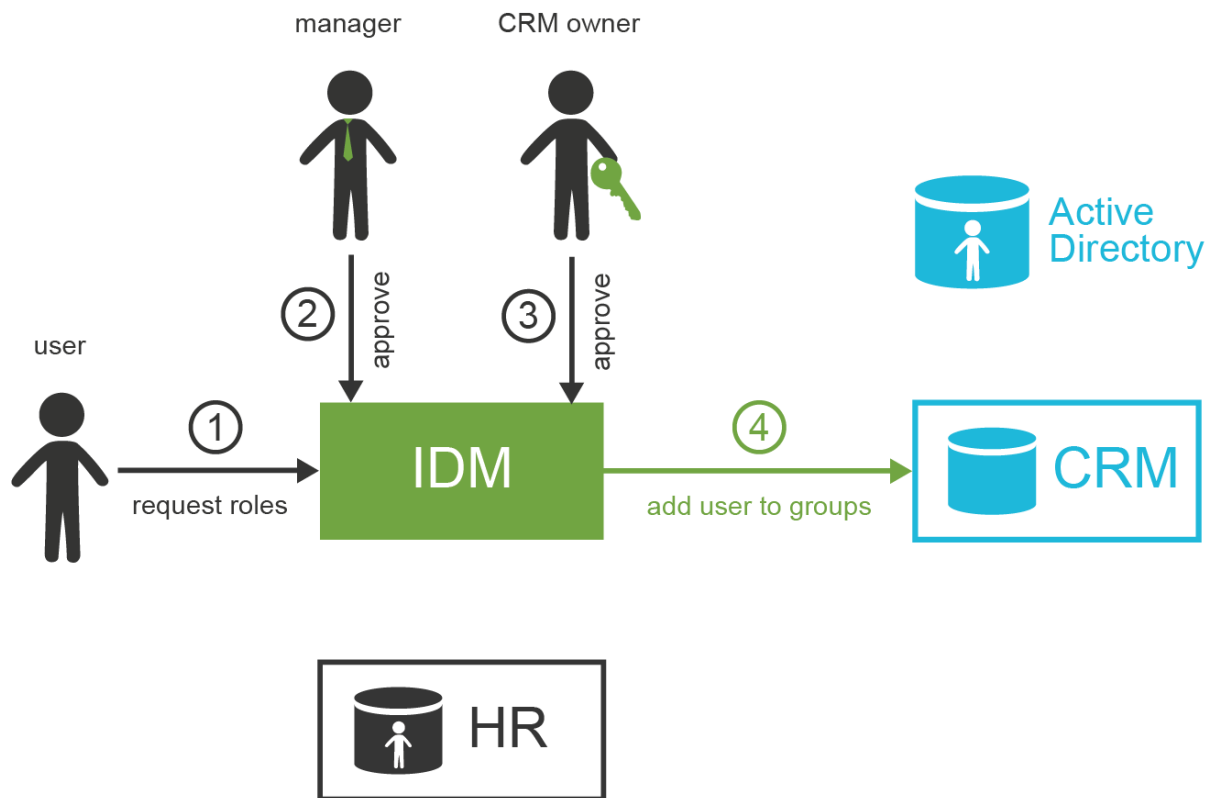
ExAmPLE hires a new employee called Alice. Alice signs an employee contract few days before she starts her employment. The contract is entered into the HR system by the ExAmPLE HR staff. The IDM system periodically scans the HR records and it discovers the record of a new hire. The IDM system pulls in the record and analyzes it. The IDM system will take user's name and employee number from the HR record, it will generate a unique username and based on that information it will create a user record in the IDM system. The IDM system also gets the organization code of 10010 from the HR record.

The IDM will look inside its organizational tree and discovers that the code 10010 belongs to sales department. Therefore IDM will automatically assign the user to the sales department. The IDM will also process the work position code of 007 in the HR record. The IDM policies say that the code 007 means sales agent and that anybody with that code should automatically receive the “Sales Agent” role. So the IDM will assign that role. As this is core employee belongs to the sales department, the IDM will automatically create an Active Directory account for the user together with the company mailbox. The account will be placed into the Sales Department organizational unit. The “Sales Agent” role entitles the user to more privileges. Therefore the Active Directory account is automatically assigned to sales groups and distribution lists. The role also gives access to the CRM system, so the CRM account is also automatically created and assigned to the appropriate groups. All of that happens in a couple of seconds after the new HR record is detected. And it all happens automatically.

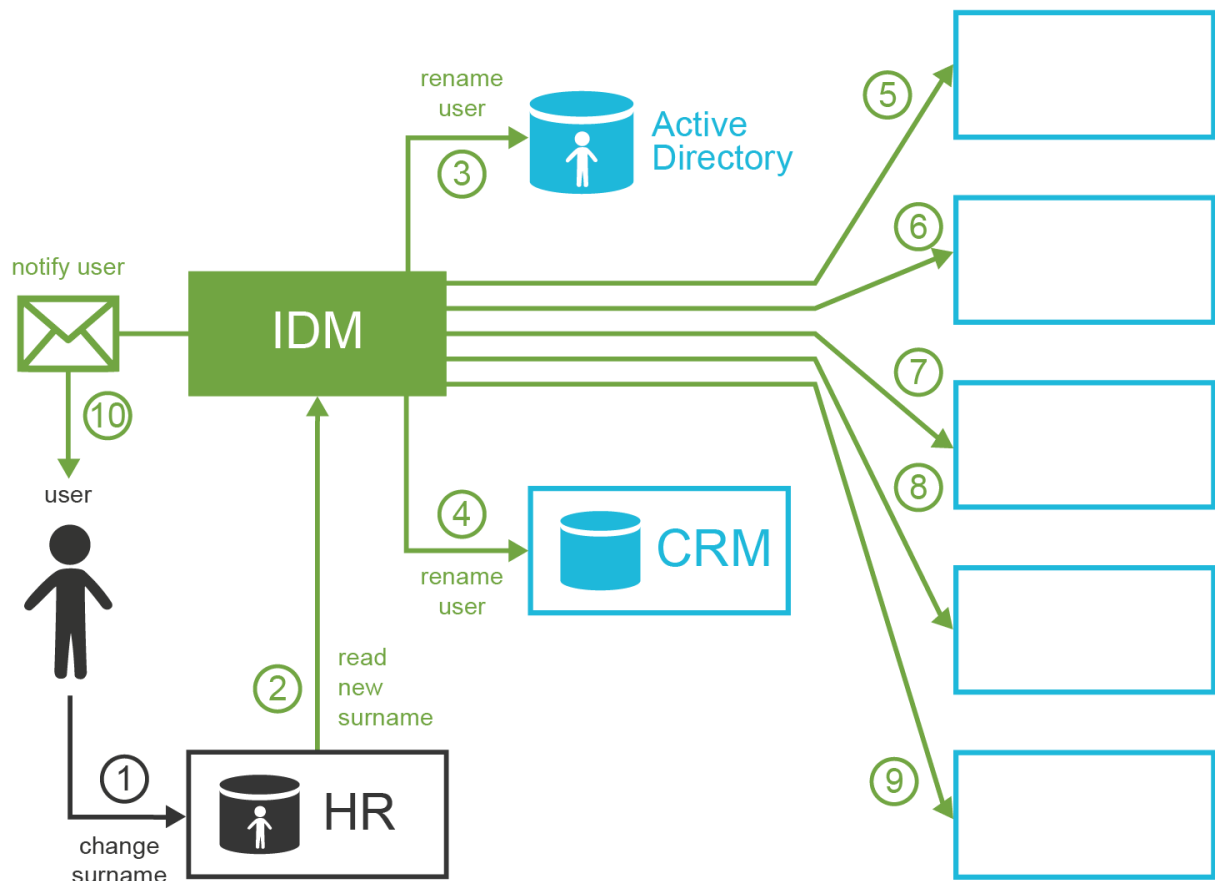


Alice starts her career and she is a really efficient employee. Therefore she gets more responsibilities. Alice is going to prepare specialized market analyses based on empirical data gathered in the field. ExAmPLE is a really flexible company, always inventing new ways how to make business operations more efficient. Therefore they invented this work position especially to take advantage of Alice's skills. Which means there is no work position code for Alice's new position. But she needs new privileges in the CRM system to do her work efficiently. And she needs that right now. Fortunately the ExAmPLE has a

flexible IDM system. Alice can log into the IDM system, select the privileges that she needs and request them. The request has to be approved by Alice's manager and also by the CRM system owner. They get the notification about the request and they can easily approve or reject it in the IDM system. Once the request is approved Alice's CRM account will be automatically assigned to appropriate CRM groups. Alice may start working on her analysis minutes or hours after she has requested the privileges.

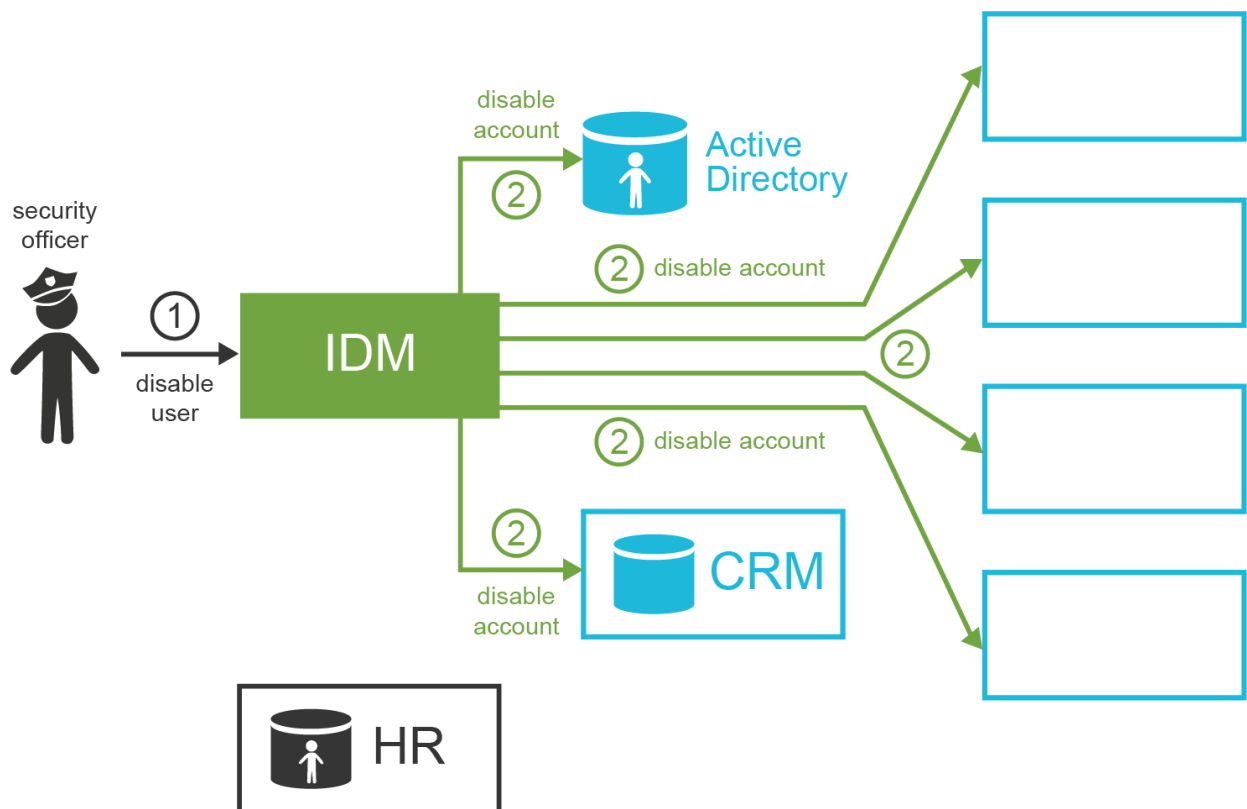


Alice lives happily ever after. And one day she decides to get married. Alice, similarly to many other women, has the strange habit of changing her surname after the marriage. But now Alice has a really responsible work position and she has accounts in a dozen information systems. This is no easy task to change her name in all of them, is it? In fact it is very easy because ExAmPLE has its IDM system. Alice goes to the HR department and the HR staff changes her surname in the HR system. The IDM system will pick up the change and propagate that to all the affected systems. Alice even automatically gets a new e-mail address with her new surname (keeping the old one as an alias). Alice will receive a notification that now she can use her new e-mail address. The change is fast, clean and effortless.



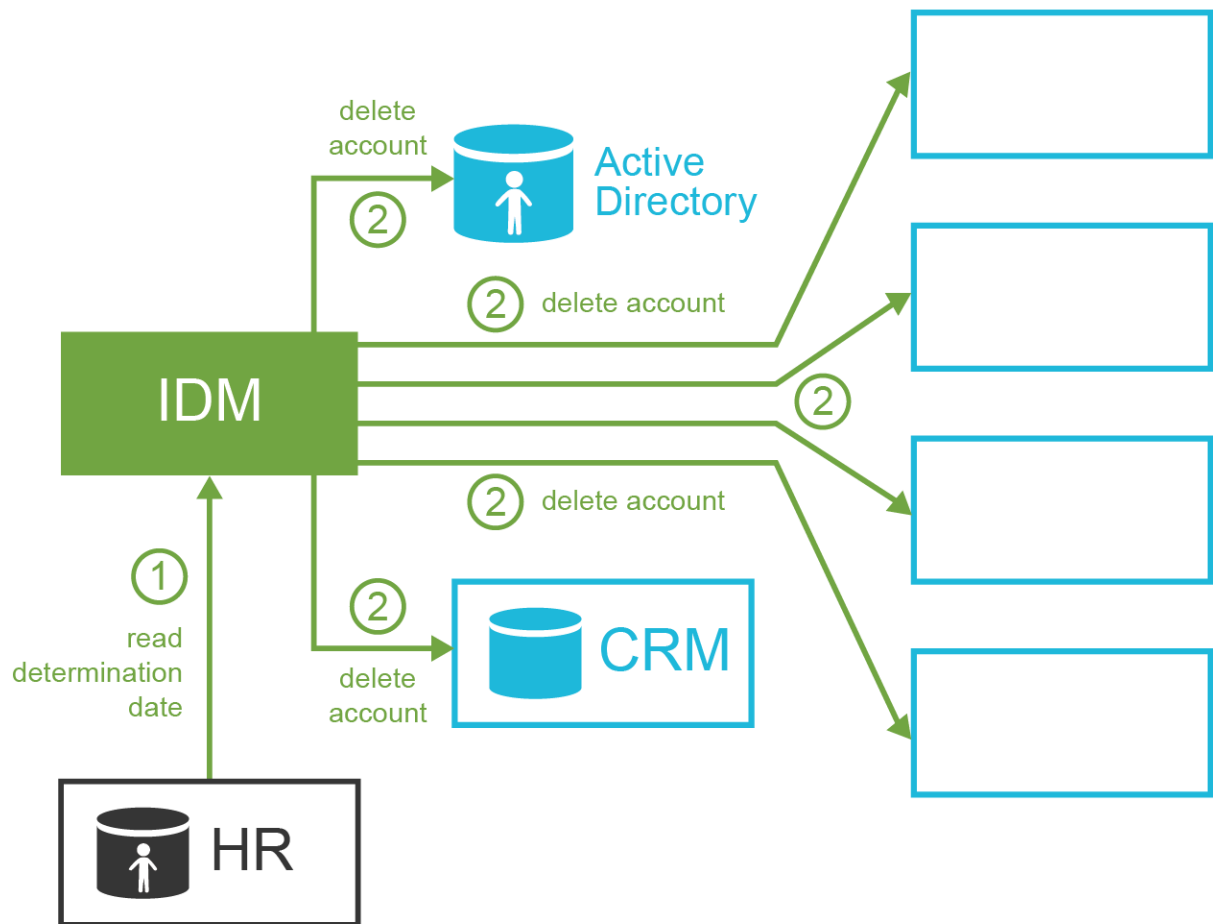
Later that day Alice discovers that her password is about to expire. Changing the password in all the applications would be a huge task. But Alice knows what to do. She logs into the IDM system and changes her password there. The password change is automatically propagated to each affected system according to policy set up by the IT security office.

The following month something unexpected happens. There is a security incident. The security office discovered that very early and now they are investigating it. It looks like it was an insider job. The security officers are using the data from the IDM system to focus their investigation on users that had privileges to access the affected assets. They isolate Mallory as a prime suspect. The interesting thing is that Mallory should not have these privileges at all. Luckily the IDM system also keeps an audit trail about every privilege change. Therefore they discover that it was Mallory's colleague Oscar that assigned these privileges to Mallory. Both men are to be interviewed. But as this incident affects sensitive assets there are some preventive measures to be executed before any word about the incident spreads. The security officers use the IDM system to immediately disable all the accounts that Mallory and Oscar have. It takes just a few seconds for IDM to disable these accounts in all the affected applications.

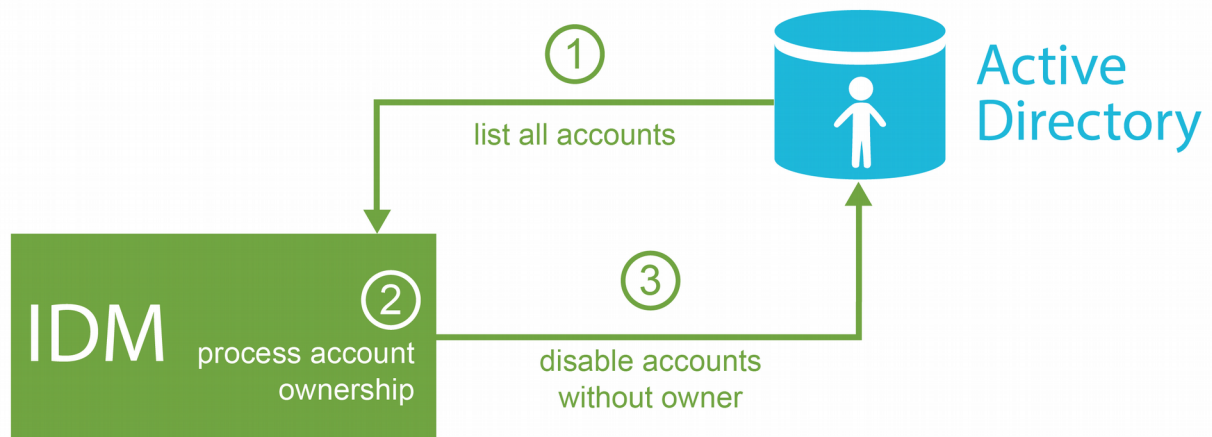


The investigation later reveals that Oscar is mostly innocent. Mallory misused Oscar's trust and tricked him to assign the extra privileges. Mallory abused the privileges to get sensitive data and he tried to sell them. The decision is that Mallory has to immediately leave the company while Oscar may stay. However, as Oscar has shown poor judgment in this case his responsibilities are reduced. The IDM is now used to permanently disable all Mallory's accounts, to re-enable Oscar's accounts and also to revoke sensitive privileges that are considered too risky for Oscar to have.

Few months later Oscar is still ashamed because of his failure. He decides not to prolong his employee contract with ExAmPLE and leave the company without causing more trouble. Oscar's contract expires at the end of the month. This date is recorded in the HR system and the IDM system takes it from there. Therefore at midnight of the last Oscar's day at work the IDM system automatically deletes all Oscar's accounts. Oscar starts a new career as a barman in New York. He is very successful.



The security office has handled the security incident in a professional way and the IDM system provided crucial data to make the security response quick and efficient. They receive praise from the board of directors. But the team always tries to improve. They try to learn from the incident and reduce the possibility of such a thing happening again. The team is using data from the IDM system to analyze the privileges assigned to individual users. The usual job of the IDM system is to create and modify accounts in the applications. But the IDM system is using bidirectional communication with the applications. Therefore this analysis is far from being yet another pointless spreadsheet exercise. The analysis is based on real application data processes and unified by the IDM system: what are the real accounts, to which user they belong, what roles they have, which groups they belong and so on. The IDM system can detect accounts that do not have any clear owner. The security team discovers quite a rich collection of testing accounts that were obviously used during the last data center outage half a year ago. The IT operations staff obviously forgot about these accounts after the outage. The security staff disables the accounts using the IDM tools and sets up an automated process to watch out for such accounts in the future.



Based on the IDM data the security officers suspect that there are users that have too many privileges. This is most likely a consequence of the request-and-approval process and these privileges simply accumulated over time. But this is just a suspicion. It is always difficult for a security staff to assess whether particular user should have certain privilege or should not have it. This is especially difficult in flexible organizations such as ExAmPLE, where work responsibilities are often cumulated and organizational structures is somehow fuzzy. Yet there are people that know what each employee should do: their managers. However, there are many managers on many departments and it would be a huge task to talk to each one of them and consult the privileges. The IDM system comes to the rescue once again. The security officers set up automated access recertification campaign. They sort all users to their managers based on the organizational structure which is maintained in the IDM system. Each manager will receive an interactive list of their users and their privileges. The manager must confirm (re-certify) that the user still needs those privileges. This campaign is executed in a very efficient manner as the work is evenly distributed through the organization. Therefore the campaign is completed in a couple of days. At the end the security officers know which privileges are no longer needed and can be removed. This reduces the exposure of the assets which is a very efficient way to reduce residual security risk.

How Does Identity Management Technology Work?

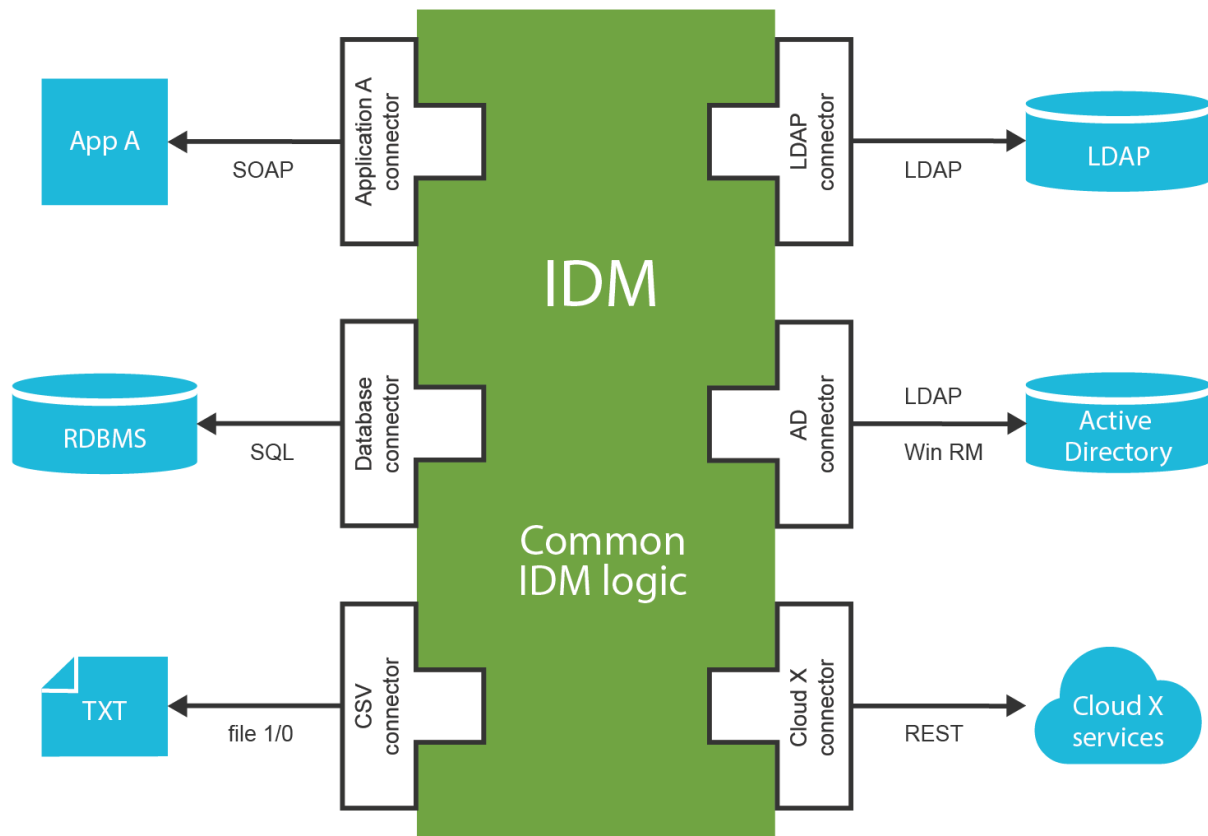
Obviously identity management systems have a lot of advantages for business, processes, efficiency and all that stuff. But how does it really work on a technological level? The basic principle is very simple: identity management system is just a sophisticated data synchronization engine.

Identity management system takes data from the source systems, such as HR databases. It is processing it, mapping values as necessary. It will figure out which records are new. The the IDM engine will do some (usually quite complex) processing on the records. That usually includes processing policies such as Role-Based Access Control (RBAC), organizational policies, password policies and so on. The result of this processing is usually creation or modification of user accounts in other systems such as Active Directory, CRM systems and so on. So basically it is all about getting the data, changing them and moving them around. This does not seem very revolutionary, does it? But it is all about the details. It is the way how the IDM system gathers the data, how it is processing the data and how it is propagating the changes that make all the difference.

Identity Management Connectors

Identity management system must connect to many different applications, databases and information systems. Typical IDM deployment has tens or even hundreds of such connections. Therefore the ease of connecting IDM system with its environment is one of its essential qualities.

Current IDM systems use *connectors* to communicate with all surrounding systems. These connectors are based on similar principles that database drivers. On one end there is unified connector interface that presents that data from all the systems using the same “protocol”. On the end of the connector is the native protocol that the application supports. Therefore there are connectors for LDAP and various LDAP variants, SQL protocols and dialects, connectors that are file-based, connectors that invoke web services or REST services and so on. Every slightly advanced IDM system has tens of different connects.



Connector is usually relatively simple piece of code. Typical responsibility of a connector is to adapt communication protocols. Therefore LDAP connector translates the LDAP protocol messages into data represented using a common connector interface. The SQL connector does the same thing with SQL-based protocols. The connector also interprets the operations invoked on the common connector interface by the IDM system. Therefore the LDAP protocol will execute the 'create' operation by sending LDAP 'add' message to the LDAP server and parsing the reply. Connectors typically implement the basic set of create-read-update-delete (CRUD) operations. Therefore a typical connector is usually not very complicated. Despite its simplicity the whole connector idea is a clever one. The IDM system does not need to deal with the communication details. The core of the IDM is free to focus on the generic identity management logic which is typically quite complex just by itself. Therefore any simplification that the connectors provide is more than welcome.

Connectors are usually accessing external interfaces of source and target systems. It is natural that the connector authors will choose interfaces that are public, well-documented and ideally based on open standards. And many systems indeed have interfaces like that. But there are notorious cases that refuse to provide such interface. But usually there is always some way. The connector may create record directly in the application database. Or it may execute a database routine. Or it may execute a command-line tool for account management. Or it may even do crazy things such as simulation of a user working with

text terminal and filling out a form to create new account. There is almost always a way to do what connector needs to do. Just some ways are nicer than others.

The connector-based architecture is pretty much standard among all advanced IDM systems. Yet the connector interfaces significantly vary therefore the connectors are not interchangeable between different IDM systems. The connector interfaces are all proprietary. And the connectors are often used as weapons to somehow artificially increase the profit from IDM solution deployment. Except for one case. The ConnId connector framework is the only connector interface that is actively used and developed by several competing IDM systems. Of course, it is an open source framework.

Even though connector-based approach is quite widespread, some older IDM systems are not using connectors. Some IDM products use agents instead of connectors. Agent does similar job than the connector does. However, agent is not part of the IDM system instance. Agents are installed in each connected application and they communicate with the IDM system using a remote network protocol. This is a major burden. The agents needs to be installed everywhere. And then they need to be maintained, upgraded, there may be subtle incompatibilities and so on. Also running a third-party code inside every application can be a major security issue. Overall the agent-based systems are too cumbersome (and too costly) to operate. The whole agent idea perhaps originated somewhere in our digital past when applications and databases haven't supported any native remote interfaces. In such a situation the agents are obviously better than connectors. Fortunately, this is a thing of the past. Today even old applications have some way to manage identities using a remote interface. This is typically some web or REST service that are easy to access from a connector. But even if the application provides only a command-line interface or interactive terminal session there are connectors that can handle that sufficiently well. Therefore today the agent-based systems are generally considered to be obsolete.

Identity Provisioning

Perhaps the most frequently used feature of the IDM system is provisioning. In the generic sense provisioning means maintenance of user account in application, databases and other target systems. This includes creation of the account, various modifications during the account lifetime and permanent disable or delete at the end of the lifetime. The IDM system is using the connectors to manipulate the accounts. And in fact good IDM systems can manage much more than just accounts. Ability to easily manage groups and group membership was quite a rare feature only a couple of years ago. Yet today an IDM system that cannot manage groups is almost useless. Almost all IDM systems work with roles. But only few IDM systems can also provision and synchronize the roles (e.g. automatically create LDAP group for each new role). Good IDM system can also manage, provision and synchronize organizational structures. However, this feature still is still not entirely common.

Synchronization and Reconciliation

Identity provisioning is often seen as the most important feature of an IDM system. And provisioning is without any doubt an essential feature of any IDM system. But if an IDM system did just the provisioning and nothing else it would be a quick and utter failure. It is not enough to create an account when a new employee is hired or delete that account when an employee leaves. Reality has many ways how to make a big mess in a very short time. Maybe there was a crash in one of the applications and the data were restored from a backup. So an account that was deleted few hours ago is unexpectedly resurrected. It stays there, alive, unchecked and dangerous. Maybe an administrator manually created an account for a new assistant because the HR people were all busy to process the papers. And the new assistant had such pretty eyes. When the record finally gets to the HR system and it is processed the IDM system discovers that there is already a conflicting account and it simply stops. Maybe few (hundred) accounts get accidentally deleted by junior system administrator trying out a novel system administration routine. There are simply too many ways how things can go wrong. And in reality they do go wrong surprisingly often. It is not enough for an IDM system to just set things up and then forget about it. One of the most important features of any self-respecting IDM system is to make sure that everything is right and also that it stays right all the time. Identity management is all about continuous maintenance of the identities. Without that the whole IDM system is almost useless.

The trick to keep the data in order is to know when they get out of order. In other words the IDM system must detect when the data in the application databases change. If an IDM system detects that there was a change then it is not that difficult to react to the change. So, the key is change detection. But now there's a slight issue with that. We cannot expect that the application will send a notification to the IDM system every time a change happens. We do not want to modify the applications, otherwise the IDM deployment will be prohibitively expensive. The application needs to be passive and the IDM system needs to be active. Fortunately, there are several ways how to do that.

Some applications already keep a track of the changes. Some databases record a timestamp of the last change for each row. Some directory servers keep a record of recent changes for the purpose of replication. Such meta-data can be used by the IDM system. The IDM system usually periodically scans the timestamps or replication logs for new changes. When the IDM detects a change it can retrieve the changed objects and react to the change based on its policies. The scanning for changes based on meta-data is usually very efficient, so it can be executed every couple of minutes or even seconds. Therefore the reaction to the change can be done almost in the real-time. This method has many names in various IDM systems. It is called "live synchronization", "active synchronization" or simply just "synchronization". This is usually the preferred method. Sadly, it is not always available. In fact this ability is quite rare.

Even if the application does not maintain good meta-data that allow near-real-time change detection there is still one way that works for almost any system. The principle is very

simple. The IDM system gets the list of all accounts in the application. Then it compares that list with the list of accounts that are supposed to be there. Therefore it compares the reality (what *is* there) with the policy (what *should be* there). The IDM system can react to any inconsistency and repair it. This method is called reconciliation. It does the job, but it is quite brutal. Listing all accounts and processing each of them may seem as a straightforward job. But it can be extremely slow if the number of accounts is high and the policies are complex. It can take anything from few minutes to few days. Therefore it cannot be executed frequently. Running that once per day is feasible only for small and simple systems. Running it once per week (on weekends) is a more common practice. But many systems cannot afford to run it more frequently than once per month.

There are also other methods. But synchronization and reconciliations are the most frequently used. The drawback of the synchronization is that it is not entirely reliable. The IDM system may miss some changes, e.g. due to change log expiration, system times not being synchronized or variety of other reasons. On the other hand reconciliation is mostly reliable, but it is a very demanding task. Therefore these two are often used together. Synchronization runs all the time and handles vast majority of the changes. Reconciliation runs weekly or monthly and it acts as a safety net to catch the changes that might have escaped during synchronization.

Good IDM systems handle the changes consistently regardless of the mechanism that was used to detect them. They simply enforce the same policy. However, some (especially older) IDM systems have separate policies for each mechanism. This significantly complicates the configuration and maintenance of an IDM system.

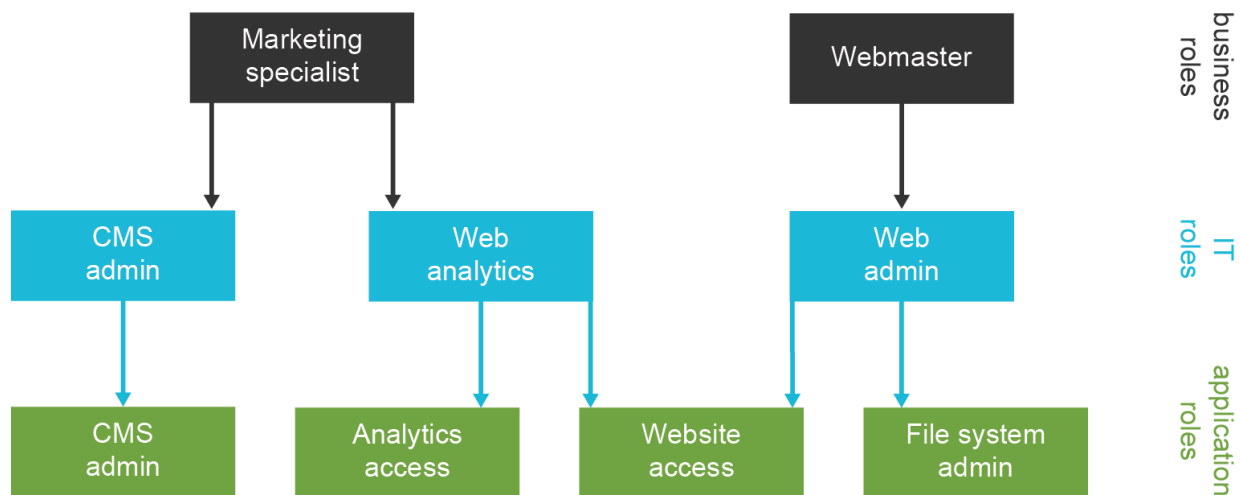
The synchronization and reconciliation are really powerful mechanisms. They can be used to scan for illegal accounts and automatically disable them. They can be used to detect changes done by application administrators and update the user profile accordingly. One way or another this is a crucial mechanism to keep IDM data consistent with the reality.

Identity Management and Role-Based Access Control

Managing permissions for every user individually is a feasible options only if the number of users is very low. Even with just a few hundreds of users the individual management of permissions becomes very difficult. When the number of users goes over a thousand it usually becomes an unbearable burden. The individual management of permissions is not only a lot of work, it is also quite error-prone. This has been known for decades. Therefore many systems unified common combinations of permissions and created roles. And the Role-Based Access Control (RBAC) concept was born. The roles usually represent work positions or responsibilities that are much closer to the “business” than permissions. A role may reflect the concepts of bank teller, website administrator or sales manager. The roles are assigned to users instead of permissions. The roles themselves contain permissions that are used for authorizations. However, these low-level permissions are hidden from the users. Users are quite happy when they deal with the business-friendly role names.

Note: Terminology. The term *RBAC* is frequently used in the industry, however the actual meaning is not always clear. The confusion is perhaps caused by the fact that there is a formal RBAC specification known as *NIST RBAC model*. When people say RBAC some of them mean that specific formal model, others means anything that is alike to that formal model and yet other mean anything that deals with roles. We use the term RBAC in quite a broad sense. Major identity management systems usually implement something that is inspired by the formal NIST RBAC model, but deviating from that model as necessary. That is what we mean when we use the term RBAC.

Most RBAC systems allow for roles to be placed inside other roles thus creating role hierarchy. On the top of the hierarchy there are business roles such as “marketing specialist”. This role contains a lower-level roles. These are often application roles such as “web site analytics” or “CMS administrator”. These lower-level roles may contain concrete permissions or other roles that are even closer to the underlying technology. Role hierarchy is often a must when the number of permissions and users gets higher.



No IDM system can be really complete without RBAC mechanism in place. Therefore vast majority of IDM systems support roles in one way or another. However, the quality of RBAC support significantly varies. Some IDM systems only support the bare minimum that is required to claim RBAC support. Other systems have excellent and very advanced dynamic and parametric hybrid RBAC systems. But most IDM systems are somewhere in between.

Role-based mechanism is very useful management tool. In fact the efficiency of role-based mechanisms often lead to their overuse. This is a real danger especially in bigger and somehow complex environments. The people that design roles in a complex environment have a strong motivation to maintain order by dividing the roles to smallest reusable pieces and then re-composing them again in a form of application and business roles. This is further amplified by the security best practices such as the principle of least privilege. This is understandable and perfectly valid motivation. However, it requires extreme care for

such RBAC system to remain maintainable. Even though this may seem counter-intuitive, it is quite common that the number of roles exceeds the number of users. Unfortunately, this approach turns the complex problem of user management to even more complex problem of role management. This phenomenon is known as *role explosion*.

Role explosion is a real danger and it is definitely not easy to avoid. The approach that prevailed in the first-generation IDM deployments was to simply live with it. Some IDM deployments even created tools that were able to automatically generate and (more-or-less successfully) manage hundreds of thousands of roles. However, this is not a sustainable approach. The second-generation IDM systems bring features that may help to avoid the role explosion in the first place. Such mechanisms are usually based on the idea to make the roles more dynamic. The roles are no longer just a set of privileges. They also contain small pieces of algorithmic logic used to construct the privileges. Input to these algorithms are parameters that are specified when the role is assigned. Therefore the same role can be reused for many related purposes without the need to duplicate the roles. This significantly limits the number of roles required to model a complex system which is the best weapon against role explosion.

Even though the RBAC system has some drawbacks it is required for almost any practical IDM solutions. There were several attempts to replace the RBAC system with a completely different approach. Such attempts have some success in the access management and related field. But they cannot easily replace RBAC in the identity management. Attribute-Based Access Control (ABAC) is one popular example. The ABAC idea is based on replacing the roles with completely algorithmic policies. Simply speaking ABAC policy is a set of algorithms that take user attributes as input, combine that with the data about operation and context and produce a decision whether an operation should be allowed or denied as their output. This approach is simple and it may work reasonably well in the access management world where the AM server knows a lot of details about the operation that just takes place. But in the IDM field we need to set up the account before the user logs in for the first time. There are no data about the operation yet. And even contextual data are very limited. That together with other issues make ABAC a very poor choice for an IDM system. Therefore whether you like it or not RBAC is the primary mechanism of any practical IDM solution. And it is here to stay.

Identity Management and Authorizations

The basic principle of authorization in the information security is quite straightforward: take the subject (user), object (the things that user is trying to access) and the operation. Evaluate whether the policy allows that subject-object-operation triple. If policy does not allow it then deny it. This is quite simple. But in the identity management field we need to think quite differently. We need to work backwards. The IDM system needs to setup an account for a user before the user initiates any operation. And when user really starts an operation then the IDM system will not know anything about it. Therefore the concept of authorization in the IDM world is somehow turned upside down.

The IDM system does not take direct part in authorization. IDM system sets up the account in the applications and databases. But the IDM system itself is not active when user logs into an application and executes the operations. Does that mean IDM system cannot do anything about authorizations? Definitely not. Yes, the IDM system does not enforce authorization decisions. But the IDM can manage the data that determine how the authorization will be evaluated. IDM system can place the account to the correct groups which will cause certain operations to be allowed and other denied. IDM system can set up an access control lists (ACLs) for each account that it manages. IDM system is not evaluating or enforcing the authorizations directly. But it indirectly manages the data that are used to evaluate authorizations. And this is extremely important feature.

Authentication and authorizations are two very prominent concepts of information security. And they are vitally important for any identity and access management solution. However, authentication is quite simple. Yes, the user may have several credential types used in adaptive multi-factor authentication. But while that description sounds a bit scary it is still not that complex. There are just a couple of policy statements that govern authentication. Also, authorization is typically quite uniform: most users are authenticating using the same mechanism. Authentication is not that difficult to centralize (although it may be expensive). Authentication is therefore relatively easy to manage.

But it is quite a different story for authorization. Every application has slightly different authorization mechanisms. And these mechanisms are not easy to unify. One of the major obstacle is that every application works with different objects that may have complex relations between them and that may also have complex relation with the subjects. The operations are also far from being straightforward. And then there is the context. There may be per-operation limits, daily limits, operations allowed only during some times or when system is in certain state and so on. This is very difficult to centralize. Also, almost every user has slightly different combination of authorizations. Which means that there is a great variability and a lot of policies to manage. And then there are two crucial aspects that add whole new dimension of complexity: performance and scalability. Authorization decisions are evaluated all the time. It is no exception if authorization is evaluated several times for each request. Processing of authorization needs to be fast. Really fast. Even a round-trip across a local network may be a performance killer. Due to performance and scalability reasons the authorization mechanisms are often tightly integrated into the fabric of each individual application. E.g. it is a common practice that authorization policies are translated to SQL and they are used as an additional clauses in application-level SQL queries. This technique is taking advantage of the database to quickly filter out the data that the user is not authorized to access. It is very efficient and it is perhaps the only practical option when dealing with large-scale data sets. However this approach is tightly bound to the application data model and it is usually almost impossible to externalize.

Therefore it is not realistic to expect that the authorization could be centralized anytime soon. The authorization policies need to be distributed into the applications. But managing partial and distributed policies is not an easy task. Someone has to make sure that the

application policies are consistent with the overall security policy of the organization. Fortunately, the IDM systems are designed especially to handle management and synchronization of data in wide range of systems. Therefore the IDM system is the obvious choice when it comes to management of authorization policies.

Organizational Structure, Roles, Services and Other Wildlife

Back in 2000s the IDM was all about managing user accounts. It was enough to create, disable and delete an account. But the world is a different place now. Managing just the accounts is simply not enough. Yes, automated account management brings significant benefits and it is a necessary condition to get at least a minimal level of security in complex systems. But even that is often not enough to justify the cost of an IDM system. And the IDM system really can do much more than just simple account management. And all the new IDM systems do that. Those old IDM systems that can only manage user accounts are very likely to completely disappear in the near future.

