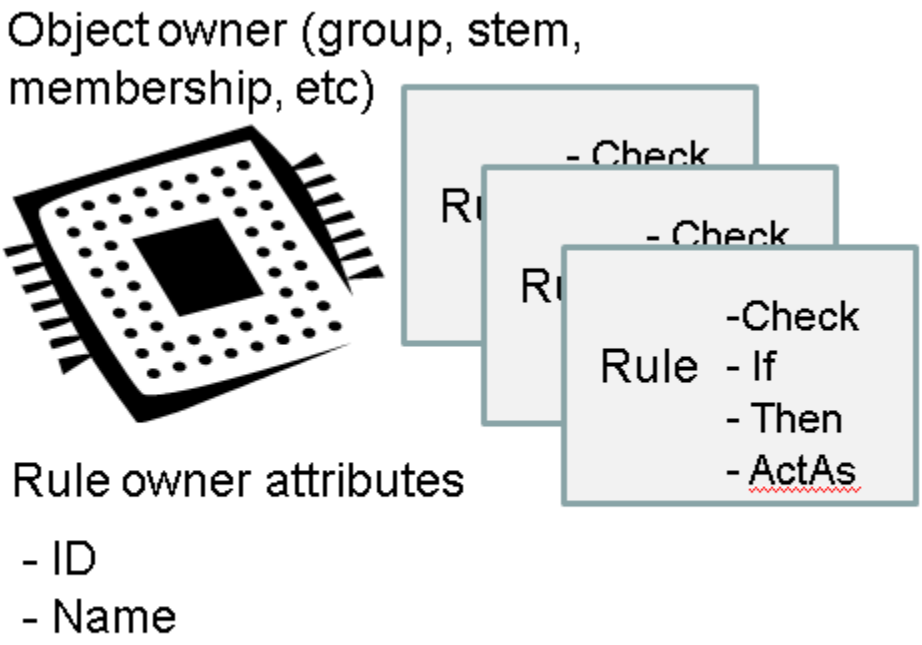
Grouper rules

Grouper rules are configurable declarative scripts which run at certain times and perform actions on the registry. They are similar to hooks though you don't have to write Java, and it does not require a change to a config file to enable a rule (i.e. anyone with authority in the folder hierarchy could enable a rule). This is similar to JBoss drools. There is no heuristic to find the best rule, it finds all matching rules to fire. Rules are unordered. Some rule "Then" clauses could kick off more rules.

Check out this page from the API docs for help with Grouper rules

Here is an object which has a rule on it. Note: rules are configured with the attribute framework. The attributes that set the rules metadata are set in a configured namespace. Ask your Grouper admins which folder holds these attributes.

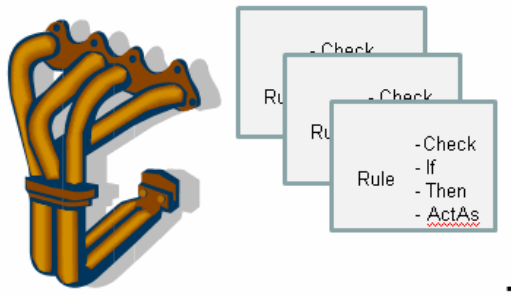


Here is a diagram which describes what happens when rules fire, and in the background

---

1 Event happens in Grouper
e.g. membershipRemove

2 Notifies rules engine to look for rules

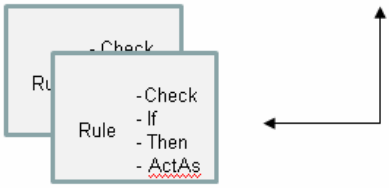


3

Rules engine finds by hash the rules which match the “check” clause against valid rules in memory

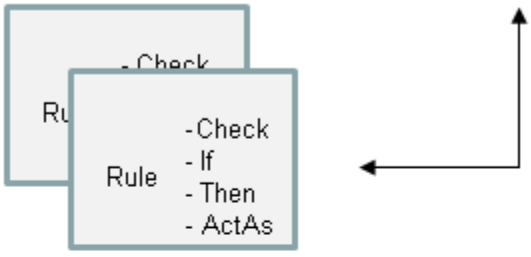
- Rules periodically refreshed from DB
- Rules daily validated in DB
- Rules immediately refreshed if edited in same VM

- Eligible rules are run periodically always (e.g. email when about to expire)
- Eligible rules are run periodically to fix data corruptions or inconsistencies



4

Subset of matching rules have their “If” clause evaluated



5

Subset of matching “Check” and “If” eligible rules have their “Then” clauses executed, in the context of the rule’s “ActAs” subject

---

Use cases

Click here to see the use cases and examples of how the Grouper rules engine can address them

