BÆRUM KOMMUNE

Best practices for IDM driver development at the municipality of Bærum (Norway)

Sammen skaper vi fremtiden

MANGFOLD - RAUSHET - BÆREKRAFT

Who are we?

- The municipality of Bærum is the 5th largest municipality in Norway with about 130' inhabitants
- It has the highest average income per inhabitant
- We rank 2nd in business and employment and 3rd in competency and economy
- We employ about 12' employees, of which 35 work in IT operations

Who are we?

- Novell/NetIQ customer for decades (mid 90s?)
  - Novell NetWare
  - Novell Open Enterprise Server
  - Novell GroupWise
  - Novell ZENworks for Desktops and Linux Management
  - Novell Storage Manager
  - Novell/NetIQ Identity Manager
  - Novell/NetIQ Access Manager
  - NetIQ Identity Governance

BÆRUM KOMMUNE

Our current IDM setup

- The following objects are stored under the o=Meta container:
  - Organizational units
  - Positions are imported as hrPosition objects
  - Students and employees are imported as Person objects
- A Business Logic driver reads data on the Person objects to:
  - Define a username and e-mail address
  - Decode the position data
  - Etc.

Our current IDM setup

- Object Creator drivers examines the newly created/updated object to determine if an object on the Active side should be created, updated or deleted
- If an object has been created/updated, Business Logic drivers make additional changes on the User account object
  - Adds default roles based on department
  - Adds FEIDE attributes (teachers and most students)

Our current IDM setup

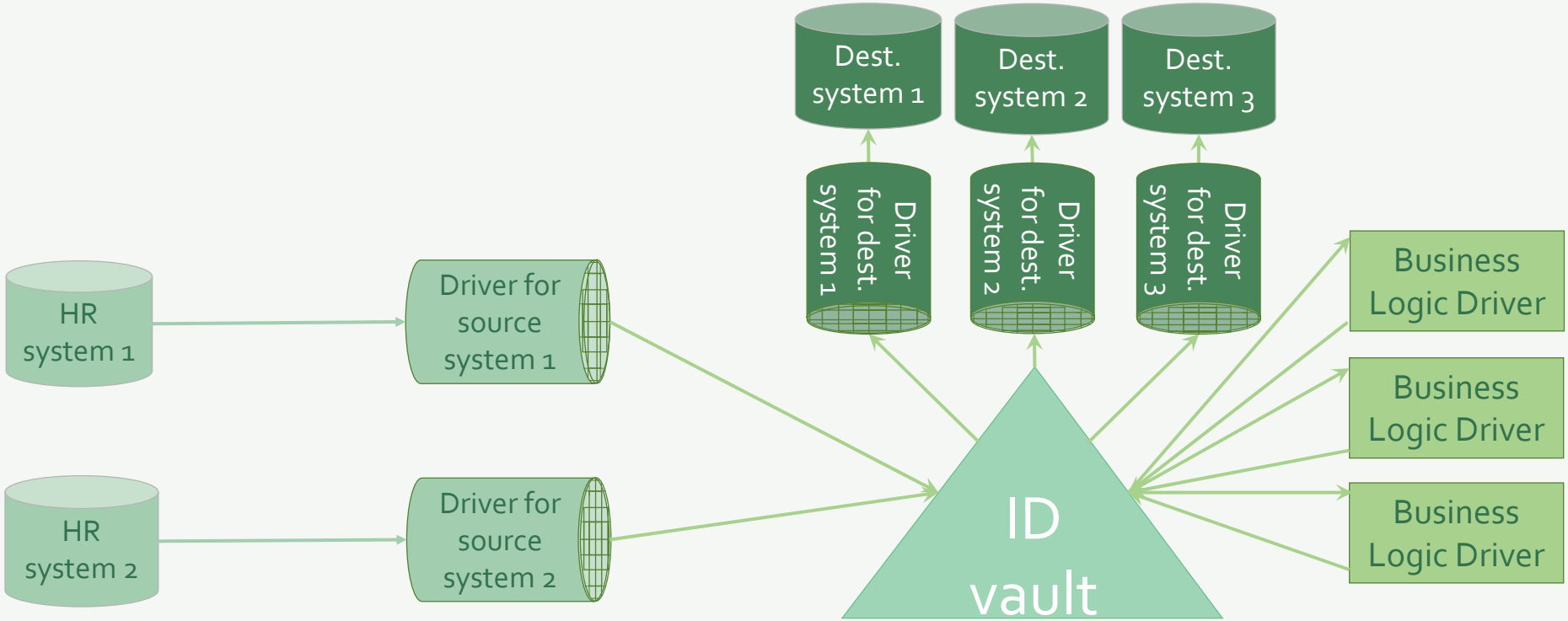
Our current IDM setup

- This setup gives us some challenges:
  - The Person objects are User objects that could be used to login
  - …and have attributes that can be problematic in complying with GDPR
  - Since employees can migrate between not needing a user account and needing one, we do need to represent them in the ID vault
  - We also have Aux classes added to driver and role objects, which isn't supported by NetIQ

Where we want to be

- To reduce the attack surface and at the same time enhance our capabilities, we wish to create a new class for the Meta objects that will be used for «shadow» objects
- Shadow objects will represent org. units, Persons, drivers, roles and possibly also resources and groups
- Each object will use the same base class and will have multiple auxiliary classes to store information (person, employee, student, license information, role approval information, etc.)