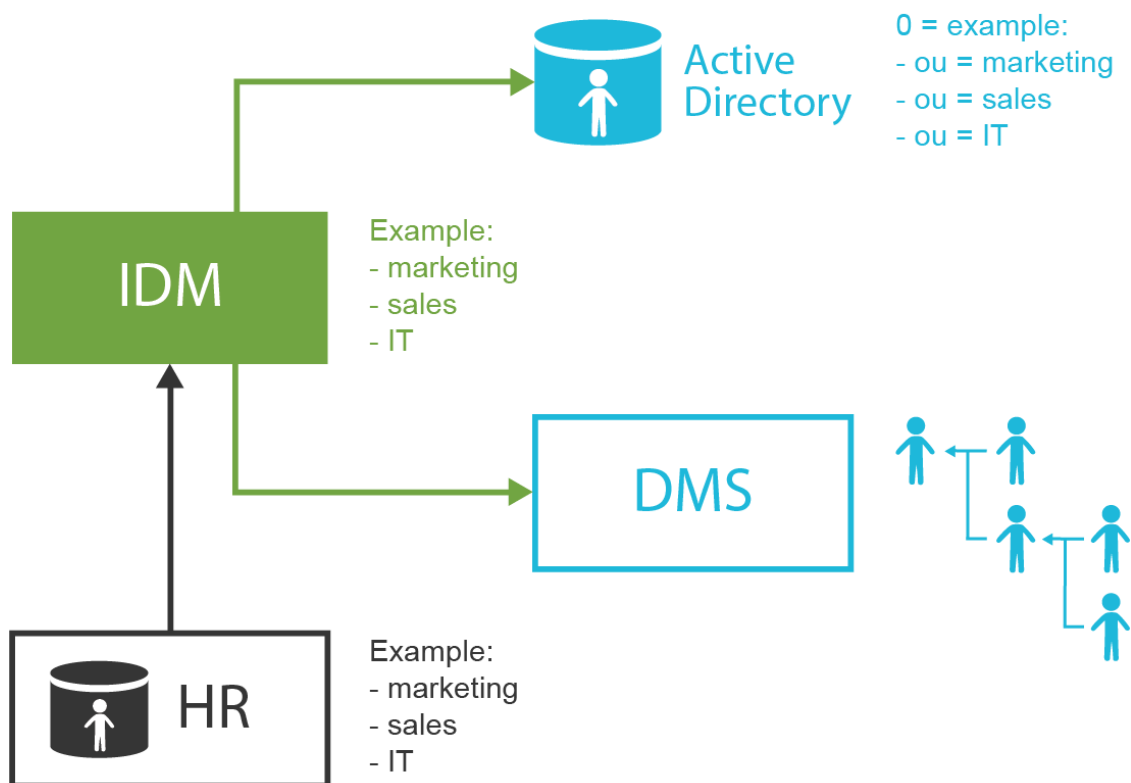
There are many things that an advanced IDM system can manage:

- Accounts. Obviously. But the age of simple account management is finally over. Now it is a standard that an IDM system can fully manage account attributes, groups membership, privileges, account status (enabled/disabled), validity dates and all the other details.
- Groups and roles. Apart from managing the membership of accounts in groups the IDM system can take care of the whole group life-cycle: create a group, manage it and delete it.
- Organizational structure. The IDM system can take organizational structure from its authoritative source (usually HR) and synchronize it to all the applications that need it. Or the IDM itself may be used to manually maintain an organizational structure.
- Servers, services and “things”. While this is not yet IDM mainstream, there are some experimental solutions that use IDM principles to manage concepts that are slightly outside the traditional IDM scope. E.g. there is an IDM-based solution that can automatically deploy predefined set of virtual machines for each new project. The new IDM systems are so flexible that they can theoretically manage everything that is at least marginally related to the concept of identity: virtual machines, networks, applications, configurations, devices ... almost anything. This is still quite a unique functionality. But it is very likely that we will see more stories about this in the future.

While all these features are interesting some of them clearly stand out. The management of groups and organizational structure are those that are absolutely critical for almost any new IDM deployment. Your organizational structure may be almost flat and project-oriented or you may have twelve levels of divisions and sections. But regardless of the size your organizational structure needs to be managed and synchronized across applications in pretty much the same way as identities are synchronized. You may need to

create groups in Active Directory for each of your organizational unit. And you want them to be correctly nested. You may want to create distribution list for each of your ad-hoc team. And you want this operation to have as little overhead as possible otherwise the teams will not be ad-hoc any more. You may want to synchronize the information about projects into your issue tracking system. You may also want to automatically create a separate wiki space and a new source code repository for each new development project. The possibilities are almost endless. Both the traditional organizations and the new lean and agile companies will benefit from that.

Organizational structure management is closely related to group management. The groups are often bound to workgroups, projects or organizational units. E.g. and IDM system can automatically maintain several groups for each project (admin and member groups). Those groups can be used for authorization. Similarly an IDM system can automatically maintain application-level roles, access control lists (ACLs) and other data structures that are usually used for authorization.



While this functionality provides benefits in almost any deployment, it is absolutely crucial for organizations that are based on tree-like functional organizational structures. These organizations heavily rely on the information derived from an organizational structure: Direct manager of the document author can review and approve the document in the document management system. Only the employees in the same division can see the

document draft. Only the employees of a marketing section can see marketing plans. And so on. Traditionally this data is encoded into an incomprehensible set of authorization groups and lists. And that contributes to the fact that reorganizations are a total nightmare. However, an IDM system can significantly improve the situation. IDM can create the groups automatically. It can make sure that the right users are assigned into these groups. It can synchronize information about the managers into all affected applications. And so on. And a good IDM system can do all of that using just a handful of configuration objects.

This all seems too good to be true. And it is fair to admit that the quality of support for these features significantly varies between IDM systems. Group management and organizational structure management seem to be the most problematic feature. Only few IDM systems support these concepts at the level that allows practical out-of-box deployment. Most IDM systems have some support for that, but any practical solution requires heavy customization. It is not clear why IDM vendors do not pay attention to features that are required for almost any IDM deployment. Therefore when it comes to a comprehensive IDM solution there is one crucial advice that we could give: choose the IDM product wisely.

Everybody Needs Identity Management

This may look like a huge exaggeration but in fact it is very close to the truth. Every non-trivial system has a need for identity management, even though the system owners may not realize that. But as you are reading this book chances are that you are one of the few that can see the need. In that case it is mostly about the costs versus the benefits. Identity management has some inherent complexity. Even very small systems need IDM. Yet for these systems the benefits are likely to be too small to justify the costs. The cost/benefit ratio is much better for mid-size. And IDM is an absolute necessity for large-scale systems. There seems to be a rule of thumb that has quite broad applicability:

Number of users	
Less than 200	You may need IDM, but the benefits are probably too small to justify the costs.
200 – 2 000	You need IDM and the benefits may be just enough to justify the costs. But you still need to look for a cost-efficient solution.
2 000 – 20 000	You really need IDM. You simply cannot manage that crowd manually. If you implement IDM properly the benefits will be much higher than the costs.
More than 20 000	I can't believe that you do not have any IDM yet. Go and get one. Right now. You can thank me later.

Identity Governance and Compliance

Identity governance is basically an identity management taken to a higher business level. The identity management proper is focused mainly on technical aspects of identity life-cycle such as automatic provisioning, synchronization, evaluating the roles and computing attributes. The identity governance abstracts from the technical details and it is focused on

policies, roles, processes and data analysis. E.g. a governance system may deal with segregation of duties policy. It may drive the process of access recertification. It may focus on automatic analysis and reporting of the identity, auditing and policy data. It will drive remediation processes to address policy violations. It will manage application of new and changed policies, evaluate how is your system compliant with policies and regulations and so on. This field is sometimes referred to as *governance, risk management and compliance* (GRC).

Almost all IDM system will need at least some governance features to be of any use in practical deployments. And many governance features are just refinement of concepts that originated in the IDM field few years ago. Therefore the boundary between identity management and identity governance is considerably blurred. It is so fuzzy that new terms were invented for the unified field that includes the identity management proper together with identity governance. *Identity governance and administration* (IGA) is one of these terms. This field (or sub-field) is still very young, therefore it is expected that the terminology and even the concepts need some time to settle down. For us the governance is just a continuation of the identity management evolution.

However, it seems to be a common practice that the governance features are implemented by specialized products that are separate from their underlying IDM platforms. Almost all commercial IDM and governance solutions are divided into (at least) two products. This strategy obviously brings new revenue streams for the vendors. But from customer point of view it makes almost no sense at all. The industry has even coined a term *closed-loop remediation* (CLR) which in fact means that the governance system is somehow integrated with the underlying IDM solution. Industry sometimes has a need for inventing fancy terms for something that should be natural part of any reasonable solution. It comes without saying that reasonable solutions offer both the IDM and governance features in one unified and well aligned product.

Complete Identity and Access Management Solution

A comprehensive Identity and Access Management solution cannot be built by using just a single component. There is no single product or solution that will provide all the necessary features. And as the requirements are so complex and often even contradictory it is very unlikely that there ever will be any single product that can do it all.

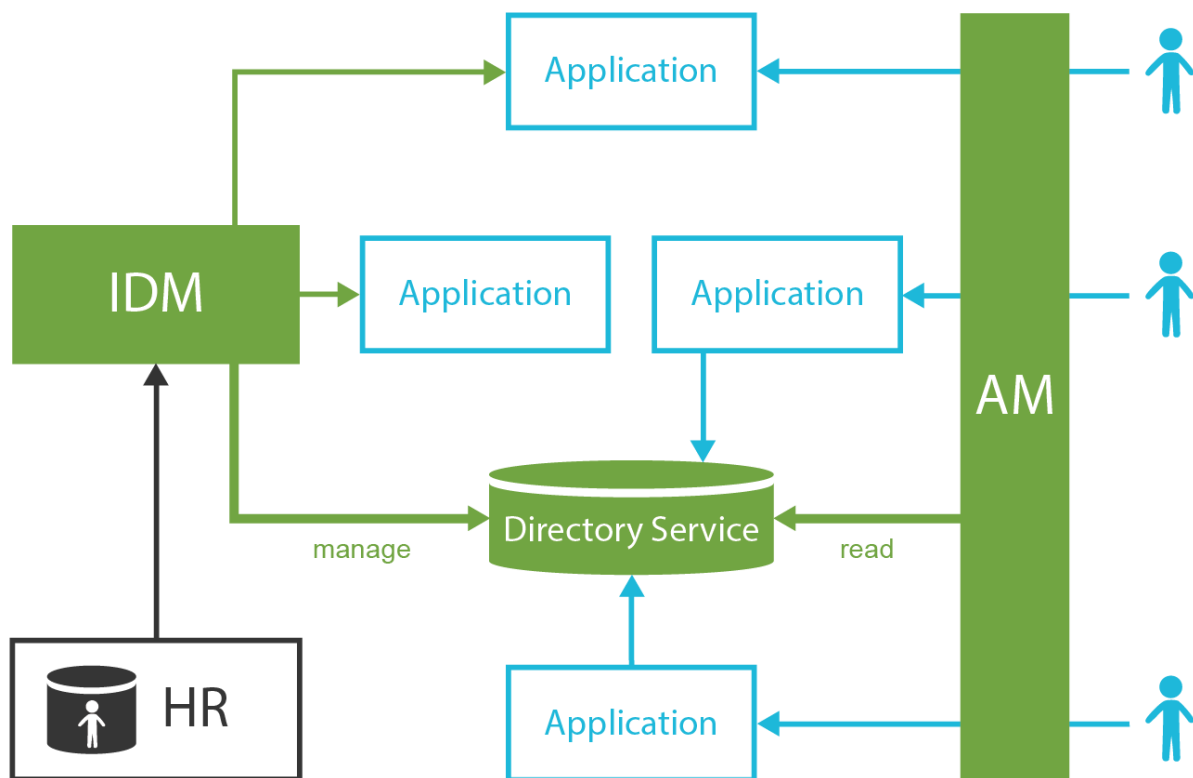
A clever combination of several components is needed to build complete solution. The combination will always be slightly different as no two IAM solutions are the same. But there are three basic components that are required for any practical IAM deployment:

- **Directory service** or similar identity store is the first component. This is the database that stores user account information. The accounts are stored there in a “clean” form that can be used by other applications. And indeed this database is widely shared by applications that are capable to connect to it. It is usually implemented as a replicated LDAP server topology or Active Directory domain. This has an advantage of relatively cheap high availability. And especially the LDAP

sever topologies can usually scale ad nauseam. But there is one major limitation: the data model needs to be simple. Very simple. And the identity store needs to be properly managed.

- **Access Management** is a second major component of the solution. It that take care of authentication and (partially) authorization. Access management unifies authentication mechanisms. If an authentication mechanism is implemented in the access management server then all integrated applications can easily benefit. It also provides Single Sign-On (SSO), centralizes access logs and so on. It is very useful component. But of course, there are limitations. It needs access to identity data. Therefore it needs reliable, very scalable and absolutely consistent identity database as a back-end. This is usually provided by the directory service. Performance and availability are the obvious obstacles here. But there is one more obstacle which is less obvious but every bit as important: data quality. The data in the directory service must be up to date and properly managed. But that is only part of the picture. As most applications locally store some pieces of identity data these data also need to be synchronized with the directory database. No access management system can do this well enough. And there is no point for AM to do it at all. The AM has a very different architectural responsibilities. Therefore yet another component is needed.
- **Identity Management** is the last but in many aspects the most important component. This is the real brain of the whole solution. The IDM system maintains the data. It is the component that keeps the entire system from falling apart. It makes sure that the data are up to date and compliant with the policies. It synchronizes all the pieces of identity data that those pesky little applications always keep creating. It maintains groups, privileges, roles, organizational structures and all the other things necessary for the directory and the access management to work properly. It maintains order in the system. And it allows living and breathing system administrators and security officers to live happily, to breath easily and to keep control over the whole solution.

The following diagram shows how all these components fit together.



This is a true composite solution. There are several components that have vastly different features and characteristics. But when bound together into one solution it will create something that is much more than just a sum of its part. The components support each other. The solution cannot be complete unless all three components are in place.

However, building a complete solution may be quite expensive and it may take a long time. You have to start somewhere. But if you have resources for just one product then choose identity management. IDM is a good start. It is not that expensive as access management. And IDM brings good value even quite early in the IAM program. Especially the second generation IDM systems are very good at repaying the investment. Going for open source product will also keep the initial investment down. Starting with IDM is usually the best choice to start the IAM program.

IAM and Security

Strictly speaking Identity and Access Management (IAM) does not entirely fit into the information security field. The IAM goes far beyond information security. IAM can bring user comfort, reduce operation costs, speed up processes and generally improve the efficiency of the organization. This is not what information security is concerned with. But even though IAM is not strictly part of information security there is a huge overlap. IAM deals with authentication, authorization, logging, role management and governance of objects that are directly related to the information security. Therefore IAM and information security have an intense and very complicated relationship.

It is perhaps not too bold to say that the IAM is a requisite to good information security. Especially the Identity Management (IDM) part is absolutely critical even though this may not be that obvious at the first sight. But it is quite clear. Security studies quite consistency rate the insider threat as one of the most severe threats for an organization. However, there is not much that the technical countermeasures can do about the insider threat. The employee, contractor, partner, serviceman - they all are getting the access rights to your systems easily and legally. They will legally pass through even the strongest encryption and authentication because they have got the keys. Firewalls and VPNs will not stop them because they are meant to pass through them.

Obviously, vulnerability is there. And with the population of thousands of users there is a good chance that there is also an attacker. Maybe one particular engineer was fired yesterday. But he still has VPN access and administration rights to the servers. And as he might not be entirely satisfied by the way how he has left the company the chances are he is quite inclined to make your life a bit harder. Maybe leaking some of the company records would do the trick. Now we have a motivated attacker who will not be stopped by any countermeasures and who can easily access the assets. Any security officer can easily predict the result even without the need for a comprehensive risk analysis.

Information security has no clear answers to the insider threat. And this is no easy issues to solve as there is obviously a major security trade-off. The business wants users to access the assets easily to do their job and keep the organization going. But security needs to protect the assets from the very same users. And there is no silver-bullet to solve this issue. However there is a couple of things that can be done:

- **Record who has access to what.** Each user has several accounts in many applications through the enterprise. Keep track which account belongs to which user. It is very difficult to do that manually. But even the worst IDM system can easily do it.
- **Remove access quickly.** If there is a security incident then the access needs to be removed in order of seconds. If an employee is fired then the accounts have to be disabled in order of minutes. It is not a problem for a system administrator to do that manually. But will be the administrator available during a security incident late in the night? Would you synchronize layoffs with the work time of system administrators? Wouldn't system administrators forget to stop all the processes and background jobs that the user might left behind? IDM system can do that easily. Security staff simply disables all the accounts by using the IDM system. Single click is all that is needed.
- **Enforce policies.** Keep track about the privileges that were assigned to users. This usually means managing assignment of roles (and other entitlements) to users. Make sure that the assignment of sensitive roles is approved before user gets the privileges. Compare the policies and the reality. System administrators that create accounts and assign entitlements are not robots. Mistakes can happen. Make sure

they are discovered and remediated. This is the best practice. But it is almost impossible to do manually. Yet even an average IDM system can do that without any problems.

- **Remove unnecessary roles.** Roles and entitlements tend to accumulate over time. Long-time employees often have access to almost any asset simply because they needed the data at some point in time. And the access to the asset was never removed since. This is a huge security risk. It can be mitigated by inventing a paper-based process to review the entitlements. But that process is very slow, costly, error-prone and it has to be repeated in regular intervals. But advanced IDM system already supports automation of this re-certification process.
- **Maintain order.** If you closely follow the principle of least privilege then you have probably realized that you have more roles than you have users. Roles are abstract concepts and they are constantly evolving. Even experienced security professionals can easily get lost in the role hierarchies and structures. The ordinary end users often have absolutely no idea what roles they need. Yet, it is not that hard to sort the roles to categories if you maintain them in a good IDM system. This creates a role catalog that is much easier to understand and use.
- **Keep track.** Keep an audit record about any privilege change. This means keeping track of all new account, account modifications, deletions, user and account renames, role assignments and unassignments, approvals, role definition changes, policy changes and so on. This is a huge task to do manually. And it is almost impossible to avoid mistakes. But a machine can do that easily and reliably.
- **Scan for vulnerabilities.** Mistakes happen. System administrators often create testing accounts with trivial passwords when diagnosing application issues. These accounts are not always cleaned up. System administrators may assign privileges to a wrong user. Help desk may enable account that should be permanently disabled. Therefore all the applications have to be permanently scanned for accounts that should not be there and for entitlements that should not be assigned. This is simply too much work to be done manually. It is not realistically feasible unless a machine can scan all the system automatically. This is called reconciliation and it is one of the basic functionalities of any decent IDM system.

Theoretically all of these things can be done manually. But it is not practically feasible. The reality is that without automation and visibility that an IDM system brings the information security seriously suffers. Good information security without an IDM system is hardly possible.

Building Identity and Access Management Solution

There is no single identity and access management solution that would suit everybody. Every deployment has specific needs and characteristics. Deployment in a big bank will probably focus on governance, role management and security. Deployment in small

enterprise will focus on cost efficiency. Cloud provider will focus on scalability, user experience and comfort. Simply speaking one size does not fit all. Almost all IAM solutions use the same principal components. But product choice and configuration will significantly vary. Do not expect that you download a product, install it and that it will solve all your problems. It won't. Customization is the key.

We consider identity management to be the heart of any IAM solution. This is one of the reasons why we have started midPoint project. The rest of this book will focus almost exclusively on identity management and the use of midPoint as the IDM component. This is the place where theory ends and practice begins.

Chapter 3: MidPoint Overview

Chaos was the law of nature; Order was the dream of man.

– Henry Adams

MidPoint is an open source identity management (IDM) and identity governance system. It is a very rich and sophisticated system that provides many advanced features. MidPoint is maintained by Evolveum – a company dedicated to open source IDM development. All midPoint core developers work for Evolveum. However, there are also partners and other developers that are contributing to midPoint development.

MidPoint is a second-generation IDM system. There are few veterans in midPoint development team that deployed first-generation IDM systems since early 2000s. That was not always a pleasant experience. Therefore in 2011 we started midPoint project to correct the mistakes of early IDM systems. One of the main differences between midPoint and other IDM systems is that midPoint is designed and implemented with one primary goal in mind: to be practical. We had been dealing (and struggling) with first-generation IDM systems in the past and we do not want to live through that experience again. Therefore practicality goes very deep into the very foundations of midPoint. To be more concrete, practicality means:

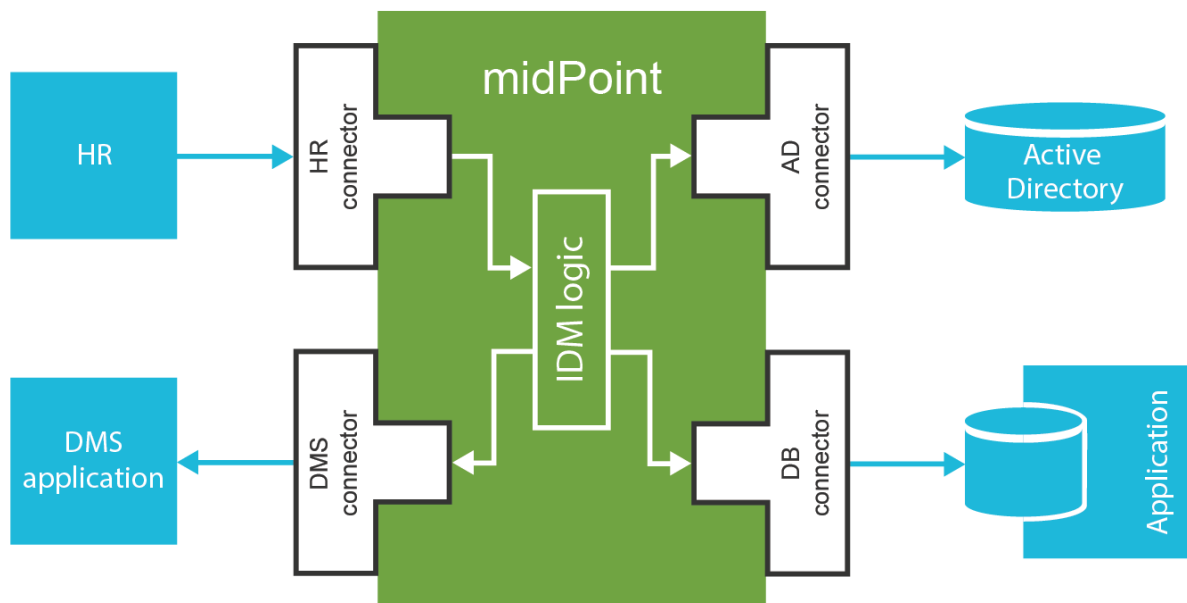
- Things that are simple or used often should be easy to configure. Propagation of changed password, user enable/disable, account synchronization – these should be as easy as possible. As simple as flicking a switch or setting few configuration properties.
- Things that are more complex or used less frequently may be a bit harder. Such as editing XML or JSON file or writing few lines of Groovy or Python script.
- Things that are very complex or very unusual must be still possible. However these might not be easy. It may require longer scripts or implementing some Java classes. It may require forking and modifying the source code. But it must be possible to do almost anything.

This means that simple solutions which do not deviate from the usual requirements will be easy to implement. Most IAM programs start like this. This approach allows to gain the benefits very early in the project. As the requirements are getting more complex and more unusual the effort grows. But it is still much lower than implementing everything from scratch. And there is always an option to stop the project at any point where the costs are getting too high to justify the benefits. MidPoint is an open source system. Therefore there is no license cost that would offset the initial costs. Even small projects are feasible with midPoint.

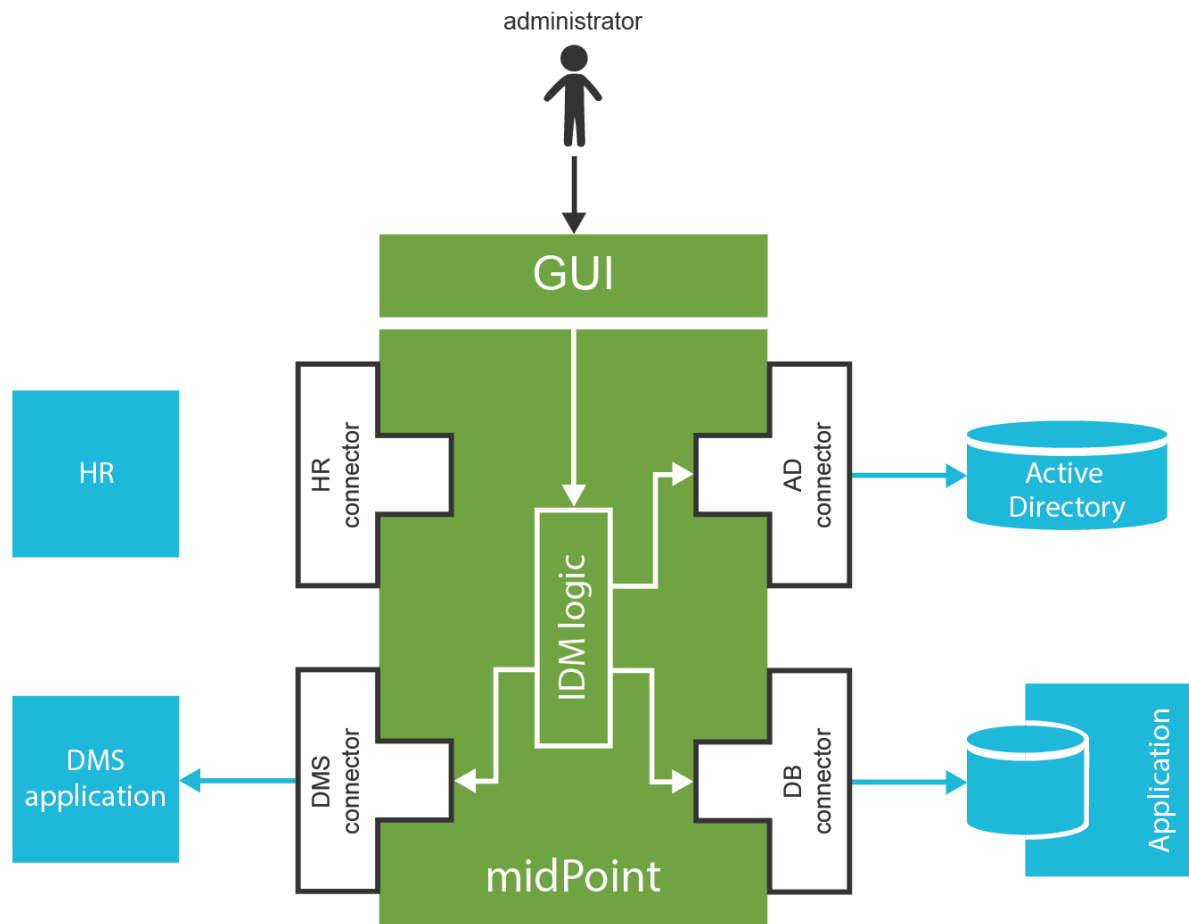
Simply speaking midPoint is following the Pareto principle: 20% effort brings 80% benefits. There are many mechanisms that support this approach. Some are based in midPoint design, some originate from midPoint development practices and some are even supported by the Evolveum business model. But more about that later.

How MidPoint Works

MidPoint does what any identity management system is supposed to do: it manages identities. The very basic functionality of midPoint is the synchronization of identity data maintained in various applications, databases, directory servers, text files and so on. We call all these systems *resources*. MidPoint is using connectors to reach the resource. MidPoint can propagate change that happened in one resource to other resources. E.g. an employee record appears in the HR system, it is picked up by midPoint, processed and new Active Directory and CRM accounts are created. This is the process that we call *synchronization*.



MidPoint also has a rich user interface that can be used to modify the identities. In that case midPoint also propagates the change to all affected resources. E.g. security officer disables a user by clicking on disable button in midPoint user interface. MidPoint makes sure that all accounts that belong to user are immediately disabled.



This is the basic principle of midPoint operation. However this description is extremely simplified. Most of the important things that make midPoint great happen before the changes are applied to target resources. This may look simple at the first sight. But it is not. For each processed change midPoint needs to evaluate:

- **Roles:** MidPoint computes where the user should have access. This is usually given by the roles that the user has. The role structure is usually quite rich. There may be hierarchical roles, parametric roles, conditional roles and a lot of other advanced mechanisms.
- **Organizational structure:** Users usually belong to some organizational units, projects, teams or groups. Some of them may give additional privileges to the user.
- **Status:** Accounts can be created, enabled, disabled or deleted. There are many situations that need to be processed. E.g. we may want to created a disabled account one week before a new employee starts his work, enable the account on his first day, disable the account on his last day and delete it three months after he leaves.
- **Attributes and identifiers:** Simple synchronization cases assume that attributes and values will be the same in the synchronized systems. That is a nice theory, bit it

almost never works in real world. Attribute names need to be translated, values need to be transformed, data types need to be converted. This is different for each system and even for each instance of each system. Small algorithms in form of script expressions are usually needed to properly transform the values.

- **Credential management:** Password changes need to be propagated to the resources. Sometimes we want to synchronize password with all systems, sometimes we want just a subset of systems. Password policies need to be evaluated. Password encoding and hashing might be needed.
- **Consistency:** The account in the target application might have changed since midPoint has updated it. The current change may no longer be applicable, it may conflict with the native change, the change may be already partially applied, the account may have attribute values that it should not have or the account may not exist at all. MidPoint has to detect such situations and react accordingly, e.g. by re-creating a deleted account before applying the changes.
- **Workflow:** MidPoint determines if any of the changes need to be approved before they are applied. If that is the case then midPoint drives the request through an approval process.
- **Notifications:** MidPoint notifies the user that he can access a new account. It notifies the administrator if something goes wrong.
- **Audit:** MidPoint records all the changes into an audit trail. This can later be used by security officers or specialized analytic engines.

All of that is processed, evaluated and executed for each change that midPoint detects. Some of these steps are quite complex. And indeed there are many complex algorithms implemented in midPoint and ready to be used. There are algorithms that evaluate complex role structures, organizational structures, temporal constraints, password policies and so on. The only thing that is needed is to configure them properly.

However, midPoint does even more than that. MidPoint does not only manage identities, it can also manage any object that is anyhow related to the identities. MidPoint can manage roles, role catalogs, organizational structures, groups, projects, teams, services, devices and almost any other object.

MidPoint is also an *identity governance* system. The identity management features make sure that the policies are consistently applied through the organization. The governance features assist with the maintenance and evolution of the policies. MidPoint implements access recertification process. This is a recurring process that requires that managers confirm that the users still need the privileges that they have received. MidPoint also contains mechanism how to sort roles into hierarchies and categories. That is necessary to maintain order during role engineering and maintenance of role definitions. MidPoint also has mechanisms for selective enforcement of role which comes very useful during

migrations and when new system is connected to midPoint. Recent midPoint versions introduced support for policy (role) lifecycle, general policy rules and so on. And more work in that direction is planned in future midPoint versions. We fully understand that it is not enough to simply apply the policies. Policies are living things and they need to evolve.

Case Study

This book is about practical identity management. Therefore we will get very close to a practice by demonstrating midPoint features using a case study. This is a case study of a fictional company ExAmPLE, Inc. The name stands for “Exemplary Amplified Placeholder Enterprise”. ExAmPLE is a mid-sized financial company. Its operation heavily relies on information technologies, therefore there is a diverse set of applications and information systems ranging from legacy applications to cloud services. As ExAmPLE has few thousand employees and there is a good potential for growth the management has decided to start an IAM program. The first step of the program is deployment of midPoint as the identity management system.

Eric is an IT engineer at ExAmPLE and he has taken the responsibility to install and configure midPoint. Eric spins up a new Linux virtual machine for midPoint. He downloads midPoint distribution package and follows the installation instructions. Couple of minutes later the midPoint instance starts up. Eric logs in to the midPoint user interface.

The screenshot shows the midPoint Dashboard. The top navigation bar includes the midPoint logo, a hamburger menu, the word "Dashboard", and a user profile icon labeled "administrator". The left sidebar is divided into "SELF SERVICE" (Home, Profile, Credentials, Request a role) and "ADMINISTRATION" (Dashboard, Users, Org. structure, Roles, Services, Resources, Work items, Certification). The main content area features six summary cards: USERS (1 enabled, 1 total), ORGANIZATIONAL UNITS (0 enabled, 0 total), ROLES (4 enabled, 4 total), SERVICES (0 enabled, 0 total), RESOURCES (0 up, 0 total), and TASKS (3 active, 3 total). Below these are two panels: "Personal info" and "System status".

Last login	
Date	September 12, 2016 6:06:19 PM
From	127.0.0.1
Last unsuccessful login	
Date	Never
From	Not defined
Other	
Account expiration date	Not defined

CPU Usage	1.1
Heap memory (used/committed/max)	468.1MB / 726.5MB / 910.5MB
Non heap memory (used/committed/max)	185.0MB / 190.3MB / -1B
Threads (live/peak/total)	58 / 58 / 62
Start time	Sep 16, 2016 4:32:10 PM
Uptime	4 minutes ago

MidPoint instance is almost empty after fresh installation. It only contains a couple of essential objects. But Eric is a smart engineer. He has already read through this book and he knows exactly what he needs to do.

First thing to do is to populate midPoint with employee data. The primary source of ExAMPLE employee data is an HR system. The HR system is quite big piece of software and it is not easy to connect to that system directly. Fortunately, it is quite easy to get a text export of the employee data in comma-separated (CSV) format. Eric plans to use this file to get employee data to midPoint.

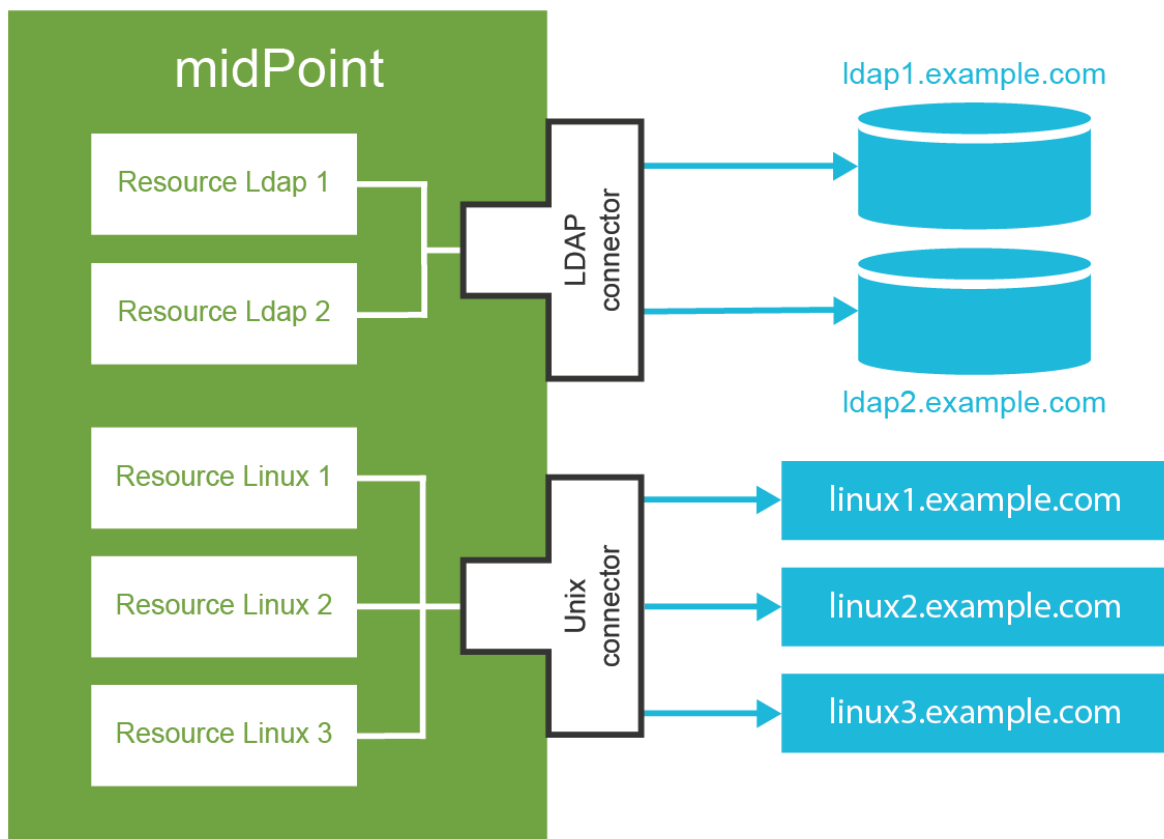
Connectors and Resources

MidPoint communicates with all the source and target systems by the means of *connectors*. Connectors are relatively small Java components that are plugged into midPoint. There is usually one connector for each type of the connected system. Therefore there are connectors for LDAP servers, Active Directory, databases, UNIX operating systems and so on. The responsibility of a connector is to translate protocols. E.g. LDAP connector will translate midPoint search commands to LDAP search requests. The UNIX connector will create an SSH session and translate midPoint create command to the invocation of Linux useradd binary. And so on. Each connector talks using its own communication protocol on one side. But on the other side the connectors all translate the information to a common format that is understood by midPoint.

There is no distinction between source and target system when it comes to the connector. The same connectors are used for source and target systems. The difference is only in midPoint configuration.

The connectors are distributed as Java binaries (JAR files). To deploy them to midPoint you just need to place them in the correct directory and restart midPoint. MidPoint will automatically discover and examine the connectors during the start-up. A handful of frequently used connectors is bundled into midPoint distribution. These connectors do not need to be deployed. They are automatically available.

Connector of a specific type works for all the systems that communicate by the protocol supported by connector. E.g. LDAP connector works for all the LDAP-compliant servers. Connector is just a very generic piece of code. It does not know the hostname, port a passwords that are needed to establish a connection to a particular server. The configurations that specify connection parameters for individual servers are stored in special configuration objects called *resources*. The term *resource* in midPoint terminology generally means any system which is connected to a midPoint instance.



Therefore what Eric the Engineer needs to do to get ExAmPLE employees into midPoint is to define a new resource. This resource will represent the CSV file exported from the HR system. MidPoint distribution already contain a CSV file connector, therefore there is no need to deploy it explicitly. Now Eric has to create a new resource definition. There are (at least) two ways how to do it. Firstly, there is a configuration wizard in midPoint user interface. Eric can use the wizard to configure a new resource from scratch. But as you will see later in this book the resource definition is very flexible and it has many configuration options. This makes the configuration wizard very rich and it may be quite confusing for new users. Therefore it is better for Eric to use the other approach: start from an example. There are examples of various resource definitions in the midPoint distribution package and even more examples are available on-line. Therefore Eric quickly locates a XML file that contains a complete example of a CSV resource. He edits the file to change the filesystem path to his CSV file and adjusts the names of the columns to match the format of his file. The very minimal resource configuration specifies just the resource name, connector and connector configuration. The XML file that Eric creates looks approximately like this (simplified for clarity):

```
<resource oid="03c3ceea-78e2-11e6-954d-dfdfa9ace0cf">
  <name>HR System</name>
  <connectorRef type="ConnectorType"> ...</connectorRef>
  <connectorConfiguration>
```

```

<configurationProperties>
  <filePath>/var/opt/midpoint-example/resources/hr.csv</filePath>
  <uniqueAttribute>id</uniqueAttribute>
</configurationProperties>
</connectorConfiguration>
</resource>

```

Then Eric goes to *Configuration* section of midPoint user interface and imports the XML file into midPoint. Import operation creates new resource definition in midPoint. Eric now navigates to *Resources* section of the midPoint user interface. The new CSV resource is there. When Eric clicks on the resource name a resource details screen appears.

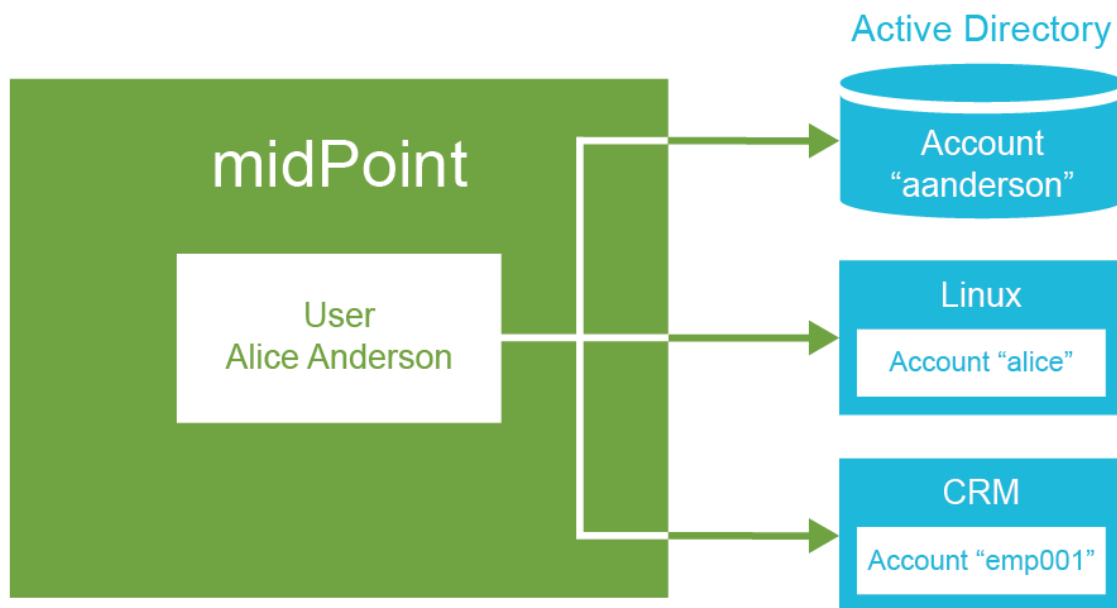
Eric can click on the buttons to test connection to the resource. As this is a local CSV file there is no real connection in this case. But the test checks that the filesystem path is correct, that the file exists and that it can be opened. The test also loads *resource schema*. MidPoint reads the CSV file header and it now knows what is the structure of the data in the CSV file. The resource is now prepared for use.

There is not yet much that Eric can do with the resource. We need to explain a couple of essential midPoint concepts before moving forward with our case study.

User and Accounts

The concept of *user* is perhaps the most important concept in the entire IDM field. The term *user* represents physical person: an employee, support engineer, temporary worker, customer, etc. On the other hand the term *account* refers to the data structure that allows

the user to access applications. This may be an account in the operating system, LDAP entry, row in the database table that stores user identifier and password and so on. One user typically has many accounts – usually one account for each resource.