Example of setting up a rule with grouper client

Example of setting up a rule with WS

Rule structure

The rule structure is custom for Grouper since we want it to be performant and secure, however it is inspired from drools. There are several parts to a rule:

- actAs: subject that the rule will act as. If blank, then it will be filled in with the user who created the rule (probably a bad idea since the person might leave at some point, unless it is a service principal). There can be configurations in the grouper.properties (details) which allow users to act as other users or GrouperSysAdmin.
- check: this is when the rule is fired. This will generally have a checkType, which tells grouper when to fire the check, and some data which narrows down the search. e.g. checkType could be flattenedMembershipRemove, and the data could be groupName: a:b:c. the data is stored in the checkOwner attribute
- ifCondition: this might not be needed if the check contains all the information about when the rule should fire. You can configure a premade check (enum) or a scriptlet or EL (Expression language). e.g.

```
${!RulesUtils.hasMember(groupName, subjectSourceId, subjectId)}
```

- then: this is a premade (enum) or scriptlet (EL: expression language). e.g. thenType is removeMember and groupName is a:b, or a scriptlet:

```
${RulesUtils.removeMember(groupName, subjectSourceId, subjectId}
```

Rule check

The check component will see if the rule should continue to the "if condition". The check part is an enum class: edu.internet2.middleware.grouper.rules.RuleCheckType

Look at the javadoc or source for the most recent check types, currently they are:

- flattenedMembershipRemove
- groupCreate
- membershipRemove
- stemCreate

Here is an example of setting a rule check:

```
AttributeAssign attributeAssign = groupA.getAttributeDelegate().assignAttribute(
    RuleUtils.ruleAttributeDefName()).getAttributeAssign();

attributeAssign.getAttributeValueDelegate().assignValue(RuleUtils.ruleCheckTypeName(), RuleCheckType.
membershipRemove.name());

attributeAssign.getAttributeValueDelegate().assignValue(RuleUtils.ruleCheckOwnerNameName(), "stem:b");
```

The second part of the check is the owner. This can either be set by name or id. If the check is for objects in a folder or subfolder, there is also a stem scope attribute for ONE or SUB

Rule data

The rule will be an attribute of a grouper object (group, stem, etc). There are attributes on the assignment which configure the params

```
//add a rule on stem:a saying if you are out of stem:b, then remove from stem:a
    AttributeAssign attributeAssign = groupA
      .getAttributeDelegate().assignAttribute(RuleUtils.ruleAttributeDefName()).getAttributeAssign();

    attributeAssign.getAttributeValueDelegate().assignValue(
        RuleUtils.ruleActAsSubjectSourceIdName(), "g:isa");
    attributeAssign.getAttributeValueDelegate().assignValue(
        RuleUtils.ruleActAsSubjectIdName(), "GrouperSystem");
    attributeAssign.getAttributeValueDelegate().assignValue(
        RuleUtils.ruleCheckOwnerNameName(), "stem:b");
    attributeAssign.getAttributeValueDelegate().assignValue(
        RuleUtils.ruleCheckTypeName(),
        RuleCheckType.membershipRemove.name());
    attributeAssign.getAttributeValueDelegate().assignValue(
        RuleUtils.ruleIfConditionEnumName(),
        RuleConditionEnum.thisGroupHasImmediateMember.name());
    attributeAssign.getAttributeValueDelegate().assignValue(
        RuleUtils.ruleThenElName(),
        "${ruleElUtils.removeMemberFromGroupId(ownerGroupId, memberId)}");
```

GSH

Print out the rules for an owner in GSH:

```
RuleApi.rulesToString(groupA)
```

Print out all rules from GSH:

```
RuleApi.rulesToString()
```

Remove a rule (id is printed out in rulesToString)

```
grouperSession = GrouperSession.startRootSession();
stem = StemFinder.findByName(grouperSession, "test:testRules");
RuleApi.inheritGroupPrivileges(SubjectFinder.findRootSubject(), stem, Stem.Scope.SUB, SubjectFinder.
findRootSubject(), Privilege.getInstances("admin"));
RuleApi.rulesToString(stem);
stem.getAttributeDelegate().removeAttributeByAssignId("af0aa3601b5149a08b71d7d82ea6a906");
```

Error handling

If the rule execution fails for some reason, it should be logged (which could include emailing administrators), but it probably should not affect the transaction of the operation that triggered the rule. Maybe this can be a setting on a per rule basis and where applicable (e.g. if it is a flattened membership rule trigger, then there is no transaction since the rule fires post commit anyways.