General driver design principles



General driver design principles

- The following principles represent our best practice for driver development:
  - Source drivers store raw data from the source system
  - Business Logic drivers process the raw data and store the transformed values in new attributes
  - Destination drivers read the transformed data and send it to the destination systems
  - Only one driver should ever write to an attribute

General driver design principles

- Our reasoning for this is:
  - Since no data transformation occurs during import or export, the processing from/to source/destination systems go quicker
  - We can create Business Logic drivers for a small number of attributes, giving us more speed and easier troubleshooting
  - Business Logic drivers create the «one truth» that is sent to destination systems
  - Since we store the raw data as well as the transformed data, we can also see at a glance where an error in the data occured

Source driver dataflow

- Data from the source system is written to an auxiliary attribute class only used by that system:

HRSystem1	SISSystem1	SISSystem2
fSrcHR1GivenName	fSrcSIS1GivenName	fSrcSIS2GivenName
fSrcHR1Surname	fSrcSIS1Surname	fSrcSIS2Surname
fSrcHR1NIN	fSrcSIS1NIN	fSrcSIS2NIN
fSrcHR1Gender	fSrcSIS1Gender	fSrcSIS2Gender
fSrcHR1Department	fSrcSIS1Department	fSrcSIS2Department
fSrcHR1PrivateEMailAddress	fSrcSIS1PrivateEMailAddress	fSrcSIS2PrivateEMailAddress
fSrcHR1PrivateMobile	fSrcSIS1PrivateMobile	fSrcSIS2PrivateMobile

Source driver dataflow

- A Business Logic Driver looks at values from all source systems, then determine which value to set as the «one truth» based on simple weighing rules::

Dest. attribute	First priority	Second priority	Third priority
Given Name	fSrcHR1GivenName	fSrcSIS1GivenName	fSrcSIS2GivenName
Surname	fSrcHR1Surname	fSrcSIS1Surname	fSrcSIS2Surname
fPersonNIN	fSrcHR1NIN	fSrcSIS1NIN	fSrcSIS2NIN
homeEMailAddress	fSrcHR1PrivateEMailAddress	fSrcSIS1PrivateEMailAddress	fSrcSIS2PrivateEMailAddress
homeMobile	fSrcHR1PrivateMobile	fSrcSIS1PrivateMobile	fSrcSIS2PrivateMobile
homeAddress	fSrcHR1PrivateAddress		
fStudentClassName	fSrcSIS1ClassName	fSrcSIS2ClassName	

Principles for using configuration values

- There are several ways to configure a driver:
  - By defining Global Configuration Values
  - By using Mapping Tables
  - By using eDirectory objects

Principles for using configuration values

- We prefer to store driver configuration in Global Configuration Values (GCVs) because:
  - Many variable types are available for GCVs (string, integer, enumeration, list, etc.). By writing a good description, a descriptive help text and using the right GCV type makes it easier for the operator to correctly configure the driver
  - It is possible to group GCVs so that one GCV is displayed only if another GCV is set to a certain value

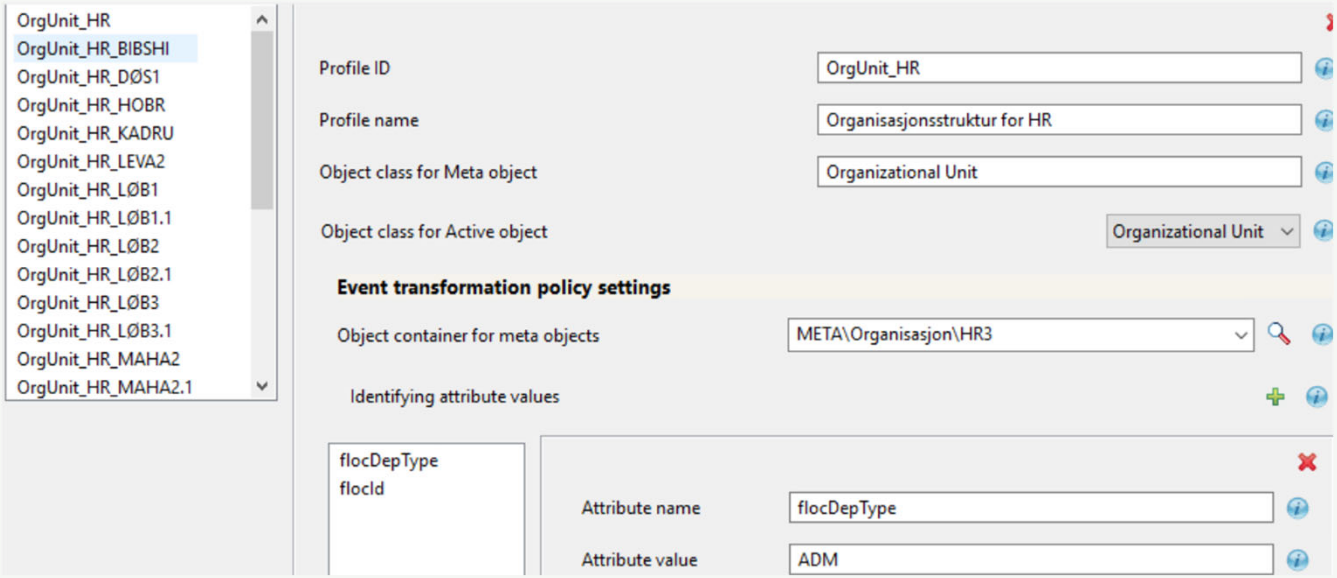
Principles for using configuration values