The data that represents users are stored directly in midPoint. While the data that represents accounts are stored “on the resource side”. Which means accounts are stored in the connected applications, databases, directories and operating systems. Accounts are not stored in midPoint. Under normal circumstances MidPoint keeps just account identifiers and some meta-data about the accounts. All other attributes are freshly retrieved when needed. MidPoint will use the connectors to fetch account data.

We will strictly distinguish the terms user and account in this book and you will find such a strong distinction is also in the midPoint user interface and documentation. It is very helpful to get used to this terminology.

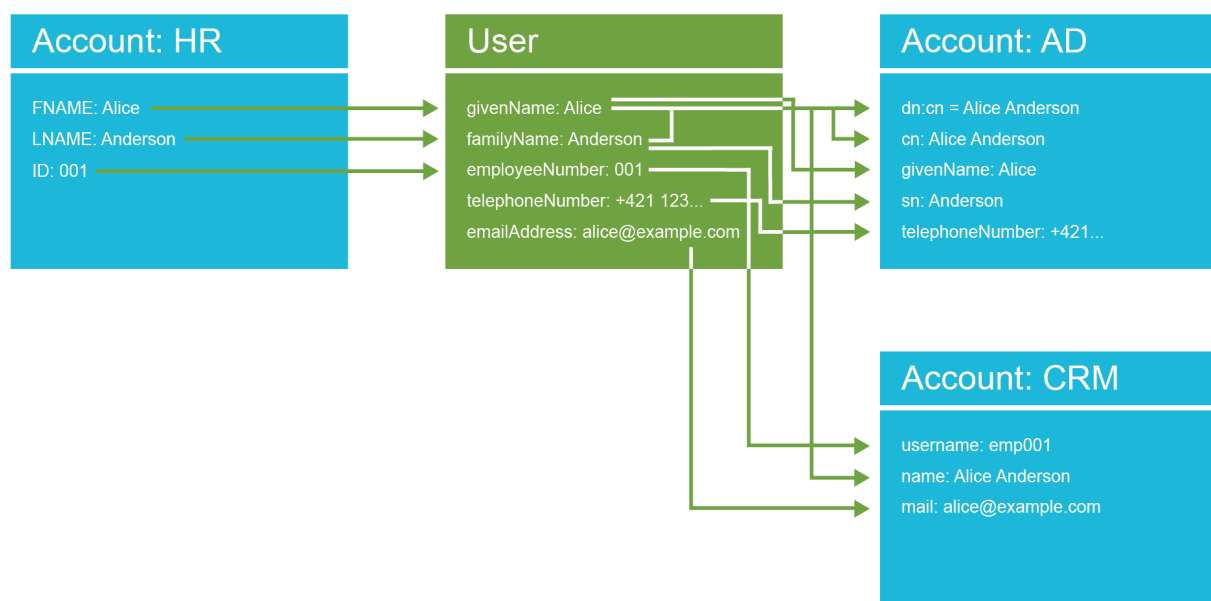
Accounts are linked to users that own the accounts. Therefore midPoint knows which account belongs to who. MidPoint can list all the accounts for any user, it can synchronize the data, it can disable all the accounts at once and so on. This link is usually automatically established and maintained by midPoint.

MidPoint comes with a built-in data model (schema) for users. It contains properties that are very often used to describe users such as full name, e-mail address and telephone number. There is a reasonable set of properties that should be a good starting point for most deployments. Of course, as most midPoint objects, the user schema can be extended with custom properties if needed.










However, there is no unified data model for accounts. There cannot be one. Every resource may have different account attributes. They may have different names, different

types and the values may have different meaning. MidPoint is designed to handle that. Schema for resource accounts is dynamically discovered when midPoint connects to the resource for the first time. MidPoint then interprets the schema in run-time and automatically adapts to it. E.g. when midPoint displays information about account the user interface fields are dynamically generated from the discovered schema. MidPoint does that all by itself. No extra configuration and definitely no coding is necessary.

Account schema in the resources may significantly differ. Yet midPoint must be able to synchronize all the accounts from any kind of resource imaginable. In this case the user schema works as a unified data model. The schema of each account is mapped to the user schema. The user schema is designed as a unified data model that fully represents users in your organization.



Getting back to our ExAmPLE story, Eric has a HR resource configured. Therefore he can see the “accounts” that the users have in the HR system. Eric opens the resource detail page in the midPoint GUI, clicks on *Accounts* tab and then on the *Resource* button (we'll explain that later). The list of accounts appears:

 HR System UP							
Details Defined Tasks Accounts Entitlements Generics Uncategorized Connector							
Intent <input type="text"/> (Object Class: AccountObjectClass) Search In: Repository Resource							
		More... <input type="button" value="Q"/> Advanced					
<input type="checkbox"/>	Name	Identifiers	Situation	Intent	Owner	Result	
<input type="checkbox"/>	 001	uid: 001 name: 001	UNMATCHED				
<input type="checkbox"/>	 002	uid: 002 name: 002	UNMATCHED				
<input type="checkbox"/>	 003	uid: 003 name: 003	UNMATCHED				
<input type="checkbox"/>	 004	uid: 004 name: 004	UNMATCHED				
<input type="checkbox"/>	 005	uid: 005 name: 005	UNMATCHED				
<input type="checkbox"/>	 006	uid: 006 name: 006	UNMATCHED				
<input type="checkbox"/>	 007	uid: 007 name: 007	UNMATCHED				

All that can be seen in this list are just employee numbers, because employee number is set as the primary identifier for the HR system. Clicking on the link will display more details. In fact these are not real accounts. These are lines in the CSV file exported from the HR database. But they describe some aspects of *identity* and therefore midPoint interprets them as accounts. For midPoint “account” is a generic term used to describe any resource-side data structure that represents the user.

Initial Import

The *user* is a central concept for any IDM system and midPoint is no exception. For midPoint to work correctly it needs reliable information about users. The HR system is a good source of user information. Eric needs to get that information from the HR system into midPoint. He has already set up a resource that connects to the CSV file exported from the HR system. But the resource does not do anything by default. It has to be configured to pull the information from the file into midPoint. What Eric needs is a set of *mappings*. Mapping is a mechanism for synchronization of attribute values between user and linked accounts. Eric needs *inbound* mappings to import the data. Inbound mappings synchronize the value in the direction from the resource and into the midPoint. Eric can open the resource definition in the configuration wizard in GUI and he can add the mappings there. Or he simply looks at the configuration samples again and add the mappings in the XML form. Inbound mapping looks like this:

```
<attribute>
```

```

<ref>ri:firstname</ref>
<inbound>
  <target>
    <path>$user/givenName</path>
  </target>
</inbound>
</attribute>


```

This is a mapping that maps the account (HR) attribute `firstname` to user (midPoint) property `givenName`. This tells midPoint to always update a value of user's given name when the mapped HR attribute changes. Eric adds similar mappings for all the attributes in the HR export file. Eric also needs to add *synchronization* section to the resource definition. The synchronization section will instruct midPoint to create a new user for each new account. This is exactly what we want: create a user for each HR account. Eric then re-imports the modified XML file into midPoint.

MidPoint is now ready to synchronize the attributes. But we still need a task to pull all the data from the HR system. Eric navigates to the page that shows the list of HR accounts. At the bottom of that page there is a big *Import* button that can be used to manage the import tasks. Eric clicks on that and creates a new import task. The task is started and it runs for a couple of seconds. After the task is done Eric can look at user in midPoint:

	Name	Given name	Family name	Full name	Email	Accounts
001	Alice	Anderson				1
002	Bob	Brown				1
003	Carol	Cooper				1
004	David	Davies				1
005	Erin	Evans				1
006	Frank	Fox				1
007	Goerge	Green				1
008	Harry	Harris				1
009	Isabella	Irvine				1
010	Jack	Jones				1
011	Kate	Knowles				1
012	Lily	Lewis				1
013	Max	Morgan				1
014	Nathan	Newman				1

By clicking on the username Eric can see details about the user:



(001)

✓ Enabled

✗ No assignments

✗ No organizations

Basic

Projections 1

Assignments 0

Tasks 0

Request a role

Properties

Name *

001

Full name

Given name

Alice

Family name


Anderson

Password

password is set

Change Remove

This page shows all the details about the user that midPoint knows about. The details are sorted to several tabs and we are going to explain all of that later in this book. For now we only care about first two tabs. The “Basic” tab shows user properties as midPoint knows them. These properties are stored in midPoint repository. MidPoint has quite a rich data model that can be used out-of-the-box, but the GUI only shows those properties that are actually used. The “name”, given name and family name were imported from the HR resource and that's what the page shows. Let's have a look at the second tab now:


(001)

✓ Enabled

✗ No assignments

✗ No organizations

Basic


Projections 1

Assignments 0

Tasks 0

Request a role

☐
Projections

☐

HR System
default, 001

Attributes

ID

001

Name

001

First name

Alice

Last name

Anderson

Associations

Password

password is set

Change Remove

The “Projections” tab shows user's accounts. Currently there is only one account and it is the HR account that was used to import the data. The data that are displayed here are the fresh data retrieved from the resource at the very moment that the account was displayed. This is the difference between user data and account data: user data are kept in midPoint repository, while account data are retrieved as needed.

The user and the account are linked. MidPoint remembers that this user originated from this specific HR account. When the HR account is modified then the change is synchronized and applied to the user data. The mappings are not just for the import. They can work continually and keep the account and user data synchronized all the time.

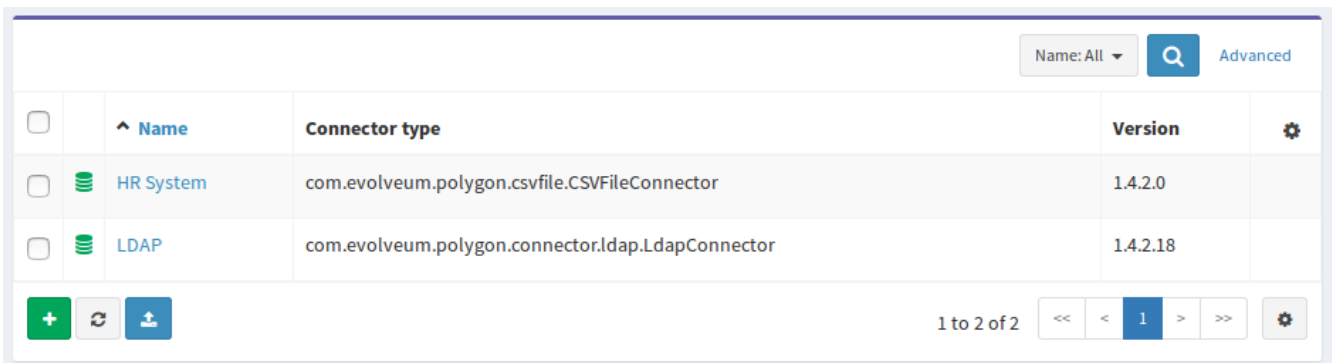
Assignments and Projections

The concepts of an account is all about the reality: it shows the data that are there, here and now. It shows what *is* there. But policies are the things that make up the major part of the identity management. Policies, by definition, specify what *should be* there. Policies specify what is right. But as every citizen knows all too well, the things that *are* and the things that *should be* do not always match perfectly. We are no idealists. Therefore we

have designed midPoint from the day one to acknowledge that there may be a difference between reality and policy. And the primary role of midPoint is to manage that difference and eliminate it completely in the long run.

This kind of thinking is easy to see in midPoint user interface. There is *Projections* tab in the user details page. This tab shows reality. It shows the accounts that the user has right now. It shows the real state in which the accounts are. It shows the reality. And then there is *Assignments* tab. This tab shows the policy. This tab shows what accounts, roles, organizations, or services are assigned to the user. This tab shows what user *should have*.

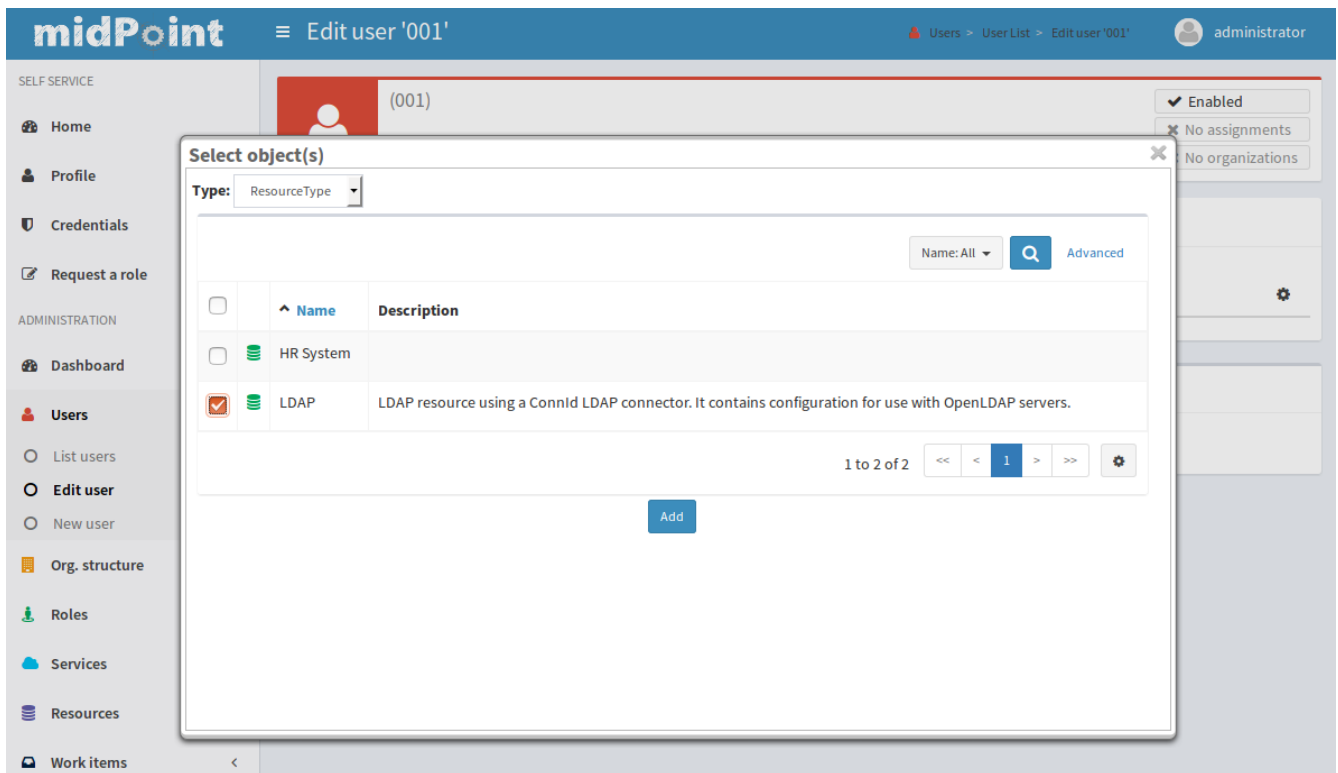
To demonstrate how the assignments work we need a new resource. Therefore let Eric connect a new resource to midPoint. This time it will be new, clean and empty LDAP server. So Eric once again locates the proper example, modifies the configuration and imports it to midPoint. In a while there is a new LDAP resource. Eric wants to synchronize all the users to the LDAP server. And once again Eric has to define mappings. But this time these will be outbound mappings as Eric wants to propagate data out of midPoint and into the (LDAP) resource. We will cover the details of mapping configuration later, so now let's just show the results. We have two resources now:



<input type="checkbox"/>	^ Name	Connector type	Version	⚙
<input type="checkbox"/>	HR System	com.evolveum.polygon.csvfile.CSVFileConnector	1.4.2.0	
<input type="checkbox"/>	LDAP	com.evolveum.polygon.connector.ldap.LdapConnector	1.4.2.18	


1 to 2 of 2

But how do we create an account on the LDAP resource? The right way to do this is to let midPoint know that a user *should have* an account on that resource. In midPoint terminology we say, that we are *assigning* the resource to the user. All that Eric needs to do is to navigate to user details page, click on the *Assignments* tab, use the cog button to add an assignment for the LDAP resource and click *Save*:



After the click on Save button a lot of complex things happen. But simply speaking midPoint will recompute what the user should have and what the user has. MidPoint will detect that the user now should have an LDAP (because there is a new assignment for it). But no such account exists. Therefore midPoint creates the account.

When Eric opens the user details again and navigates to the Projections tab he can see that there are two accounts now:


Alice Anderson (001)

☒ Enabled

☒ No organizations

Basic

Projections 2

Assignments 1

Tasks 0

Request a role

☐ Projections

☐ **HR System** default, 001

☐ **LDAP** default, uid=001,ou=people,dc=example,dc=com

Attributes

Entry UUID	14dc6610-0d50-1036-9b1a-d35c135ff39d	
Distinguished Name	uid=001,ou=people,dc=example,dc=com	
description	Created by midPoint	+
createTimestamp	1473697021000	
Login Name	001	+
Surname	Anderson	+
Given Name	Alice	+
Common Name	Alice Anderson	+

There is an HR account that was used to create the user in the first place. And there is also LDAP account that was created as a reaction to a new assignment.

Info: Careful reader will notice that the two accounts have vastly different attributes. And that's right. Every account has a different *schema*. MidPoint automatically discovers that schema when it connects to the resource for the first time. And then midPoint will dynamically interpret the schema to display the attributes in GUI, to validate the inputs, to check for errors in mappings and so on. MidPoint will do everything by itself without any need to write a single line of code. MidPoint is completely based on the concept of schema and it takes full advantage of that.

There is the reality and there is policy. There are accounts and there are assignments. Ideally these two things should match. And midPoint will try really hard to make them match. But there may be exceptions. Careful reader will once again notice that there is the HR account but there is no assignment for that account. And still midPoint has not deleted the account. That's because the HR system is what we call a “pure source” system. MidPoint does not write to the HR, it only reads from it. Writes to the CSV export file would be overwritten by the next export anyway, so there is no point in writing there. Therefore

the HR resource has an exception specified in its configuration: it allows the HR account to exist even if there is no assignment for it. Using this method we can keep the HR account linked to the user. We can see the data that were used to create the user. This improves overall visibility and it greatly helps with diagnostics of configuration issues.

Roles

It would be a daunting task if Eric had to assign every individual account for every individual resource to every user. Typical IDM deployment has thousands of users and dozens of resources. Such deployment would be very difficult to manage using just the direct assignments that were described above.

But of course, there is a better way: roles. The concept of *role-based access control* (RBAC) is a well-established practice and the roles are really the bread-and-butter of identity management. The basic idea of RBAC is to group privileges into roles. Then the roles are assigned to the users instead of privileges. E.g. let's create a "Webmaster" role. Then put all the privileges that webmaster should have into that role. And let's assign the role to every user that works as a webmaster. This simplifies the privilege management. If there are two webmasters there is no need to think about the individual privileges that a webmaster should have. Just assign the role and the role has everything that is needed. It is also easy to change webmasters: unassign role from one user, assign to another user. It is also easy if you add a new web server. Just add the privilege for accessing new server into the Webmaster role. And all webmasters will have it.

That's the theory. But how does it work for Eric? First of all let's add a handful of new resources – to get some material for the roles. So now we have four resources: HR, LDAP, CRM and Portal. That's a good start. Let's do some role engineering now.

Every organization usually has one role that almost everybody has. It is usually "Employee" or "Intern" role. This role gives access to all the systems that an employee should have access to: Windows domain login, e-mail, employee portal – things like that. The ExAmPLE company is no exception. In this case the basic role should create accounts in two systems:

- LDAP server: many applications are connected to LDAP and use that for authentication. We want every ExAmPLE employee to have account there.
- Portal: this is enterprise intranet portal with lots of small services essential for every employee.

This is simple to do in midPoint user interface. Eric navigates to Roles > New role. Fills in the name of the new role ("Employee") and description. Then he goes to the *Inducements* tab. This is where the role definition takes place. Inducements are almost the same as assignments. However, they do not give access to the role itself. Inducements give access to the users that have this role. So they are kind of indirect assignments. Eric clicks on the cog button and adds the inducements for the two resources into the role:

Basic

Projections 0

Assignments 0

Tasks 0

Request a role

Inducements 0

Policy constraints

☐ Inducements

☐ Portal -

+

☐ LDAP -

+

Options

☐ Force

☐ Reconcile

☒ Execute after all approvals

☐ Keep displaying results

Back

Preview changes

Save

Eric clicks on Save button and the new role is created. Now it is ready to be assigned to the users. Eric goes on and assigns *Employee* role to user Bob. MidPoint automatically creates all the accounts given by the role:

Bob Brown (002)

✓ Enabled

✗ No organizations

Basic

Projections 3

Assignments 1

Tasks 0

Request a role

☐ Projections

☐ HR System default, 002

+

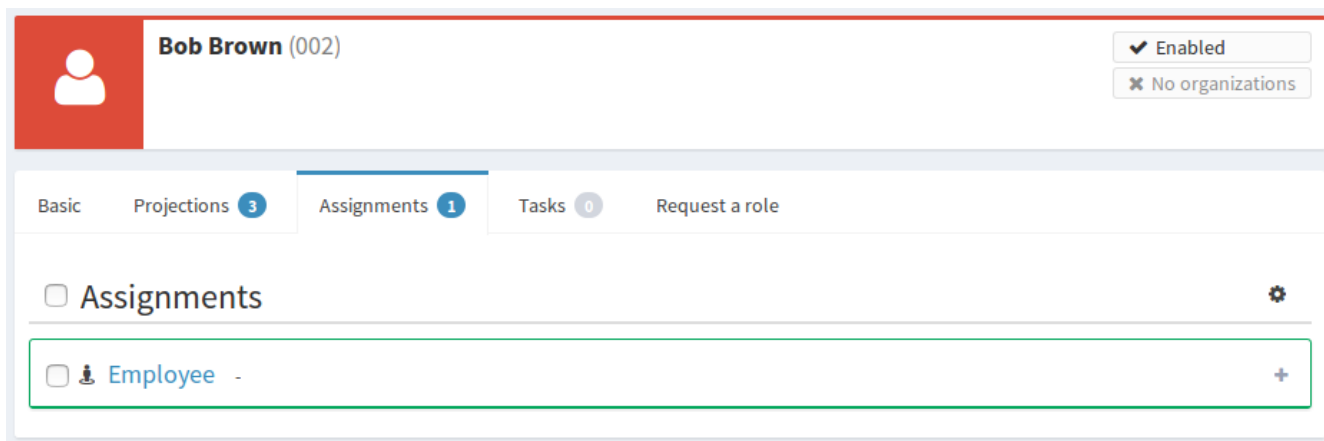
☐ Portal default, 002

+

☐ LDAP default, uid=002,ou=people,dc=example,dc=com

+

There is the HR account that was used to create the Bob user record in the first place. And then there are the two accounts that were created because Bob has the *Employee* role:



This operation works in both directions: if Eric unassigns the *Employee* role, the accounts given by the role will be deleted. Eric can create any number of roles like this: roles for Sales agents with CRM access, roles for Sales managers with higher CRM privileges and so on. MidPoint is designed to handle large number of roles. Each role can have its own combination of resources. MidPoint will seamlessly merge the privileges given by all the roles a user has. E.g. if two roles give CRM access to the user, only one CRM account will be created. If one of these roles will be unassigned the CRM account remains there. It is not deleted yet because it is needed by the other role. Only when the last CRM role is removed that's the point where the account gets deleted. MidPoint takes care of all that logic.

Of course, there much more that the roles can do:

- Roles can assign accounts to groups, give the privileges and generally they can manage account entitlements.
- Roles can mandate specific account attribute values, e.g. clearance levels, compartments, etc.
- Roles may contain custom logic (scripts).
- Roles may be hierarchical: there may be roles within roles.
- Roles may be assigned for only for a specified time.
- Roles may be conditional and parametric.
- ... and much much more.

Roles are really the essence of identity management. We will be dealing with roles in almost all the parts of this book.

There Is Much More

Eric the Engineer has done a few basic steps to configure midPoint as an identity management system for his company. But this is still a very basic configuration. Careful

readers have already noticed a lot of things that need to be done. E.g. employee full name is not automatically generated. Employee numbers are used as identifiers and we would like something more user-friendly here. We would like to automatically assign the *Employee* role instead of doing that manually. And so on. There are still a lot of things to improve. Fortunately, all of that is very easy to do with midPoint once you know where to look. And we will be dealing with all these things in the rest of this book. The new functionality will be administered to the ExAmPLE solution in small doses in each chapter – together with a proper explanation of midPoint principles. MidPoint is a very flexible and comprehensive system and there are still a lot of things to learn. This chapter covered only a minuscule part of midPoint functionality.

What MidPoint Is Not

Now you probably have some idea what midPoint is. However, it is also very important to understand what midPoint is not. Identity and Access Management (IAM) field may sometimes be quite confusing. That is perhaps the reason why the midPoint team occasionally gets questions about midPoint functionality that simply do not make much sense.

First of all, midPoint is not an authentication server. MidPoint is not designed to validate your username and password. Yes, midPoint maintains data about users (including passwords). But the data model that midPoint maintains is quite complex. It is not meant to be exposed to applications directly. That would not be efficient.

If you want midPoint to manage the users but you also want your applications to have a centralized authentication services there is a solution: publish the data to the LDAP server. MidPoint can easily populate the LDAP sever with data. MidPoint will maintain accounts for all the users in the LDAP server. MidPoint will manage the passwords. But the application will not talk to midPoint directly. They will talk to the LDAP server. This is better for everybody: LDAP is a standard protocol well supported in many applications. LDAP servers are also extremely fast and scalable ad nauseam. Therefore use the combination of midPoint and an LDAP server of your choice here. That's what people usually do and it works perfectly.

As midPoint is not an authentication server it obviously is not an Single Sign-On (SSO) server either. If you want SSO you will still need the management capabilities of midPoint. You will also need the scalable directory system (LDAP). But to get real SSO you will also need one additional component: the SSO server. There are plenty to choose from in both the closed-source and open-source worlds. If they work with LDAP they will most likely work in the solution which is managed by midPoint.

One of the things that seems to be shrouded in a lot of confusion is authorization. To get the record straight from the beginning: midPoint is not an authorization server. It is not a policy decision point (PDP) and it definitely is not a policy enforcement point (PEP). You cannot rip out authorization out of your application and just “use midPoint for that”. That does not work.

You can think about midPoint as a policy management point (PMP). MidPoint has a lot of sophisticated authorization-related logic inside its core. But that logic is not designed to answer questions such as “Is subject S authorized to execute operation O on object X?”. MidPoint logic is different. MidPoint is not concerned with making authorization decisions. It is concerned about managing the authorization policies. MidPoint sets up the authorization policies in the target applications. And the applications then evaluate these policies themselves. This is a much more efficient and more reliable method. Unlike authentication, the authorizations decisions are done all the time: at least once per every request, but usually several times per request. If the application makes these decision internally then there is no need to a round-trip to the authorization server. Performance is significantly increased. And there is no single point of failure. As the application has all the data inside then midPoint failure will not interrupt the authorization flow. One less component to cause a failure. The system is more reliable. And still the policies are centrally managed in midPoint. When a policy changes midPoint will update all the affected applications. You get all the benefits without the usual drawbacks.

MidPoint does what it is supposed to do: it manages identities, entitlements, organizational structures and policies. But it does not do things that are not necessary. It does not do the things that other technologies already do well. MidPoint does not reinvent the wheel. There is no need for this. MidPoint is not the wheel. MidPoint sits above all the wheels. MidPoint is the chauffeur.

Chapter 4: Installation and Configuration Principles

The Guide is definitive. Reality is frequently inaccurate.

*– The Hitchhiker's Guide to the Galaxy
(The Restaurant at the End of the Universe
by Douglas Adams)*

This chapter provides instructions for installation and initial configuration of a midPoint system. The instructions describe installation on Linux system because that is by far the most common operating environment for midPoint. However, midPoint is platform-independent and it can run on any environment where Java is running. Any experienced engineer will have no trouble adapting these instructions to fit a different operating environment.

MidPoint installation described in this chapter is a very basic one. It is ideal for initial exploration of midPoint, development of midPoint configurations, demonstrations and similar purposes. It is a very convenient installation and we use it every day for development work. However, to use midPoint in a production deployment the installation need to be slightly adjusted. The adjustments are mentioned in this chapter, but the full description of the production-ready installation is provided in later chapters. This chapter gives you midPoint installation that is ideal to get you started.

Requirements

MidPoint will run on almost any machine. All you need is approximately 4GB RAM. That's perhaps the only real limiting factor. If you look for more formal system requirements definition then you will find that in midPoint wiki (see chapter “Additional Information”).

From the software side you will need:

- **Java 8** runtime environment (JRE) or development environment (JDK). Any JRE or JDK should work. You can use the packages from your operating system distribution. Or you can download Java and install it as a standalone package. Both should work. Just do not forget to set your PATH and JAVA_HOME environment variables to point to the Java installation. You may also want to download and install JCE extension for unlimited-strength cryptography. But midPoint can work even without that.
- **MidPoint distribution package.** Download the latest version of midPoint from the Evolveum website. You are looking for an archive that looks like midpoint - 3.7.1-dist.zip. This archive contains everything that you will need to run midPoint. Extract the archive to some convenient temporary working directory.

We recommend to use the latest available versions of all software packages when dealing with midPoint. We are trying really hard to always keep midPoint up-to-date with the rest of the technologies.

MidPoint Installation

MidPoint is Java web application. However, since version 3.7 midPoint is distributed in a stand-alone package. The distribution package contains everything that midPoint needs to run – except for Java platform itself. Therefore as long as the Java platform is installed all that is needed to run midPoint is to start it:

1. Locate `start.sh` (Unix) or `start.bat` (Windows) script in midPoint distribution package. It should be located in `bin` directory.
2. Execute the script

And that is pretty much it. MidPoint will start. It will initialize the embedded web container, database and all the other midPoint components. That can take a minute or two. After the application is initialized you can access it by connecting to midPoint HTTP port, which defaults to 8080. You can start working with midPoint now.

MidPoint User Interface

Use the following URL to access midPoint user interface:

`http://hostname:8080/midpoint`

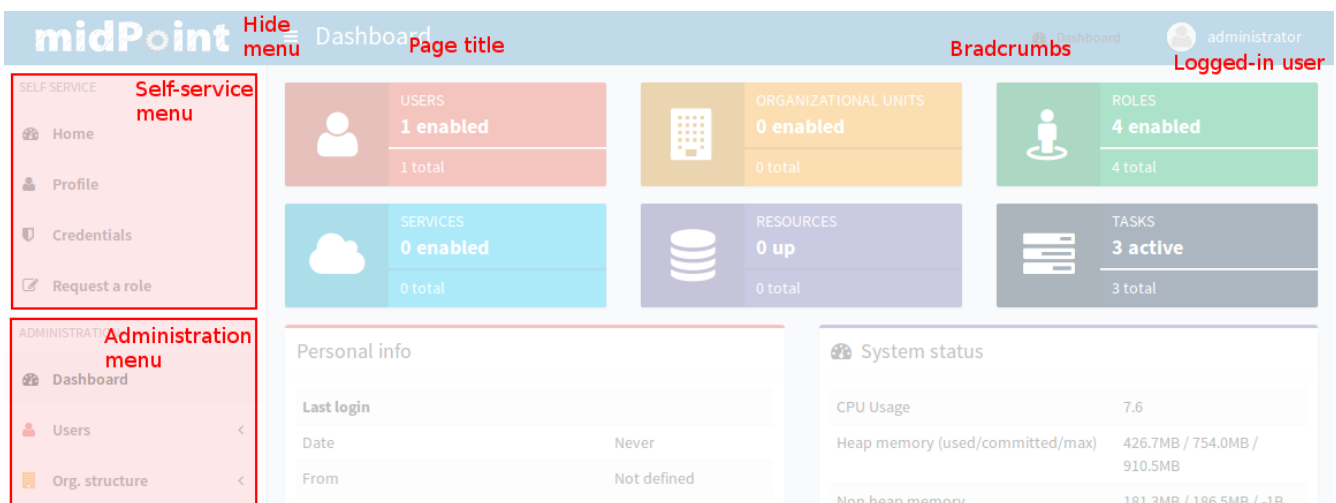
Log in with the following credentials:

Username: administrator

Password: 5ecr3t

Now you are logged-in as the administrator. This user has superuser privileges therefore you can see everything and you can do anything in the midPoint GUI.

MidPoint GUI look and feel is structured, it has the same layout and controls for all the user interface areas:



Primary tool for user interface interaction is the menu. MidPoint user interface is functionally divided into three parts, therefore there are also three parts of the menu:

- **Self-service** user interface deals with the things that the user can do for oneself: displaying list of account, changing password, requesting a role and so on. This is relatively simple part of the user interface. It is often accessible to all the users.
- **Administration** user interface deals with management of other users, roles, organizational structures and similar midPoint objects. This is a very comprehensive and considerably complex user interface. Usually only a privileged user has access to this part of the user interface. Parts of this interface are often used to support delegated administration and role management therefore it is also meant for security officers, resource owners, role engineers and similar expert users.
- **Configuration** user interface deals with configuration of midPoint itself. It is used to customize midPoint behavior, set fundamental policies and rules that form the foundation of midPoint deployments. This part of user interface is usually used only by identity management engineers.

User Interface Areas

MidPoint user interface is quite rich. The following list provides short description of the most important parts of the user interface.

- **Home** page gives a brief status about user's own accounts, requests, work items and so on. This is a page designed to be the first page that will be displayed to the end user after log in to midPoint.
- **Profile** page allows users to see or edit their own profile.
- **Credentials** page allows users to change their own credentials, such as password.
- **Role request** page allows users to select the roles that they need and then request assignment of the roles.
- **Dashboard** page shows statistics about midPoint installation. This is the system dashboard where the system administrators can see the state of the system at a first glance.
- **User** pages list users in midPoint, edit users and allow to create a new user.
- **Organizational structure** pages show the organizational structure trees. Many parallel organizational structures can be managed here, such as tree-like functional organizational structure, flat project-oriented structure, role catalogs and so on.
- **Role** pages allow to list and manage roles. Roles can be created and defined in this part of the user interface.

- **Service** pages allow to list and define services, such as devices, servers, applications and so on.
- **Resource** pages list and manage resources. New resource can be defined here, associated with the connector, tested, etc.
- **Work item** pages list the things that the users have to do. Work items are created if user has to approve something or if there is some manual step in the workflow.
- **Certification** pages deal with access certification (re-certification, attestation). Certification campaigns can be created and managed here.
- **Server task** pages show the tasks that are running on midPoint servers. These may be scheduled synchronization tasks, import tasks, running user requests – everything that runs on the servers and cannot be executed immediately in a synchronous way.
- **Report** pages allow to define and run reports. These pages typically deal with scheduled printable reports.
- **Configuration** area contains many pages that manage midPoint configuration: system default configuration, repository objects, logging, bulk actions and so on.

User Interface Concepts

MidPoint user interface is using the same concepts in all the parts of the user interface when possible. For example all the lists of all the objects (users, roles, ...) look like this:

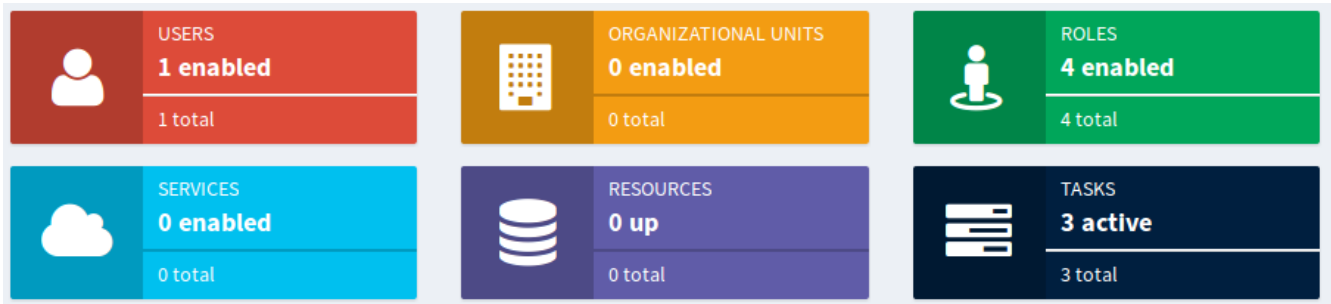
The screenshot displays a web-based table interface for managing roles. The table has columns for 'Name', 'Display name', 'Identifier', and 'Description'. It includes a search bar at the top, a 'Select all' checkbox, and a 'Cog button' for each row. A context menu is shown for the 'End user' role, listing actions like 'Enable', 'Disable', 'Reconcile', and 'Delete'. At the bottom, there are buttons for adding, refreshing, and importing data, along with pagination controls showing '1 to 4 of 4' items.

	Name	Display name	Identifier	Description
<input type="checkbox"/>	Approvers			Role authorizing users to make approval decisions on work items.
<input type="checkbox"/>	End user			Role authorizing end users to log in, change their passwords and reset accounts.
<input type="checkbox"/>	Reviewer			Role authorizing users to make decisions on certification cases.
<input type="checkbox"/>	Superuser			Role that gives user full authorization in MidPoint.

Each table row represents one object: user, roles, service, task, etc. A click on the object name will usually open a page that shows object details. There is also a color-coded object icon. The search bar at the top can be used to look for a specific object or to filter the object view. The “cog button” is a universal tool in midPoint for doing something. A click on the cog button will usually open a context menu with actions. Cog button in the table

header contains actions that apply to all selected objects. Cog button in each table row contains actions that apply only to that individual object. The button in the bottom-left corner execute global actions, such as creating or importing new object and refreshing the view. The “import” button is especially useful. It allows to import new object in XML/JSON/YAML form. The paging controls are in the bottom-right corner.

MidPoint has a unified color-code that makes the navigation easier. Users, roles and other object types have their specific color and icon. This indicates the object type and it is used whenever possible: menu, information boxes, lists, box title accents and so on. The primary colors and icons are shown in the dashboard:



All user-related controls are red, all controls that deal with organizational structure are yellow. Roles are green. And so on. This color code is applied mostly consistently through the midPoint user interface.

Similar color code applies to object icons when displayed in user lists. However, the color that is used there does not indicate object type (as that is obvious from the icon on the page). The icon color indicates the status of the object:

- Black icons indicate normal state. It suggests that there is nothing special to see here.
- Gray icons indicate non-active state. It suggests that the object is disabled, archived or there is another reason why the object is not active.
- Red icons indicate privileged state. It suggests that the object has higher privileges or that it may be sensitive from a security point of view. E.g. administrative users, superuser role, etc.
- Blue icons indicate typical end-user access. It suggests that the object has an access, but the access is limited only to safe, non-privileges operations. E.g. users with end-user role.
- Yellow icons indicate management capabilities. E.g. users that are managers of organizational units.

Info: Almost all objects are equal in midPoint. MidPoint will handle users, roles, organizational units and services in almost the same way. The lists used to display these

objects are the same, the pages that display object details are the same. All the objects have properties, they can be enabled/disabled in the same way, they are subject to authorizations in almost the same way and so on. It is a midPoint philosophy to design several powerful principles and then apply them over and over again.

Object Details Page

Typically when a user clicks on a name of any object the object details page appears. The detail pages for common midPoint objects such as user or role are very similar to each other. They have the same layout and controls. E.g. user details page looks like this:

The screenshot displays the 'Object Details Page' for a user named Ing. Katarína Valalíková. The page is structured as follows:

- Header:** Includes a profile picture placeholder labeled 'Icon or photo', the user's name 'Ing. Katarína Valalíková (katkav)', title 'Software Developer', and organizational unit 'Development Section'. On the right, there are 'Tags' (End user, Manager) and status indicators (Enabled).
- Details tabs:** A navigation bar with tabs for 'Basic', 'Projections' (1), 'Assignments' (5), 'Tasks' (2), and 'Request a role'.
- Properties section:** A large form area with a red border containing various input fields for user information: Name (katkav), Full name (Ing. Katarína Valalíková), Given name (Katarína), Family name (Valalíková), Honorific Prefix (Ing.), Title (Software Developer), Email Address (katarika.valalikova@evolveum.com), Employee Number (003), Locality (Bratislava), and Jpeg photo (Browse... No file selected.). To the right of this section are 'Sort and hide buttons'.
- Activation section:** A section titled 'Activation' showing 'Lock-out Status' as 'Normal'.
- Password section:** A section titled 'Password' showing 'password is set' with 'Change' and 'Remove' buttons.
- Options section:** A section titled 'Options' with checkboxes for 'Force', 'Reconcile', 'Execute after all approvals' (checked), and 'Keep displaying results'.
- Operation buttons:** A bottom bar with 'Back', 'Preview changes', and 'Save' buttons.

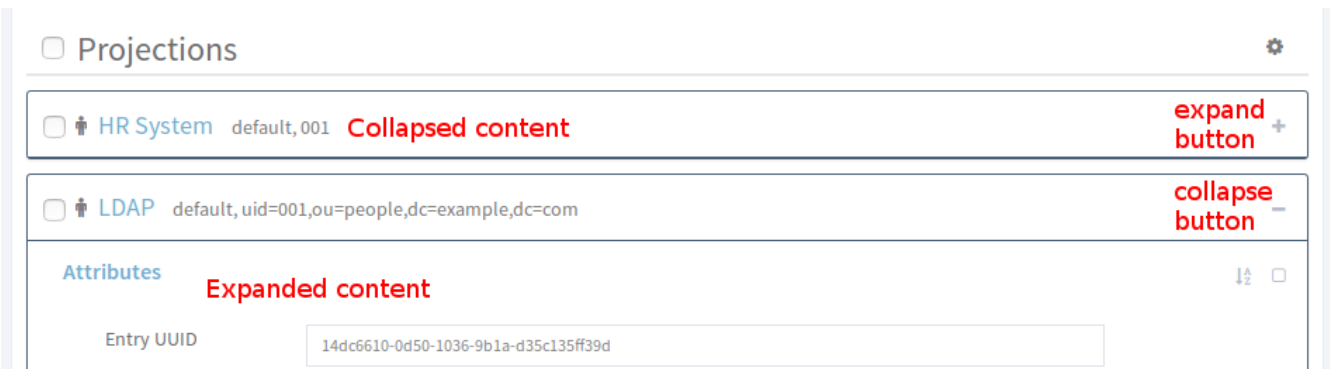
At the top of the page there is an information area. That area shows user photo (or icon) and provides some basic information such as user name and identifier. It also shows where the object belongs in the organizational structure. There is also a couple of “tags” that show interesting details about the object: whether the object is enabled, whether it has special privileges and so on.

