Consistency mechanism

MidPoint 3.9 and later

This page describes consistency mechanism used in midPoint 3.9 and later versions. MidPoint has consistency mechanism since almost the beginning. However, the mechanism was revised and significantly updated in midPoint 3.9. For information about the earlier versions please see Consistency mechanism (3.8 and earlier) page.

Introduction

Consistency mechanism is a part of midPoint that takes care of cases when midPoint cannot carry out the operation as it wants to. Inconsistencies form in these cases. Inconsistencies are the difference between what *midPoint thinks that should be* and what *really is*. Inconsistencies may happen if midPoint wants to create an account but the resource is down, if midPoint try to create an account but it finds that the account already exists and in many more cases. These may be caused by network outages, resource downtime, changes that happen outside midPoint, parallel and concurrent execution of tasks and also by technology limitations (e.g. connector unable to report all the changes). In practice inconsistencies happen all the time. Therefore a mechanism to handle them is an essential part of any IDM solution.

Please see the IDM Consistency page for short introduction to one of the consistency issues and Consistency Theory for an theoretical introduction to the consistency problem.

Consistency in midPoint

MidPoint implements consistency mechanisms at several layers:

- Operations that cannot be executed synchronously are postponed (e.g. in case a resource is down)
- Inconsistencies discovered during execution of operations are processed before the operation is finished (e.g. midPoint finds existing account while trying to create it)
- All the inconsistencies are detected and processed during a reconciliation process

Postponing the operations

Many types of errors may happen during provisioning operations: the resource may in inaccessible, the resource is refusing the operation due to security issues, the operation is malformed and so on. Provisioning errors are divided into groups of known problems, such as communication errors, schema errors, configuration errors and so on. One of these groups are problems in communication with the external resource, for example the resource is down, timeout occur during processing request etc. In this situation, when we recognize that the resource is unavailable, we postpone the actually processed operation to be able to process it when the resource goes online. The postponing operation means that we store actually processed object (shadow) to the local midpoint's repository. This object contains some additional information that make it clear what went wrong and what need to be done to finish the operation successfully.

Pending Operations

Operations that cannot be executed immediately (synchronously) are stored in shadow objects in a form of *pending operations*. Pending operation is a data structure that contains details about unfinished operation. This data structure contains:

- Delta describing the operation
- Status of the operation. Simply speaking the status indicates whether the operation is executing or whether it is already finished.

- Outcome (result status) of the operation indicating whether the operation was successful or whether it has failed.
- Metadata, such as timestamps, attempt counter and so on.

If an operation fails due to a communication issue, midPoint remembers that operation in *pending operations* data structure and returns immediately. MidPoint will continue with other operations on other resources that could still be executed - unless there are provisioning dependencies prohibiting that. MidPoint will re-try the failed operation later.

The same data structure is used for all asynchronous operations in midPoint, such as operations on manual resources. This is an example of synergy of several midPoint features.

Configuration

Operation retries can be configured in resource definition in the `consistency` section. However, it is often not necessary to provide any explicit configuration as midPoint has reasonable default settings. If a configuration is needed, it can be specified like this:

Consistency mechanism configuration in resource

```
<resource>
    <consistency>
        <pendingOperationRetentionPeriod>P3D</pendingOperationRetentionPeriod>
        <operationRetryPeriod>PT15M</operationRetryPeriod>
    </consistency>
</resource>
```

Following table is summarizing configuration properties that are applicable to the consistency mechanism:

Property	Default value	Description
`pendingOperationRetentionPeriod`	1 day	Duration for which the completed asynchronous operations will be kept in the shadow objects. This may be desirable for visibility, e.g. if the administrator wants to inspect result of an asynchronous operation.
`operationRetryPeriod`	30 minutes	Duration for which the system will wait before re-trying failed operation.
`operationRetryMaxAttempts`	3	Maximum number of attempts to re-try a failed operation. If set to 0 then operation re-tries are disabled.
`deadShadowRetentionPeriod`	7 days	Duration for which the system will keep dead shadows. After this interval has passed the dead shadows are deleted. Note: this may also be influenced by pendingOperationGracePeriod. Dead shadow may be kept for longer than the interval specified in deadShadowRetentionPeriod if that is needed to for evaluation of grace period.
`reshreshOnRead`	false	If set to true then midPoint will always refresh shadow when it is retrieved. Refreshing a shadow means that the status of asynchronous (e.g. manual) opearions is checked, unfinished operations may be retried and so on. In this case shadow will always be as fresh as it can be. But read may be slower and there may be strange errors (e.g. reading a shadow may cause "already exists" error because pending ADD operation in the shadow was executed during that read). If set to false (which is the default) then refresh will not be executed during read operations - unless the refresh is explicitly requested by midPoint code (e.g. during reconciliation).