- If we use structured GCVs, it is possible to design a very complex configuration. We can even design nested structured GCVs – with groups
- Mapping tables – which could be an alternative - were actually designed to hold massive amounts of data (dozens of megabytes) – far too much for configuration data

Driver configuration

- Example of a nested GCV:
  - The Object Creator driver creates Organizational Unit, User and Group objects based on configuration
  - All object types share some of the same configuration values – naming attribute, naming pattern, etc.
  - Since we don't want to process the whole nested GCV every time, we write a startup policy that stores a condensed version of the configuration in a driver wide variable

Driver configuration



Driver configuration

```
<group>
    <definition critical-change="true" display-name="Object class for Active object" name="CW_GCV_Prof
        <description>This GCV stores the object class used for Active objects that this profile is for
        <value>User</value>
        <enum-choice display-name="User">User</enum-choice>
        <enum-choice display-name="Group">Group</enum-choice>
        <enum-choice display-name="Organizational Unit">Organizational Unit</enum-choice>
    </definition>
    <subordinates active-value="User">
        <header display-name="Event transformation policy settings"/>
        <definition critical-change="true" display-name="Object container for meta objects" dn-space="
            <description>This GCV stores the DN where objects matching this profile originate</descrip
        </definition>
        <definition display-name="Identifying attribute values" instance-separator="$" name="CW_GCV_Pr
            <template min-count="0">
                <definition display-name="Attribute name" name="CW_GCV_Prof_User_IAV_AttributeName" ty
                    <description>This GCV stores the name of the attribute used to identify this objec
                </definition>
                <definition display-name="Attribute value" name="CW_GCV_Prof_User_IAV_AttributeValue"
                    <description>This GCV stores the value of the attribute used to identify this obje
                </definition>
            </template>
            <description>This GCV stores the attribute values required to identity this object profile
        </definition>
```

```
<subordinates active-value="Organizational Unit">
    <header display-name="Event transformation policy settings"/>
    <definition critical-change="true" display-name="Object container for meta objects" dn-space="di
        <description>This GCV stores the DN where objects matching this profile originate</descriptio
    </definition>
    <definition display-name="Identifying attribute values" instance-separator="$" name="CW_GCV_Prof_
        <template min-count="0">
            <definition display-name="Attribute name" name="CW_GCV_Prof_OrgUnit_IAV_AttributeName" ty
                <description>This GCV stores the name of the attribute used to identify this group pr
            </definition>
            <definition display-name="Attribute value" name="CW_GCV_Prof_OrgUnit_IAV_AttributeValue"
                <description>This GCV stores the value of the attribute used to identify this group p
            </definition>
        </template>
        <description>This GCV stores the attribute values required to identity this profile</descript
    </definition>
```

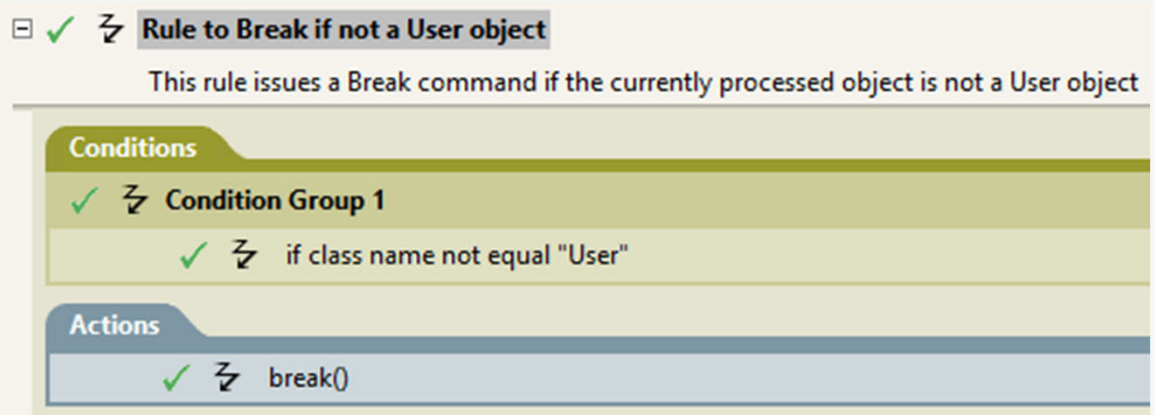
```
<subordinates active-value="Group">
    <header display-name="Event transformation policy settings"/>
    <definition critical-change="true" display-name="Object container for meta objects" dn-space="
        <description>This GCV stores the DN where objects matching this profile originate</descrip
    </definition>
    <definition display-name="Identifying attribute values" instance-separator="$" name="CW_GCV_Pr
        <template min-count="0">
            <definition display-name="Attribute name" name="CW_GCV_Prof_Group_IAV_AttributeName" ty
                <description>This GCV stores the name of the attribute used to identify this group
            </definition>
            <definition display-name="Attribute value" name="CW_GCV_Prof_Group_IAV_AttributeValue"
                <description>This GCV stores the value of the attribute used to identify this grou
            </definition>
        </template>
        <description>This GCV stores the attribute values required to identity this profile</descr
    </definition>
```