The screen below the information area is divided into several tabs. Each tab shows one aspect of the object. The first tab usually shows object properties. There are tabs that show projections, assignments, inducements – we will come to that later in this book. But all of the tabs show some aspect of the displayed object. The first tab is perhaps the most interesting now. It contains a dialog that shows object properties: the attributes of the object. They are displayed or they can be edited – depending on the authorizations of currently logged-in user. In addition to the basic properties there are also other sections. E.g. activation section shows whether the object is enabled or disabled, it shows the activation dates and other activation details. The credentials section allows to change password and other credentials. At the top of the properties section there are two little buttons. The first one can be used to change the ordering in which the properties are shown. The other button toggles the display between showing all the properties or only some of the properties.

Operation options and buttons are located at the bottom of the details page. The buttons initiate the operations. The most common operation is just to save the changes and that's what the “Save” button is for. Saving the changes is a universal way how to start almost any operation: change of user properties, assignment of roles, change of password, user disable, etc. When you make edits in any of the tabs on the details page then nothing really happens yet. MidPoint just remembers what you are editing. The real things happen only when you click the Save button. This is our method how to execute several things in one operation. It may require some time to get used to it. Just do not forget to click the save Button.

The operation options above the buttons are used to modify the behavior of the operation. These options may force to execute operations that fail to pass midPoint internal checks. There is an option to reconcile the data even if midPoint thinks that reconciliation is not needed. And so on. Checking or unchecking these options influences the way how midPoint executes the operation.

MidPoint user interface often needs to present objects that are internally quite complex. But these objects need to be presented in quite a compact form. This applies to list of user's accounts, assignments, role inducements, etc. In that case midPoint is using expandable views. The objects are initially displayed as collapsed, displaying only the basic data. Such objects may be expanded to show more details:



Watch out for expand/collapse buttons in midPoin user interface. Clicking on these buttons may provide the additional details that you are looking for. Also, clicking on the object label usually also expands the view.

MidPoint Configuration Basics

The principle of midPoint configuration is quite different from what would a typical system administrator expect. There are almost no configuration files in midPoint. MidPoint is storing vast majority of its configuration in its configuration database. There are several reasons for this:

- MidPoint configuration is **complex**. This is not what a typical system administrator would think of like a “configuration”. MidPoint configuration contains numerous resource definitions that in turn contains mappings that in turn may contain scripts. There are roles, policies, templates, ... and these objects are too complex to be expressed in simple configuration files.
- MidPoint configuration is **scalable**. It is no exception that a midPoint deployment has thousands of role definitions or organizational units, tens of resource definitions and a number of templates and policies. All of that needs to be stored efficiently, so midPoint can handle deployments with millions of users. The configuration also needs to be searchable. Managing thousands of roles in the text files simply won't work.
- MidPoint configuration needs to be **available**. There are midPoint deployments with several nodes working as a cluster. Configuration change done on one node has to be seen by the other node. Simple configuration files would not work here.

Therefore midPoint has a completely different approach to configuration. The configuration is stored in a form of well-defined structured objects in the midPoint database. We call that database *midPoint repository*.

Configuration Objects

Everything is an object in midPoint. Every piece of configuration is represented as a structured object and stored in midPoint repository. It may look like this:

```
<role oid="8ebab0bc-7e7e-11e6-a7bc-57de1cd45ecc">
```

```

<name>Basic User</name>
<requestable>true</requestable>
<roleType>business</roleType>
<inducement>
  <targetRef oid="f92e67c2-7e7e-11e6-a306-7bf6aa2e8c61" type="RoleType"/>
</inducement>
</role>

```

Every object has its identifier. We call that identifier **OID** which is simply a shorthand notation for “object identifier” (it has nothing to do with LDAP or ASN.1 OIDs). This is usually randomly-generated UUID value. It has to be a value that is unique in a whole system. This identifier is once assigned to the object and it should never change. This is what is known as *persistent* identifier. It is used for internal purposes and it is almost never displayed to midPoint user.

Every object also has a **name**. Name is human-readable and it can change any time. This is the value that is displayed to users.

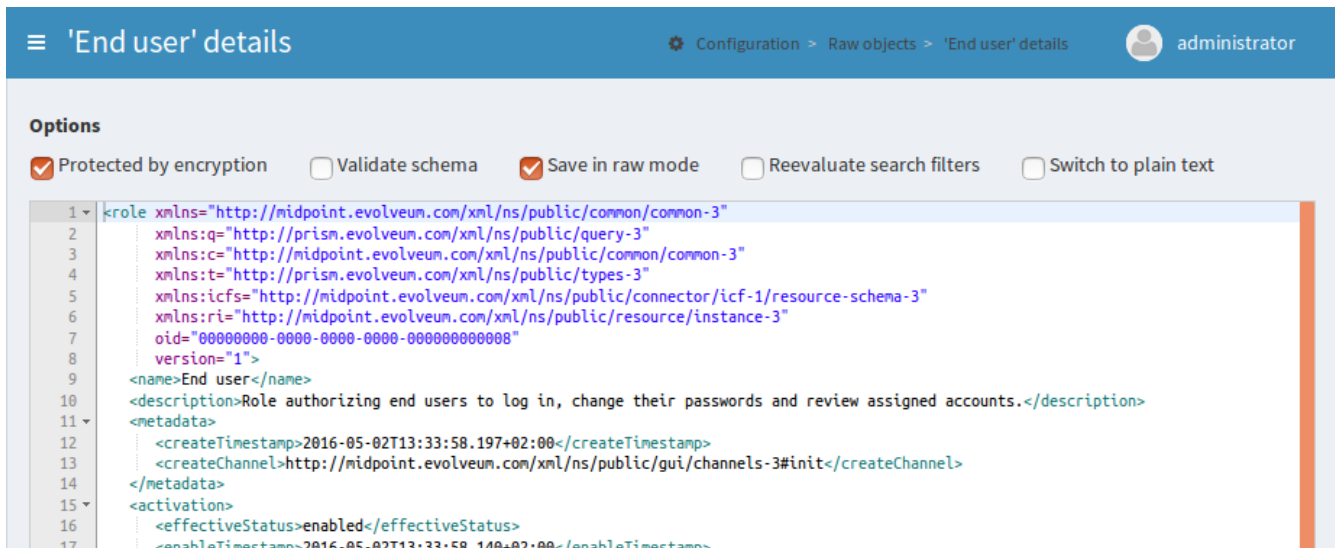
And then the objects have properties. Or rather “items”. Each type of midPoint object has a slightly different set of these items. That’s what we call *schema*. The items may be simple properties such as string, integer or boolean values. But there also may be complex structures and references between objects. The data model is quite rich. It is in fact so rich that its description will take better part of this book: because description of the data model is also description of midPoint features.

You can see midPoint configuration objects in midPoint user interface by navigating to *Configuration > Repository Objects* and selecting object type. The following picture shows objects of type “Role”:

Options <input type="checkbox"/> Use zip				
Role		Name: All	More...	Advanced
<input type="checkbox"/>	^ Name	Description		
<input type="checkbox"/>	Approver 00000000-0000-0000-0000-00000000000a	Role authorizing users to make approval decisions on work items.	Export	Delete
<input type="checkbox"/>	Blogger a73b0386-1cf3-11e6-ac6e-dfedc87cdda3	Author of Evolveum blog posts. Requestable role with an approver.	Export	Delete
<input type="checkbox"/>	Contributor 9ff31e4c-1cf3-11e6-bc5d-0727c08b96ed	Contributor to Evolveum projects. Requestable role with an approver.	Export	Delete
<input type="checkbox"/>	Delegated Identity Administrator b613c706-3889-11e6-b175-d78cc67d7066	Allows full identity administration for organizations where the user is a manager.	Export	Delete
<input type="checkbox"/>	Domain Administrator 601c0f9e-ba0c-11e5-bc34-5f83c73a7961	Sample parametric role. It expects "domain" parameter that is used to compute values on resource.	Export	Delete
<input type="checkbox"/>	Domain Auditor c965cac8-ba1e-11e5-abca-1fa0da899e1f	Sample parametric role. It expects "domain" parameter that is used to compute values on resource.	Export	Delete

XML, JSON and YAML

The objects are stored in the midPoint repository in a native form which is hidden from midPoint users. However, the objects also have a human-readable representation. They can be represented in XML form and (starting with midPoint 3.5) also in JSON a YAML forms. All the objects can be imported into midPoint in the XML form. They can be exported from midPoint in XML form. They can be even edited directly in midPoint using embedded editor. Just click on any object in the *Repository objects* page:



The ability to export, import and edit objects in XML/JSON/YAML form is absolutely essential, because:

- It is **human-readable** (or rather administrator-readable). The configuration can be created, edited and maintained in your favorite editor and then imported into midPoint. It can be reviewed. It can be copied and pasted. Especially that. No system administrator can live efficiently without an ability to copy and paste.
- It is **transferable**. It can be exported from one system (e.g. development environment) and easily transferred to another system (e.g. testing environment). It can be easily backed-up and restored. It can be easily shared, e.g. in a form of configuration samples.
- It is **versionable**. The exported configuration can be easily put under any ordinary version control system. This is essential for deployment projects and configuration management.

Therefore the midPoint configuration has the best of both worlds. It is stored in database, so it can be processed efficiently, it can be made available and so on. But it also has a text form, so it can be easily managed.

The XML, JSON and YAML forms are considered to be equivalent. Starting with midPoint 3.5 the objects can be written in any of these forms.

XML form:

```
<role oid="8ebab0bc-7e7e-11e6-a7bc-57de1cd45ecc">
  <name>Basic User</name>
  <requestable>true</requestable>
  <roleType>business</roleType>
  <inducement>
    <targetRef oid="f92e67c2-7e7e-11e6-a306-7bf6aa2e8c61" type="RoleType"/>
  </inducement>
</role>
```

JSON form:

```
{
  "role" : {
    "oid" : "8ebab0bc-7e7e-11e6-a7bc-57de1cd45ecc",
    "name" : "Basic User",
    "requestable" : true,
    "roleType" : "business",
    "inducement" : {
      "targetRef" : {
        "oid" : "f92e67c2-7e7e-11e6-a306-7bf6aa2e8c61",
        "type" : "RoleType"
      }
    }
  }
}
```

YAML form:

```
role:
  oid: "8ebab0bc-7e7e-11e6-a7bc-57de1cd45ecc"
  name: "Basic User"
  requestable: true
  roleType: "business"
  inducement:
    - targetRef:
        oid: "f92e67c2-7e7e-11e6-a306-7bf6aa2e8c61"
        type: "RoleType"
```

The examples in this book use mostly XML notation. The XML form is almost always simplified for clarity: there are no namespace definition, no namespace prefixes and so on. The complete files with all the details can be found in midPoint distribution package, midPoint source code or in other places. See Additional Information chapter for more details.

Maintaining MidPoint Configuration

When it comes to maintenance of midPoint configuration there are two practical methods how to do that.

First method is to maintain the configuration in midPoint: use midPoint wizards and user interface tools to create new objects and modify them. Export the objects in regular intervals so they are backed up, placed under version control and so on. This is an easy method to start with. But sooner or later you will probably figure out that you need the ability to copy and paste parts of the configuration. That you need to share the

configuration with other team members. And that the no user interface is ever as efficient as an experienced engineer with a good text editor.

Then there is a second method: maintain the configuration files in text form outside of midPoint. Import them to midPoint as needed. The objects can be imported in midPoint user interface by going to *Configuration > Import object* page. There are also blue import icons in almost all the object list tables that also lead to that page. If you are re-importing an object do not forget to check the overwrite option.

It is much easier to maintain a proper version control and a good teamwork using the import method. It also seems to be more efficient once you get used to midPoint: pieces of configuration can be copied from samples, documentation or from other projects. This makes the work efficient. Although work with midPoint is “just” configuration and there is usually almost no programming, this method of work is quite close to the methods that software developers use. And we know that these methods work quite efficiently.

If you maintain the configuration files out of midPoint you can import them individually using midPoint user interface. This may seem like quite an uncomfortable way. But it works surprisingly well even for a mid-size projects. However, starting from midPoint 3.5 there is a much better way. MidPoint has a plug-in into Eclipse IDE environment that allows you to maintain the configuration files in form of Eclipse projects. The plugin allows easy download and upload of changed configuration files. As Eclipse also has good integration for version control systems and other development tools this seems like an ideal approach for large and complex projects.

Looking Around The Installation

We have a running midPoint installation now and you should have some understanding of how to configure it. But before we plunge into more details about configuration let's have a look at midPoint installation. There are few things that need to be understood before going on. They will save a lot of time later on.

MidPoint needs its own database to work. It is used to store the configuration, users, resource definitions, account links, audit trails and a lot of other things. Proper relational database (PostgreSQL, MySQL/MariaDB, Microsoft SQL or Oracle) is strongly recommended for production deployment. But for development and demonstration purposes midPoint contains an embedded database engine (H2 database). This embedded database is initialized by default when midPoint is installed. And it is that embedded database that is used to store the configuration objects right now in your fresh midPoint deployment. This database does not need any special configuration, the database schema is applied automatically and it is started and stopped together with midPoint. Therefore it is ideal for development purposes.

Vast majority of midPoint configuration is stored in the database (midPoint repository). But there are few things that cannot be stored in the database. Such as connection parameters to the database itself. For that purpose midPoint has a small configuration file

called `config.xml`. MidPoint also needs a place where to store other data that cannot be in the database, such as cryptographic keys, connector binaries and so on.

For that purpose midPoint needs a special directory on a filesystem. We call it *midPoint home directory*. If no location is explicitly specified then a new directory with the name `var` is created in directory where midPoint distribution package was installed, i.e. at the same level as the `bin` directory where midPoint start scripts are located. Assuming that midPoint was installed in `/opt/midpoint` directory then midPoint home will be located in `/opt/midpoint/var` directory.

The location of midPoint home directory can be changed by using the `midpoint.home` Java system property. This is done by specifying `-Dmidpoint.home` in the JVM command-line. Or in case that the default midPoint start scripts are used `MIDPOINT_HOME` environment variable can be used to set the location of midPoint home directory.

When midPoint starts for the first time it will find that the database is empty. MidPoint will then populate the database with a minimal set of configuration objects. This set contains objects such as the Superuser role and the user administrator. These objects get imported automatically because if they are not there you will not be able to log into the new midPoint instance. These objects are imported only if they are not already present in the database. If you modify them later then midPoint will not overwrite them.

Logging

Logging is perhaps the most important mechanism to diagnose any issues with midPoint. Logging should be *the* thing that comes to your mind anytime you cast a puzzled look at midPoint user interface. We try to make midPoint user interface convenient to use and we pay a lot of attention to good error reporting. But there are limits. The error that the user interface displays may be just a result of a long chain of causes and effects and therefore it may not directly point to the primary cause. Or maybe there is no error at all, just midPoint does not do what it is supposed to do. That is the point where logging comes to the rescue.

MidPoint is using Java logging facilities to log its messages. MidPoint log file name is `midpoint.log` and it is stored in the `log` subdirectory in midPoint home directory (`/opt/midpoint/var/log/midpoint.log`). The default logging level is set up more-or-less to suit normal midPoint operation. This means that the messages on level `INFO` and above are logged while the finer levels are not logged. If you want to diagnose midPoint issues you will need to switch the logging levels to `DEBUG`, or in extreme cases even to `TRACE`. The logging levels can be adjusted in midPoint user interface. Navigate to *Configuration > Logging*.

Chapter 5: Resources and Mappings

The pessimist complains about the wind; the optimist expects it to change; the realist adjusts the sails.

– William Arthur Ward

Reading and writing resource objects, attribute synchronization, mapping of attribute values, their transformation using scripts – these are the basic midPoint features. These features are absolutely essential for any self-respecting IDM deployment and all IDM engineers should be more than familiar with them. And this is exactly the purpose of this chapter: describe the configuration needed to use midPoint as a provisioning engine.

It is unrealistic to expect that all the systems will agree on the same interface, communication protocol and schema. There were several attempts to unify the IAM landscape. But they were not entirely successful. The LDAP protocol is here since 1990s and the implementation are still not 100% interoperable. It is even worse in the identity provisioning. There were several attempts to specify a standard provisioning protocol, but all of them failed to deliver complete interoperability. The worst pain point of identity integration is undoubtedly the schema. Every application has its own data model for representation of accounts, groups, privileges and other identity-related objects. Even if the application tries to expose that data model using a some kind of standard schema there will always be small (but important) details and differences. MidPoint provides a practical solution to this problem. Application interfaces and their schemas need to be aligned or *mapped* to a common identity schema that you choose to use for your deployment. There are nice applications and mapping of these takes not more than few minutes. And there are naughty applications that require days or even weeks of hard work. But some integration and configuration work is always required. And this chapter will tell you how to do it.

Resource Definitions

The *resource* is one of the most important concepts in midPoint. Resource is any system that midPoint is connected with. Resources are typically target systems where midPoint manages accounts. But source systems such as HR databases are also defined as resources. There is no strict distinction between source and target resource in midPoint. Both source and target resources are defined in exactly the same manner. The resources can even act as both sources and targets at the same time. And they can be both sources and targets even for the same attribute. This is given by midPoint unique relative change model (see below).

Any system connected to midPoint is a resource. MidPoint needs a way how to communicate with the resource. MidPoint has to know communication protocol, hostname,

passwords, etc. For that purpose midPoint has *resource definition objects*. These are ordinary midPoint configuration objects stored in midPoint repository. The resource definition usually contains:

- **Name** of the resource and description
- **Reference to the connector** used to communicate with the resource
- **Connector configuration** properties that define resource host name, port, communication settings and so on. There are used to initialize the connector.
- Definition of **object types** that are interesting for midPoint. This is typically a definition that describes how a typical account looks like. But there may be much more: groups, roles, organizational units, ...
- Object type definitions typically contain **mappings**. Mappings define how attributes are synchronized from midPoint to resource (target resources) or from resource to midPoint (source resources).
- **Synchronization** settings that define what midPoint should do if it discovers unknown account, if the account is deleted on the resource and so on.

Resource definition looks like this in its XML form (simplified):

```
<resource oid="b4101662-7902-11e6-9f14-53e18426fe81">
  <name>LDAP</name>
  <connectorRef oid="028159cc-f976-457f-be70-9e9fa079bcf7"/>
  <connectorConfiguration>
    <configurationProperties>
      <port>389</port>
      <host>localhost</host>
      <baseContext>dc=example,dc=com</baseContext>
      ...
    </configurationProperties>
  </connectorConfiguration>
  <schemaHandling>
    <objectType>
      <kind>account</kind>
      <default>true</default>
      <objectClass>ri:inetOrgPerson</objectClass>
      ...
    </objectType>
  </schemaHandling>
</resource>
```

Resource definition is a very rich (and powerful) configuration object. It is maybe the richest configuration object in the entire midPoint system. Creating resource definition from scratch is usually no easy task. There is a lot of things to consider here: connector configuration, identifier conventions, mandatory attributes, attribute value formats and so on. What we usually do is to locate a resource definition sample that is using the same connector. Then we modify the sample to suit our needs. However, to do this you need to understand how the resource definitions work. Next few sections will explain that.

For those that do not like XML/JSON/YAML or for those that really want to start creating the resource from scratch there is a resource wizard in the midPoint user interface. The wizard can be used to create and edit resource using a graphical user interface.

The screenshot shows the midPoint Configuration wizard. The top navigation bar has steps: Resource basics, Configuration (active), Schema, Schema handling, Synchronization, and Capabilities. Below the navigation bar, there are tabs for Configuration, Connector pool, Timeouts, and Results handlers. The Configuration tab is selected, showing fields for Host (localhost), Port number (389), Connection security, and SSL protocol.

However, even if resource wizard is your preferred way, it may be still easier to start with an existing sample. Find the sample that is the best match for your situation, import it in midPoint and then use the wizard to modify it.

There are many resource samples to start from. Most of them are located in midPoint distribution package. But there are other places to look for samples. Please see the Additional Information chapter.

Connectors

Every resource needs a connector to work. Connectors are small pieces of Java code that are used to communicate with the resource. MidPoint looks for available connectors when it starts up. MidPoint will automatically create new configuration object for each connector that it discovers during startup. The list of discovered connectors can be seen in midPoint user interface in Configuration > Repository objects > Connector. There is one object for each connector that midPoint discovers. The connector objects look like this (simplified):

```
<connector oid="028159cc-f976-457f-be70-9e9fa079bcf7">
  <name>ICF com.evolveum.polygon.connector.ldap.LdapConnector v1.4.2.18</name>
  <framework>http://midpoint.evolveum.com/xml/ns/public/connector/icf-
1</framework>

<connectorType>com.evolveum.polygon.connector.ldap.LdapConnector</connectorType>
  <connectorVersion>1.4.2.18</connectorVersion>
  <connectorBundle>com.evolveum.polygon.connector-ldap</connectorBundle>
  <namespace>http://midpoint.evolveum.com/xml/ns/...</namespace>
  <schema>
    ...
  </schema>
</connector>
```

The resource definition needs to point to the appropriate connector object. Therefore select the right connector from the connector list and remember its OID. Then use the connector OID in the resource configuration like this:

```
<resource>
  <name>LDAP</name>
  <connectorRef oid="028159cc-f976-457f-be70-9e9fa079bcf7"/>
  ...
</resource>
```

This is a straightforward way how to link connector and resource. However, it is not the most convenient one. MidPoint creates connector objects automatically. Therefore the OIDs of the connector objects are not fixed. Every midPoint instance will have different OID for the discovered connectors. Therefore if we want a resource that is always using the LDAP connector in all the midPoint instances we cannot do that by just using OIDs. But there is another way. You can use search filter instead of fixed OID:

```
<resource>
  <name>LDAP</name>
  <connectorRef type="ConnectorType">
    <filter>
      <q:equal>
        <q:path>connectorType</q:path>
        <q:value>com.evolveum.polygon.connector.ldap.LdapConnector</q:value>
      </q:equal>
    </filter>
  </connectorRef>
  ...
</resource>
```

The detailed explanation of the search filters will come later. For now it is important to know just few basic principles. When this resource definition is imported, midPoint will notice that there is no OID in the connectorRef reference. It will also notice that there is a search filter. So midPoint will execute that search filter. In this case it will look for object of ConnectorType type that has property connectorType with value com.evolveum.polygon.connector.ldap.LdapConnector. Therefore midPoint will find LDAP connector regardless of the OID that was generated when midPoint discovered that connector. Then midPoint takes the OID of the object that it has found. The OID will be placed to the connectorRef reference, so midPoint will find the connector directly and it will not need to execute the search every time the resource is used.

This is the method that is almost always used to bind resource definition to a specific connector type. It has the advantage that it works in all midPoint deployments. Therefore it is also used in all the samples.

Bundled and Deployed Connectors

Each class of resources needs its own connector. There is an LDAP connector that supports all the common LDAP servers. There are connectors that work with generic

database tables. These connectors are quite generic. But most connectors are built for a specific application or software system: SAP R/3, LifeRay portal, Drupal, etc.

There is a handful of connectors that are so generic that they are used in almost all midPoint deployments. These connectors are bundled with midPoint. That means that they are part of the midPoint application and they are always available. These three connector bundles are part of midPoint:

- **LDAP Connector bundle**, which contains:
 - **LDAP connector** that works with most LDAPv3-compliant servers.
 - **Active Directory** connector that can work with Microsoft Active Directory over LDAP protocol.
 - **eDirectory** connector which is a modified LDAP connector to support peculiarities of eDirectory LDAP implementation.
- **DatabaseTable** connector bundle with a connector that can connect to a generic relational database table.
- **CSV** connector bundle with a connector that works with comma-separated (CSV) text files.

These connectors are always available in midPoint. Other connectors must be deployed into midPoint. The deployment is a very straightforward process:

1. Locate the connector binary (JAR file).
2. Copy the binary into the `icf-connectors` directory which is located in midPoint home directory.
3. Restart midPoint

MidPoint will scan the `icf-connectors` directory when it starts up. It will discover any new connectors and create a connector configuration objects for them.

Connector Configuration Properties

Connectors need a configuration to be able to work with resources. This configuration usually consists of connection parameters such as hostname, port, administrative username, password, connection security settings and so on. The connection configuration properties are specified in resource definition object. In a simplified form it looks like this:

```
<resource oid="b4101662-7902-11e6-9f14-53e18426fe81">
  <name>LDAP</name>
  <connectorRef oid="028159cc-f976-457f-be70-9e9fa079bcf7"/>
  <connectorConfiguration>
    <configurationProperties>
      <port>389</port>
      <host>localhost</host>
      <baseContext>dc=example,dc=com</baseContext>
    </configurationProperties>
  </connectorConfiguration>
</resource>
```

```

    ...
    </configurationProperties>
  </connectorConfiguration>
  ...
</resource>

```

There may be a very broad range of configuration properties – and every connector has its own set. While working just with the text representation of the resource definition you will need to find out the names of the configuration properties by looking at the samples, connector documentation or maybe even connector source code. It may look difficult but this is perfectly viable approach. However, there is also other way. Firstly there is the resource wizard. The wizard knows all the connector configuration properties and it will present the properties in a configuration form. The wizard takes the definition of the configuration properties from *connector schema*. Connector schema is a definition of the properties that the connector supports: their names, types, multiplicity and so on. The connector schema is stored in the connector configuration object under the `schema` tag. Therefore even if you are working only with the XML/JSON/YAML files you can have a look at that schema to figure out what connector configuration properties are supported.

The connector schema also defines connector namespace. Generally speaking namespaces in midPoint are used to isolate schema extensions that might conflict and they are also used for data model versioning. The use of namespaces is optional in almost all parts of midPoint – but not yet in all the parts. Connector configuration is one of the few parts where namespaces should still be used. And it also makes some sense, as namespaces are used here as an additional safety mechanism. To keep a long story short, the configuration properties should be properly namespace-qualified:

```

<resource oid="b4101662-7902-11e6-9f14-53e18426fe81">
  <name>LDAP</name>
  <connectorRef oid="028159cc-f976-457f-be70-9e9fa079bcf7"/>
  <connectorConfiguration
    xmlns:icfc="http://midpoint.evolveum.com/xml/ns/public/connector/icf-
1/connector-schema-3"
    xmlns:icfclldap="http://midpoint.evolveum.com/xml/ns/public/connector/icf-
1/bundle/com.evolveum.polygon.connector-
ldap/com.evolveum.polygon.connector.ldap.LdapConnector">
    <icfc:configurationProperties>
      <icfclldap:port>389</icfclldap:port>
      <icfclldap:host>localhost</icfclldap:host>
      <icfclldap:baseContext>dc=example,dc=com</icfclldap:baseContext>
      ...
    </icfc:configurationProperties>
  </connectorConfiguration>
  ...
</resource>

```

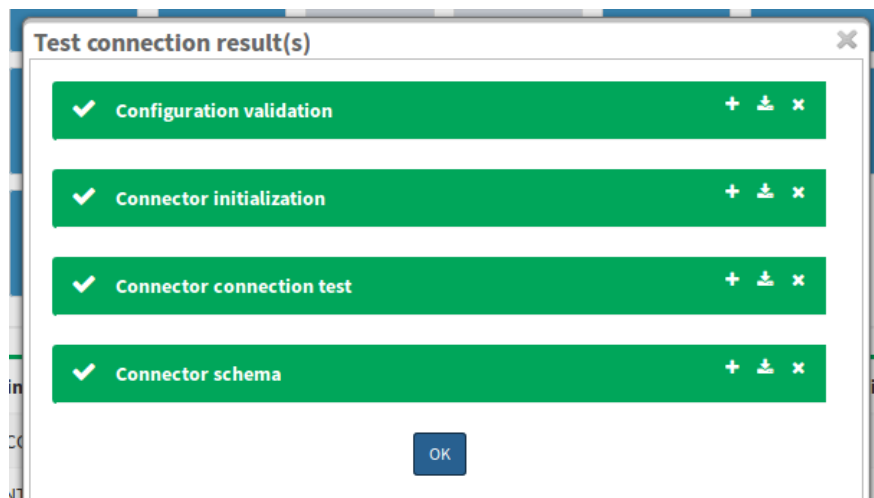
The use of namespaces will be completely optional in later midPoint versions. For now just copy the namespace URIs from the samples. You do not have to completely understand what is going on. Just one thing: the namespace of the configuration properties should be the same as the namespace defined in the connector object. This is a long URI that is composed from connector bundle name and connector name. E.g.

```
http://midpoint.evolveum.com/xml/ns/public/connector/icf-1/bundle/com.evolveum.polygon.connector-ldap/com.evolveum.polygon.connector.ldap.LdapConnector
```

If this namespace does not match then the connector will refuse to work. This is a safety mechanism that prohibits accidental use of configuration from one connector in another connector where the configuration properties may have the same name but a completely different meaning.

Testing the Resource

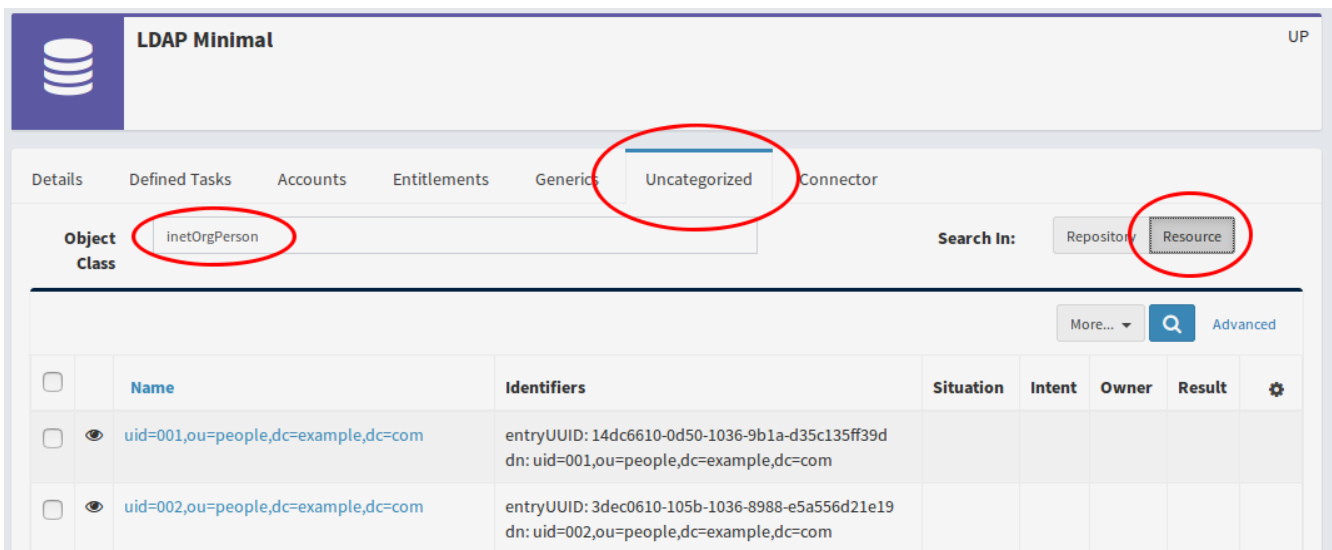
Minimal resource definition has just the name, connector reference and connector configuration properties. If you do this then the resource should show the first signs of life. So now select a suitable sample file. Strip it down to the minimum, modify the parameters and import the resource into midPoint. Now you should see your resource in the list in *Resources > List resources*. The icon next to your resource is most likely black – not green and not red. Green icon means that the resource is working, red icon means that there is an error, black means “I do not know yet”. Now click on the resource label. Resource details page should appear. There is a *Test Connection* button at the bottom of the page. Click on that button. It may take a while now. MidPoint is initializing the connector with the parameters that you have specified. Then the connector will be used to check connection to the resource. If the parameters were correct and midPoint can reach the resource you will see green lights:



If there are any errors during connector initialization, parameters or network connection you will see the errors here. In that case correct the configuration and try again. If everything works well then the resource icon turns green. Now we have a minimal working resource.

There are more things that you can do with such a minimal resource. For example you can look at the resource content. Navigate to the resource details page and switch to *Uncategorized* tab. Select one of the object classes that the resource supports. Just click inside the *Object class* input box and the suggestions will appear. Now click on the *Resource* button on the right side. MidPoint then goes to the resource, lists all the objects

of the given object class and displays the list. Now you can click on any object to see the details.



That is a very useful feature for several reasons. Firstly, you can check that not just the resource connection works, but also that the connector can actually see the objects. Secondly, you will get some idea about the object classes that the resource supports. And thirdly, by looking at several objects you can get basic overview of how the data are structured: what attributes are used and what are the typical values. You will appreciate that information later on when we will be setting up mappings.

Resource Schema Basics

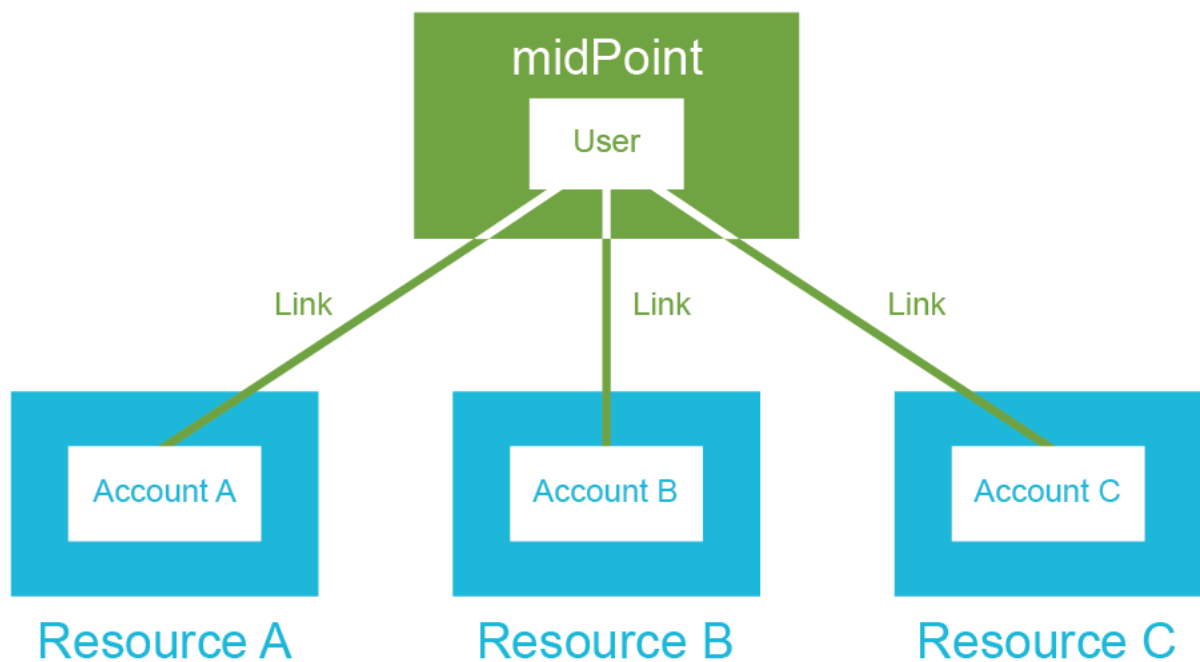
The only resource object that traditional identity management systems dealt with was an *account*. But that is not sufficient any more. Good identity management system needs to manage many different types of resource objects: accounts, groups, organizational units, privileges, roles, access control lists and so on. In midPoint these are the *object classes*: types of resource objects that are made accessible to midPoint by the connector. A typical resource supports at least the *account* object class and it may support more object classes. Each object class may have a completely different set of attributes: different names, different types, some may be mandatory, some optional.

The collective definition of the object classes and their attributes is what we call *resource schema*. Obviously, resource schema is different for every resource. Even resources that are using the same connector may have different resource schema (e.g. two LDAP servers with different custom schema extensions). MidPoint is a smart system and it is capable of automatic resource schema discovery. When the resource is used for the first time midPoint will reach out to the resource and retrieve the schema. Retrieved resource schema is stored under the schema tag in the resource definition object. You can have a look and examine the schema there. But beware, the schema may be quite rich and big.

Resource schema is crucial. MidPoint will use resource schema whenever it needs to work with resource objects such as accounts or groups. MidPoint will use resource schema to validate mappings. The schema will be used for automatic type conversions. And most importantly of all: resource schema will be used to display resource objects in the user interface. MidPoint will adapt to resource schema automatically. Not a single line of custom code is needed to do that.

Hub and Spoke

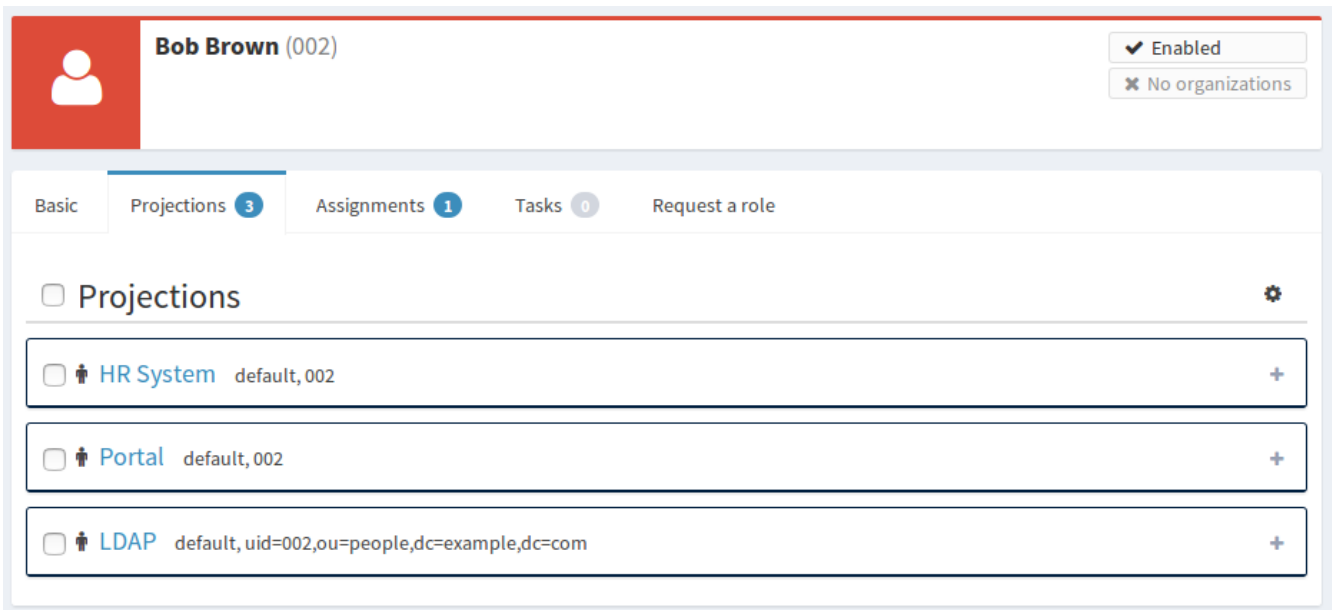
MidPoint topology is a star (a.k.a. “hub and spoke”) with midPoint at the center. This is both physical and logical topology.



This means that the *account A* can be synchronized with *midPoint user* and then *midPoint user* can be synchronized with *account B*. But *account A* cannot be synchronized directly to *account B*. This is deliberate decision that was made very early in *midPoint* design and we have very good reasons for it. But do not be afraid. There is no need to set up a separate synchronization process for every trivial operation on every resource. *MidPoint* is smart enough to take care of that. All you need is to set up the synchronization policies. *MidPoint* will take care of their execution. Just remember that you can synchronize resource(s) with *midPoint*. But not two resources directly.

Accounts and *user* that represent the same person are *linked* together. This *link* is a relation that *midPoint* creates and maintains. Therefore *midPoint* knows who is the owner of a particular account. *MidPoint* also knows which accounts the user has. That is how *midPoint* know which account needs to be synchronized with which user. It is critical that the links are correct otherwise *midPoint* cannot reliably synchronize the data. Therefore *midPoint* takes great care to maintain the links – and that is not always easy. There are

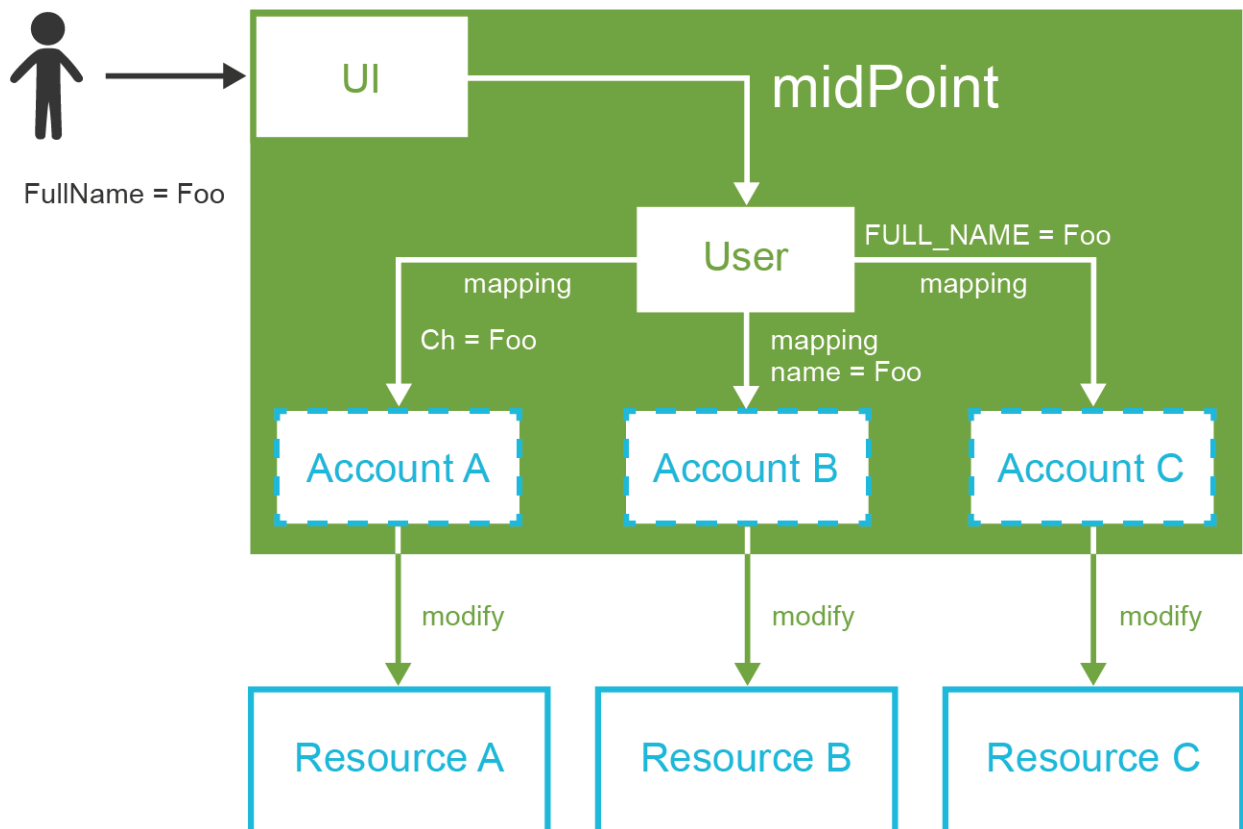
strange corner cases such as renamed accounts, accounts that were deleted by mistake and re-created and so on. But midPoint is built to handle such cases. The links are maintained. And it is the link that allows midPoint to list all user's accounts in the user interface.