Act as

Note that the subject source should be set before the subject id or identifier (if the id or identifier arent unique). Anyways, you can act as yourself, though I dont know why you would want to do that since if you leave the institution the rule might break. You can configure in the grouper.properties what the act as rules are, similar to the grouper WS act as.

```
# Rules users who are in the following group can use the actAs field to act as someone else
# You can put multiple groups separated by commas.  e.g. a:b:c, e:f:g
# You can put a single entry as the group the calling user has to be in, and the grouper the actAs has to be in
# separated by 4 colons
# e.g. if the configured values is:       a:b:c, e:f:d :::: r:e:w, x:e:w
# then if the calling user is in a:b:c or x:e:w, then the actAs can be anyone
# if not, then if the calling user is in e:f:d, then the actAs must be in r:e:w.  If multiple rules, then
# if one passes, then it is a success, if they all fail, then fail.
rules.act.as.group = etc:rulesActAsGroup
```

Validation

There are certain validation constraints to make a rule valid. i.e. you need some check, you need some then, you need an act as subject, etc. So each time you change a rule attribute value, all the attributes are validated, and the attribute "ruleValid" is managed by that hook. If the rule attributes are not valid, you will get a ruleValid value of something like: "INVALID: Rule check type required", if they are valid, then the value will be "T". Only rules with a value of T will be processed. The attribute stores this state so the rules dont have to be validated each time they are read from the DB, and so the user can get some feedback.

Allow users to be able to assign rules

This is a normal attribute framework NG concept. The user needs to be able to assign attributes to the owner object. e.g. for a group, the user needs ADMIN access on the group. Then the user needs UPDATE/READ on the attributeDefs (there are two). Also note, the user needs privileges in the actAs. Maybe add to an actAs group, if acting as the user itself, might need access to READ another group where the rule is fired from, etc.

```
stem2.grantPriv(SubjectTestHelper.SUBJ9, NamingPrivilege.CREATE, false);
    stem2.grantPriv(SubjectTestHelper.SUBJ9, NamingPrivilege.STEM, false);

    RuleUtils.ruleTypeAttributeDef().getPrivilegeDelegate().grantPriv(SubjectTestHelper.SUBJ9,
AttributeDefPrivilege.ATTR_UPDATE, false);
    RuleUtils.ruleAttrAttributeDef().getPrivilegeDelegate().grantPriv(SubjectTestHelper.SUBJ9,
AttributeDefPrivilege.ATTR_UPDATE, false);
    RuleUtils.ruleTypeAttributeDef().getPrivilegeDelegate().grantPriv(SubjectTestHelper.SUBJ9,
AttributeDefPrivilege.ATTR_READ, false);
    RuleUtils.ruleAttrAttributeDef().getPrivilegeDelegate().grantPriv(SubjectTestHelper.SUBJ9,
AttributeDefPrivilege.ATTR_READ, false);
```

Logging

You can turn debug logging on to see information about rules which fire. log4j.properties

```
log4j.logger.edu.internet2.middleware.grouper.rules = DEBUG
```

If you want to only log certain rules, you can specify them in the grouper.properties. (and you need to set the RulesEngine to INFO level at least)

```
# uuids (comma separated) of the attribute assign record which is the rule type to the owner object
# e.g. SELECT gaagv.attribute_assign_id FROM grouper_attr_asn_group_v gaagv WHERE gaagv.attribute_def_name_name
LIKE '%:rule' AND gaagv.group_name = 'stem:a'
# make sure log info level is set for RuleEngine
# log4j.logger.edu.internet2.middleware.grouper.rules.RuleEngine = INFO
rules.attributeAssignTypeIdsToLog = 446bb6b3bbd8417b9a3e386b3bc894c1
```

You will see log messages like this:

```
2010-08-21 15:24:13,032: [main] INFO  RuleEngine.fireRule(248) - Rules engine processing rulesBean: group: stem:
b, membership:
Membership[createTime=1282418648019,creatorUuid=8b10ad84a2ab4e4d912aeca154866bbc,depth=0,listName=members,
listType=list,
memberUuid=ddbbbb1615964f109e4b5f85c05098f7,groupId=291dbf3b736e42de9985a70e2ac11177,type=immediate,
uuid=4f249fd2636247a78158fc358aa58a32:bb46e541e12049618c199e162056e715], subject: Subject id: test.subject.0,
sourceId: jdbc, ,
found 1 matching rule definitions, ruleDefinition should fire: attributeAssignTypeId:
446bb6b3bbd8417b9a3e386b3bc894c1,
sourceId: g:isa, subjectId: GrouperSystem, checkOwnerName: stem:b, checkType: membershipRemove,
ifConditionEnum: thisGroupHasImmediateEnabledMembership, thenEl: ${ruleElUtils.removeMemberFromGroupId
(ownerGroupId, memberId)}, ,
EL variables: membershipId(4f249fd2636247a78158fc358aa58a32:bb46e541e12049618c199e162056e715),groupId
(291dbf3b736e42de9985a70e2ac11177),
groupName(stem:b),ruleElUtils,ownerGroupId(b38004ccf99d44f08f5a0971153ad6a9),subjectId(test.subject.0),memberId
(ddbbbb1615964f109e4b5f85c05098f7),
checkOwnerName(stem:b),sourceId(jdbc),, elResult: true, shouldFire count: 1
```