Driver configuration

```
iverConfigXML is '<Profiles>
        <Profile>
                <ProfileID>OrgUnit_HR</ProfileID>
                <ProfileDescription>Organisasjonsstruktur for HR</ProfileDescrip
tion>
                <MetaObjectClass>Organizational Unit</MetaObjectClass>
                <ActiveObjectClass>Organizational Unit</ActiveObjectClass>
                <MetaObjectContainerDN>META\Organisasjon\HR3</MetaObjectContaine
rDN>
                <IdentifyingAttributeValues>
                        <IdentifyingAttributeValue>
                                <AttributeName>flocDepType</AttributeName>
                                <AttributeValue>ADM</AttributeValue>
                        </IdentifyingAttributeValue>
                        <IdentifyingAttributeValue>
                                <AttributeName>flocId</AttributeName>
                                <AttributeValue>^(?!BIBSHI|DØS1|HOBR|KADRU|LEVA2
|LØB1\.1|LØB1|LØB2\.1|LØB2|LØB3\.1|LØB3|MAHA2\.1|MAHA2|SOLVA5|STB1\.1|STB1|STB2\
.1|STB2|STHO\.1|STHO|VAAA|VAAB|VAAC)</AttributeValue>
                        </IdentifyingAttributeValue>
                </IdentifyingAttributeValues>
                <AttributeMappingTableDN>system\driverset1\Object Creator Driver
 v2\CW MT AttributeSync OrgUnit HR</AttributeMappingTableDN>
```

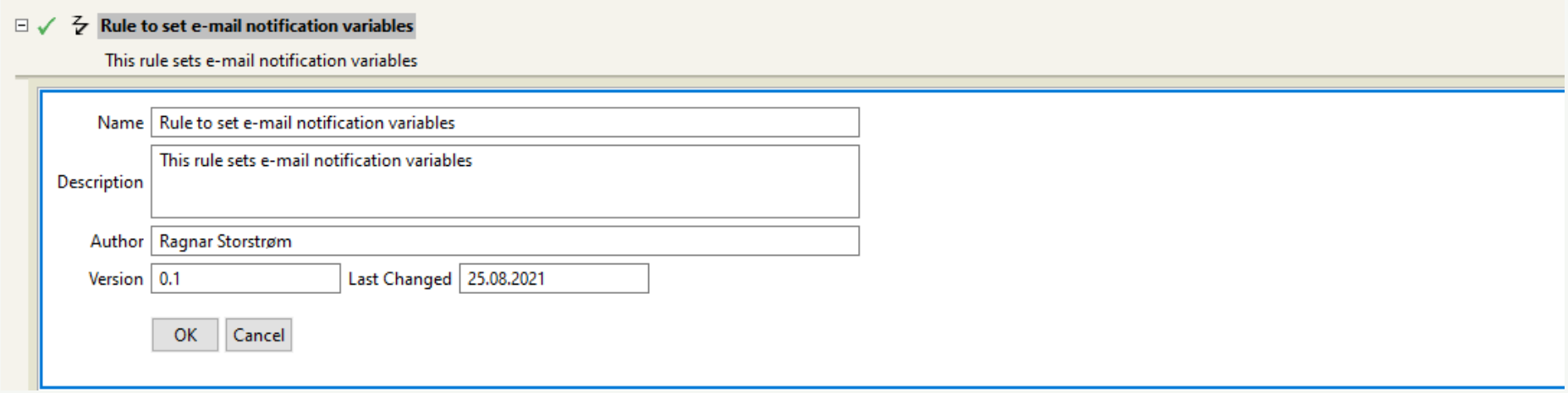
Policy development

- A policy can manage a single object class or multiple object classes. What is chosen is not important as long as you implement proper scoping:
  - The operation type
  - The object class
  - The attribute value(s)



Principles for coding rules

- The first thing you should do when you create a new rule is describe it:



Principles for coding rules

- The fields should be used as follows:

Field name	Usage
Name	A descriptive name for the rule, e.g. **Rule to set default attribute values**[1]
Description	A description of what the rule does + a short description of changes done since the rule was created.
Author	The name of the developer that created the rule
Version	The current version number of the rule[3]
Last Changed	The date that the rule was last changed

[1] It is recommended to use a naming standard for the name as this makes it easier to see when a BK employee wrote the rule
[2] The change log should include the date, developers name and short description of what was done
[3] For each change, add 0.1 to the current number

Principles for coding rules

- The reason we use comments for each rule is much the same as for the driver:
  - Although the code + driver trace will tell us how a rule works, it won't tell you anything about why it was written the way it is. The only person who can tell you that is the driver designer
  - To allow developers that come after to get advice on how best to make changes to the rule, the driver designer writes his/her name in the rule description

Principles for coding rules

- Each rule should contain a (short) change log that allow developers to see which change was made, which date the change was done and who the developer was
- In case a rule used in multiple drivers has a bug, we must be able to see who wrote it so that that person can tell the developer where else it should be changed

Principles for coding rules

- Since Designer has problems rendering large rules, try to use the following principles when writing rules:
  - If at all possible, keep the length of the rule to a page
  - Also remember that the description of the rule counts towards the size

Principles for coding rules

- Using the Veto token has some shortcomings:
  - While it is possible to specify a message when issuing a Veto, you can only see it in a trace level 3
  - It is not possible to specify a message when using «Veto if operational attribute not available», however
- To get around this, we simulate a Veto by issuing a Status message + Strip operation. This allows us to trap the error

Principles for writing driver code

- To ensure that the driver works as intended, the driver code should only process changes:
  - When required objects are present
  - When the object has required attributes
  - When the attribute value has an allowed/expected value
  - When the driver encounters a situation that does not conform to the above, the transaction should be reported and then discarded.