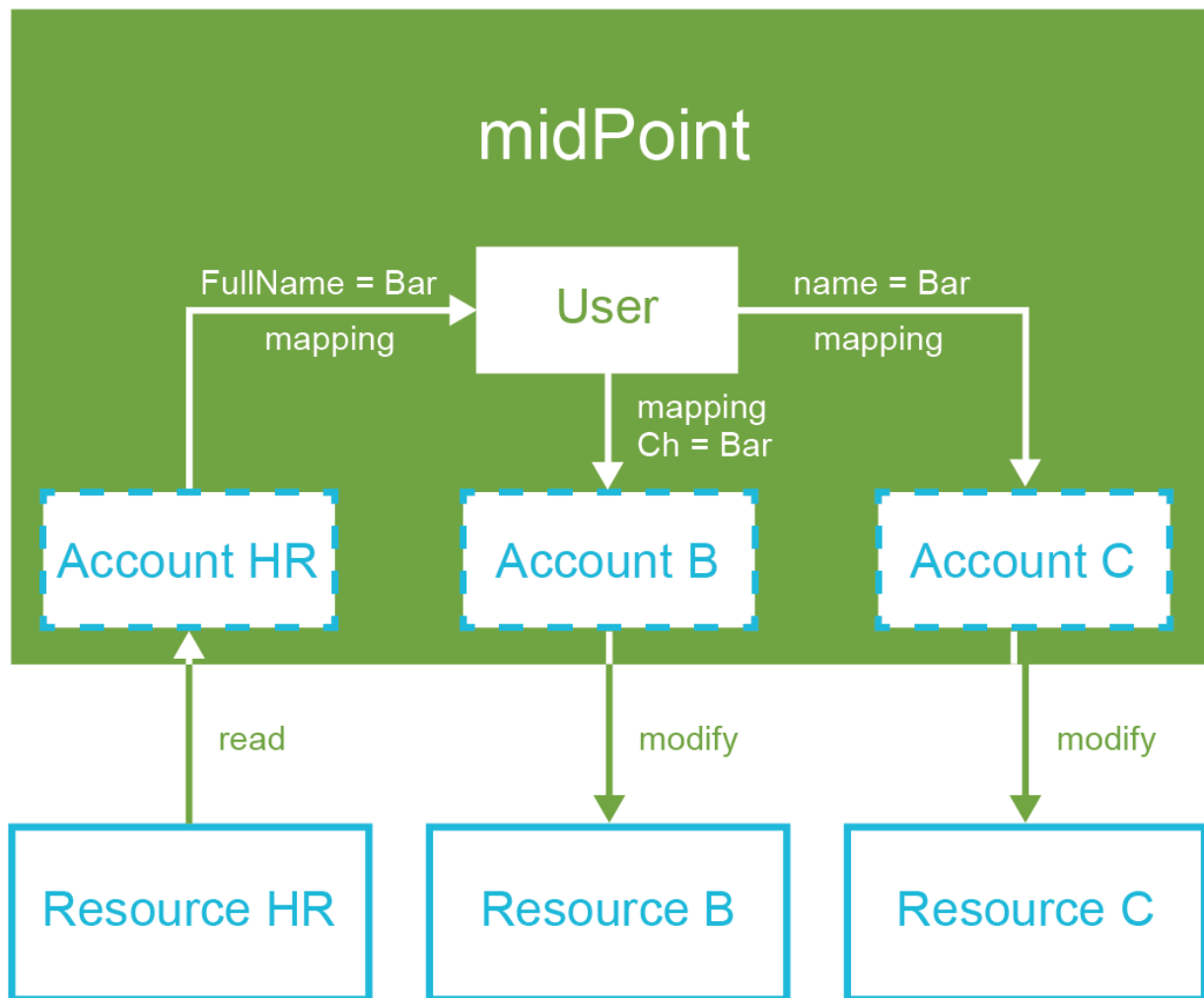
The user in midPoint is known as *focus* in midPoint terminology. The accounts are known as *projections*. You can imagine a light projector that sends many light beams from its focal point to create a projection on the screen. This is the metaphor that we have chosen when building midPoint. And for the lack of better words this terminology remains in use even now. We will get back to the concept of focus and projections many times in this book. For now you just need to remember that *projection* means an *account*.

MidPoint knows which account belongs to which user by following links that it maintains. But how does midPoint know which attributes to synchronize? How to transform the values? And which side is the authoritative one? Mappings take care of that. Mapping is something like a very flexible data replication recipe. MidPoint allows to define mappings for each attribute in any direction. The mappings are used to control the synchronization on a very fine granularity.

Perhaps the best way to summarize the synchronization principles is to use examples. The first example is a modification of user properties in midPoint user interface. When the Save button is pressed then midPoint user interface sends the modification to midPoint core. The synchronization code in the midPoint core follows the links to find all the accounts that belong to this specific user. Then the mappings are applied to synchronize the changed user properties to the accounts. Account changes are propagated to the resources and user changes are stored in the midPoint repository.



The second example is slightly different. In this case the change of account data is the trigger. This may be a change of the employee record in HR system. MidPoint detects that change and reads the new account. MidPoint will use the link to find the user to which the account belongs. Then it will use other links to find all the other accounts that may be affected. Similarly to the previous case the mappings are applied. The mappings from the HR account to the user are applied first. The result is a modification of user properties. Then an process identical to the previous case takes place. The user modifications are automatically applied to all affected accounts.



This cases are very different use-cases for midPoint. First case is a manual change of data by system administrator. Second case is an automatic data feed from the HR system. But as you can see the principles that are used to implement it are almost exactly the same. This is the consequence of midPoint philosophy: radical reuse of functionality, generic application of principles. You just need to define what you want to be done (the policy). MidPoint takes care that it is done when it needs to be done.

Info: Why the star topology? The star or “hub and spoke” were (and still are) the big buzzwords of system integration. And the basic idea makes a lot of sense. If every node needs to be synchronized with every other node then the number of required connections grows quite steeply. It is in fact proportional to the square of the number of nodes. Mathematicians say that is has $O(n^2)$ complexity. However, if you rearrange the connections so that they all point to the central “hub” then the number of connections is significantly reduced. It is proportional to the number of nodes: $O(n)$ complexity. This is a huge difference especially in deployments with many resources. However, this approach works well only if the star topology is both physical and logical. I.e. it makes very little sense to connect all resources to a central “hub” if that hub still internally needs $O(n^2)$

policies to synchronize the data. In that case we have only hidden the complexity in a black box but we have not really solved it. But midPoint is different. MidPoint is a real “hub”. This is the reason why midPoint does not support synchronization of accounts directly with each other. We want to have simple, clean and maintainable system, both externally and internally.

Schema Handling

Resource schema is a very important concept. It defines what object classes the resource supports and how do they look like. We have mentioned that before. But it is not important only to know how the objects look like. It is also important to know what to do with them. And that is what the *schema handling* is all about.

Schema handling is a part of resource definition. It specifies which object classes from the resource schema are actually used by midPoint. And most importantly of all it specifies how they are used. This is the place where mappings are stored. This is the place where account-group associations are defined. This is the place where schema can be augmented and tweaked. Simply speaking, this is the place where most of the resource-related configuration takes place.

Schema handling section contains definition of several *object types*. Each *object type* refers to one “thing” that midPoint will work with: default account, testing account, group, organizational unit and so on. Let’s start with something simple and let’s define just one object type now: default account. It looks like this:

```
<resource oid="b4101662-7902-11e6-9f14-53e18426fe81">
  <name>LDAP</name>
  ...
  <schemaHandling>
    <objectType>
      <kind>account</kind>
      <default>true</default>
      <objectClass>ri:inetOrgPerson</objectClass>
    </objectType>
  </schemaHandling>
</resource>
```

This may seem trivial, but even such minimal definition is important for midPoint. This definition tells midPoint that default account on this resource has `inetOrgPerson` object class. Resources such as LDAP servers may have dozens of object classes. Most of them are not used at all. There are often several alternative object classes that can be used to create accounts. It is important to tell midPoint which object class is the right one. And that’s what this definition does. Once this definition is in place, the accounts appear on the *Accounts* tab of the resource details page (they were visible only on the *Generics* tab before). This is a sign that the definition works correctly.

A clever reader will notice the definition of *kind* in the above example. Setting *kind* to `account` indicates that this object type definition represents (surprisingly) an account. MidPoint supports many types of objects. But two types have a special place: *accounts*

that represents the users and *entitlements* that give privileges to accounts. MidPoint can handle the objects in a smart way if it knows that it is either account or entitlement. And the *kind* definition tells just that. There is also optional *intent* setting that can be used to define subtypes. But more on that later.

The schema handling section can also be used to augment (or even override) some parts of the resource schema. E.g. the following example sets a display name for this object type. The display name will be used by the user interface when it displays the account.

```
<resource oid="b4101662-7902-11e6-9f14-53e18426fe81">
  <name>LDAP</name>
  ...
  <schemaHandling>
    <objectType>
      <kind>account</kind>
      <displayName>Default account</displayName>
      <default>true</default>
      <objectClass>ri:inetOrgPerson</objectClass>
    </objectType>
  </schemaHandling>
</resource>
```

However, the most powerful feature that is used in the schema handling is the ability to Deal with attributes. Next couple of sections is all about that.

Attribute Handling

Resource objects such as accounts or groups are mostly just a bunch of attributes. Almost all of the IDM magic is about setting the correct attribute to the correct value. The schema handling section of the resource definition is the place where the basic attribute behavior is defined.

The object type definition contains sections that define behavior of each attribute that we care about:

```
<resource oid="b4101662-7902-11e6-9f14-53e18426fe81">
  <name>LDAP</name>
  ...
  <schemaHandling>
    <objectType>
      <kind>account</kind>
      <default>true</default>
      <objectClass>ri:inetOrgPerson</objectClass>
      <attribute>
        <ref>ri:dn</ref>
        <!-- behavior of "dn" attribute defined here -->
      </attribute>
      <attribute>
        <ref>ri:cn</ref>
        <!-- behavior of "cn" attribute defined here -->
      </attribute>
      ...
    </objectType>
  </schemaHandling>
</resource>
```

There is an `attribute` section for every attribute that we need. Lot of details can be defined here: display name of the attribute that will be used by the user interface, limitations and schema augmentation and override settings and so on. But the most important things that go there are mappings. In the simplest form a mapping looks like this:

```
<resource oid="b4101662-7902-11e6-9f14-53e18426fe81">
  <name>LDAP</name>
  ...
  <schemaHandling>
    <objectType>
      <kind>account</kind>
      <default>true</default>
      <objectClass>ri:inetOrgPerson</objectClass>
      ...
      <attribute>
        <ref>ri:cn</ref>
        <outbound>
          <source>
            <path>$focus/fullName</path>
          </source>
        </outbound>
      </attribute>
      ...
    </objectType>
  </schemaHandling>
</resource>
```

This means that the value of the `cn` attribute will be taken from the `fullName` property of the focal object (which is typically a user). This is a very simple mapping, there is no value transformation, no condition – nothing complicated at all. This is how a lot of mappings look like. But mappings can be very powerful and complex. And that will be described in the next section.

The attribute sections are typically used to set up the attributes that a typical user account on the resource has. So it will assign the identifiers, set up full name, set description and telephone number attributes and things like that. It is a very convenient approach to have this directly in the resource definition. If we do that then we can simply assign the accounts to the user without specifying any details. MidPoint will use the mappings in this section to populate account attributes with the correct values. Now it is perhaps a good time to have a look at some sample resource definitions to get a feel how a real-world resource definition looks like. The samples are located in the midPoint distribution package or you can find them on-line. See the Additional Information chapter for more details.

Info: The “ri” namespace. You may have noticed that “ri” namespace prefix is used whenever we refer to the object classes or attributes. In a strict sense this is the correct notation. Object classes and attributes are defined in resource schema and the “ri” is the namespace of that schema. While the use of namespaces should be optional in almost all parts of midPoint, we are still using the “ri” namespace in samples. Mostly due to the nostalgic reasons. By the way, “ri” stands for “resource instance”.

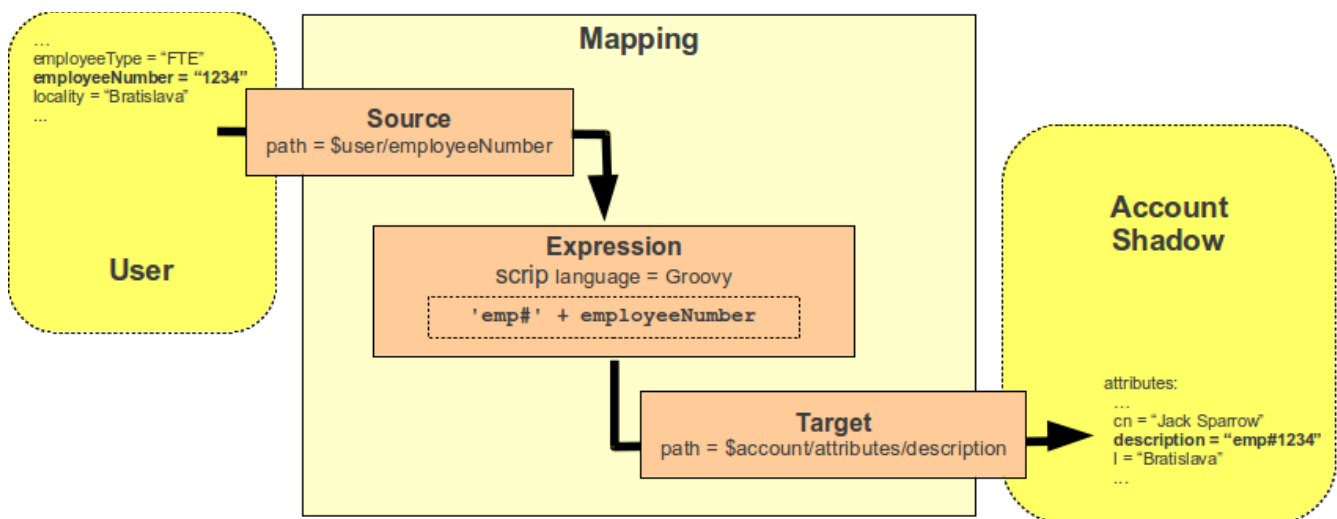
Mappings

Mapping is a very flexible mechanism that takes one or more input properties, transforms them and puts the result in another property. Mappings are used all over midPoint. But perhaps the most important use of mappings is in the schema handling part of the resource definition where they set up account attribute values. We have already seen a very simple mapping that simply copies the values from one place to another. Now it is the time to look at mapping in its entirety.

Mapping consists of the three basic parts:

- **Source** part defines the data sources of the mapping. These are usually understood as mapping input variables. Source defines where mapping gets its data from.
- **Expression** part defines how the data are transformed, generated or passed on to the "other side". This is the most flexible part of the mapping as it contains logic. There is a broad variety of possibilities.
- **Target** part defines what to do with the results of the mapping, where the computed values should go.

The three parts of the mapping as well as the basic principle is illustrated in the following diagram:



The diagram shows simple mapping that takes employeeNumber user property and transforms it to description account attribute by using a simple Groovy script expression.

The source part of the mapping defines that there is a single source which is based on employeeNumber user property. Source definitions are important for the mapping to correctly process relative changes (deltas), mapping dependencies, etc. The source definition tells mapping that the value of employeeNumber user property should be passed to an expression.

The expression part contains a simple Groovy script that prepends the prefix `emp#` to the employee number value specified by the source definition. The expression part of the mapping is very flexible and there is a lot of way that can be used to transform a value, generate new value, use a fixed value, pass a value without any change and so on.

The target part defines how the result of the expression should be used. In this case the result is to be used as a `description` account attribute. The target definition is necessary so the mapping can locate appropriate definition of the target property and therefore make sure that the expression produces a correct data type and that other schema constraints are maintained (e.g. single vs multiple values).

This example mapping can be expressed in XML using the same structure:

```
<mapping>
  <source>
    <path>$focus/employeeNumber</path>
  </source>
  <expression>
    <script>
      <code>'emp#' + employeeNumber</code>
    </script>
  </expression>
  <target>
    <path>$projection/attributes/description</path>
  </target>
</mapping>
```

Not all parts of the mapping are mandatory. If the expression is not present then “*as is*” expression is assumed that just copies the source to target without any change. Some parts of the mapping may be implicitly defined by the surrounding context, e.g. the target or source is implicit if the mapping is used to define attribute behavior in the schema handling section. Therefore for these mappings it is usually sufficient to define either source or target:

```
<attribute>
  <ref>ri:sn</ref>
  <outbound>
    <source>
      <path>$focus/familyName</path>
    </source>
  </outbound>
</attribute>
```

This is almost exactly the notation that you have seen in the previous section. Mapping source is explicitly specified as the `familyName` property of the user. Mapping target is implicitly set to be the attribute for which the mapping is defined. And as no expression is explicitly defined it defaults to a simple copy of the value without any transformation.

In this case the mapping notation can even be shorted a bit more. As it is quite clear that the mapping source will be one of the properties of the focal object (user), then the `$focus` prefix can be omitted:

```
<attribute>
```

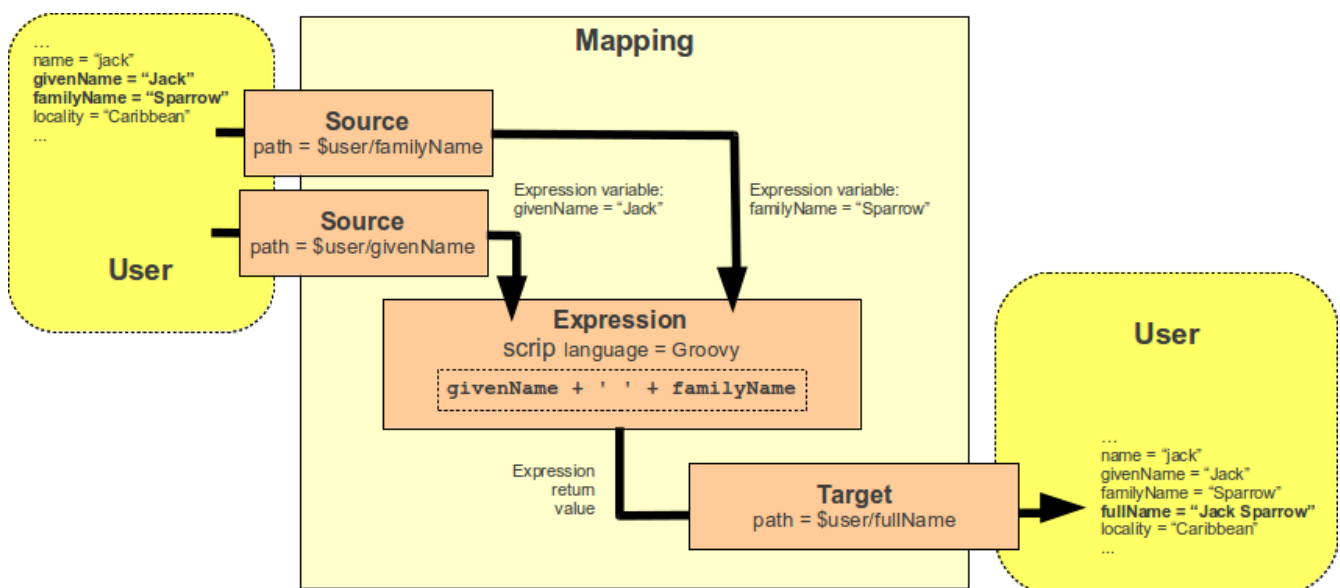


```

<ref>ri:sn</ref>
<outbound>
  <source>
    <path>familyName</path>
  </source>
</outbound>
</attribute>

```

These are a very simple examples. Mappings can do much more – as you will learn later on. But there is one more thing that we need to explain here. Mappings are designed to work with more than just a single source. The following diagram illustrates a mapping that takes two arguments: given name and a family name. The mapping produces a full name by concatenating these value with a space in between. This is the kind of mapping that is frequently used to construct user's full name from its components. While the mapping may seem simple there are some sophisticated mechanisms hidden inside.



The mapping is represented in the XML form as follows:

```

<mapping>
  <source>
    <path>givenName</path>
  </source>
  <source>
    <path>familyName</path>
  </source>
  <expression>
    <script>
      <code>givenName + ' ' + familyName</code>
    </script>
  </expression>
  <target>
    <path>fullName</path>
  </target>
</mapping>

```

There are two sources specified by the source definitions: user properties `givenName` and `familyName`. The mapping is using a *script expression* to combine the values into a single value which is used to populate user's `fullName` property.

This example also illustrates that the mappings are smart. The mapping will be evaluated only if one of the sources changes or if a full recompute is requested. In case that neither givenName nor familyName changes there is no need to re-evaluate that expression. This is the primary reason for requiring explicit source definition in the mappings. Without such definitions it is not (realistically) possible to reliably determine when and how the expression should be re-evaluated.

Info: \$user and \$account variables. The variable \$focus and \$projection were introduced in midPoint 3.0 as a consequence of the *generic synchronization* feature. The objects that the expression works with might not be just user or account. A much broader range of objects may be used there. Therefore generic concepts of *focus* and *projections* were introduced and the variable names were changed to reflect that. The old variables \$user and \$account can still be used, but their use is deprecated. And they are still used in some older examples.

Mappings are used in many situations. Sometimes the mapping needs to be really authoritative. It has to enforce the value to the target. But sometimes we want only to provide a default value and the mapping should never change the target value once it is set. Therefore mapping can be set to various *strength*: from weak to strong. Following table describes how that works:

Strength	Description
weak	Applied only if the target has no value. Weak mappings are usually used to set <i>default values</i> .
normal	Apply mapping only if there is a change in the source properties. Normal mappings are used to implement the <i>last change wins</i> strategy. If the value was modified in midPoint then the mapping will be applied and target will be modified. If the target is modified directly then the mapping will not overwrite it – until the next change in midPoint. This is the default setting. If no strength is specified then <i>normal</i> is assumed.
strong	Always applied. Strong mappings <i>enforce</i> particular values.

The strength can be specified in any mapping by using the strength tag:

```
<attribute>
  <ref>ri:sn</ref>
  <outbound>
    <strength>strong</strength>
    <source>
      <path>$focus/familyName</path>
    </source>
  </outbound>
</attribute>
```

When it comes to the mapping strength then the following rule of the thumb may be useful: If you want to enforce policy use *strong* mappings. If you just want to set a default value

use *weak* mapping. If you are not sure what you are doing then *normal* mappings will probably work just fine.

Expressions

Expression is the most flexible part of the mapping. There are approximately dozen different type of expressions ranging from the simplest *as is* expression through the *scripting expressions* all the way to a special purpose expressions that search through midPoint repository. The expression type that is used is determined by the tag that is used inside the expression section of the mapping. We call them *expression evaluators*. You can find the detailed description of expression evaluators in midPoint Wiki. Now we are going to deal only with few types that are most frequently used:

Expression Evaluator	Tag	Description
As is	asIs	Copies the value without any transformation.
Literal	value	Stores literal (constant) value in the target.
Generate	generate	Generates a random value.
Script	script	Executes a script, stores script output in the target.

The simplest expression evaluator is `asIs`. It simply takes the source and copies that to the target. It obviously works only if there is just one source. It is also the default expression evaluator. If no expression is specified in the mapping then `asIs` is assumed. It is used like this:

```
<attribute>
  <ref>ri:sn</ref>
  <outbound>
    <source>
      <path>familyName</path>
    </source>
    <expression>
      <asIs/>
    </expression>
  </outbound>
</attribute>
```

The literal expression is used to place a constant value in the target. This expression does not need any source at all. It always produces the same value. It looks like this:

```
<attribute>
  <ref>ri:o</ref>
  <outbound>
    <expression>
      <value>ExAmPLE, Inc.</value>
    </expression>
  </outbound>
</attribute>
```

The generate expression is used to generate a random value. As such it is used almost exclusively to generate passwords. We will deal with that expression later when we will be dealing with credentials.

Script Expressions

The most interesting and also the most flexible expression evaluator is undoubtedly the script expression evaluator. It allows to execute arbitrary scripting code to transform the value. The basic principle is simple: values from source properties are stored in the script variables. Script is executed and it produces output. The output is stored in the target.

We have already seen a mapping that has a scripting expression:

```
<mapping>
  <source>
    <path>givenName</path>
  </source>
  <source>
    <path>familyName</path>
  </source>
  <expression>
    <script>
      <code>givenName + ' ' + familyName</code>
    </script>
  </expression>
  <target>
    <path>fullName</path>
  </target>
</mapping>
```

There are two sources: givenName and familyName. The values of these user properties are placed in the scripting variables that have the same names: givenName and familyName. Then the script may do whatever it does with the variables. At the end the script has to return a value. The script above is written in Groovy, therefore the return value is the value of the last evaluated expression. In this case it is the only expression in the script which concatenates the two variables with a space in between. Script return value is placed in the output, which in this case is the fullName user property.

Scripts are often used to transform the values before they are stored in the attributes. One very common case is the construction of LDAP distinguished name (DN). The DN is a complex value in the form of uid=foobar,ou=people,dc=example,dc=com. However it is easy to construct such value using a simple script:

```
<attribute>
  <ref>ri:dn</ref>
  <outbound>
    <source>
      <path>name</path>
    </source>
    <expression>
      <script>
        <code>
          'uid=' + name + ',ou=people,dc=example,dc=com'
        </code>
      </script>
    </expression>
  </outbound>
</attribute>
```

```

        </script>
      </expression>
    </outbound>
  </attribute>

```

(A clever reader surely has a suspicious look on his face now. Yes, this is not entirely correct way how to compose LDAP DN. But please bear with us. We will correct that later.)

Midpoint supports three scripting languages:

- **Groovy:** This is the default scripting language.
- **JavaScript (ECMAScript)**
- **Python**

All three languages can be arbitrarily mixed even in a single midPoint deployment. Although quite understandably such a practice is not recommended. The language can be selected for each individual expression by using the language URI:

```

      <expression>
        <script>
          <language>http://midpoint.evolveum.com/xml/ns/public/expression/
language#python</language>
          <code>
            "Python is %s, name is %s" % ("king", name)
          </code>
        </script>
      </expression>

```

When writing scripting expression please keep in mind that some characters must be properly escaped in the text format that you are using (XML, JSON or YAML). E.g. the ampersand character so frequently used for logical operations needs to be escaped as `&` in XML.

The scripting expressions can do almost anything. And there is still more to them. This section provided only the very basic description to get you started. Will get back to the scripting expressions many times in this book.

Activation

The term *activation* is used in midPoint to denote a set of properties that describe whether the user is active. This includes properties that describe whether the user is enabled, disabled, archived, since when he should be enabled, to what date he should be active and so on. The simple enabled/disabled flag might be sufficient in 1990s. But now we need much more than that. Therefore the *activation* is quite a rich data structure in midPoint. We will get into the details later. We will describe just the basic idea now.

The most important activation concept is the *administrative status*. Administrative status defines the "administrative state" of the object (user). I.e. the explicit decision of the administrator whether the user is enabled, disabled or archived. Except for administrative

status there are also validity times, lockout status, various timestamps and metadata. But we will come to that later.

The important thing to realize is that both user and the accounts have activation properties – and they are almost the same. The user and account activation is using the same property names, meaning and data formats. This is important, because you would probably want account activation to follow user activation. E.g. if user is disabled then also all his accounts are disabled. This is very easy to do in midPoint because the user and account activation are compatible. Therefore all it takes is a simple mapping. There is a special place in the resource schema handling section for that:

```
<resource oid="b4101662-7902-11e6-9f14-53e18426fe81">
  <name>LDAP</name>
  ...
  <schemaHandling>
    <objectType>
      <kind>account</kind>
      <default>true</default>
      <objectClass>ri:inetOrgPerson</objectClass>
      <!-- attribute handling comes here --->
      <activation>
        <administrativeStatus>
          <outbound/>
        </administrativeStatus>
      </activation>
    </objectType>
  </schemaHandling>
</resource>
```

It is as simple as that. Nothing more is needed. User has `administrativeStatus` property, account has `administrativeStatus` property, so midPoint knows what is the source and target of the mapping. The values of the `administrativeStatus` property are also the same on both side. Therefore the default `asIs` mapping is just fine. All that midPoint needs to know is that the mapping exists at all. That we want to map the value. MidPoint will fill in all the details.

When this mapping is in place and the user is disabled, the account will be disabled as well. When the user is enabled, the account will also follow suit.

Credentials

Credential management is important part of identity management. MidPoint is built to easily synchronize credentials to many accounts. Similarly to activation, credential data structures of user and accounts are also aligned. Therefore all that is needed to synchronize the password is a simple mapping:

```
<resource oid="b4101662-7902-11e6-9f14-53e18426fe81">
  <name>LDAP</name>
  ...
  <schemaHandling>
    <objectType>
      <kind>account</kind>
      <default>true</default>
```

```

        <objectClass>ri:inetOrgPerson</objectClass>
        <!-- attribute handling comes here --->
        <credentials>
            <password>
                <outbound/>
            </password>
        </credentials>
    </objectType>
</schemaHandling>
</resource>

```

When the user password in midPoint is changed the changed password will be propagated to all the resources that have a mapping like this.

Complete Provisioning Example

This section describes a complete working example of connection to the LDAP directory. The configuration below is used to automatically create accounts in the OpenLDAP server. The entire configuration is contained in a single resource definition file. Following paragraphs explain individual parts of the file. Simplified XML notation is used for clarity. The complete file in a form directly usable in midPoint can be found at the same place as all the other samples in this book (see Additional Information chapter for details).

Resource definition begins with object type, OID, name and description. These are self-explanatory:

```

<resource oid="8a83b1a4-be18-11e6-ae84-7301fdab1d7c">

    <name>OpenLDAP</name>

    <description>
        LDAP resource using a ConnId LDAP connector. It contains configuration
        for use with OpenLDAP servers.
        This is a sample used in the "Practical Identity Management with
MidPoint"
        book, chapter 4.
    </description>
    ...

```

Connector reference comes next. We want to point to the LDAP connector. Here we use dynamic reference that is using search filter to locate the connector:

```

...
    <connectorRef type="ConnectorType">
        <filter>
            <q:equal>
                <q:path>c:connectorType</q:path>
                <q:value>com.evolveum.polygon.connector.ldap.LdapConnector</q:va
lue>
            </q:equal>
        </filter>
    </connectorRef>
    ...

```

When this object is imported the reference is resolved. The resolution process will use the search filter and it will look for connector object with the connectorType specified in the filter.

The connector configuration goes next. This block specifies connector configuration properties such as hostname, port, passwords and so on.

```
<connectorConfiguration>
  <icfc:configurationProperties>
    <icfcldap:port>389</icfcldap:port>
    <icfcldap:host>localhost</icfcldap:host>
    <icfcldap:baseContext>dc=example,dc=com</icfcldap:baseContext>
    <icfcldap:bindDn>cn=idm,ou=Administrators,dc=example,dc=com</icfcldap:bindDn>
    <icfcldap:bindPassword><t:clearValue>secret</t:clearValue></icfcldap:bindPassword>
    <icfcldap:passwordHashAlgorithm>SSHA</icfcldap:passwordHashAlgorithm>
  >
    <icfcldap:vlvSortAttribute>uid,cn,ou,dc</icfcldap:vlvSortAttribute>
    <icfcldap:vlvSortOrderingRule>2.5.13.3</icfcldap:vlvSortOrderingRule>
  >
    <icfcldap:operationalAttributes>memberOf</icfcldap:operationalAttributes>
    <icfcldap:operationalAttributes>createTimestamp</icfcldap:operationalAttributes>
  </icfc:configurationProperties>
  <icfc:resultsHandlerConfiguration>
    <icfc:enableNormalizingResultsHandler>false</icfc:enableNormalizingResultsHandler>
    <icfc:enableFilteredResultsHandler>false</icfc:enableFilteredResultsHandler>
    <icfc:enableAttributesToGetSearchResultsHandler>false</icfc:enableAttributesToGetSearchResultsHandler>
  </icfc:resultsHandlerConfiguration>
</connectorConfiguration>
```

The last part of this block defines that the Connld framework result handlers should be disabled. The Connld result filtering is a legacy mechanism and most connectors do not need that any more. It is even harmful in many cases. Unfortunately this mechanism is turned on by default. Therefore most resource configurations contain this part to explicitly turn all the handlers off.

These parts alone should already define a minimal resource. If you define just the name, connector reference and connector configuration you should be able to import the resource to midPoint. The connection test should also pass and you should be able to browse resource content. However there is absolutely no IDM logic or automation yet. That is what we are going to add next.

The next element that usually follows connector configuration is schema. However if you look at almost any file that contains resource definition you will find no such element. The schema element is automatically generated by midPoint when midPoint connects to the resource for the first time. Therefore there is no need to include it in the definition.

What we have to include in the definition is the way how midPoint handles the schema. This is defined in the `schemaHandling` section. The `schemaHandling` section contains just one `objectType` definition. We are going to define how to handle ordinary user accounts on our OpenLDAP server.

```
...
<schemaHandling>
  <objectType>
    <kind>account</kind>
    <displayName>Normal Account</displayName>
    <default>true</default>
    <objectClass>ri:inetOrgPerson</objectClass>
  ...

```

This is where we define the *kind* of objects that we are going to handle. In this case it is `account`. This is *default* account. Which means that if the account type is not explicitly specified then this definition will be used. There is also specification of a display name. This is not used in the automation logic. It is used by the user interface when referring to this definition. And finally there is specification of the *object class*. The `inetOrgPerson` object class will be used to create new accounts. The object class specification determines what attributes the account can have.

The `objectType` definition also includes the specification of attribute handling. There is one section for each attribute that we want to handle in automated or special way. It starts with the most important attribute: LDAP distinguished name (DN):

```
...
<attribute>
  <ref>ri:dn</ref>
  <displayName>Distinguished Name</displayName>
  <limitations>
    <minOccurs>0</minOccurs>
  </limitations>
  <outbound>
    <source>
      <path>$focus/name</path>
    </source>
    <expression>
      <script>
        <code>
          basic.composeDnWithSuffix('uid', name,
'ou=people,dc=example,dc=com')
        </code>
      </script>
    </expression>
  </outbound>
</attribute>
...

```

The `ref` element specifies the name of the attribute that we are going to work with. In fact this is the reference to the automatically-generated schema part of resource definition. The definition of display name follows. This name will be used by the user interface as a label for the fields that work with this attribute. When we define it like this then a nice

“Distinguished Name” label will be used instead of cryptic “dn” which would be used by default.

Let's skip the limitation definition now. We will come back to that later.

The outbound mapping definition follows. This is where the automation logic is specified. This is the place where the DN value is computed. The name property of the user object is the source for this mapping. The name property usually contains username (login name). This value is used by the expression in this mapping. The expression is supposed to create a DN in the form:

```
uid=username,ou=people,dc=example,dc=com
```

The expression here is clever. It does do the work itself. It invokes a library function to compose the DN. It may look like a good idea to use simple string concatenation to construct a DN. However that will fail in case that the DN components contain certain characters that need to be escaped in the final DN. The `composeDnWithSuffix` library function takes care of that and creates a proper DN.

The outbound mapping will be evaluated whenever we need to construct a DN. This obviously happens when a new object is created. But the same mapping will be also used when a user is renamed (his username changes). This is the reason that the mapping needs specification of source. Rename is often quite tricky and complicated operation. It may not be cheap and in some applications it may not be entirely reversible. We definitely do not want to trigger DN changes unless they are really needed. The specification of the mapping source tells us *when* the DN change is needed. In this case it tells us to change the DN in the name property of the user object changes.

Now it is the right time to go back to the `limitations` section. The `dn` attribute is defines as mandatory attribute by the schema. And that is perfectly correct: LDAP object cannot be created without a DN. `MidPoint` is using schema for everything. Therefore when `midPoint` displays a form to edit this LDAP account it will check that DN has a value. It is a mandatory attribute. However, normally we do not want users to enter the DN in the form. We want to compute it automatically. And that is exactly the point of the outbound mapping. Yet `midPoint` does not know when the expression computes a value and when it does not. The expression is a generic piece of Groovy code. As far as `midPoint` can see the expression can do anything. Therefore even if there is an expression `midPoint` will stick to the schema and it will still require the DN to be entered by the user. However we have written the expression and we know that it will produce a value for any (reasonable) input. Therefore we want to tell `midPoint` that the DN is no longer mandatory – that it is OK for user to enter do DN in the GUI forms. And that is exactly what the `limitations` section does. This section overrides the automatically generated schema and it turns the `dn` attribute from mandatory to optional.

And that is all. Now we have defined the behavior of the `dn` attribute. We can use similar approach to define the behavior of other attributes as well. E.g. the handling of the `cn` attribute has similar definition:

```
...
<attribute>
  <ref>ri:cn</ref>
  <displayName>Common Name</displayName>
  <limitations>
    <minOccurs>0</minOccurs>
  </limitations>
  <outbound>
    <source>
      <path>$focus/fullName</path>
    </source>
  </outbound>
</attribute>
...
```

In this case there is outbound mapping, but no explicit expression. If there is no expression then the value is taken from the source without any change (“as is”). Therefore the attribute `cn` will have the same value as the property `fullName` of the user.

It is also possible to define an attribute without any mapping:

```
...
<attribute>
  <ref>ri:entryUUID</ref>
  <displayName>Entry UUID</displayName>
</attribute>
...
```

This means that `midPoint` will not provide any automatic handling for the `entryUUID` attribute. This definition is used just to set a user-friendly display name for the attribute.

The mapping and expressions have almost unlimited flexibility. E.g. the following definition sets a static value for the `description` attribute:

```
...
<attribute>
  <ref>ri:description</ref>
  <outbound>
    <strength>weak</strength>
    <expression>
      <value>Created by midPoint</value>
    </expression>
  </outbound>
</attribute>
...
```

The mapping has no source because the source does not make any sense for static value. They are always the same. You can also notice that this mapping is *weak*. It will be used to set the `description` attribute only if that attribute does not have any value. It will not overwrite existing values.

The `inetOrgPerson` object class has much more attributes than those defined in the `schemaHandling` section. Those attributes will be automatically displayed in the user interface. MidPoint will use the generated resource schema to determine their names and types. MidPoint will display these attributes, the user can change them, midPoint will execute those changes but apart from that midPoint will not do any special handling on those attributes. It is all right not to enumerate all the attributes in `schemaHandling` section. You only need to define those attributes which you want to handle in a special way.

There are two more definitions to describe before our example is complete. First definition is the *activation* definition. It is very simple:

```
...
<activation>
  <administrativeStatus>
    <outbound/>
  </administrativeStatus>
</activation>
...
```

This is a definition that specifies handling of the activation administrative status. This status property specifies whether account is enabled or disabled. Activation properties are somehow special in midPoint. MidPoint understands the meaning and the values of activation properties. MidPoint also expects that user activation and account activation will be usually mapped together. Therefore it is enough to tell midPoint that you want such mapping. MidPoint already knows the source (user activation) and the target (account activation). If the user is disabled then the account will be also disabled. If the user is enabled then the account will also be enabled.

Note: Clever reader is surely scratching his head now. There is no LDAP standard that specifies how to enable or disable accounts. In addition to this OpenLDAP does not even have a concept of disabled account. Therefore how does midPoint know how to disable an LDAP account? To tell the truth midPoint does not know it. We have taken a bit of a poetic license here as we wanted to demonstrate the simple activation mapping. But in this case it will not work just by itself. OpenLDAP resource simply does not have this *capability*. However there is a way. This capability can be simulated. We will deal with that later. For now let's just marvel in the beauty of this activation mapping that does absolutely nothing.

For the resource to work well we also need to define a mapping for *credentials*. In this case it is a password mapping:

```
...
<credentials>
  <password>
    <outbound/>
  </password>
</credentials>
...
```

Similarly to activation also the credentials are special in midPoint. MidPoint understands how credentials work, what are their values and so on. MidPoint also expects that user credentials such as passwords will be normally mapped to the account credentials. Therefore all that midPoint needs to know is that you want that mapping. It can automatically determine the source and target. The account will have the same password as the user. User's password is used when a new account is created. And when user changes his password the change is also propagated to the account.

And that is all. Now you have your first (almost) working resource. You can import the definition to midPoint and test it. Simply assign the resource to a user. The OpenLDAP account will be created – the DN and all the essential attributes will be automatically computed. When midPoint creates an account it remembers it. Therefore it can be easily deleted. Unassign the resource and the account will get deleted. This is how automated provisioning and deprovisioning works. No real programming was needed. Just a declarative specification and a line or two of very simple scripting. This is all configuration that can be done in a couple of minutes. And this is essentially the same process for all the applications. Just the connector is different, it has different configuration properties, the attribute names are different – but the principles and the tools are the same. It is easy to connect many heterogeneous applications in this way. The connectors and the mappings are hiding the differences. In the end all the “resources” look the same to midPoint. The same principles are used to manage them. Therefore the management can be done efficiently even at a large scale.