Shadow Refresh

Pending operations are re-tried when two conditions are met:

1. Operation retry period has passed. This means that enough time has passed from the last attempt to execute the operation (30min by default).
2. Shadow *refresh* is initiated. This usually means that another modification operation is executed on the shadow. But it also happens when shadow read operation is forcing the refresh. That does not usually happen during normal read. But some read operations are forcing the refresh such as reconciliation operations. This may be also explicitly invoked by a refresh task (see below).

Refresh Task

Main page: Shadow Refresh Task

If the shadow refresh operation is left entirely to a chance then strange things may happen. Some shadows may not get refreshed for a long time. Therefore it is always a good idea to make sure that the shadows are refreshed periodically. This is often achieved by the means of a reconciliation process . Reconciliation is forcing refresh of all shadows to make sure that all data in midPoint are up to date. However, reconciliation is quite a heavy-weight process. It may be an overkill to run reconciliation just to make sure pending operations are retried. Therefore there is also a lighter-weight refresh task. The refresh task is just looking for shadows with pending operations and the task is forcing refresh of such shadows. Therefore running the refresh task can make sure that pending operations are retried.

Refresh task is very simple:

```
<task>
    <name>Shadow refresh</name>
    ...
    <handlerUri>http://midpoint.evolveum.com/xml/ns/public/model/shadowRefresh/handler-3</handlerUri>
    <recurrence>recurring</recurrence>
    <schedule>
        <interval>10</interval>
    </schedule>
</task>
```

Refresh task is quite lightweight and efficient. Therefore it can usually be scheduled for quite a frequent execution, usually executing every few minutes.

This is the same refresh task that is used for manual resources. In fact the mechanism of *pending operations* is the same for both consistency mechanism and manual resources, therefore also the same refresh task is used.

See Shadow Refresh Task page for more details.

Discovery

Discovery is used as one way to detect and eliminate the inconsistencies. It runs while executing operation when we recognize that something with the processed object is not okay. For example, the user tries to get account that is not actually present on the resource but only the shadow exists in the midpoint's repository. This shadow was created during resource unavailability and the account needs to be created on the resource when it goes online. Another example is, when the shadow contains pending modification. In this case, when the resource is not up, we first try to apply this modifications to the account and then return the most fresh account. Also, if the administrator gets the account that is not found on the resource, but the shadow exists we run discovery to find out what to do with this present shadow.

Handling different situations

Get – with previous problem in communication with resource while adding account

When the provisioning is requested to get account from the resource and only incomplete shadow exists (this shadow does not have the identifier of the real account on the resource), we run the discovery and tries to complete the previous operation (in this case, create account on the resource). Discovery may be a quite a long process and it is not good to run it by every get request when we are not sure that the resource is up now. Therefore, the discovery by get operation runs only if the resource is up (the resource has last availability status which tells us if the resource is up or down). After finishing the operation successfully, the new account is created on the resource and we return this, most fresh, account. If the resource is still unavailable we do not run discovery and we return the incomplete shadow.

Get – with previous problem in communication with resource while modifying account

This situation is similar to previous one, with one difference that if the resource is up and we run discovery, the account is not created, but pending modifications are applied to the account on the resource and the most fresh object is returned.

Get – object is not found on the external resource

In this scenario, administrator tries to get account, that is not present on the external resource. Or example, such situation can be formatted if the external resource does not support synchronization and someone deletes the account directly from the resource. Now, we have shadow in the midpoint's repository that has invalid link to the real account. We run discovery to find out what to do with such shadow. There are two possibilities, either the shadow is deleted or the account on the resource is re-created. It depends on the way, how the original account was created. If it was created using assignment, the result of the discovery is re-created account on the external resource which we return to administrator. If the account was created directly (not using assignment), the shadow is deleted.

Configuration

For the discovery mechanism we do not need any additional settings. As discovery is yet another way how midPoint detects changes, all what you need is to have configured synchronization part in the resource description.

Reconciliation

Reconciliation process (also called synchronization) is a standard way how identity management systems solve possible inconsistencies. It is used to scan external resource and find out changes that have been not applied yet because of some reason, e.q. when administrator made changes on external resource, synchronization was suspended. In the midPoint we use this standard reconciliation process, but we also add another option. In out reconciliation process we process not only changes from the external resource but also the changes from the local midPoint's repository. In this way, we add opaque direction for handling. In the direction from midpoint's repository to external resource, we search through shadows and if the one with additional information is found, we try to process it and complete previous failed operation.

See Also

- IDM Consistency
- Consistency Theory
- Relativity
- Reconciliation process
- Shadow Refresh Task
- Connector Development Guide - DiscoverySupport