Principles for writing driver code

- Note:
  - Code to ensure this should be as simple as possible
  - To re-process the transaction at a later date, the trigger attribute should be used.

Principles for writing driver code

- Some drivers have been designed in a way that makes them run fine on an IDM engine server, but not in a Remote Loader configuration. Since the general recommendation from NetIQ is to always run drivers in a Remote Loader configuration, we ALWAYS* test that this works as the final step in driver development

- Drivers in a Remote Loader configuration perform commands on the Subscriber channel on the IDM engine server and commands on the Publisher channel on the Remote Loader side. As such, there is no point in testing Null drivers in a Remote Loader configuration.
- Starting with NetIQ Identity Manager 4.8, Loopback drivers are no longer supported in a Remote Loader configuration.

Principles for use of WorkOrder objects

- At first glance, WorkOrder objects seems like a great idea:
  - They allow a task to be performed at a specified time – with no human intervention!

Principles for use of WorkOrder objects

- There are, however some drawbacks:
  - The task may not be performed if the object it should process has been moved
  - The task may be performed incorrectly if information stored on the WorkOrder object has been updated elsewhere
  - An object can end up with dozens of future tasks assigned
  - The tasks defined in WorkOrder objects are usually «invisible» to Helpdesk personell

Principles for use of WorkOrder objects

- To fix some of these problems, use the following guidelines:
  - You should only use WorkOrder objects when:
    - It is not feasible to define a Job to do the same task
    - As a rule, the task should be performed a maximum of 14 days from now (and absolutely NEVER more than 4 months from now)

Principles for use of WorkOrder objects

- The WorkOrder object should contain only the minimum amount of information to perform the task, e.g.:
  - The task to be performed
  - The time and date when the task should be performed
  - The DN of the object that should be processed[1]
  - The immutable ID of the object that should be processed[2]

[1] If the object has not been moved since the WorkOrder object was created, using the DN is the fastest way to execute the task
[2] If the object has been moved since the WorkOrder object was created, the immutable ID will help us refind the object

Principles for using Jobs

- Any attribute change that trigger other updates could go wrong. To allow an operator to fix these situations, jobs should be created that re-process value updates
- Jobs should also be created to handle periodic events, e.g. employees leaving the company
- Jobs should always be used for events far into the future

Principles for managing objects

- Since the object creation time will change every time the object is moved in the directory tree, all objects should be timestamped with the time it was originally created
- It is also a good idea if each driver that creates a new object should adds a «source» value (e.g. Tieto HR system) that indicates which component created it

Principles for managing objects

- All drivers should update the object it modifies with a timestamp
  - A timestamp that tells operators when the driver last updated the object in the application
  - A timestamp that tells operators when the driver last updated the object in the ID vault

Principles for managing objects

- An IDM system can contain many drivers/connectors/integration, which can make troubleshooting incorrect values very difficult. But there are a few things we should do that will make it easier:
  - ALWAYS associate the object with the driver

    It is much easier to know which log files you need to look at if you know which drivers the object has a connection to

Principles for managing objects

- ALWAYS add attributes processed by the driver in the driver filter
  When you are trying to find which drivers update a specific attribute, it is helpful to search for the attribute in the driver filters. But when a source object is updated – or an object is updated directly, attributes don't pass through the filter. So it is possible to forget to add the attribute to the filter since the driver works without doing so. But to aid troubleshooting, this should always be done

Principles for managing objects

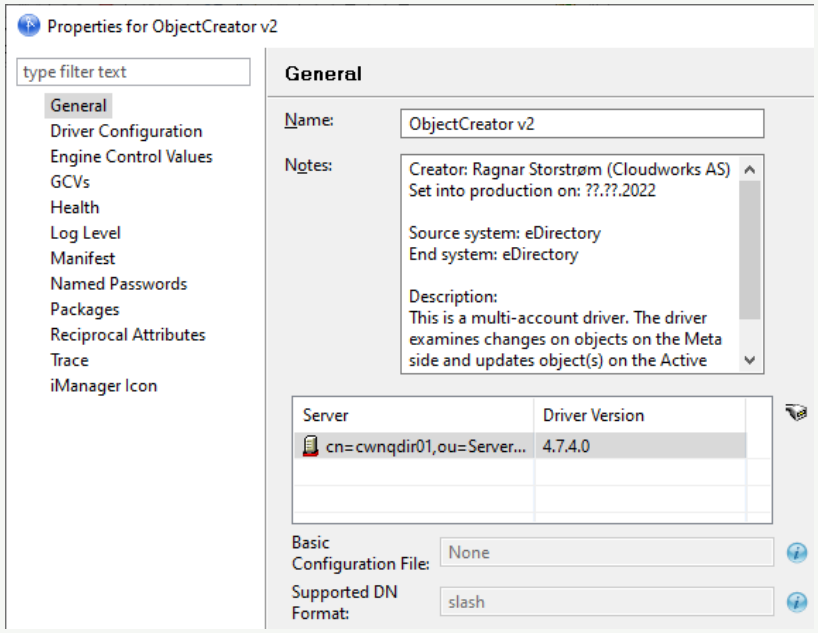
- ALWAYS add attributes to the Schema Mapping
  If an attribute isn't in the Schema Mapping policy, it will pass through the driver with no changes. This means that you don't have to add all of the attributes in the Schema Mapping policies, but to aid troubleshooting you really should. The reason is that by adding the attribute you are telling troubleshooters that the driver is processing a certain attribute (which might not show up except in a driver trace)