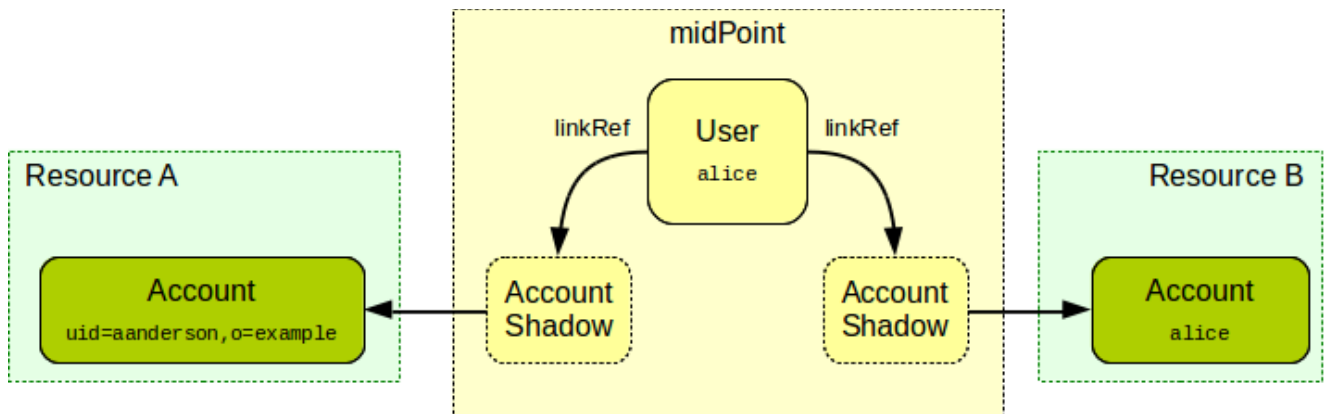
Yet this configuration is still extremely basic. We are just scratching the surface of what midPoint can do. There is much more to see in next chapters.

Shadows

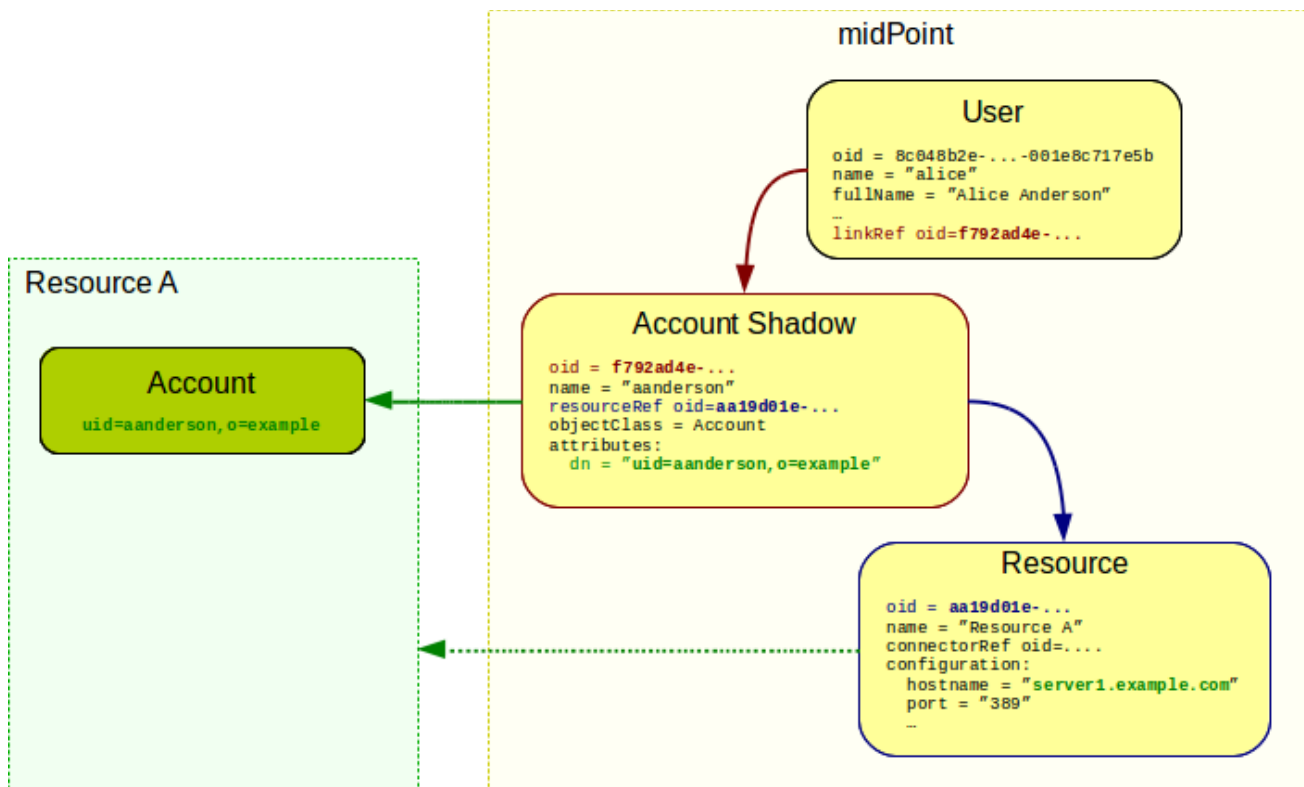
Linking users and accounts is one of the basic principle of any decent identity management system. However it is surprisingly difficult to implement this link. There are numerous methods how to reliably identify accounts and they significantly vary from system to system. Some system can identify the accounts only by username – which makes reliable detection of renames quite difficult. Other systems improve on that by introducing another identifier that is persistent. It is assigned by the resource and it never changes. However, username is still used as secondary identifier and it has to be unique. Yet another system has compound identifiers that consists of two or more values. Some system have globally-unique identifiers while other systems have identifiers that are only unique in their own object class. Some systems have hierarchical structured identifiers, others have flat unstructured identifiers. Some identifiers are case-sensitive strings, others are case-insensitive and some identifiers are even binary. And so on and so on. To make long story short: reliable identification is really complicated.

We do not want to pollute user object with all the delicate details of account identification. Therefore we have created a separate midPoint object that hides all the resource-related details and identification complexities. We call it simply *shadow* because it behaves as a

shadow of the real account. When midPoint detects new account a new *shadow* is created for it. When midPoint detects that account has changed (e.g. it was renamed) then midPoint automatically updates the shadow. When the account is deleted midPoint also deletes the shadow. Yet technically shadow is still an ordinary midPoint object. Therefore it has object identifier (OID). Other objects can simply point to the shadow using ordinary object reference. And that is exactly how user-account links are implemented:



The *shadow* objects contain all the information that is needed to reliably identify the account – or any other resource object such as group or organizational unit. They also point to the corresponding resource definition to make the identification complete. The shadows are multi-purpose objects. Shadows also have other uses in midPoint. They record meta-data about resource objects. They shadows are used to hold cached values of the attributes (this functionality is still experimental in midPoint 3.5). And in the near future the shadows will be used to hold the state of the resource objects for which midPoint does not have on-line communication channel (connector). Therefore the shadows are quite complex objects. The following picture provides a deeper example of a shadow.



If that picture looks a bit scary then do not worry. Shadows may be complex but they are almost invisible to the midPoint user. Shadows are maintained by midPoint automatically. Under normal circumstances MidPoint takes all that is needed to maintain the shadow and no special configuration is needed for that. We describe the mechanics of the shadow objects here mostly for the sake of completeness. Also there may be situations when this knowledge may be useful. These are usually situations when midPoint was mis-configured and the shadows were created incorrectly. In that case you may need to purge all shadows and start over. But beware. Shadows are used to link users and accounts therefore if you purge the shadows you will lose the links. But even that is usually not critical. The synchronization methods described in the next chapter may be used to easily re-create the links.

Chapter 6: Synchronization

It is a capital mistake to theorize before one has data.

*– Sherlock Holmes
(The Adventures of Sherlock Holmes
by Arthur Conan Doyle)*

Data are the lifeblood of any software system. Managing the data is one the primary responsibilities of a software architect. But data management can be very tricky – as any experienced software architect knows only too well. One of the important principle of software architecture is often formulated as “do not repeat yourself”. The data should not be repeated. There should not be copies of the data. If there are no copies then there is one universal source of truth. If there are no copies the data are always consistent. No copies means no contradictions. Just one truth, precise and crystal-clear. Keep data in one place and one place only. That is the theory.

However, practice has a different opinion to offer. In practice there are many incompatible technologies. Applications built on relational databases cannot directly use data from directory services. Even relational databases do not fit together easily. Each application is designed with a different data model in mind. There are data translation and bridging technologies that work as an adapters to resolve compatibility issues. But there is a cost to pay. The data bridges add latencies and they almost always have a major negative impact on performance. Transaction handling and consistency is very problematic. The adapters are additional components on a critical path and they failures are very painful. The resulting systems are often operationally fragile: failure of even a minor component means a failure of the entire system. Not to speak about the enormous complexity and cost of the solution.

On the other hand copying all the necessary data into my application is so convenient. The application can access it easily, using just one homogeneous mechanism. Failure of other components are not affecting the critical path. And it is all so much better for performance. Copying the data solves almost all of the troublesome issues. Except for one small detail: keeping the data up to date. And that is where the synchronization mechanisms come in.

Several copies of the data are unavoidable in practice. Identity data are no exception. In fact identity data are often the most heavily affected by copying. And it makes a lot of sense. Almost every application keeps some kind of data about users. And such data are usually very sensitive from security and privacy point of view.

We cannot avoid copying the data. The best thing that we can do is to keep the copies synchronized. Some applications have built-in support for LDAP or directory synchronization. But those mechanisms are too often inferior. For example many

applications provide capability for on-demand synchronization with directory service on login time. It usually works like this:

1. User enters username and password to application login dialog.
2. The application connects to the directory service to validate the password.
3. If the password is correct then the application retrieves user data from the directory.
4. The application stores copy of user data locally.
5. Business as usual. Local copy of the data is used ever since.

This approach works quite well at the beginning. But after a while the data begin to stink. Users are renamed, but the local copies are not updated. Users are deleted, but the local copies stay around forever. There are local accounts and privileges that are not reflected back to the directory service and therefore remain undetected for years.

Some applications have more advanced synchronization processes that can do better than this. But seeing an application that does this well is an extremely rare sight. And there is a good reason for this. Synchronization much harder than it seems. There may be data inconsistencies on both sides. There may be networking errors. Configuration errors. The data models are evolving in time. The policies are changing. This is no easy task to reliably synchronize the data. Therefore there is a special breed of systems that specialize in synchronization of identity data: identity management systems.

Synchronization in MidPoint

Synchronization is itself one of the basic midPoint principle. Synchronization mechanisms are integral part of midPoint design from its very beginning. Many of the things that midPoint normally does are in fact synchronization. There are obvious cases, e.g. reconciliation process will synchronize the account attributes on resource with the midPoint user properties. But there are also less obvious cases. Such as a completely ordinary provisioning case when midPoint needs to create a new account for a user. But also that case is in fact a synchronization: midPoint user properties are synchronized with a new empty account on the resource. Majority of midPoint operations are directly or indirectly using the synchronization mechanisms.

Info: Reuse of the mechanisms is one of the basic midPoint principles. When we have designed midPoint we have not invented a separate principle for every midPoint feature. We have rather designed few very generic principles that are re-used at many places in midPoint. Synchronization is one of these principles. There is one code that implements the core of the synchronization logic. And that code is used whenever we need to “align” objects that relate to each other. The same code is used for *user-account* reconciliation, ordinary provisioning, role-based provisioning, live synchronization, consistency ... almost everywhere.

MidPoint synchronization is almost a continuous functionality spectrum that can be tweaked and tuned to specific needs. Yet, it can be divided to several broad and overlapping categories:

- **Live synchronization** is almost real-time synchronization mechanism. MidPoint continually scans the resource for changes. If changes are detected then those changes are immediately processed by midPoint. The usual latencies depend on the capabilities of the resource but usual numbers range from few seconds to few minutes. Only the changes are processed by live synchronization. Therefore it is a very efficient mechanism that has fast responses even in large-scale deployments.
- **Reconciliation** is a process that compares the data and corrects them. When an account is reconciled midPoint computes the attribute values that the account should have. The computed values are compared to the real values that the account has. Any differences are corrected. Reconciliation is quite heavy-weight mechanism, comparing all the accounts one-by-one. But it is also a very reliable mechanism. It can correct mistakes that were missed by live synchronization, it can correct data after major failures and corruptions and so on. Reconciliation is usually executed in regular intervals. However due to its nature it is usually executed during off-peak times (nights and weekends).
- **Import** is usually a one-time process to get data from the resource to midPoint. Import is used to populate midPoint with initial data or it may be used to connect a new resource to midPoint. Import is almost the same as reconciliation with only a few minor differences. However its purpose is different and therefore there is usually also a slightly different configuration of import policies (mappings). Import is usually not scheduled, it is manually triggered when needed.
- **Opportunistic synchronization** is a very special kind of animal which is quite unique to midPoint. Opportunistic synchronization is triggered automatically when midPoint discovers that something is not in order. For example if midPoint tries to modify an account, but it discovers that the account is not there. Then a synchronization mechanism is triggered just for that single account. Which usually means that the account is re-created. The opportunistic synchronization is also triggered when midPoint tries to create a new account, but the account is already there. This approach makes midPoint a self-healing system. If midPoint runs into a problem it can often correct the problem by itself.

Individual mechanisms differ in a way data inconsistency is discovered: livesync will actively look for new changes, reconciliation will compare the data one-by-one and opportunistic synchronization will discover inconsistency by pure chance. But all the mechanism react to inconsistency in the same way. There is only one policy that specifies how to fix the system. Of course, there may be slight deviations in the behavior. For example we usually want *import* to behave in slightly different way than *reconciliation*. And midPoint allows that. But overall there just one big scheme how to do synchronization. And

this has a good reason. It does not really matter how the problem was discovered. What really matters is that it gets fixed. And we do not want to maintain four separate configuration for that. Having one policy is enough. MidPoint knows which part of the configuration need to be applied in each specific situation. And it does so automatically. This unifying approach significantly simplifies the configuration of midPoint synchronization mechanisms. And that is also a reason why the boundaries of individual synchronization mechanisms are quite fuzzy. In fact this is just a single big mechanism with several facets.

Sources, Targets And Other Creatures

The idealistic identity management deployment usually starts with a human resources (HR) system. The HR system has records for all the identities therefore it is *authoritative source* resource. Identity management system pull in all the data from HR, recomputes them and create accounts on *target resources*. And they lived happily ever after.

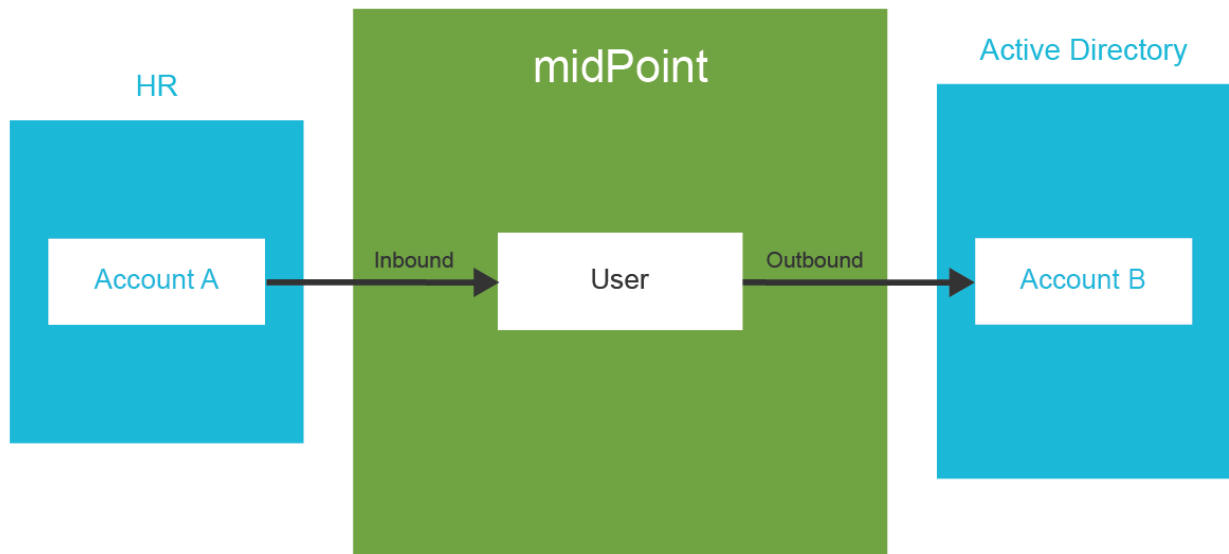
Now, let's get back to reality. The HR resource is indeed an authoritative source in most cases. But it is a limited source. It contains only data about employees. And it has only a partial information. For example there is not username, this has to be generated by the logic. There is no initial password. Organizational structure assignment is often incomplete, missing or unreliable. Therefore it is only a *partially-authoritative source*. There may be additional authoritative sources for contractors, partners, suppliers, support engineers and other identities that need to access our systems. These are *additional source systems*. Then there is a directory system which is often Active Directory. This should be a *target resource* in theory. But there usually there are pieces of authoritative information in here. For example an algorithm to generate a username may be based on the usernames that are already taken in the Active Directory. The active directory may also be needed to create an e-mail address. Directory systems are also used as a semi-authoritative sources for telephone numbers, office numbers and so on. Therefore such resource are both *target* and *source* resources. And then there are finally target resources. These are not authoritative in any way. Identity management system will only write to these. Or ... will it? What happens when a conflicting account already exists on such resource and therefore we cannot create a new account for a new employee. And how do we check if there are no accounts that are not supposed to be there? It turns out that even the target systems contain valuable information after all.

The reality brings a wild mix of source, target, semi-source, target/source and quasi-target resources that are almost impossible to put into a pre-defined boxes. Therefore midPoint does not bother to define a concept of source or target resource. All resources can be both sources and targets and the authoritativeness of each attribute can be controlled on a very fine level. Almost every real-world situation can easily fit into this model.

Inbound and Outbound Mappings

MidPoint is firmly based on the principle of reuse. Previous chapter explained that behavior of attributes during provisioning is controlled by *mappings*. Therefore it is perhaps no big surprise that the behavior of attributes during synchronization is also controlled by

mappings. In fact, provisioning is just a special case of synchronization. Following picture explains the combined mechanism.



There are two types of mappings:

- **Inbound mappings** map data *into* midPoint. These mappings take the data from the source resources, transform them and apply the result to the user object.
- **Outbound mappings** map data *out of* midPoint. These mappings take user properties, transform them and apply the result to attributes in target systems.

The mappings themselves are almost the same regardless whether they are inbound or outbound. They have sources, targets, expressions, conditions, etc. Just the sources and targets are reversed:

	Inbound mapping	Outbound mapping
Direction	resource → midPoint	midPoint → resource
Mapping source	resource object (e.g. account)	focal object (e.g. user)
Mapping target	focal object (e.g. user)	resource object (e.g. account)

That is it. Just think about the same mappings that were used in previous chapter, just flip the direction. Now the mapping will take data from the account and the results will be applied to user object. Like this:

```
<attribute>
  <ref>ri:lastname</ref>
  <inbound>
    <target>
      <path>$focus/familyName</path>
    </target>
  </inbound>
</attribute>
```

This mapping will take the value of `lastName` attribute from the resource and store the value in `familyName` property of `midPoint` user.

The rest is the same as outbound mappings. All the expressions and evaluators can be used for inbound mappings in the same way as for outbound mappings. For example a Groovy expression can be used to sanitize the value before it is stored in `midPoint`:

```
<attribute>
  <ref>ri:lastName</ref>
  <inbound>
    <expression>
      <script>
        <code>lastName?.trim()</code>
      </script>
    </expression>
    <target>
      <path>$focus/familyName</path>
    </target>
  </inbound>
</attribute>
```

The same approach can also be taken for activation and even password mappings. However, there is one difference for password mappings. Password are usually write-only value. When the password is written it is usually hashed and the original value cannot be retrieved any longer. Then there are resource such as HR systems that do not store employee passwords at all because those are not really accounts that we are reading. Those are just regular database entries that the connector presents as accounts. Inbound password synchronization is almost never easy and it often requires a lot of planning and ingenuity. However, there is one method that is used quite often. The initial user passwords are usually randomly generated. As this is a very common case `midPoint` can do this easily:

```
<credentials>
  <password>
    <inbound>
      <strength>weak</strength>
      <expression>
        <generate/>
      </expression>
    </inbound>
  </password>
</credentials>
```

This mapping will generate a random password for a user. Both the mapping and the `generate` expression evaluators are quite smart. The mapping knows that the target is user password without any need to explicitly specify that. In addition to that the `generate` expression evaluator will take user password policy into consideration. It does not make sense to generate any random password. If we do not consider password policy then we can generate password that is too short, too long, too weak to pass the policy or too strong to be useful in any way. Therefore the `generate` expression will look for password policy and generate a random password that just matches the specified password policy.

There is more important detail to see here. The inbound password mapping is *weak*. And there is good reason for this. We do not want midPoint password to be replaced by randomly generated password. We only want to set a random password in case that it is an initial password. And that is exactly what a weak mapping does: it sets new value only if the target does not have any existing value. Therefore this mapping will not overwrite passwords that are already set.

Note: There is no direct account-account synchronization in midPoint. As explained before, midPoint follows start topology (a.k.a. “hub and spoke”). Therefore the synchronization is either from account to user (inbound) or from user to account (outbound). The effect of account-account synchronization is achieved by combining inbound and outbound synchronization mechanisms.

Correlation

It is all quite easy if we try to import all HR records into an empty midPoint. Set up inbound mappings, start import task and all is done. But practical situations are much more complex. Synchronization algorithm almost never work with empty midPoint. Live synchronization and reconciliation are supposed to work well with existing midPoint users. And even import can be quite complex for example in cases when we try to import data from an additional data source into a running midPoint deployment. Some users in the import set are new, but there may be accounts for existing users. We need to tell the difference and we need to handle the situation in a different way. Of course, midPoint has an easy solution for this: correlation mechanism.

Correlation expression is a method how to connect new accounts and existing users. It works like this: whenever midPoint discovers new account it will try to link that account to an existing user. Correlation expression is used to do this. Correlation expression is in fact a parametric search query. Such search query will be constructed for every new account and it will be used to look for users that the account belongs to. The easiest form of the correlation expression is to look by using an identifier:

```
<correlation>
  <q:equal>
    <q:path>employeeNumber</q:path>
    <expression>
      <path>$shadow/attributes/empno</path>
    </expression>
  </q:equal>
</correlation>
```

This correlation query will take the value of empno attribute of the account. This value will be placed into the search query that midPoint computes in memory. For an account with empno attribute set to 007 the resulting search filter will look like this:

```
<q:equal>
  <q:path>employeeNumber</q:path>
  <q:value>007</q:value>
</q:equal>
```

MidPoint will use this search filter to look for users. If there is an user with `employeeNumber` property set to 007 then such user will be considered to be an owner of the account.

MidPoint has its own data representation mechanism and object structure. Therefore midPoint also has its own query language that is designed to work well with the object structure. The query language is not difficult to learn as it follows the structure of most other query languages. The language itself is described later in the book and also in midPoint documentation. But do not worry about this too much. Vast majority of correlation expressions is very easy. In fact it is usually just a single `equal` clause just like that one used in the example above.

This may seem a bit complex, but it is necessary. The correlation expression must be a search filter because that is the only efficient way how to find single user in a large set of other users. We cannot scan the accounts one-by-one. We need to utilize the search power of the database for this.

Synchronization Situations and Reactions

Correlation expression can be used to find an owner for a new account. That is part of the solution but not entire solution. If the owner is found then the action is quite obvious: link the account to the user and proceed as usual. But what to do if the owner is not found? This resource may be an authoritative resource and therefore we want to create a new user based on the account. Or this may be a reconciliation with a target resource and in that case this means that we have found an illegal account. We probably want to disable such account. And what to do if more than one owner is found? This can all become quite complicated. Therefore midPoint as a concept of *synchronization situations* to make it understandable and manageable.

Whenever midPoint deals with a change on an account the *situation* of that account is determined. The situation reflects whether this account is already linked to the user, whether we know the user, but it is not linked yet, whether we cannot determine the user and so on. Individual situations are explained in the following table.

Situation	Description
linked	The account is properly linked to the owner. This is the normal situation.
unlinked	The account is not linked to the owner, but we know who the owner is. Correlation expression told us who the owner is. In this case midPoint thinks that the link should exist, but it is not linked yet.
unmatched	The account is not linked and we not even know who the owner is. The correlation expression haven't returned any candidates.
disputed	The account is not linked, but there are more potential owners. The correlation expression returned more than one candidate.
collision	The account is linked to more than one owner. This should not happen under normal circumstances. This is usually caused by faulty customizations or

	software bugs.
deleted	There was an existing account but it was deleted on the resource.

When synchronization *situation* is determined, midPoint will figure out what a proper *reaction* is. The reaction is quite clear for some situations such as unlinked situation. But there is a lot of variability for other situations, such as unmatched situation. This variability is a reason that midPoint allow to set a reaction for each situation individually. There are several pre-defined reactions:

Action	Description
Add focus	New midPoint user will be created and linked to the account. This is usually a reaction configured for authoritative resources when an account is created.
Delete focus	MidPoint user that owns the account will be deleted. This is usually a reaction configured for authoritative resources when an account is deleted.
Inactivate focus	MidPoint user that owns the account will be disabled. This is also used for authoritative resources. But this is a milder reaction.
Link	The user-account link will be created.
Unlink	The user-account link will be removed. The account will no longer be linked to the user.
Delete shadow	The account will be deleted. This is the usual reaction when illegal account is detected on non-authoritative resource.
Inactivate shadow	The account will be disabled. Also used when an illegal account is found.

If no reaction is explicitly configured for a situation then midPoint does nothing. Just the situation is recorded in midPoint repository. This is part of midPoint philosophy not to change the data unless the action was explicitly configured.

The reactions can be defined in the synchronization section of resource configuration:

```
<synchronization>
  <objectSynchronization>
    <correlation>...</correlation>
    <reaction>
      <situation>linked</situation>
      <synchronize>true</synchronize>
    </reaction>
    <reaction>
      <situation>deleted</situation>
      <action>
        <handlerUri>http://midpoint.evolveum.com/xml/ns/public/model/action-3#deleteFocus</handlerUri>
      </action>
    </reaction>
    <reaction>
      <situation>unlinked</situation>
      <action>
        <handlerUri>http://midpoint.evolveum.com/xml/ns/public/model/action-3#deleteFocus</handlerUri>
      </action>
    </reaction>
  </objectSynchronization>
</synchronization>
```

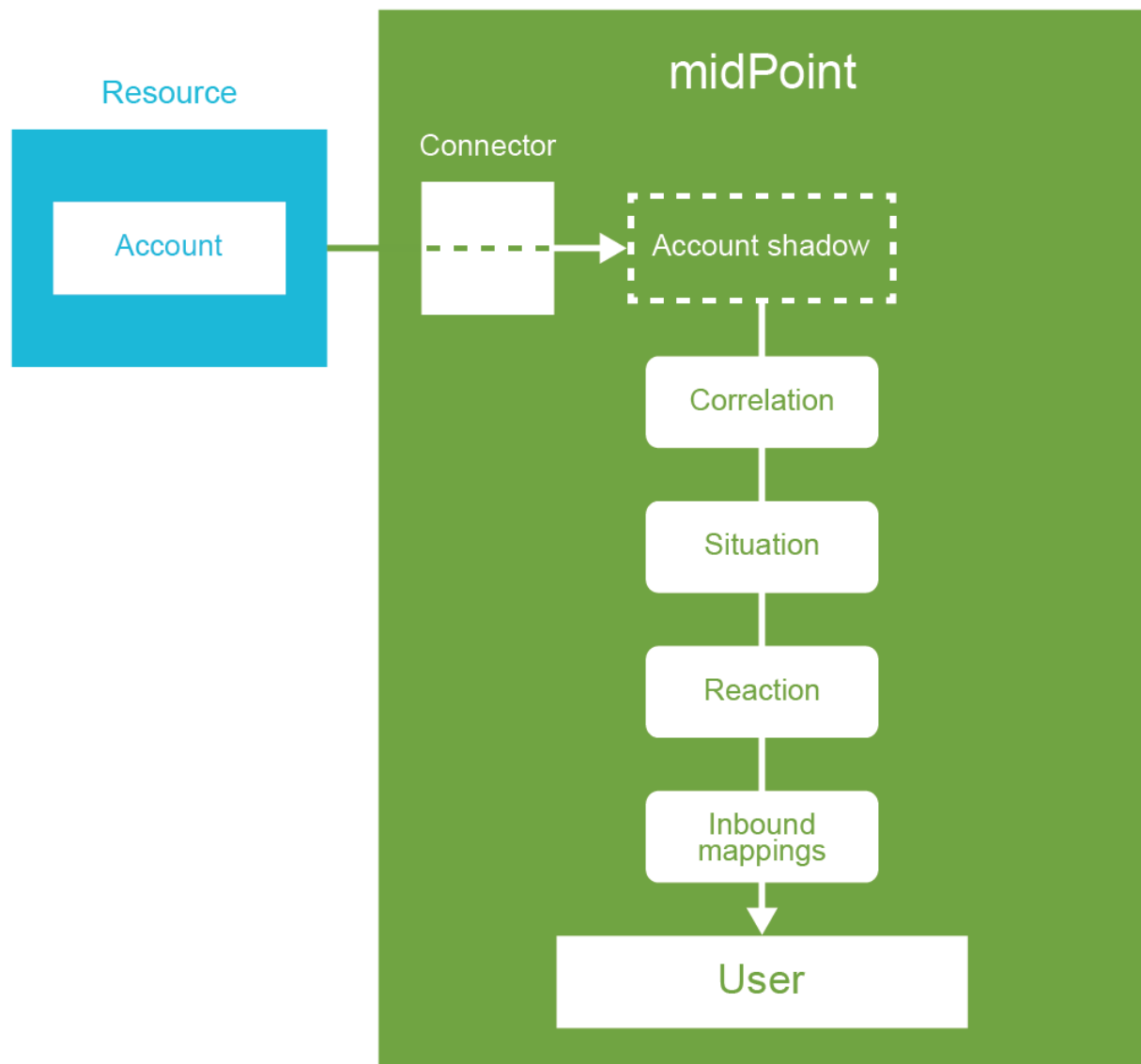


```

ion-3#link</handlerUri>
    </action>
</reaction>
<reaction>
    <situation>unmatched</situation>
    <action>
        <handlerUri>http://midpoint.evolveum.com/xml/ns/public/model/act
ion-3#addFocus</handlerUri>
    </action>
</reaction>
</objectSynchronization>
</synchronization>

```

Most of the configuration is perhaps self-explanatory. This is a typical authoritative resource. If there is a new account on the resource and we do not have an owner (situation: unmatched) then create a new user (action: addFocus). If the account is deleted from the resource (situation: deleted) then also delete the user (action: deleteFocus). If we happen to find an account which should be linked but it is not (situation: unlinked) then link it (action: link). The only thing that deserves an explanation is the reaction to linked situation. In this case there is not much to do. Everything seems to be in order. However there still may be attributes that are not correctly set. Remember the inbound mappings? The inbound mappings were not even mentioned in this section yet. And for a good reason. The inbound mappings are not evaluated at this stage. Evaluation of inbound mappings happen only after the situations and reactions are evaluated. We need this so all the accounts are properly linked (or unlinked) so the inbound mappings have valid sources and targets. But the evaluation of inbound and outbound mappings do not happen by itself. MidPoint does not change the data unless it is explicitly configured to. There are reactions for unmatched, deleted and unlinked situations. So in this case midPoint assumes that it is expected to fully synchronize everything and therefore all the mappings and policies are evaluated automatically. But there is no reaction for linked situation. In that case midPoint assumes that it should do nothing as nothing is explicitly configured. Hence the synchronize property. This property can be used to force midPoint to do full synchronization even if there is no explicit action configured. And it can also be used to void full synchronization even if explicit action is configured.



The figure above illustrates the usual sequence of events during inbound synchronization:

1. Account is stored in the resource database.
2. Appropriate identity connector is used to read the account.
3. Account shadow is created in midPoint.
4. Correlation expression is evaluated to determine account ownership (if the account is not already linked to a user).
5. Synchronization situation is determined based on account ownership and state of the account.
6. Appropriate reaction to the situation is determined based on resource configuration.
7. Inbound mappings are evaluated to map account values to the user.

Please note that the description of this process is slightly simplified for clarity. There are also obvious deviations from this process. E.g. inbound mappings are skipped in case that the user is about to be deleted, the mappings are also skipped if the reaction does not include “synchronization” and so on. But generally this is what usually happens during inbound synchronization.

Note: MidPoint is an extensible system. There are several prefabricated synchronization reactions described above. Those reactions can handle vast majority of situations that happen during synchronization. However, there is a possibility to extend the system with completely custom reactions. MidPoint was designed for this. This is the theory. However, currently this part of midPoint is only partially extensible. Full extensibility feature was planned, but it was never implemented. Therefore extensibility of synchronization reaction is possible, but it might be quite hard to achieve this in practice and it may require significant development effort. But there is another way. MidPoint development team would absolutely love to finish this extensibility feature as it was originally planned. However, existing midPoint customers had so far prioritized other features. MidPoint subscribers and sponsors are funding the development therefore midPoint development must follow their priorities. Therefore if you are interested in full synchronization reaction extensibility (or any other feature) please consider purchasing midPoint subscription or sponsoring the feature.

Synchronization Tasks

Now we know how the inbound synchronization works: midPoint reads the account, then correlation is applied, situation determined and reaction executed. However, we have not yet discussed the details of the very first step: how does midPoint actually read the account? Nothing happens without a reason, therefore there must be some active component in midPoint that actually looks for the new, changed and deleted accounts. And that component is a *synchronization task*.

MidPoint *task* is an active process that is running inside midPoint server. This is the first time that we encountered the concept of a *task*, but it is definitely not the last one. Tasks are used for many purposes in midPoint. They are used to track long-running operations, workflows, actions that work on large sets of objects (bulk actions), there are tasks that execute cleanup jobs, reporting and various other functions. The concept of tasks is a very powerful and flexible one. Tasks can be used to track execution of a short one-off operations. Tasks can be used to execute scheduled actions in regular intervals. Or tasks can be used to track long-running processes. We will be using tasks in almost every chapter of this book.

Tasks are used as an active component to “run” almost all synchronization mechanisms:

- **Reconciliation task** is listing all the accounts from a specific resource in sequence. The task executes reconciliation for every account that is found. This means that midPoint computes how that particular account should look like and then the computed values are compared with real account attributes. This task is usually

scheduled for regular execution. However the execution interval is quite long (days or weeks).

- **Live synchronization task** is looking (polling) for changes in a specific resource. The task will look for created, modified and deleted accounts. The task will get a description of the change and pass that to midPoint synchronization mechanisms. This task is almost always scheduled for regular execution and the execution interval is very short (minutes or even seconds).
- **Import from resource task** is listing all the accounts from a specific resource. The task will pretend that the accounts were just created. This usually motivates midPoint to create users based on those accounts or link these accounts to existing users. This task is usually not scheduled and it is almost always executed manually.

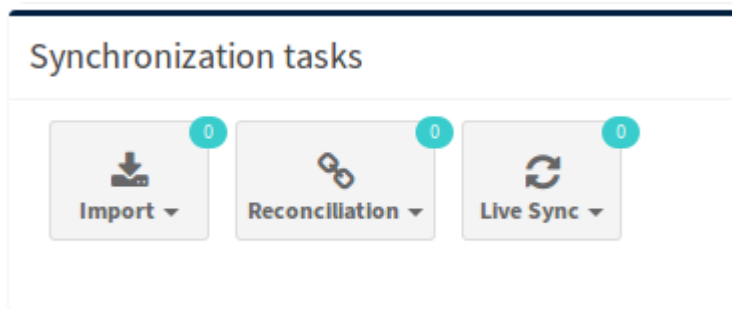
Each type of synchronization task is detecting the change using a different mechanism. However, once the task got the account (or *delta*, see below) then the processing is the same for all tasks. All the tasks lead to the same algorithms based on the same configuration and policies. Therefore it does not matter whether it has all started in reconciliation or live synchronization task. It will all end up in the same correlation-situation-reaction-mapping process.

However, the tasks are necessary to initiate the synchronization. They are the active part, the spark that starts the synchronization process. Without the tasks the synchronization does not really work. There are ways how the synchronization can “happen” even without a task, e.g. as a reaction to user interface operation or if a new account is discovered during an unrelated operation. But practical deployments need at least one synchronization task to work properly. This task takes care of vast majority of synchronization cases.

Strictly speaking tasks are quite a strange kind of animal. Tasks have their data and configuration as most other midPoint objects. But tasks are active. Therefore there is a CPU thread associated with the tasks when the task is running. There are mechanisms how to monitor task progress. The tasks needs to be cluster-aware so they can fail over to a different midPoint node if one node fails. The tasks are quite rich and a bit tricky to handle. But midPoint is making task handling quite simple. Tasks are represented as usual midPoint objects. Therefore they can be imported to midPoint in XML/JSON/YAML form as any other object. Tasks can be easily edited in their XML/JSON/YAML form to change the scheduling, modify the parameters and so on. Of course, there are some special functions that only the tasks have (such as suspend and resume) that cannot be controlled using the XML/JSON/YAML format. But vast majority of task management can be done using the very same methods as for other midPoint objects.

Therefore tasks can be created by simple taking the XML and importing that to midPoint. And that's the way how synchronization tasks are often managed. When an XML-formatted resource definition is created then there is often an associated synchronization

task. So both resource and all the necessary synchronization tasks can be imported together. However, synchronization tasks can also be created from midPoint user interface. Synchronization tasks are usually created by using special-purpose buttons in resource detail pages.



Once the synchronization tasks are created they can be managed in the same way as other tasks are managed: in the “Server tasks” part of the midPoint user interface.

Synchronization Example: HR Feed

This section describes a complete working example that feeds the HR data to midPoint. The ExAmPLE company HR system is an old and complex system. Therefore the easiest integration method is to use structured text exports. The HR system is set up to export the employee data to a comma-separated text file (CSV) every night. MidPoint takes this export file and updates the data about users.

This configuration is used in three stages. First we will use a simple setup to import the data into midPoint. This is an operation that is executed only once. The system administrator starts it manually. Then the configuration will be updated for scheduled reconciliation. The reconciliation compares all the data records every time and makes any necessary updates. That is perfectly acceptable in this case. Yet at the end we show how to use the configuration with live synchronization.

The core of the configuration is contained in a single resource definition file. Following paragraphs explain individual parts of the file. There are few additional configuration files for reconciliation and live synchronization tasks. Simplified XML notation is used for clarity. The complete file in a form directly usable in midPoint can be found at the same place as all the other samples in this book (see Additional Information chapter for details).

This HR resource is a source resource. It will be used to “pull” the data inside midPoint. However, as we have described previously, there is no fundamental difference between source and target resources in midPoint. Therefore this HR resource starts in entirely ordinary way. There is a reference to the CSV connector and the connector configuration:

```
<resource oid="03c3ceea-78e2-11e6-954d-dfdfa9ace0cf">
  <name>HR System</name>
  <connectorRef>...</connectorRef>
  <connectorConfiguration>
    <configurationProperties>
      <filePath>/var/opt/midpoint/resources/hr.csv</filePath>
      <encoding>utf-8</encoding>
```

```

        <fieldDelimiter>,</fieldDelimiter>
        <multivalueDelimiter>;</multivalueDelimiter>
        <uniqueAttribute>empno</uniqueAttribute>
        <passwordAttribute>password</passwordAttribute>
    </configurationProperties>
</connectorConfiguration>
...

```

The next section is schema handling configuration. That is where it becomes slightly more interesting. The schema handling section contains inbound mappings for HR account attributes:

```

<schemaHandling>
  <objectType>
    <objectClass>ri:AccountObjectClass</objectClass>
    <attribute>
      <ref>ri:empno</ref>
      <inbound>
        <target>
          <path>$focus/name</path>
        </target>
      </inbound>
      <inbound>
        <target>
          <path>$focus/employeeNumber</path>
        </target>
      </inbound>
    </attribute>
    <attribute>
      <ref>ri:firstname</ref>
      <inbound>
        <target>
          <path>$focus/givenName</path>
        </target>
      </inbound>
    </attribute>
    <attribute>
      <ref>ri:lastname</ref>
      <inbound>
        <target>
          <path>$focus/familyName</path>
        </target>
      </inbound>
    </attribute>
  ...

```

The account attribute empno is mapped to midPoint user properties name and employeeNumber. Account attributes firstname and lastname are mapped to givenName and familyName properties respectively. This is perhaps self-explanatory.

The next part of the configuration specifies mappings for activation and credentials:

```

<activation>
  <administrativeStatus>
    <inbound/>
  </administrativeStatus>
</activation>

<credentials>

```

```

    <password>
      <inbound>
        <strength>weak</strength>
        <expression>
          <generate/>
        </expression>
      </inbound>
    </password>
  </credentials>
  ...

```

The activation mapping is very simple. Activation is a very specific concept in midPoint. MidPoint knows activation attributes and their meaning. Therefore there is no need to specify a lot of details. The activation mapping simply specifies that the administrative status should be mapped in the inbound direction. And that is it.

However the mapping for credentials needs a bit of explanation. What midPoint sees as HR accounts are not exactly accounts. They are usually just records in the HR database. Nobody is using these HR records to log into the HR systems. Therefore there is no password associated with them. But we need a password for the users in midPoint. Therefore we are going to generate them. And for that we are going to use the weak mapping with generate expression that was explained above.

The mappings are undoubtedly important. The mappings specify how are the account data reflected to midPoint user. But the mappings do not specify whether the accounts should be created or deleted. Mappings control the data, but they do not control the *lifecycle*. It is the next configuration section that makes this resource really authoritative:

```

<synchronization>
  <objectSynchronization>
    <enabled>true</enabled>
    <correlation>
      <q:equal>
        <q:path>employeeNumber</q:path>
        <expression>
          <path>$shadow/attributes/empno</path>
        </expression>
      </q:equal>
    </correlation>
    <reaction>
      <situation>linked</situation>
      <synchronize>true</synchronize>
    </reaction>
    <reaction>
      <situation>deleted</situation>
      <synchronize>true</synchronize>
      <action>
        <handlerUri>http://midpoint.evolveum.com/xml/ns/public/model
/action-3#deleteFocus</handlerUri>
      </action>
    </reaction>
    <reaction>
      <situation>unlinked</situation>
      <synchronize>true</synchronize>
      <action>
        <handlerUri>http://midpoint.evolveum.com/xml/ns/public/model

```

```

/action-3#link</handlerUri>
    </action>
</reaction>
<reaction>
    <situation>unmatched</situation>
    <synchronize>true</synchronize>
    <action>
        <handlerUri>http://midpoint.evolveum.com/xml/ns/public/mode
1/action-3#addFocus</handlerUri>
    </action>
</reaction>
</objectSynchronization>
</synchronization>

```

Given the information in this chapter this configuration should be quite easy to read. This is how a typical authoritative resource usually works. If there is a new account on the resource and we do not have an owner (situation: unmatched) then we create a new user (action: addFocus). If there is a new account for which we can find existing owner (situation: unlinked) then simply link it (reaction: link). If the account is linked already (situation: linked) then we just synchronize the data. In fact, we will synchronize data for all the other situations as well. Except the last one. If the account is deleted in the HR system (situation: deleted) then we want to delete midPoint user as well (reaction: deleteFocus). As the user gets deleted there is point in synchronizing the data. MidPoint knows that and skips application of mappings.

The ownership of the accounts that are not already linked is determined by the correlation expression. In this case the expression will be comparing account attribute empno with user property employeeNumber. If the values match then the user is considered to be an owner of the account.

There is one more detail in this resource that we have skipped:

```

<projection>
    <assignmentPolicyEnforcement>none</assignmentPolicyEnforcement>
</projection>

```

This is a setting that adjusts the behavior of midPoint *assignments*. As was already mentioned all resources in midPoint are created equal. The source resources must follow the same rules as target resources. And one of the fundamental rules of midPoint is that there should not be any account without a specific reason to exist. In midPoint terminology every account exists because there is an *assignment* that justifies its existence. While this approach is exactly what we want for vast majority of (well behaving) resources, it is not exactly ideal for source resources. Those resources work the other way around. The HR account is in fact a cause for midPoint user existence, not its effect. Therefore there is really useful assignmentPolicyEnforcement setting that controls the behavior of assignments. This setting is used in a variety of scenarios, mostly for data migration and to tame resources that just won't behave in a civilized manner. But in this case the setting is used to turn off the assignment enforcement for this resource entirely. As this resource is

an authoritative source the assignment enforcement does not make much sense. Behavior of this resource is defined by the `synchronization` section of resource configuration.

Now we have resource configuration complete. This configuration sets up the connector, mappings and synchronization policies. This configuration is the same for all the synchronization flavors: import, reconciliation and live sync – they will all use the same settings. When it comes to configuration the only difference between those synchronization flavors is the way how the synchronization tasks are set up. If an import task is set up then import of resource accounts will be executed. If reconciliation task is set up the reconciliation will be executed. It is all just in the tasks. And in fact those tasks can be easily set up using those convenient buttons in the user interface. But we like to make our lives a bit painful in our part of the world. Therefore we are going to go hardcore and import the tasks in the XML form.

First task is an import task. This task will list all the accounts in the HR CSV file. The task will pretend that each of the accounts was just created. If the task is executed for the first time then resulting situation is going to be either unmatched or unlinked. So, we are going to create new midPoint users or link the accounts to existing users.

```
<task oid="7c57adc2-a857-11e7-83ac-0f212d965f5b">
  <name>HR Import</name>
  <taskIdentifier>7c57adc2-a857-11e7-83ac-0f212d965f5b</taskIdentifier>
  <ownerRef oid="00000000-0000-0000-0000-000000000002"/>
  <executionStatus>runnable</executionStatus>
  <handlerUri>http://midpoint.evolveum.com/xml/ns/public/model/synchronization
/task/import/handler-3</handlerUri>
  <objectRef oid="03c3ceea-78e2-11e6-954d-dfdfa9ace0cf"
type="c:ResourceType"/>
  <recurrence>single</recurrence>
</task>
```

This is a very basic structure of the task. Similarly to all midPoint objects task has a name. Then there is task identifier which is used for internal task management purposes. This is usually the same as task object OID. Task needs definition of an owner. The owner is a user that is executing the task. This is important, because authorizations that this user has determine what the task is allowed to do. This is also the identity that will be recorded in the audit log. In this case administrator is owner of this task. Task execution status tells whether the task is running, it is suspended or finished. Then there is a handler URI. The handler URI specifies what the task really does. It (indirectly) refers to the code that the server will execute. In this case the task URI specifies that this is a synchronization task that imports accounts from the resource. And the resource to import the accounts from is specified by the `objectRef` reference. This points to our HR resource. And the last item is recurrence. Recurrence specifies whether the task runs only once (single) or whether the execution should be repeated (recurring).

When this XML definition of the task is imported to midPoint, the server will try to immediately execute the task. That means that import of accounts from the HR resource starts immediately. Progress of the task can be monitored in the Server tasks section of

midPoint user interface. The import task is not a recurring task. Therefore it will run only once. If you need to re-run the task you can do that from midPoint user interface. But the task will not be executed unless you explicitly tell midPoint to do so. This is how typical import tasks work. They are usually executed when a new resource is connected to the system. And once everything is set up, correlated and linked then the import task is not needed any more.

A clever reader may ask what happens when the import task is executed once more. The answer is simple: not much. Even if the task pretends that the accounts were just created, midPoint is not fooled easily. In fact it is hard to believe that the account was just created if midPoint already has shadow for that account and it is linked to a user, isn't it? Therefore midPoint is going to stay calm and carry on. If there is any change in the account attribute than the change will be reflected to the user. But that is it. No big drama here.

Import task will get the data from the resource into midPoint. But as import is not a recurring task it will not keep the data synchronized. It is not designed to do so. But there are other tasks that are designed for this purpose. Reconciliation task is one of these. Reconciliation task will list all the accounts on a resource and compare that with data in midPoint.

```
<task oid="bbe4ceac-a85c-11e7-a49f-0f5777d22906">
  <name>HR Reconciliation</name>
  <taskIdentifier>bbe4ceac-a85c-11e7-a49f-0f5777d22906</taskIdentifier>
  <ownerRef oid="00000000-0000-0000-0000-000000000002"/>
  <executionStatus>runnable</executionStatus>
  <handlerUri>http://midpoint.evolveum.com/xml/ns/public/model/synchronization/task/rec
onciliation/handler-3</handlerUri>
  <objectRef oid="03c3ceea-78e2-11e6-954d-dfd9a9ace0cf" type="c:ResourceType"/>
  <recurrence>recurring</recurrence>
  <schedule>
    <cronLikePattern>0 0 1 ? * SAT</cronLikePattern>
    <misfireAction>executeImmediately</misfireAction>
  </schedule>
</task>
```

Definition of a reconciliation task is almost the same as the definition of import task. But there are crucial differences. First of all there is different handler URI. This is what makes this task a reconciliation task. Then the task is recurring. This means that midPoint will repeat execution of the task. Therefore there is also execution schedule so the server knows when to execute the task. Reconciliation tasks are usually resource-intensive therefore we usually want to execute them at a very specific off-peak times. For that reason the execution schedule is defined using a cron-like pattern. UNIX-friendly readers will be surely familiar with this. The format is “seconds minutes hours day-of-month month day-of-week year”. Therefore this task will be executed every Saturday at 01:00:00am. There is also definition of misfire action. Misfire is a situation when the server is down at the time when the task is supposed to run. Therefore if the server is down in the early hours of Saturday this task will be executed as soon as the server starts up.

Reconciliation task is a real workhorse of identity management. It can be used in almost any resource. It is very reliable. It is often used to fix many problems, apply new policies, look for missing accounts, illegal accounts and so on. It is indeed a really useful tool. But it has its downsides. Reconciliation iterates through all the accounts, it recomputes all the applicable policies for every account one-by-one. Therefore it may be quite resource-intensive. Even quite brutal if the policies are complex, user population is high and the resources are slow. This can take hours or even days in extreme cases. But even for smaller deployments reconciliation is not entirely easy. The problem is not in midPoint. MidPoint can be scaled to handle the load. But listing all the accounts often may put unacceptable load on the resources. Therefore reconciliation is not executed often. Daily, weekly or even monthly reconciliation seems to be a common approach. Reconciliation is reliable, but it is not entirely what we would call real-time. But of course, midPoint has a faster alternative.

Live synchronization is the option for real-time synchronization. Or rather *almost* real-time synchronization. Practical latencies for live synchronization are in the range of seconds or minutes which is fast enough for most practical cases. Live synchronization is also quite resource-efficient. Overall it is much faster and much lighter than reconciliation. But live synchronization is not available for all resources. Live synchronization depends on the ability to get recent changes from the resource. Therefore it is only available for resources that record the changes. The specific mechanism to record the changes may vary from resource to resource. It may be as basic as a simple modification timestamp or it may be a complex change log. But it has to be good enough for the connector to discover recent changes and it must be efficient enough for the connector to do that every couple of seconds. If such mechanism is available and the connector knows how to use it then setting up live synchronization is easy. All that is needed is synchronization task.

```
<task oid="7c57adc2-a857-11e7-83ac-0f212d965f5b">
  <name>HR Live Synchronization</name>
  <extension>
    <mext:kind>account</mext:kind>
  </extension>
  <taskIdentifier>7c57adc2-a857-11e7-83ac-0f212d965f5b</taskIdentifier>
  <ownerRef oid="00000000-0000-0000-0000-000000000002"/>
  <executionStatus>runnable</executionStatus>
  <handlerUri>http://midpoint.evolveum.com/xml/ns/public/model/synchronization/task/live-sync/handler-3</handlerUri>
  <objectRef oid="03c3ceea-78e2-11e6-954d-dfd9a9ace0cf" type="c:ResourceType"/>
  <recurrence>recurring</recurrence>
  <schedule>
    <interval>10</interval>
  </schedule>
</task>
```

Now the task definition should be easy to understand. There is a different handler URI that makes this a live synchronization task. There is also a different type of scheduling. We do

not want to execute this task at a specific time. We rather want to execute it all the time at regular intervals. In this case the interval is set to 10 seconds. And that is all that is needed to have live synchronization running.

Setting up configuration flavors in midPoint such as reconciliation or live synchronization is just a matter of setting up the tasks. The rest of the configuration is the same for all flavors. Therefore it is very easy to run both live synchronization and reconciliation for the same resource. Just create two tasks. And it fact this is quite common configuration. Live synchronization is used to get the changes quickly and efficiently. And reconciliation is used to make sure all the changes were processed and that the policies are applied consistently.

That is all. Now we have the HR feed up and running. However, there are still few issues. A clever reader would surely notice that this is not a very good HR resource. MidPoint users created from this HR feed have given name and family name. But the full name field is empty. But do not worry. We will sort that out in later chapters with the help of *user template*. Also the users have employee number as their username. This may be in fact a very good approach for some deployments as it avoids the need to rename accounts. However, it is not a very user-friendly approach. Therefore most deployments need to generate more convenient usernames. This is easy to do with midPoint and we will also address that later. There is still a lot of things to learn before we get to a complete synchronization set up.

HR Feed Recommendations

All resources are created equal in midPoint. However, the source resources almost always have slightly special standing. Even though midPoint mechanisms are the same for all resources, the data coming from the sources usually have significant impact on the entire solution. There is this traditional computer engineering wisdom: *garbage in, garbage out*. An error in data feed may cause a lot of problems everywhere. Therefore it is important to get the data sources right. And this is usually one of the first steps in an IDM project.

Setting up a data source is usually an iterative process. And there may be many iterations, especially if the quality of the source data is unknown. The process usually goes like this:

1. Set up initial source resource definition based on the information you have. Set up connector and test connection. Check that you can see the accounts. Set up mappings and synchronization policy.
2. Test the import process on a couple of individual accounts. Navigate to the resource details pages, click on Accounts tab to list accounts, select an account and choose *Import* from the context menu next to the account (small cog button). Import of that individual account will start. Just that one account. Therefore it is easier to see the errors (see step 6).
3. Create an import task and run import of all accounts.

4. Examine task errors. You can use task details page to get the summary.
5. If there are no errors then examine the users. If everything is OK then congratulations. You have a good import. However, this is unlikely to happen on first few couple of attempts.
6. You will probably need to have a look into system logs to learn the details of individual import failures. MidPoint heavily relies on logs for detailed error analysis. See the troubleshooting section of this book to learn how to adjust log levels and how to get understand the log messages.
7. Some errors are likely to be caused by the errors in your mappings and policies. Those are usually easy to fix. Almost always there are error caused by wrong or unexpected input data. The right way would be to fix the data. But that is not always possible (in fact it is almost never a feasible option). But most of the input data errors can be fixed in midPoint with a bit of ingenuity. Just use the power of the mappings.
8. Rinse and repeat. If the errors were not severe then you may simple re-run the import task. This often works just fine. But if the error was in a mapping that completely ruined all the data then it is perhaps best to start with a blank slate. We are all just human and this situation happens quite often, especially in the beginning. Therefore there is a special feature to help you out. Navigate to Configuration > Repository Objects. There is a small cog button that opens a context menu. Select "Delete all identities" item. That is what we lovingly call "laxative button". A brief dialog will pop up asking you to specify which identities exactly are to be deleted (users, shadows, ...). This is a very convenient way how to get back to a black slate, but keep all the configuration (resources, templates, tasks).
9. Goto step 2. Repeat until done.

If the initial IDM deployment step includes an HR feed we strongly recommend to start with that HR feed. It is usually easier to correlate other resources to midPoint users created from a reasonably-reliable HR data. Also, it will usually take some tweaking to get the HR import right. The possibility easily to clean up midPoint to get to a clean slate is extremely useful. But that is possible only if the HR feed is the first resource that is connected to midPoint.

A clever reader would notice, that we assumed that the source feed will be taken from a CSV file. And this is indeed the case in vast majority of cases. If a new employee or contractor is about to join the company there is usually no hurry. This information is entered into the HR system at least few days in advance therefore daily CVS export is perfectly acceptable. However, there may be cases when we want a faster response. Or maybe we do not want additional burden of dealing with CSV exports. Of course, there is solution. In theory any connector can be used for source resource. And in fact there are

specialized connectors for HR systems that are taking data directly from the HR system. For example there is a connector for Oracle HCM system. Unfortunately, there is no connector that can take data from SAP HR system yet.

Synchronization and Provisioning

Synchronization and provisioning are intimately connected. Everything that we have explained about provisioning in the previous chapter also applies to synchronization. In fact provisioning and synchronization are just applications of the same basic mechanisms. Provisioning starts with modification of a user. Synchronization starts a bit earlier: inbound mappings are used to map values from source system to the user. The result of inbound mapping evaluation is that the user object is modified. And according to midPoint principles it does not matter how the user was modified. The reaction is the same: accounts are provisioned, modified or deleted as needed.

The synchronization (*inbound* processing) and provisioning (*outbound* processing) usually happen in the same seamless operation. For example the HR connector detects update in the last name of an employee. That modification is applied to midPoint user, so the family name of midPoint user is updated. The operation continues by evaluating all templates, roles and outbound mappings. The outbound mappings usually map the family name change to the resource attributes. Therefore the resource accounts linked to the user are immediately updated. All of that happens in a single operation. That is how midPoint works. MidPoint is not a human. It will never procrastinate. MidPoint will never postpone the operation for later if the operation can be executed immediately. MidPoint always tries to get the data right on a first try. Therefore there are no specialized propagation or provisioning tasks that you might know from older IDM systems. MidPoint does not need them.

There are other advantages in doing everything in one operation. If it is one operation midPoint knows all the details: what was the cause, what is the effect, what exactly has been changed. This is important for troubleshooting. Some IDM systems decouple the cause and the effect. Such a divided approach may have some advantages, but it is an absolute nightmare when an engineer needs to figure out why a certain effect happened. But midPoint has both the cause and the effects correlated together in a single operation. Therefore it is much easier to figure out what is going on. And it can also be neatly recorded in the audit trail. And there is another huge advantage: midPoint knows exactly what has been changed. This means that midPoint does not only know the new value of a property. MidPoint knows also the old value and values that were added or removed. This is a complete description of the change that we call *delta*. This is recorded at the beginning of the operation and propagated all the way until the operation is done. Therefore the mappings may be smart. This approach enables a lot of interesting behavioral patterns. For example it is quite easy for midPoint to implement the “last change wins” policy. In this case midPoint will simply overwrite only those attributes that are really changed in operation. MidPoint can leave other values untouched. In fact, this is

the default behavior of midPoint. And it is a very useful behavior during deployment of a new IDM system.

Careful processing of the operations allows configurations that are not feasible with older IDM systems. Such as resource that is both a source and a target. In fact a lot of IDM systems can have resource that is both a source and a target – as long as it is a source for one attribute and a target for another attribute. But midPoint can live with a resource where the same attribute is both a source and a target. And in fact there may be many sources and many targets for the same property at the same time. And this is indeed very useful configuration. Just think about telephone number property. This is usually something that the user sets up himself. This may be set up by some kind of specialized self-service, it may be updated by a call center call, the user may update that in his Active Directory profile ... there are many ways how this information is changed. But we want this property to be consistent. We want telephone number to be the same everywhere. And we do not care where it was changed. We just want to propagate the last change from anywhere to all the other systems. MidPoint can easily do this. Just specify both inbound and outbound mappings for the same attribute:

```
<attribute>
  <ref>ri:mobile</ref>
  <outbound>
    <source>
      <path>$focus/telephoneNumber</path>
    </source>
  </outbound>
  <inbound>
    <target>
      <path>$focus/telephoneNumber</path>
    </target>
  </inbound>
</attribute>
```

In this case the change in user property `telephoneNumber` will be propagated to the account attribute `mobile` (outbound change). But also a change in the account attribute `mobile` will be propagated back to user property `telephoneNumber` (inbound change). Last change wins. A clever reader now surely grumbles something about infinite loops. But do not worry. As operations span both the inbound and outbound parts midPoint knows when to stop. There are even mechanism how to avoid loops caused by connectors detecting changes caused by the connector itself. MidPoint will break those loops automatically.

Info: Synchronization and provisioning are intimately connected in midPoint. They are in fact almost the same mechanism applied in a different direction. Then why there two sections in the resource configuration? Why there is `schemaHandling` and `synchronization`? Why not just one? The answer is simple: history. No software is created perfect on a day one. As all practical systems midPoint went through an evolutionary process of continuous improvement. MidPoint had a very good design at the beginning. Looking back at the initial design it is quite clear that almost all of midPoint

development were correctly foreseen and accounted for in the design. However, there are occasional mistakes. The initial midPoint data model design expected that there will be major differences between synchronization and provisioning mechanisms. Therefore there were two sections for that. But midPoint evolution improved over the initial design and we have found a way how to unify synchronization and provisioning mechanisms. However, we have not modified the initial data model because we wanted to keep compatibility. Having two sections instead of one is only a cosmetic imperfection. It does not cause any major trouble. But incompatible change will certainly affect continuity of midPoint deployments. And we highly value midPoint continuity and upgradeability. Therefore the two sectioned remained to this day. However, they will not be here forever. We are not going to dwell on old mistakes forever. These two section will be reunited once there is a proper time to make incompatible changes. Which will probably happen in the future when the time comes to release midPoint 4.0.

Mapping and Expression Tips and Tricks

Mappings and expressions together form a very powerful mechanism. In fact most of midPoint configuration is about setting up correct mappings. But with great powers come great responsibilities and mappings may look a bit intimidating at a first sight. However, there are few tip and tricks that may life with mappings and expressions a bit easier.

Most mappings are aware of the context in which they are used. Therefore paths of mapping sources and targets can be shortened – or even left out entirely. This is obvious for activation and credential mappings used in the HR feed example. But even ordinary mapping paths may be shortened. For example take the outbound mapping source:

```
<outbound>
  <source>
    <path>$focus/telephoneNumber</path>
  </source>
</outbound>
```

As the mapping knows that its source is a focus (user) the definition may be shortened:

```
<outbound>
  <source>
    <path>telephoneNumber</path>
  </source>
</outbound>
```

Typical midPoint deployment has tens or hundreds of mappings. Deployments with thousands of mappings are definitely feasible. Therefore it may not be entirely easy to maintain the mappings. Therefore there are two things that can make this easier. There is an ability to optionally specify mapping name. Mapping name will appear in the log files and some error messages. Therefore it may be easier to identify which mapping is causing problems or it may help locate the trace of mapping execution in the log file. Mapping can also have a description. The description can be used as a general-purpose comment or a documentation of the mapping. The description can be used to explain what the mapping does.


```

<attribute>
  <ref>ri:mobile</ref>
  <outbound>
    <name>ldap-mobile</name>
    <description>
      Mapping that sets value for LDAP mobile attribute based on
      user's telephone number.
    </description>
    <source>
      <path>$focus/telephoneNumber</path>
    </source>
  </outbound>
</attribute>

```

Mappings can become quite complex. There may be multi-line scripting expression in the mapping and it may not entirely obvious what is the input and output. Therefore each mapping and each expression have an ability to enable tracing:

```

<attribute>
  <ref>ri:mobile</ref>
  <outbound>
    <trace>true</trace>
    <source>
      <path>$focus/telephoneNumber</path>
    </source>
    <expression>
      <trace>true</trace>
      <script>
        <code>...</code>
      </script>
    </expression>
  </outbound>
</attribute>

```

If tracing is enabled then the mapping or expression execution will be recorded in the log files. Tracing can be enabled at both mapping level and expression level. Mapping tracing is shorter. It provides overview of the mapping inputs and outputs. Expression-level tracing is much more detailed.

However, even this level of tracing may not be enough to debug expression code. Therefore there is a special expression function for logging. Arbitrary messages may be logged by script expression code:

```

<expression>
  <script>
    <code>
      ...
      log.info("Value of foo is {}", foo)
      ...
    </code>
  </script>
</expression>

```

Expression Functions

Expressions in general and scripting expressions in particular are the place where most midPoint customization happens. Scripting expressions are able to execute any code in a

general-purpose programming language. Therefore the script can transform the data in any way or execute any function. However, there are things that are frequently used in the scripts. Therefore midPoint provide convenient scripting libraries full of useful methods that the scripts can use.

There are two scripting libraries that are used often:

- **Basic script library** provides very basic functions for string operations, object property retrieval, etc. These are simple, efficient stand-alone functions. These functions can be used in every expression.
- **MidPoint script library** provides access to higher-level midPoint functions contain IDM-specific and midPoint-specific logic. This library can be used to access almost all midPoint functionality. But there are few places where this library may not work reliably (e.g. correlation expression).

The libraries are designed to be very easy to use from the scripting code. While the specific details how to invoke the library depend on the scripting language, the libraries are usually accessible by the use of `basic` and `midpoint` symbols. Function `norm()` from the basic library can be invoked in a Groovy script like this:

```
<expression>
  <script>
    <code>
      ...
      basic.norm('Guľôčka v jamôčke!')
      ...
    </code>
  </script>
</expression>
```

Invocation of the libraries from JavaScript and Python is almost the same and we are sure that a clever reader will have no trouble figuring that out. What is more difficult to figure out are the functions that are in the libraries. For that purpose there is a page in midPoint wiki that lists all the libraries and this page also has a link to library function documentation.

Only two libraries were mentioned in this section so far. However, this is not a whole story. A clever reader has certainly figured out that the logging function described in previous section is also a scripting library. And there may be more libraries in the future.

Resource Capabilities

Source and target systems that midPoint connects to are not created equal. In fact, those system significantly differ in their capabilities. Most systems can create accounts. But not all of them can delete accounts. There are systems that keep the accounts forever, they just can be permanently disabled. And yet another systems cannot enable or disable accounts. While most systems support password authentication, other system do not. And so on. And even the connector may introduce limitations. Even if target system supports a particular feature, connector may not have appropriate code to use it. MidPoint needs to

take all these differences into consideration when executing synchronization and provisioning operations.

MidPoint refers to these features of the systems and connectors as *resource capabilities*. Although capabilities may in fact be quite complex, they are essentially just a list of things that a connector and resource can do. MidPoint is aware of the resource capabilities. Therefore midPoint can present the data correctly. E.g. midPoint will not try to modify account on a read-only resource.

Capabilities are usually automatically discovered by midPoint and everything just work. There is usually no extra work to maintain the capabilities. But sometimes there is a need to tweak the capabilities a bit. Maybe the connector cannot detect resource capabilities well enough. Maybe there is a read-only resource, but the connector has no way of knowing this. Therefore the write capabilities have to be manually disabled in midPoint. For that reason there are two sets of capabilities:

- **Native capabilities** are capabilities detected by the connector. Those are always automatically generated by midPoint. Those capabilities should not be modified by administrator.
- **Configured capabilities** are the capabilities modified by the administrator. Configured capabilities are used to override native capabilities. Configured capabilities are usually empty, which means that only native capabilities are used.

There are many ways how the capabilities can be tweaked by the administrator. But there is one case that is particularly interested for synchronization and provisioning: simulated activation.

MidPoint connectors can be tailored specifically for a particular system, such as a connector for custom enterprise application. At the other side of the spectrum are generic connectors that can fit a wide variety of systems and applications. LDAP, CSV and database table are examples of such generic connectors. Such connectors are very useful and they are used in almost every midPoint deployment. However, there is no standardized way how to disable an account in database table or a CSV file. Various columns and various values are used to represent account activation status. There is even no standardized way how to disable an account in LDAP directory. However, midPoint takes a significant advantage from knowing whether account is disabled or enabled. Therefore there is way how to tell midPoint which attribute and what values are used to represent account activation status. Configured capability is used for that purpose:

```
<capabilities>
  <configured>
    <cap:activation>
      <cap:status>
        <cap:attribute>ri:active</cap:attribute>
        <cap:enableValue>true</cap:enableValue>
        <cap:disableValue>>false</cap:disableValue>
      </cap:status>
    </cap:activation>
  </configured>
</capabilities>
```

```
</configured>  
</capabilities>
```

Configured capability above specifies resource attribute `active` as the attribute that controls account activation status. If this attribute is set to value `true` then the account is enabled. If the attribute is set to value `false` then the account is disabled. That is it. Once this configured capability is part of resource definition then midPoint will pretend that the resource can enable and disable accounts. Attempt to disable account will be transparently translated to modification of attribute `active`. But it also works the other way around. If an account has attribute `active` set to `false` value midPoint will display that account as disabled. No extra logic or mapping is needed to achieve that. The capability does it all.

Synchronization Example: LDAP Account Correlation

Previous example demonstrated the use of synchronization for HR feed. That is obvious use of synchronization mechanisms. However, midPoint synchronization is much more flexible than just feeding data to midPoint. Synchronization can be used even for typical target resources. In that case the synchronization is typically used for several purposes:

- **Initial migration:** This is a process of connecting new resource to midPoint. There are usually accounts that already exist in the resource at the time when a resource is connected to midPoint. At least some of the accounts usually correspond to the users that are present in midPoint (e.g. users created from the HR feed). Therefore the accounts from the resource need to be correlated to the users that already exist in midPoint. Synchronization is the right mechanism for this.
- **Detection of illegal accounts:** Security policies are usually set up in such a way that only those people that need an account on a particular resource should have an account on that resource. This is known as the *principle of least privilege*. However, in typical IDM deployment there is nothing that would prohibit system administrator to create any accounts at will. And this is often desirable because there are emergency situations where full control over the system is crucial. But even for emergency cases we want to make sure that the situation is aligned with policies when the emergency is over. MidPoint can easily do that by scanning the target systems in regular intervals. Synchronization mechanisms can be used to detect accounts that do not have any legal basis and delete or disable such accounts. Again, synchronization mechanism can do that easily.
- **Attribute value synchronization:** Accounts in target resources are usually created as a result of midPoint provisioning action. However, account attribute values are in fact copies of the data in midPoint. Attribute values can easily be changed by system administrator, may be set to old values during data recovery procedure or they can get out of sync by a variety of other means. MidPoint can make sure that the attributes are synchronized and that they stay synchronized for a long time. Synchronization mechanisms are ideal for this purpose.

Older IDM systems used synchronization mostly to get data from the source resources to IDM system. But synchronization in midPoint is much more powerful. It can be applied to source systems and target systems, it can pull data, push data, detect inconsistencies and fix them. Synchronization is a general purpose mechanism. This is the principle of reuse again. Synchronization mechanism can be reused for variety of purposes.

In this example we will be using synchronization to connect existing LDAP server to midPoint. We assume that our midPoint is already connected to the HR system. We have imported the HR data. Now we have midPoint users for all our employees. And then there is this LDAP server. It is really important LDAP server. This server is used by company intranet portal and also by a variety of smaller web applications. Those application are using the LDAP server for user authentication and access authorization. The LDAP server was deployed few years ago. Initially it was populated by the HR data. But the LDAP server was managed manually by a system administrator during all these years. Therefore it is expected that there will be some accounts that belong to former employees. Also, it might have happened that some accounts are missing. And it is quite likely that some of the accounts have wrong data.

First task is to set up the connector for this resource. As LDAP servers are used for identity management purposed all the time, MidPoint comes with a really good LDAP connector. All we need is to set up the resource to use that connector:

```
<resource oid="8a83b1a4-be18-11e6-ae84-7301fdab1d7c">
  <name>OpenLDAP</name>

  <connectorRef type="ConnectorType">
    <filter>
      <q:equal>
        <q:path>connectorType</q:path>
        <q:value>com.evolveum.polygon.connector.ldap.LdapConnector</q:va
lue>
      </q:equal>
    </filter>
  </connectorRef>
```

What we can see here is a slightly more sophisticated method to reference the connector. So far we have seen only a direct reference by OID. This works well for almost all the references in midPoint because OID never changes. But connectors slightly tricky. Objects that represent connectors are dynamically created by midPoint when a connector is discovered. Therefore the OID is generated at random at that time. There is no practical way how a system administrator can predict that OID. But we still want our resource definitions to refer to a particular connector when we import the definition. Therefore there is an alternative way how to specify object references. This method is using a search filter instead of direct OID reference. When this resource definition is imported to midPoint then midPoint will use that filter and look for LDAP connector. If that connector is found then the OID of that connector is placed in the reference (connectorRef). So next time midPoint will be using this resource it can follow the OID directly. This is a very convenient method. But there are few limitations. Firstly, the filter is resolved only during import. That means it

is resolved only once. If the connector is not present at import time then the reference needs to be corrected manually. Secondly, this approach works if there is only one LDAP connector deployed to midPoint. This is usually the case. But the connector framework can contain several connectors of the same type in different versions. This is a very useful feature for gradual connector upgrades, testing of new connector versions and so on. But in case that the filter matches more than one object the import will fail. In that case the connector reference has to be set up manually.

Once we have proper reference to LDAP connector we need is to configure the resource so the connector can access our LDAP server:

```
<connectorConfiguration>
  <icfc:configurationProperties>
    <cc:port>389</cc:port>
    <cc:host>localhost</cc:host>
    <cc:baseContext>dc=example,dc=com</cc:baseContext>
    <cc:bindDn>cn=idm,ou=Administrators,dc=example,dc=com</cc:bindDn>
    <cc:bindPassword><t:clearValue>secret</t:clearValue></cc:bindPassword>
    ...
  </icfc:configurationProperties>
  <icfc:resultsHandlerConfiguration>
    <icfc:enableNormalizingResultsHandler>false</icfc:enableNormalizingR
resultsHandler>
    <icfc:enableFilteredResultsHandler>false</icfc:enableFilteredResults
Handler>
    <icfc:enableAttributesToGetSearchResultsHandler>false</icfc:enableAt
tributesToGetSearchResultsHandler>
  </icfc:resultsHandlerConfiguration>
</connectorConfiguration>
...
```

This is all very similar to the configuration of the other resource that were already presented in this book. It should be quite self-explanatory – except perhaps for the configuration of result handlers. *Result handlers* are little helpers that come with the ConnId connector framework. The purpose of the result handlers is to assist simpler connectors in filtering and post-processing search results. But LDAP connector is no ordinary simple connector. LDAP connector is mature and full-featured connector that can do everything without any help from such annoying creatures as those result handlers. ConnId result handlers do not add any value here. In fact they may even be harmful. LDAP protocol has a lot of peculiarities such as case-insensitivity that applies to almost all the aspects of LDAP data – except for some notable exceptions. The connector is aware of those peculiarities but the handlers are not. Therefore if the handlers are turned on (which is the default) they may get in the way and ruin the data. Therefore it is always strongly recommended to explicitly turn off the handler when a full-featured connector is used.

Note: the XML example above, as all other examples in this book, is simplified and shortened for clarity. You will not be able to import the example in this form into midPoint. For a full importable examples see the files that are supposed to accompany this book. Please see additional information chapter.

The basic resource configuration above is sufficient to connect to the resource. Therefore the test connection button on resource details page should be successful. This configuration may also be used to list the accounts. However, LDAP servers support many object classes and midPoint does not yet know which object class represents account. Therefore we need to schema handling section to our resource:

```
<schemaHandling>
  <objectType>
    <kind>account</kind>
    <displayName>Normal Account</displayName>
    <default>true</default>
    <objectClass>ri:inetOrgPerson</objectClass>
    <attribute>
      <ref>ri:dn</ref>
      <displayName>Distinguished Name</displayName>
      <limitations>
        <minOccurs>0</minOccurs>
      </limitations>
      <outbound>
        <source>
          <path>$focus/name</path>
        </source>
        <expression>
          <script>
            <code>
              basic.composeDnWithSuffix('uid', name,
'ou=people,dc=example,dc=com')
            </code>
          </script>
        </expression>
      </outbound>
    </attribute>
  ...
</schemaHandling>
```

There should be outbound mapping for each mandatory LDAP attribute for the `inetOrgPerson` object class. Those mapping are typical for a target resource definition. Once we do this we should be able to conveniently list LDAP accounts in midPoint. However, we need to switch to the “Resource” view instead of “Repository” view. The accounts are stored in the LDAP server and midPoint can access them. Therefore the accounts are listed in the “Resource” view. But midPoint have not processed the accounts yet. Therefore there are no account shadows in midPoint repository. And that is the reason that the “Repository” view is empty. But now we are going to do something about it.

We are going to import (or reconcile) the resource accounts. But if we try to do this now nothing would really happen. The accounts are not linked to users therefore midPoint will not synchronize the attributes. And midPoint was not told to do anything with the accounts. Therefore midPoint will do nothing. That is one of midPoint principles: midPoint will not change the accounts in any way unless it is explicitly told to do so. Default midPoint configuration is to do nothing. We would rather do nothing than to destroy the data.

Before we can import the accounts we need to set up the synchronization configuration for this resource. There are accounts in the LDAP server that should belong to users that are

already in midPoint. We want to link them, but we do not want to do the linking manually. We would rather set up a correlation expression that does this automatically:

```
<synchronization>
  <objectSynchronization>
    <objectClass>ri:inetOrgPerson</objectClass>
    <kind>account</kind>
    <intent>default</intent>
    <focusType>UserType</focusType>
    <enabled>true</enabled>
    <correlation>
      <q:equal>
        <q:path>employeeNumber</q:path>
        <expression>
          <path>$account/attributes/employeeNumber</path>
        </expression>
      </q:equal>
    </correlation>
  </objectSynchronization>
</synchronization>
```

This correlation expression is going to match account attribute employeeNumber and user property that is also named employeeNumber. Simply speaking: if account and user employee numbers match then we assume that they should be linked. In that case midPoint decides that synchronization situation is unlinked (they should be linked, but they are not yet linked). We want midPoint to link the account in this case, therefore we define appropriate reaction:

```
<reaction>
  <situation>unlinked</situation>
  <synchronize>true</synchronize>
  <action>
    <handlerUri>http://midpoint.evolveum.com/xml/ns/public/model
/action-3#link</handlerUri>
  </action>
</reaction>
```

This will take care of accounts for whose we can find an owner. But what to do with other accounts? For now we will do nothing. Therefore we do not need to define any other reactions. This may be somehow surprising. We do not want illegal accounts, do we? So perhaps we want to define a reaction to delete unmatched accounts. That is a good approach, but it is just too early for this. We do not want to delete unmatched account just yet. There may be accounts that are perfectly legal, just the employeeNumber attribute is missing or mistyped. So, we will go just with this one reaction for now.

Now it is the right time to start import or reconciliation task. After the task is finished the situation may look like this:

OpenLDAP
UP

Details
Defined Tasks
Accounts
Entitlements
Generics
Uncategorized
Connector

Intent
(Object Class: inetOrgPerson)
Search In:
Repository
Resource

Name: All
More...
Advanced

	Name	Identifiers	Situation	Intent	Owner	Result	
<input type="checkbox"/>	uid=anderson,ou=People,dc=example,dc=com	entryUUID: ca3425f4-5727-1036-9dd9-0362a137a7f9 dn: uid=anderson,ou=people,dc=example,dc=com	LINKED	default	001		
<input type="checkbox"/>	uid=brown,ou=People,dc=example,dc=com	entryUUID: ca4f66de-5727-1036-9dda-0362a137a7f9 dn: uid=brown,ou=people,dc=example,dc=com	LINKED	default	002		
<input type="checkbox"/>	uid=carol,ou=People,dc=example,dc=com	entryUUID: ca674c36-5727-1036-9ddb-0362a137a7f9 dn: uid=carol,ou=people,dc=example,dc=com	LINKED	default	003		
<input type="checkbox"/>	uid=davies,ou=People,dc=example,dc=com	entryUUID: ca83d59a-5727-1036-9ddc-0362a137a7f9 dn: uid=davies,ou=people,dc=example,dc=com	LINKED	default	004		
<input type="checkbox"/>	uid=evans,ou=People,dc=example,dc=com	entryUUID: ca8d7c12-5727-1036-9ddd-0362a137a7f9 dn: uid=evans,ou=people,dc=example,dc=com	LINKED	default	005		
<input type="checkbox"/>	uid=fox,ou=People,dc=example,dc=com	entryUUID: ca968460-5727-1036-9dde-0362a137a7f9 dn: uid=fox,ou=people,dc=example,dc=com	LINKED	default	administrator		
<input type="checkbox"/>	uid=green,ou=People,dc=example,dc=com	entryUUID: ca9f7ad4-5727-1036-9ddf-0362a137a7f9 dn: uid=green,ou=people,dc=example,dc=com	LINKED	default	007		
<input type="checkbox"/>	uid=harris,ou=People,dc=example,dc=com	entryUUID: caa70d44-5727-1036-9de0-0362a137a7f9 dn: uid=harris,ou=people,dc=example,dc=com	LINKED	default	008		
<input type="checkbox"/>	uid=irvine,ou=People,dc=example,dc=com	entryUUID: cab2f69a-5727-1036-9de1-0362a137a7f9 dn: uid=irvine,ou=people,dc=example,dc=com	UNMATCHED	default			

It looks like had quite a good data in the LDAP server. Most of the accounts were successfully correlated and linked to their owners. But there are few accounts that were not correlated. Those accounts ended up in unmatched situation. You can resolve this situation by manually linking the unmatched accounts to their users. Simply click on the cog button next to the unmatched entry and select “Change owner” from the context menu. Then select the right user (Isabella Irvine) in the dialog that appears. After that the account is linked to the user. Repeat this process to link all unmatched accounts.

There is one interesting thing in the screenshot above. Have a look at the LDAP account identified by uid=carol. While most other accounts have their uid value taken from the surname of the user, this account is an exception. Even though the uid is obviously wrong, midPoint have linked the account correctly to the user (Carol Cooper). The reason is that we have set up midPoint to use employeeNumber for correlation. The result is that even accounts whose usernames violate the convention can be automatically linked to their owners – as long as there is any reliable piece of information that can be used for correlation.

When all the accounts are all linked to their owners it is the right time to complete the synchronization policy. Now we can tell midPoint to disable any unmatched account. That is the case when an illegal account is created in LDAP server. We can also tell midPoint to unlink any account that was deleted in LDAP server:

```
<reaction>
  <situation>unmatched</situation>
  <action>
    <handlerUri>http://midpoint.evolveum.com/xml/ns/public/model
/action-3#deleteShadow</handlerUri>
  </action>
</reaction>
<reaction>
  <situation>deleted</situation>
  <action>
    <handlerUri>http://midpoint.evolveum.com/xml/ns/public/model
/action-3#unlink</handlerUri>
  </action>
</reaction>
```

There may be some accounts in the LDAP server that have wrong attribute values. By “wrong” we mean that they have different values than the values that are computed by the outbound mappings. But midPoint will not correct those values just yet. Remember the midPoint principle that it will not change the account unless we have explicitly told to do so? Those accounts are in the `linked` situation. And we have not configured any reaction for this situation. Therefore now we need to tell midPoint to synchronize the values:

```
<reaction>
  <situation>linked</situation>
  <synchronize>true</synchronize>
</reaction>
```

A clever readers is now surely wondering whether we have forgotten something. And indeed we have. Attribute values are synchronized by running reconciliation process. But our outbound mappings will not work in reconciliation. They do not have any explicit definition of strength, therefore midPoint assumes *normal* strength. Those mappings are supposed to implement the “last change wins” strategy. Therefore reconciliation cannot overwrite the account data as midPoint does not know whether it was account attribute or user property that was the last to change. If midPoint is not sure about something then it will do nothing. We do not want to destroy the data. Therefore what we need to do now it so let midPoint know that we really mean it, that the mappings are really *strong*:

```
<attribute>
  <ref>ri:cn</ref>
  <displayName>Common Name</displayName>
  <limitations>
    <minOccurs>0</minOccurs>
    <maxOccurs>1</maxOccurs>
  </limitations>
  <outbound>
    <strength>strong</strength>
    <source>
      <path>$focus/fullName</path>
    </source>
```

```
        </outbound>
    </attribute>
    ...

```

Clever reader is once again uneasy. What is this `limitations` thing here? Simply speaking, the `limitations` specify that the attribute is optional (`minOccurs=0`) and that it is single-valued (`maxOccurs=1`). But, isn't `midPoint` supposed to be completely schema-aware and figure that all by itself? Yes, it is. And that is the reason why we need to override the information from the schema using this `limitations` element here. The `cn` attribute is specified in LDAP schema as mandatory. However, we have just specified outbound mapping for that attribute. Therefore even if `midPoint` user does not provide any value for attribute `cn` we can still determine that value by using the expression. Therefore even though LDAP schema specifies attribute `cn` as mandatory, we want to present that attribute as optional in `midPoint`. Hence the `minOccurs` limitation. And the `maxOccurs` limitation is surely immediately obvious to anyone who is intimately familiar with LDAP peculiarities. In the LDAP world everything is multi-valued by default. Therefore even commonly used attributes for account identifiers and names are multi-valued. Nobody is really using them as multi-valued attributes because vast majority of applications will probably explode if they ever encounter two values in the `cn` attribute. But those attributes are formally defined as multi-valued in LDAP schema and that is what `midPoint` gets from LDAP connector. The `maxOccurs` limitation is overriding the schema and forcing `midPoint` to handle this attribute as if it was single-value.

That is all. Now you can schedule reconciliation tasks to keep an eye on the LDAP server, correct any attribute values that step out of line and delete any illegal accounts. This is how synchronization tasks can be useful even in case of pure target resources.

However, there is one last word of warning. Those accounts were synchronized and linked to existing `midPoint` users. The accounts were not created by `midPoint`. Therefore there is nothing in `midPoint` that would say that those accounts should exist. In `midPoint` parlance there is no *assignment* for those accounts. `MidPoint` makes clear distinction between policy and reality. Therefore `midPoint` is aware that those accounts exist, but there is no policy statement that would justify their existence. By default `midPoint` does nothing and it will let the accounts live. By default the accounts will be created or deleted only if there is an explicit change in the assignments. But this is a fragile situation. Accounts that are linked but not assigned can easily get deleted if `midPoint` administrator is not careful. Of course, there are methods to handle such situations. One way would be to create the assignments together with the links. Those that are interested in this method should look up keyword "legalize" in `midPoint` wiki. But there are much better methods how to handle this. Perhaps the best approach would be to utilize the roles (RBAC). Which is the topic of the next chapters. But there are still more things to learn about synchronization until we get there.