Driver design principles

- During development, design choices should be well documented
- Doing this will let other developers know what methods were tested to solve challenges and why they were rejected

Driver design principles

- Before the driver is deployed in production, it must be described:
  - Who wrote the driver
  - When it was set into production
  - What the source and destination systems are
  - A description of what the driver does



Shadow objects

- Shadow objects are required for a variety of reasons:
  - You cannot reuse usernames
    - Example:
      If an application ties log entries to a username, all log entries belonging to a former employee will automatically be assigned to the new employee if they have the same username

Shadow objects

- You need to preserve user information in case an employee is rehired
  - Example:
    If an employee leaves, no employee should get the former employee's e-mail address with 6 months after he/she left to ensure there is no confusion
    If an employee is rehired, the employee's original user ID in Workplace from Facebook must be connected to the new user account

Shadow objects

- You need to add information to an object class where class modifications are not supported
  - Example:
    You want to add license cost information to a role or a driver

Attributes defined per driver

- ErrorsFrom <DriverName>
  This attribute stores error messages seen by the driver related to the object
- LastUpdatedInIDVBy<DriverName>
  This attribute stores the date and time the source driver updated the object in the ID vault
- LastUpdatedInAppBy<DriverName>
  This attribute stores the date and time the destination driver updated the object in the application

Attributes defined per driver

- IsDisabledIn<DriverName>
  This attribute is used to stop a destination driver from updating an object in the application
- OverrideIn<DriverName>
  This attribute is used to stop a source driver from updating an object in the ID vault
- ImmutableIDFor<DriverName>
  This attribute stores the immutable ID for the object in the application

Attributes defined per driver

- TriggerAttributeFor<DriverName>
  This attribute is used to fix problems on the object

How do we fix out of sync values?

- Option #1
  You can set the value manually in the destination system
  → This works fine if you are talking about a few users but it's not scalable
- Option #2
  You can set the value to the value in the destination system in the ID vault, then set the value to the correct value again
  → This is can cause an update storm in multiple systems

How do we fix out of sync values?

- Option #3:
  You can sync all values in the ID vault and the destination system by setting a trigger attribute that affects a single driver

Trigger attribue - bonus feature

- Occasionally, an object will get an invalid attribute value. There can be a number of ways this can happen:
  - Source system administrators change their registration procedures
  - A driver updates an attribute based on incomplete data
  - A human operator makes a mistake
- To detect this, we have made a Data Checking Driver. And when it finds a problem, it sets the correct trigger attribute to fix it

Trace level usage

- We tend to use the following trace levels:
  - Level 0
    Success/fail messages
  - Level 3
    Result values
  - Level 10
    Variable contents, ECMAscript variable values, XSLT messages

Error handling challenges

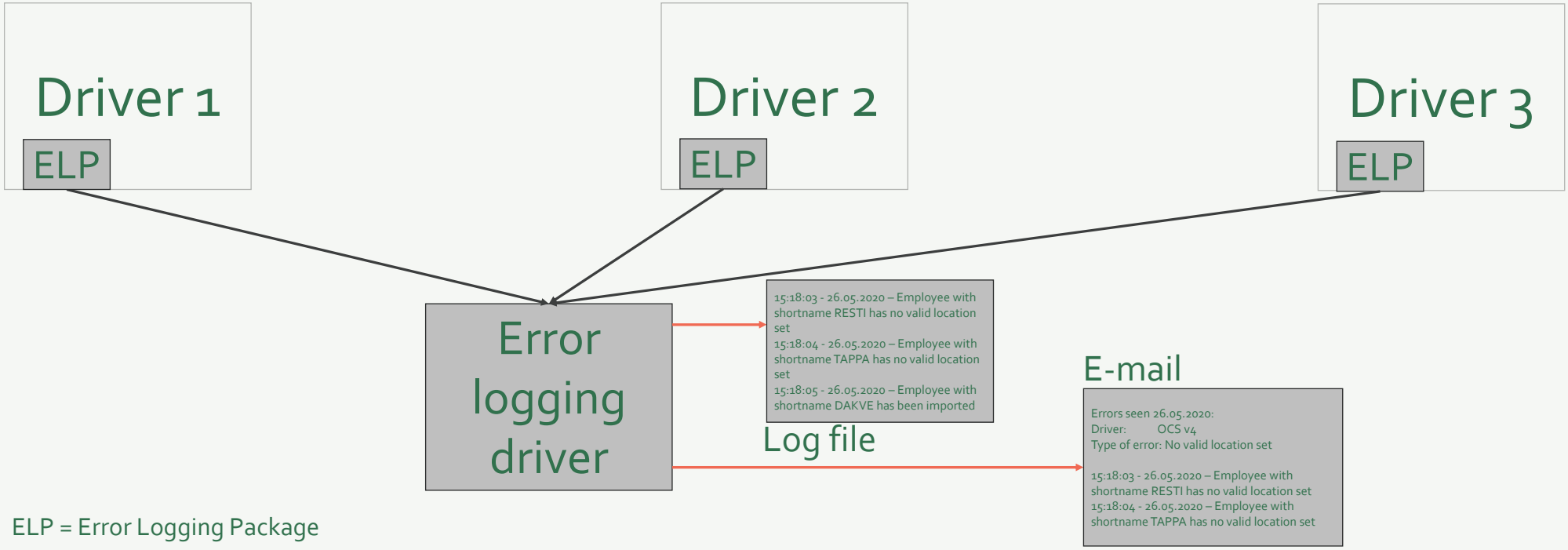
- Issue #1
  To see driver errors, you have to look in the trace log
- Issue #2
  When using a Veto token, you don't necessarily see why the operation was Veto'ed
- Issue 3#
  This article describes how to trap errors and notify by e-mail:
  https://ldapwiki.com/wiki/Wiki.jsp?page=Detect%20Status%20message%20and%20Send%20Email
  Warning: Operators could get swamped…

BÆRUM KOMMUNE

Error handling improvements

- To fix this problem, we are implementing the following:
  - An Error Trapping Package that will be installed in every driver. This package will trap errors, log them to file and can even forward them to an Error Logging Driver
  - The Error Logging Driver will filter out errors that can be ignored, accumulate errors to avoid operators getting swamped, forward errors according to system, type, time of day, etc.