Reconciliation

Reconciliation is a process of comparing that state of an account that it is (reality) to a state of the account as it should be (policy). Reconciliation does not only compare the accounts, it is fixing the inconsistencies. Therefore reconciliation can correct wrong data on resources. But it also works the other way. It can correct the data in midPoint. Therefore reconciliation is one the most useful tools in the identity management toolbox.

Reconciliation can be used in a variety of ways in midPoint. Reconciliation can be initiated for one specific user by using midPoint user interface. In that case midPoint will compare the values of all user's accounts to the values that were computed using the mappings. If there is difference midPoint will correct account values. This approach is perfect for testing reconciliation setting on just a single user. This feature is also useful for fixing values of one specific user.

Reconciliation of a specific user may be useful, but it is an ad-hoc approach. But in identity management we usually favor systemic approach. Therefore reconciliation can be used in a form of a reconciliation task. Reconciliation task will list all the accounts on the resource and then it will reconcile each account, one by one. This is a way how to keep all resource accounts synchronized.

There is a couple of things about reconciliation that can be somehow surprising. Firstly, reconciliation of an account may cause modification of a user. This happens if there are inbound mappings for that account and this is perhaps quite expected. But if user is changed then such change may propagate to other accounts, usually by the means of outbound mapping. MidPoint does not like procrastination and therefore it will try to execute those changes immediately. Therefore reconciliation of one account may cause changes to other accounts. Secondly, reconciliation will skip any normal-strength mappings. We have already explained the reasons for that, but this is something that can surprise even an experienced midPoint engineer from time to time. If we are sure that we want the mapped value to be present in the account all the time then strong mappings are the way to go.

A curious reader that has already explored midPoint user interface has surely noticed "recompute" function. What recompute does looks almost exactly the same as reconciliation. But there are subtle differences. Recompute will not force the fetch of account data. In this case the account attributes will be fetched from the resource only if midPoint inevitably needs them for the computation. This usually happens if *weak* or *strong* mappings are used. But if there are *normal* mappings only then recompute may not read account data. MidPoint will compare and correct account attribute values only for those accounts that are fetched from resource during this process. That is how recompute works. The purpose of a recompute is to correct data of midPoint *users*. Correcting account data is more or less just a side effect of a recompute. On the other hand, reconciliation will try to read all the accounts regardless whether they are needed for computation or not. Therefore all the attributes on all the accounts are fixed. That is the purpose of reconciliation: correct the account data.

There is yet another difference between recompute and reconcile *tasks*. The purpose of a recompute task is to correct user data. Therefore recompute task will iterate over midPoint users. Therefore recompute task will not detect new accounts on the resource and it may even overlook if an account is deleted. But reconciliation task is different. In fact reconciliation task has several stages. First stage will list all resource accounts. It will determine owner of each account, compare the attributes and correct them. But as this process iterates over real accounts on a resource, it can also detect new accounts. When the first stage is completed then a second phase starts. Second phase is looking at account shadows stored in midPoint. The task will look for shadows that have not been processed in the first phase. Those are accounts that used to be on the resource some time ago but that have disappeared. That is how reconciliation detects deleted accounts.

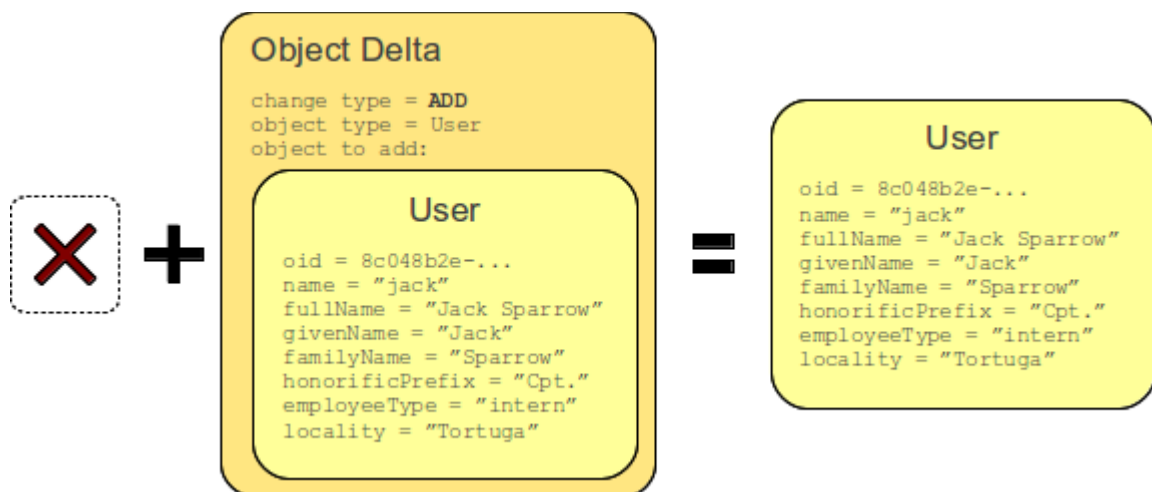
Deltas

Reconciliation is really useful mechanism. It is reliable and thorough, but it is also quite slow and it consumes a lot of computation and network resources. And there are reasons why reconciliation is such a heavyweight beast. Reconciliation works with *absolute state* of accounts. It means that reconciliation is reading all the accounts with all the values of all the attributes. Then it recomputes everything. Even those attributes and values that were not changed. This is a very traditional and reliable way of computation and that is also the way how most older identity management systems work.

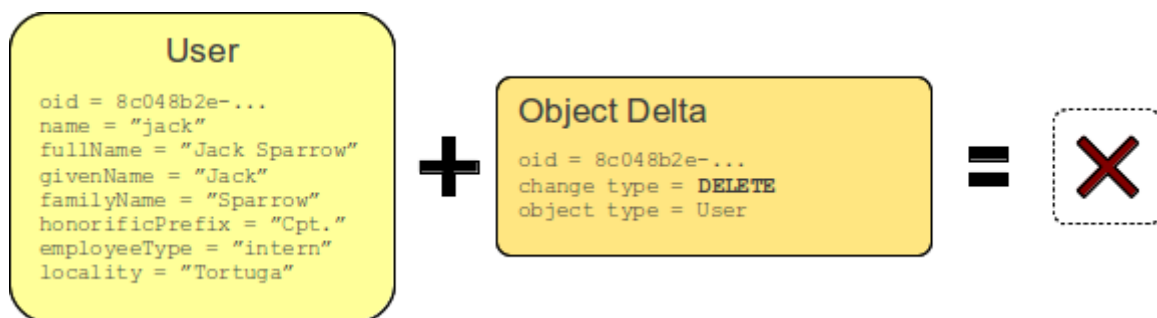
But there is also a better way. If we know that just one attribute was changed we can recompute just that one attribute. We do not need to care about other attributes. And if we know that attribute foo has changed in such a way, that there is a new value bar then it gets even better. We just need to recompute the value bar and do not care about anything else. This is what we like to call a *relative change*. We just care about what was changed. And we do not care that much about the attributes and values that were not changed. That is how midPoint works internally. MidPoint is *relativistic*.

This is where *delta* comes in. Delta is a data structure that describes the change of a single midPoint object. *Add delta* describes a new midPoint object that is about to be created. *Modify delta* describes existing midPoint object where some properties have changed. *Delete delta* describes an object that is going to be deleted. This is a very powerful mechanism. Just remember that everything in midPoint can be represented as an object: user, account, resource, role, security policy ... everything. Therefore delta can represent any change. It may be a change of user password, deletion of an account, change of connector configuration or introduction of a new password policy. If all the changes can be represent in a uniform way then they can also be handled in a uniform way. Therefore it is easy for midPoint to record all the changes in an audit trail – including configuration changes. It is easy to route any change through an approval process. And so on. MidPoint can create a relatively simple mechanisms to handle changes and then those mechanisms can be applied to changes of (almost) any object.

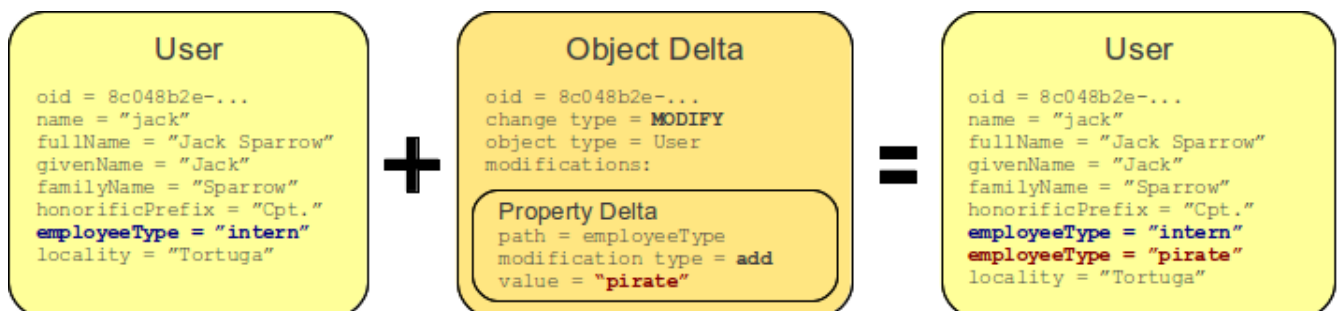
Let's have a closer look at an anatomy of a delta. There three types of delta: *add*, *modify* and *delete*. *Add delta* is quite simple. It contains a new object to be created.



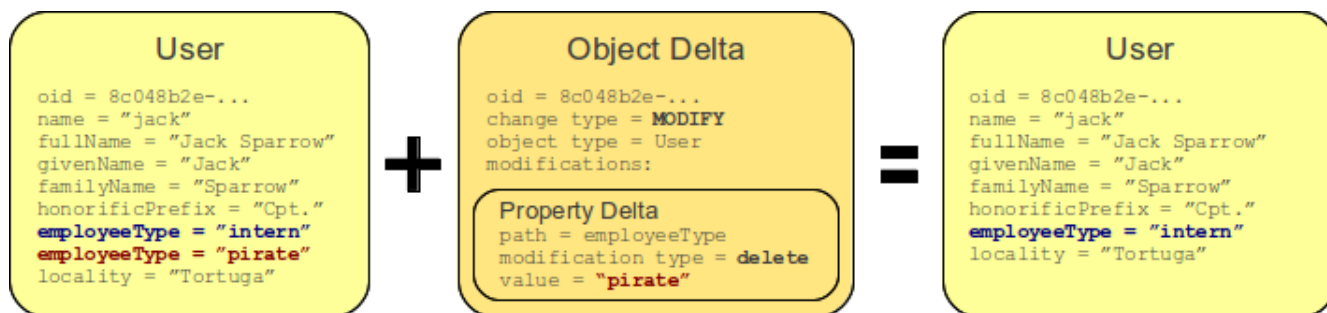
Delete delta is even simpler. It contains just object identifier (OID) of an object to be deleted.



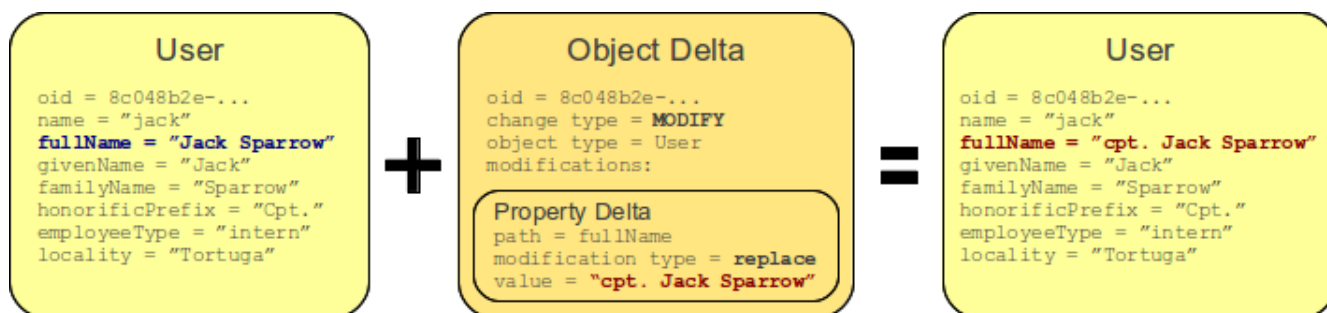
Last one is *modify delta*. This delta contains a description of modified properties of an existing object. But as the object can change in a variety of ways the *modify delta* is the most complex of the tree. Modify delta contains a list of *item deltas*. Each *item delta* describes how a particular part of an object changes. For example following delta describes that a new value *pirate* is added to a user property *employeeType*.



The item delta may have three modification types: *add*, *delete* and *replace*. *Add modification* means that new value or values are added to an item. *Delete modification* means that value or values are removed from an item.



In both *add* and *delete* cases the values that are not mentioned in the delta are not affected. However, *replace modification* is different. This means that all existing values of the item are going to be discarded and they are replaced with the value or values from the delta.



The delta are designed to work with both single-valued and multi-valued items. In fact *add modification* and *delete modification* deltas are specifically designed with multi-value items in mind. Those deltas can work on efficiently even in cases that there is a multi-valued attribute that has a very large number of values. And there is a good reason for this. Multi-valued properties are quite common in the identity management field. Just think about how roles, groups, privileges and access control lists are usually implemented. Everybody that ever managed a large group in LDAP server will surely remember that experience in vivid colors. But midPoint is designed to handle situations like those.

Everything in midPoint is designed to work with deltas: user interface, mappings, authorizations, auditing ... all the way down to the data storage components. Mappings are designed in a relativistic ways. That is the reason why we need to explicitly specify sources of the mapping. Mapping source definitions are matched with items in the delta to control execution of the mapping. Deltas permeate entire midPoint computation. Deltas are input to the mappings, but mapping produce deltas as output. Therefore we can have a complete chain: deltas that are result of inbound mappings is applied to the user object, but those deltas are also input to outbound mappings. Everything is *relativistic* in midPoint.

This might seem complex at the beginning, but do not worry. You will get used to it. And clearly this approach has major advantages. But a clever reader does not seem to be impressed. How can this relativistic approach conserve any significant portion of computational resources? We usually fetch the entire account from the resource anyway. Therefore there is no harm to recompute all the attributes. The computation itself is fast, it is the fetch operation that is slow. Isn't it? The clever reader is indeed right. At least partially right. Most resources really fetch all the account attributes in a single efficient operation. And for those cases there is no big increase in efficiency if we go with the relativistic methods. But there are exceptions. For example some resources will not return all the values of big attributes, e.g. all the members of a large group. Additional requests are needed to fetch all the values – and there may indeed be a lot of requests if the group is really large. Relativistic approach has significant benefit in those cases. And the benefits will be even more obvious when we get to the live synchronization in the next section. But performance is not the primary motivation for the relativistic approach. There is one extremely strong reason to go relativistic: consistency. Consistency is something that brings ugly nightmares to many engineers that try to design distributed system. And identity management solution is in fact a distributed system. But it is a very loosely-coupled distributed system. There is no support for locking or transactions in the connector. And even if there was some support, vast majority of resource cannot provide those consistency mechanisms on their identity management APIs. This means that midPoint cannot rely on traditional consistency mechanisms. And that is where relativistic approach is so useful. Relativistic computation has a very high probability of achieving correct result even without locking or transactions. This is more than acceptable for typical identity management deployments. And for those rare cases where relativistic computation can fluctuate there is always reconciliation as a last resort. But thank to the relativistic nature of midPoint the need for reconciliation is significantly reduced.

That was a lot of long words, but clever reader seems to be satisfied now. At least for a while. But there is quite a simple summary: relativistic approach of midPoint can do miracles. For example, midPoint resource can be both sources and targets, even a single attribute can be both source and target of information. It is the relativistic approach that allows features like this. The principle of relativity is relatively simple. But its effect in midPoint is nothing short of being revolutionary.

Live Synchronization

MidPoint has a range of synchronization mechanism. Slow, brutal but reliable reconciliation is at one end. Live synchronization is on the other. Live synchronization is a lightweight mechanism that can provide almost-real-time synchronization capabilities. Live synchronization is specifically looking for recent changes on a resource. When such changes are detected, live synchronization is designed to process those changes immediately. When live synchronization is used properly then the synchronization delay is usually in order of seconds or minutes.

Unlike reconciliation, live synchronization cannot be triggered manually. Triggering it manually makes very little sense. Live synchronization works in a long-running task repeatedly looking for fresh changes in short time intervals. If the resource is configured for synchronization then all that is needed to run live synchronization is to set up a live synchronization task. MidPoint user interface can be used to do that easily. And an example of live synchronization task was provided in the HR feed section above.

Live synchronization task wakes up at regular intervals. Each time the task wakes up it invokes the connector. Connectors that are capable of live synchronization have special operation that is used to get fresh changes from the resource. The connector can support any reasonable mechanism in theory. But two mechanisms are commonly used by the resources:

- **Timestamp-based synchronization:** Resource keeps track of last modification timestamp for each account. The connector looks for all accounts that have been modified since last scan. This is very simple and relatively efficient method. But it has one major limitation: it cannot detect deleted accounts. If an account is deleted then there is no timestamp for that account and therefore the connector will not find it in the live synchronization scan.
- **Changelog-based synchronization:** Resource keeps a “log” of recent changes. The connector is looking at the log and it is processing all the changes that were added to the log since the last scan. This is a very efficient and flexible method. But it is not simple and only few resources support it.

All live synchronization methods need to keep the track of what changes are “recent”, i.e. which changes were already processed by midPoint and which were not processed yet. There is usually some value that needs to be remembered by midPoint: timestamp of last scan, last sequence number in the change log, serial number of last processed change and so on. Each connector has a different value with a connector-specific meaning. MidPoint refers to those values as “tokens”. The latest token is stored in the live synchronization task. That is how midPoint keeps track of processed changes. There are (hopefully quite rare) cases when resource and midPoint token get out of alignment. This may happen in cases such as the resource database is restored from a backup, if network time gets out of synchronization and so on. If that happens then deleting the token from the live synchronization task is usually all it takes to get the synchronization running again.

Live synchronization is fast and very efficient. But it is not entirely reliable. MidPoint may miss some changes. This is quite a rare situation, but it may happen. Reconciliation will surely remedy the situation in such a case. Just remember, all the synchronization mechanisms share the same configuration. And it is perfectly acceptable to run live synchronization and reconciliation on the same resource at the same time. But of course, it would be a good idea to run reconciliation less frequently than live synchronization.

Conclusion

Synchronization is one of the most important mechanisms in the entire identity management field. Primary purpose of synchronization is to get the data into midPoint. And that is really good approach when an identity management deployment begins: get your data into midPoint first. Get the data from the HR system. Correlate that with Active Directory. Connect all the major resources to midPoint and correlate the data. MidPoint does not need to make any changes at this stage. In fact it is perfectly good approach to make all the resource read-only at this stage. The point is to let midPoint see the data. But why do we need that?

- We will see what is the real quality of the data. Most system owners have at least some idea what data sets are there. But it is almost impossible to estimate data quality until the data are processed and verified. That is exactly what midPoint can do at this stage. This is essential information to plan data cleanup and sanitation.
- We will learn how many accounts and account types are there. It is perhaps quite obvious that there are employee accounts. But are there accounts for contractors, suppliers, support engineers? Are those accounts active? What is the naming convention? Do system administrators use employee accounts for administration. Or are they using dedicated high-privilege accounts? This information is crucial to set up provisioning policies.
- We will learn distribution of accounts and their entitlements. Do all employees have accounts in Active Directory? Are there any frequently-used groups? How does organizational structure influence the accounts? This information is very useful to design a role-based access control structures and other policies.
- We will surely learn some security vulnerabilities. Are there orphaned accounts that should have been deleted long time ago? Are there testing accounts that were left unattended after the last night-time emergency? Indeed, there is no security without identity management.

This is a good start. But even if this is all that you do in the first step of the deployment it is still a major benefit. You will get better visibility and with that comes better security. And you have the data to analyze your environment and plan next step of the identity management deployment. You won't be blind any longer. And that is really important. It is indeed a capital mistake to theorize before one has data.

Chapter 7: MidPoint Development, Maintenance and Support

*Those who do nothing but observe from the shadows
cannot complain about the brightness of the sun.*

– Frank Herbert

MidPoint is a *professional open source* project. This means that midPoint is developed by using *professional* methods, but the product is still available under *open source* license.

Professional Development

MidPoint is developed by a professional developers. The development is lead by senior developers in the midPoint core team that have decades of software engineering experience. There are also few younger developers in early stages of their careers. MidPoint development team is first of all a community of developers that enjoy working together and developing next-generation software. Professionalism is a strict requirement for all midPoint development, but it is mostly the engineering passion that really moves the project forward.

All midPoint core developers work for Evolveum. Evolveum is the company that created midPoint. Evolveum also maintains midPoint. Vast majority of work on midPoint is being done by midPoint core team. All the core developers are paid for their work on midPoint. The developers can pay their bills from the income that midPoint generates. Evolveum income from midPoint makes sure that the developers have all their time available for midPoint development. This means that midPoint is properly maintained.

Professional development also means that professional software engineering methods are used to develop and maintain midPoint. MidPoint development is firmly founded on principles of continuous integrations. There are literally thousands of automated integration tests integrated directly into midPoint build process. Thousands of other automated tests are running every day. There are tests that closely reflect real-world configurations. There are tests with real resources. Everything is an integral part of midPoint development. MidPoint is a comprehensive and very flexible system. Professional quality assurance is essential for midPoint to work reliably.

Open Source

MidPoint is an open source project. This means that all of midPoint source code is available under an open source license. We have chosen Apache License 2.0 as this is one of the most liberal licenses out there. But there is much more to open source than just a license. Evolveum is fully committed to the open source approach. MidPoint is completely developed in public. Entire history of midPoint source code is public. Every

commit of every developer is immediately available to anyone. Complete midPoint source code is available. There are no private parts that are held back by purpose. There are no private branches with extra features. Even all the support branches are completely public. When it comes to the source code midPoint is as true to the open source methods as it gets.

Even though vast majority of midPoint development is done by Evolveum, open source is absolutely critical for the success of midPoint. Open source is the only way that allows midPoint users to understand midPoint completely. All non-trivial software needs to be customized in some way and open source brings the ultimate power of customization. Open source allows participation. Open source is great approach to avoid vendor lock-in. Open source brings longevity to the project. Open source has so much advantages. Evolveum is completely committed to the open source approach.

MidPoint has started as an open source project. MidPoint source code was available from the day one. And as far as we have something to say about it midPoint will remain open source forever.

MidPoint Release Cycle

MidPoint has stable development cycle. There are two *feature releases* every year. As the name suggest those releases are bringing new features and major improvements. In addition to that there are several *maintenance releases*. Those releases bring bugfixes and minor improvements. Maintenance releases are published as needed, there is no strict schedule. Timing of maintenance releases is influenced by midPoint subscribers.

MidPoint Subscriptions

MidPoint subscription is a service provided by Evolveum. There are several subscription offering with different scope and service level. But generally speaking most part of subscription service is 3rd-line support. Which basically means that we will fix midPoint bugs. Obviously this includes assistance with diagnostics of difficult issues where it is not entirely clear whether it is a bug or configuration issue. Simply speaking, midPoint subscription is a way how to make sure that your midPoint deployment will run without any problems. There are also subscription offerings designed to help you deploy midPoint in the first place. Some subscription offerings also contain feature development and improvements (a.k.a. 4th-line support). Those subscriptions are ideal way to make sure midPoint will be able to do anything that you need for your project.

MidPoint subscriptions provide significant funding for midPoint development and maintenance. Therefore it is perfectly natural that midPoint subscribers get high priority for resolution of their issues, feature requests and so on. This limits the time that midPoint core team has available for other tasks. Therefore there are some rules:

- Every new midPoint feature must be *sponsored*. This means that a customer with an active midPoint subscription has endorsed the feature. Of course, this has to be a high-level subscription that includes feature development. Or, alternatively,

someone have to pay for the development cost of that feature. However, direct feature sponsoring is very limited as most of the midPoint development capacity is reserved for subscribers.

- MidPoint architecture and quality is the primary responsibility of Evolveum team. Part of Evolveum income is reserved to maintain midPoint – to keep the architecture up to date, to make systemic quality improvements, to maintain midPoint in the long run and so on. In rare cases Evolveum will sponsor some features. Those are usually strategic features that point midPoint development in the right direction. Or those may be experimental features mean to explore a particularly interesting functionality. However, Evolveum will not sponsor any “customer” features. Those need to be covered by subscriptions.
- Evolveum will eventually fix any bugs in midPoint. Those bugfixes will be committed to midPoint primary development branch (*master branch*). The fixes that make it will be part of the next feature release. However, as midPoint release cycle is fixed, not all of the bugs will be fixed in each release. The bugs that affect subscribers will be fixed first. If there is still time then other (non-subscriber) bugs will be fixed as well. But there are no guarantees that that. If the time before the release runs out features reported by non-subscribers will not get fixed. In fact such non-subscriber fixes may have to wait for several releases until they finally get fixed.
- Every feature release has a *support branch*. This is where the maintenance releases come from. However, every bugfix or improvement is developed on master branch first. It has to be backported to the support branch. Which takes time. Therefore there are very strict rules. Any bugfix, improvement or any other update will go to the support branch only if:
 - Backport to support branch is explicitly requested by midPoint subscriber.
 - It is a security issue. Security issues have absolute priority. Those will be fixed immediately regardless of who reported them (subscriber or non-subscriber). Security fixes will also get automatically backported to any active support branch.
 - It fixes a severe issue that affect large number of users.

Simply speaking, if you want to make sure that midPoint works for you then get a subscription. Subscription will help you. That what subscription is for. But the money from subscriptions also enable long-term midPoint maintenance and new feature development. Getting midPoint subscription is the right thing to do.

MidPoint Community

MidPoint is a good open source project. And as all good open source projects midPoint has a vibrant community. This is both engineering community and business community. The primary communication channel of the engineering community is midPoint mailing list.

Mailing list is used to discuss midPoint future, announce new release, discuss configuration issues, provide feedback to the development team and so on. MidPoint community is open to anyone.

Business community is formed mostly from Evolveum partners. Evolveum partners deliver midPoint solutions, provide 1st-line and 2nd-line support services, provide professional services, customized solutions based on midPoint and so on. The possibilities are endless. Even the business community is open. Entry-level partnership is open to anyone. However, there are several partnership levels and it takes some effort for a partner to level up. There is a rich (and growing) network of midPoint partners. The partners can deliver solutions based on midPoint almost anywhere on planet Earth.

Chapter 8: To Be Continued

Hanc marginis exiguitas non caperet.

(There is not enough space in the margin to write it.)

– Pierre de Fermat

That is it then. That was the last real chapter of this book. But that we have not yet covered all capabilities of midPoint. In fact we are not even close. We have only scratched the surface of what midPoint can provide. The other chapters are not written yet. There is still so much to write about:

- **Role-based access control (RBAC):** Assignments, roles, conditional roles, parametric roles, role-based provisioning and meta-roles. If you like roles you should really look forward to this because we will be putting roles into roles, make roles for roles and quite literally take the role-based concepts to the nth.
- **Schema:** MidPoint has quite a rich identity data model by default. MidPoint user has a lot of useful properties to choose from. But no default data model is ever good enough. It needs to be extended. So this chapter is going to be just about that: how to extend user properties ... and also roles and essentially any other midPoint object.
- **Object template:** MidPoint can automatically recompute objects, e.g. to compute full name from first name and last name. But this simple concept can be used to a variety of purposes: assign roles automatically, look up values, consolidate values from several sources, get the values from external services and so on.
- **Organizational structure:** Organizations, divisions, departments, but also projects, teams, ad-hoc workgroups, realms, tenants and in fact anything else that you can imagine. This is all about organizing people in groups and applying policies. But midPoint organizational structure is even more flexible than that. Even roles and other midPoint objects can be organized. Organizational structures in all their forms are fundamental building blocks that can be used to create really interesting configurations.
- **Authorizations:** MidPoint works with sensitive data and there is a need to strictly control access to that data. Therefore midPoint has a fine-grained authorization system for controlling access to the midPoint itself. Authorization mechanisms are very powerful allowing many scenarios from delegated administration to partial multi-tenancy.
- **Identity management miscellanea:** There are various interesting features that were not mentioned yet: iteration, password policies, notifications, auxiliary object

classes, provisioning dependencies, deputies, constants, function libraries, provisioning scripts and so on. Those may be little features, but they are essential pieces of the puzzle. It is almost impossible to have a complete identity management solution and not to use any of those features.

- **Workflow and approvals:** Browse available roles, select role, request role, approve role, assign role, provision accounts. That is the basic mantra of many identity management deployments. Of course this is easy to do in midPoint. But there is much more: multi level approvals, optional approval steps, dynamic approver selection, escalation and so on. MidPoint has most of the features already built-in, you just need to configure them.
- **Entitlements:** Managing accounts is fine. But it is not the whole story. There is huge difference between a regular account and an administrator account. Fortunately, midPoint can easily manage membership in groups, roles, assignment of privileges and other entitlements. In this case we really mean entitlements on the resources, such as Active Directory groups, distribution lists, Unix groups and so on. MidPoint is designed to this quite easily.
- **Generic synchronization:** Any self-respecting identity management system can synchronize accounts and users. But what about roles, organizational units, groups and things like that? You guessed it, midPoint can do it. In fact, the same principles that are used for user-account synchronization can be directly reused. Do you want your departments to automatically appear as OU objects in Active Directory? Here you go. Do you want to place the users there. That is easy. Do you also want to create nested LDAP groups for the departments. No problem. MidPoint can do it all.
- **Manual resources:** Obviously, you want most of the resource to be connected to midPoint by a connector so they can be automatically managed. But there are always few bothersome resources that just won't comply. Maybe they are too small to justify the cost of building a connector. Maybe there is just no good way for the connector to manage the resource. But midPoint one again comes to the rescue. MidPoint has a concept of *manual resource* where the work is done by system administrator instead of connector. There is even a way how to create semi-manual resource that can read the data, but provisioning is still manual. And there is a way how to integrate this with ITSM system.
- **Auditing and history:** No identity management system can be complete without an auditing facility. MidPoint can store every operation to the audit trail: changes in users, accounts, roles – even internal configuration changes. This is stored in a format that can be used to integrate midPoint with a data warehouse or a SIEM system. Also midPoint user interface has a facility to display the audit trail. And it can even look into the past: it can reconstruct the objects as they were at a certain point in time.

- **Policy rules:** MidPoint is much more than just an identity management system. The identity governance features of midPoint are based on a powerful and universal concept of *policy rules*. The rules can be used to express role exclusions, thus defining a segregation of duties (SoD) policy. The rules can be used to define policy-based approval. The rules can control role lifecycle. The rules can define compliance policies. The rules can do it all. The rules are here to *govern* the identities.
- **Access certification:** This is known by many names: certification, re-certification, attestation, ... but whatever the name is it is still the same process. Simply speaking, this is a method to review roles assigned to the users to make sure the users still need the roles that they have. This is a method how to get a grip on the *principle of least privilege* even in environments that are naturally inclined to ad-hoc operation. But it is very useful mechanism in almost all environments. And midPoint provides many flavors of certification mechanisms from a scheduled mass recertification campaigns focused on roles assignments to an ad-hoc recertification of a single user after he is moved to a new organization.
- **Data protection:** Identity management is no longer a wild west where anybody can do anything. Now there are strict data protection rules, regulations and legislation. Being a good identity governance system midPoint can assist in managing the data protection and privacy policies. MidPoint can be really helpful in managing compliance to the data protection regulations such as European GDPR legislation.
- **User interface customizations:** MidPoint has a general-purpose user interface that can be used for user self-service, identity administration and system configuration. The user interface is designed to be dynamic. It will automatically adapt to resource schemas, extension of midPoint schema, authorizations and so on. Therefore usually there is no need to customize user interface at all. But there are cases when the deployment need to deviate from the default behavior. And midPoint is prepared to that. There are many ways how to customize user interface: colors, stylesheets, localization, custom forms, tabs, whole new custom pages. In extreme cases midPoint can be customized beyond recognition.
- **Integration with midPoint services:** MidPoint is a great system. But even great software does not live in isolation. There is always need to integrate the systems together. Integration runs through midPoint veins, because that is what the connectors really do. But often there is a need to integrate midPoint with other systems that is beyond the capabilities of a connector. Maybe there is a password reset application that needs to interact with midPoint. Maybe there is an analytic software that needs to get midPoint data. MidPoint was designed from the day one to be a service-based application. Therefore there are REST and SOAP services packed with features. Actually almost anything that midPoint does can be controlled by using those services.

- **Troubleshooting:** MidPoint is build on a sound design, it has clear principles, the mechanisms are generic, reusable and flexible, there is a order and meaning in midPoint design. Despite all of that midPoint is still a big and complex system. To err is human. And given all the flexibility of midPoint mechanisms configuration mistakes just happen. Effects of the mistakes may not be entirely obvious and the root cause may be difficult to find. Fortunately, midPoint has strong diagnostics mechanisms. MidPoint is almost pedantic about how the data are logged and how diagnostics data are presented. There is diagnostic capability in all midPoint components and all the levels, from the connectors all the way up to the user interface. However it may not be entirely obvious where to start and how to use the diagnostic facilities efficiently. Explaining all of that is the goal of this chapter.
- **Advanced concepts:** There are still some features that were not explained in previous chapters: consistency mechanisms, personas, multi-connector resources and so on. Some of those features are seldom used, but they may save your project. Other features are used all the times, but they are a natural part of midPoint and therefore they are almost invisible. But all those features deserve explanation. And there is also a need to describe how midPoint itself is developed – as there is a lot of experimental and incomplete features. But, as this is midPoint, even those features may be extremely useful. This chapter may also be interesting to people who would like to extended midPoint in a unusual way or those that want to contribute to midPoint development.
- **MidPoint Deployment:** There are many path from downloading midPoint packages to a working system. Some of those paths are easier than other. MidPoint design was build on many years of practical identity management experience. Therefore midPoint has mechanisms that can be used to efficiently overcome some of the notorious problems in identity management – provided that midPoint is used correctly. This chapter aims at giving advice how midPoint should be used in practical projects. How to plan the project, what information to gather, how to design the deployment, how to prepare the environment, plan the migration, handle project extensions and changes and so on.
- **Management of IAM program:** Identity management is very similar to information security: Identity management has no end. Identity management is not a project. It is a *program*. It is an endless cycle of gathering data, planning and execution. The environment around identity management is always changing, therefore identity management must change as well. But midPoint is designed for this kind of longevity. This chapter will describe how to handle this endless cycle. How to make midPoint configuration open to extensions. How to gather data. How to handle new feature requests. How to do upgrades. How to keep identity management solution sustainable.

- **Deployment examples:** This book is using a lot of examples in all the chapters. But those are examples designed to demonstrate one specific aspect of midPoint functionality. This chapter will be different. There will be complete examples of practical midPoint solutions. After all, the way that copying and pasting is one of the best ways how to learn.
- **Glossary:** Identity management and governance parlance may sound like an alien language to a newcomer. Therefore perhaps preparing IDMimsh-to-english dictionary might be a good idea.

Those chapters are still missing. They are not written yet. Obviously, the best people to write those chapters are the people from the Evolveum team: the people that designed and implemented midPoint, the people that support midPoint deployments, people that work with midPoint every day, people that eat, breathe and sleep midPoint. But those people are just engineers. They need to pay their bills. They cannot put away their day-to-day responsibilities to work on this book. Obviously, funding is needed to finish the book. As this book is available for free there is no direct income that could provide the funding for next chapters. There is only one way: sponsoring.

If you like this book then please consider sponsoring some of the next chapters. The market economy is, unfortunately, quite ruthless. Therefore, it is pretty straightforward: if there are no sponsors, it is very unlikely that there will be any new chapters. Therefore please sponsor this book if you can. If you cannot afford to sponsor this book then please at least help us to spread the word: a word about midPoint and a word about this book. Any form of help is more than appreciated.

Chapter 9: Additional Information

Logic's useless unless it's armed with essential data.

– Leto II
Children of Dune by Frank Herbert

We have tried to make this book as comprehensive as possible. But no book can possibly include all the information that a humble IDM engineer would need. Therefore this chapter describes the sources of additional information about midPoint.

MidPoint Wiki

MidPoint wiki is the most comprehensive information about midPoint. This is where all the midPoint documentation is stored. But there is much more. There is connector documentation, midPoint architecture, developer documentation, midPoint internals and all kinds of information including midPoint release planning and roadmap. Everything is public.

However the wiki is so comprehensive that it is often not entirely easy to find the right page. We have done what we could to organize the information. The pages are organized hierarchically. Many pages have “See Also” section that points to additional information. Yet the practice shows that if you want to find something in the wiki you need to have at least a faint idea what you are looking for. This book should give you that idea. If you know what you are looking for then the wiki search bar is your friend. If you enter the correct search term then there is high probability that you will quickly find the right page.

URL: <http://wiki.evolveum.com/>

Samples

The midPoint projects maintains quite a rich collection of samples. These are sample resource definitions, role and organizational structure examples and other various samples. They are usually provided in the XML form. The samples are maintained together with midPoint source code on GitHub. They are also part of midPoint distribution package.

URL (latest version): <https://github.com/Evolveum/midpoint/tree/master/samples>

Book Samples

This book contains many examples and configuration snippets that are taken from various places. Some of the smaller snippets are taken from midPoint wiki or from the sample files (see above).

Some chapters contain mostly complete configurations of the midPoint deployment. These configurations have a separate folder in the midPoint samples. Look for a folder “book” in

midPoint samples. All the important files used in this book are there, sorted by chapter number.

URL: <https://github.com/Evolveum/midpoint/tree/master/samples/book>

Story Tests

MidPoint developers like to create and maintain complete end-to-end automated tests. These tests are usually inspired by real-world midPoint deployments. We call them *story tests*. These tests are important to maintain midPoint quality and continuity. However they are also excellent source of inspiration and they have often proved useful as examples of midPoint configuration.

Wiki description: <https://wiki.evolveum.com/display/midPoint/Story+Tests>

Code and configuration: <https://github.com/Evolveum/midpoint/tree/master/testing/story>

MidPoint Mailing List

MidPoint project attracted a vibrant community during the years. The main community communication channel is midPoint mailing list. The mailing list is used for announcements, user suggestions and also what we at Evolveum call *community support*. The mailing list is used to ask questions about midPoint. Experienced community members usually answer these questions and provide pointers to additional information. The whole midPoint development team is also subscribed to the mailing list and they provide answers when needed. However this is a best effort service. Please do not abuse this communication channel and try keep the following community guidelines:

1. **Be polite.** Mailing list is a best effort service. Nobody is (directly) paid to answer mailing list questions. The engineers that answer the questions are doing that in addition to their day-to-day responsibilities and they are doing that because they want to help the community. Therefore if you are asking for help, do so politely. If you are answering a question please respect other members. Everybody started somewhere and it is natural that novice users do not know everything. Please tolerate the differences in skill sets.
2. **Do some research before asking a question.** Do not ask trivial question that can be easily answered by googling the question, by searching for it in the midPoint wiki or mailing list archive. If you are getting an error try to read error message very carefully and try to think about the possible causes. Try to experiment with the configuration a bit. Look at troubleshooting section of this wiki. Spend at least a couple of minutes to make your own research before asking the question. If that research does not provide the answer then it is a good question for the mailing list.
3. **Provide context.** If your post looks like *"my midpoint is broken, please help"* then it is very unlikely that you will get any answers. Try to describe your problem in more details. Make sure to describe relevant bits of your configuration. Be sure to include error message. Look in the log files if necessary. And most importantly: describe what are you trying to achieve. Maybe the root of your problem is that you are using

completely wrong approach. The community may point your nose in the right direction - but only if they know what is your goal.

4. **Give back.** Mailing list is not one way communication channel where users ask questions and developers answer them. There is already a significant body of knowledge distributed among community members that are not midPoint developers. If you adhere to these guidelines and ask a question it will most likely be answered. But for that there needs to be someone who is answering. Therefore do not just ask the questions. If you know the answer for the question that someone else asks then please go ahead and answer it. Do not worry that your answer may not be perfect. Even a partial answer will be greatly appreciated by any novice user. Simply speaking: Do not only take from the community. Try to repay what the community gave you.

You may also be tempted to send your questions directly to Evolveum or midPoint developers. However the developers have many midPoint users, partners, customers and contributors to deal with in their day-to-day job. The first responsibility of any midPoint core developer is to make sure that midPoint development will continue. The developers naturally prefer to spend time doing tasks that bring funding to the midPoint project. Therefore the developers will strictly prioritize the communication. Answers to midPoint subscribers are highest priority, mailing list is second and answers to private messages from the community are absolutely lowest priority. We prefer efficient spread of knowledge about midPoint. Mailing list is good for that, but private communication is not. That's the primary reason for this priority setup. Besides, if you contact a developer directly then only that developer can answer your question. But if you send the question to the mailing list there are more people that can potentially answer the question. Therefore unless you have active subscription the mailing list is your best option.

Mailing list URL: <http://lists.evolveum.com/mailman/listinfo/midpoint>

Chapter 10: Conclusion

This book is not finished yet. In fact it is just a beginning. The book is written continually in an incremental and iterative fashion as fast as time and money allow.

Your contributions and donations will surely speed up completion of this book. Also consider getting midPoint subscription. That is the source of funding for midPoint development. Evolveum is no mega-corporation that can fund open source development from huge profits in other areas. There are no other areas. Evolveum is open-source-only company. Everything we do is open source. Evolveum is not a start-up either. We are not funded by venture capital and we do not have millions to spend. Evolveum is a self-funded company. We can spend only what we earn on subscriptions, sponsored features and services. There is no other income. MidPoint development can only go as fast as the money allow. The same principle applies to midPoint documentation and this book. It will grow proportionally to the Evolveum income. Therefore if you liked this book please consider supporting us. Both money and your time are more than appreciated.

We hope that you have enjoyed reading this book at least as much as we have enjoyed writing it – and as we have enjoyed creating midPoint in the first place. MidPoint is quite a unique software project and it would not be possible without you. The whole Evolveum team would like to thank all past, present and future midPoint supporters for making this exciting project a reality. Together we have created interesting and useful software product. We hope that together we can make midPoint even better.

Thank you all.