Error handling improvements



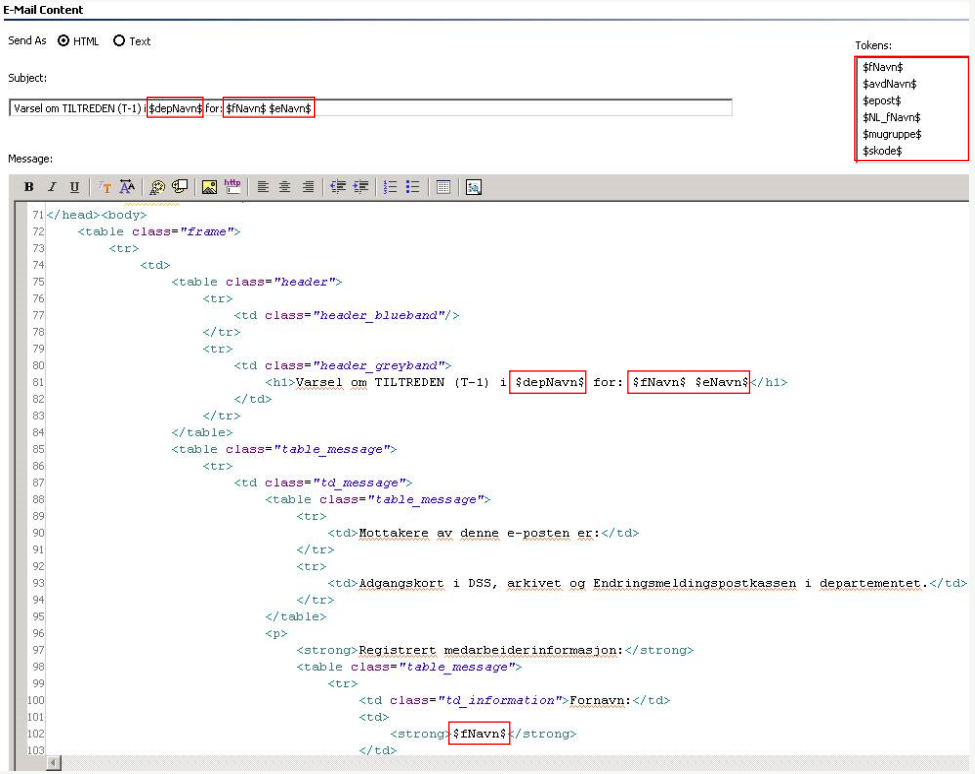
ELP = Error Logging Package

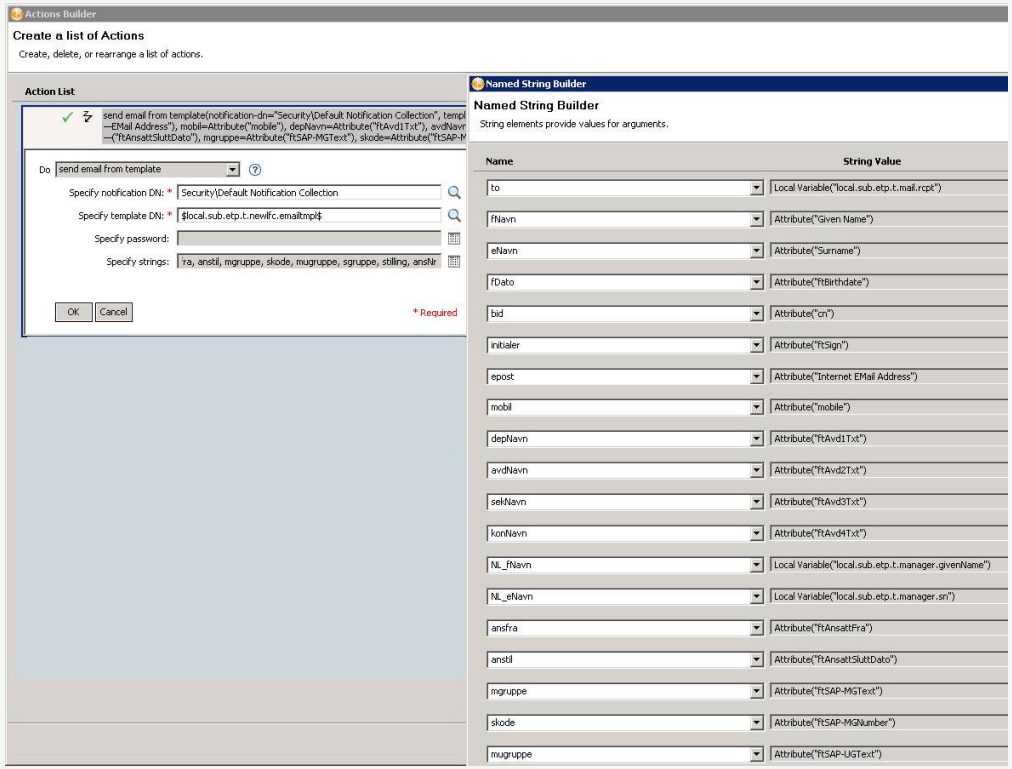
BÆRUM KOMMUNE

E-mail issues

- Issue #1:
  You have to define one token per value you wish to include in the subject or message body
- Issue #2:
  The value for every token used in the template must be separately assigned/coded
- Issue #3:
  If a token is deleted/not defined or doesn't have a value, there will be an empty space in the subject/message body

E-mail issues





E-mail improvements #1

- Our solution uses two templates instead of one:
  - The first template contains the subject and message of the e-mail (but has no tokens defined)
  - The second template is almost empty (and has two tokens defined – subject and message)
- To send an e-mail, our code first replaces values in the first template with an attribute ([[Given Name]]), GCV ({{idv.dit.users.data}}), Operation Property, variable or regular expression

E-mail improvements #1

- Once the value replacement is complete, the second template is used to send the e-mail, using only the subject and message tokens

E-mail improvements #1

E-Mail Template Editor

New template idea - part 1.Default Notification Collection.CWNQLABTREE

E-Mail Content

Send As ◉ HTML ○ Text

Tokens:

Subject:

Varsel om TILTREDEN (T-1) i [[ftAvdeling1]] for: [[Given Name]] [[Surname]]

Message:

```
71 </head><body>
72     <table class="frame">
73         <tr>
74             <td>
75                 <table class="header">
76                     <tr>
77                         <td class="header_blueband"/>
78                     </tr>
79                     <tr>
80                         <td class="header_greyband">
81                             <h1>Varsel om TILTREDEN (T-1) i [[ftAvdeling1]] for: [[Given Name]] [[Surname]]</h1>
82                         </td>
83                     </tr>
84                 </table>
85                 <table class="table_message">
86                     <tr>
87                         <td class="td_message">
88                             <table class="table_message">
89                                 <tr>
90                                     <td>Mottakere av denne e-posten er:</td>
91                                 </tr>
92                                 <tr>
93                                     <td>Adgangskort i DSS, arkivet og Endringsmeldingspostkassen i departementet.</td>
94                                 </tr>
95                             </table>
96                             <p>
97                                 <strong>Registrert medarbeiderinformasjon:</strong>
98                                 <table class="table_message">
99                                     <tr>
100                                        <td class="td_information">Fornavn:</td>
101                                        <td>
102                                            <strong>[[Given Name]]</strong>
103                                        </td>
104                                    </tr>
```

E-Mail Template Editor

New template idea - part 2.Default Notification Collection.CWNQLABTREE

E-Mail Content

Send As ◉ HTML ○ Text

Tokens:

$Subject$
$Message$

Subject:

$Subject$

Message:

```
1 <head>
2     <title>$Subject$</title>
3     <style>
4         <!-- body { font-family: Trebuchet MS } -->
5     </style>
6 </head><body BGCOLOR="#FFFFFF">
7     <p>$Message$</p>
8 </body>
```

E-mail improvements #2

- To further improve upon this setup, we created an advanced version where an e-mail was put together from mapping tables based on attribute values:

	Weight Numeric	RequiredAttrs Case Insensitive	MessageBlock Case Insensitive
1	10		Hei, Det er registrert at en ny medarbeider tiltrer [[fUserPosStartDate]] med deg som l
2	15		 <dl><dt>En medarbeider kan være en intern ansatt eller en eksternt tilknyttet/ innlei
3	20		 Medarbeiderens detaljer:
4	25		<table border="1"><tr><td>Fornavn</td><td>[[Given Name]]</td></tr><tr><td>Etternavn</t
5	26		 Merk:<i>Brukernavn skytjenester er ikke en e-postadresse, men et brukernavn ger
6	28		 Medarbeiderens stillinger: [[UserPositions]]
7	30	fUserM365License=SKOLE|KONTOR|TERMINAL	 Den nye medarbeideren har fått tildelt e-postadresse [[Internet Email Address]]. Du ka
8	35	fUserM365License=SKOLE|KONTOR|TERMINAL	 [[Given Name]] kan fra og med første arbeidsdag lese og besvare e-post og møteinnkalling
9	36	fUserM365License=NONE|ON REQ	 Den nye medarbeideren er ansatt i en stilling som ikke automatisk gir tilgang til Offic
10	40		 Følgende organisasjonsroller og tilgangsroller er definert basert på medarbeidere
11	45	fUserPOBCases	 Forespørsler sendt for din nye medarbeider: [[fUserPOBCases]]
12	50	nrfMemberOf=OR2100|OR2151|OR2115|OR2116|OR2118|or2108|OR2117|OR2121|Elektronisk arkiv saksbehandler|OR_276444|OR2123|OR2124	 Tilganger som ikke er automatisk tildelt:<div style="width:1330px;height:22
13	55	nrfMemberOf=OR2100|OR2151|OR2115|OR2118|or2108|OR2117|OR2121|OR_276444|OR2123|OR2124	Det kan være tilganger som ikke er automatisert som kan være relevant for din medarbeider. <br /
14	60	nrfMemberOf=OR2100|OR2151|OR2115|OR2118|or2108|OR_276444	 Trykk her for å gå til Ansattportalen <br /