Veto

You can have the "then" clause veto an action (if it is a transactional check), by using the grouper util veto EL method. Note, if you are writing a custom EL class and want a veto, return the exception, dont throw it. Also the exception should be a RuleVeto exception (which is runtime) or a subclass. This example will veto an add to group A if the person is not a member of group B

```
//act as GrouperSystem
    attributeAssign.getAttributeValueDelegate().assignValue(
        RuleUtils.ruleActAsSubjectSourceIdName(), "g:isa");
    attributeAssign.getAttributeValueDelegate().assignValue(
        RuleUtils.ruleActAsSubjectIdName(), "GrouperSystem");

    //fire the rule when a membership is added to group A
    attributeAssign.getAttributeValueDelegate().assignValue(
        RuleUtils.ruleCheckOwnerNameName(), "stem:a");
    attributeAssign.getAttributeValueDelegate().assignValue(
        RuleUtils.ruleCheckTypeName(),
        RuleCheckType.membershipAdd.name());

    //continue with the rule if the member is not a member of B
    attributeAssign.getAttributeValueDelegate().assignValue(
        RuleUtils.ruleIfConditionEnumName(),
        RuleIfConditionEnum.groupHasNoImmediateEnabledMembership.name());
    attributeAssign.getAttributeValueDelegate().assignValue(
        RuleUtils.ruleIfOwnerNameName(),
        "stem:b");

    //if we get this far, veto the action with a descriptive reason
    attributeAssign.getAttributeValueDelegate().assignValue(
        RuleUtils.ruleThenElName(),
        "${ruleElUtils.veto('rule.entity.must.be.a.member.of.stem.b', 'Entity cannot be a member of stem:a if
not a member of stem:b')}");
```

Daemon

There is a daemon which runs on the loader which validates the rules and marks invalid ones as invalid. Those need manual fixes to get them valid again (due to actas permissions). You can configure the quartz cron in the grouper-loader.properties:

```
###################################
## Rules config
###################################

# when the rules validations and daemons run.  Leave blank to not run
rules.quartz.cron = 0 0 7 * * ?
```

The daemon will also run rule logic to sync up data inconsistencies (if it slipped by the rule somehow, or existed before the rule did). The rule must be eligibile for daemon logic, meaning it must have an enum for the CHECK part, or either a blank or enum IF condition. Also, the CHECK and IF enum must support daemon logic (basically it needs to be implemented), and the "ruleRunDaemon" attribute must be blank or T, and not F.

You can run the rules on an owner (daemon mode) with GSH:

```
RuleApi.runRulesForOwner(groupA)
```

Email config

To get emails to be sent from EL, you need to configure grouper email and rules email

In the grouper.proeprties set the SMTP settings for your institution's SMTP server

```
#####################################
## mail settings (optional, e.g. for daily report form loader)
#####################################

#smtp server is a domain name or dns name
mail.smtp.server = server.school.edu

#leave blank if unauthenticated
#mail.smtp.user =

#leave blank if unauthenticated
#mail.smtp.pass =

#this is the default email address where mail from grouper will come from
mail.from.address = noreply@school.edu

#this is the subject prefix of emails, which will help differentiate prod vs test vs dev etc
mail.subject.prefix = TEST:

#need to identify the email address attributes of each subject source
mail.source.someName.name = jdbc
mail.source.someName.emailAttributeName = email
#
#mail.source.someName2.name = jdbc2
#mail.source.someName2.emailAttributeName = EMAIL_ADDRESS
```

To test an email, run this in gsh:

```
gsh 0% new GrouperEmail().setTo("something@somewhere.edu").setBody("email body").setSubject("email subject").
send();
```

Extended EL API

There is a special group which has access to more objects in EL:

```
# any actAs subject in this group has access to more objects when the EL fires on
# the IF or THEN EL clause
rules.accessToApiInEl.group =
```

This is because the RuleUtils class might be too limiting in some cases, but if everyone had access to the API, it might not be secure. So if you need this, configure a group here, put in trusted admins/users, then act as those users in your rule. e.g.in this case the attributeAssignType object is in scope

```
attributeAssign.getAttributeValueDelegate().assignValue(
        RuleUtils.ruleIfConditionElName(),
        "${ruleElUtils.hasMembershipByGroupId(attributeAssignType.getOwnerGroupId(), memberId, null, 'true')}");
```

Note, to see which objects are in EL scope, turn debug logging on for rules and check the logs

```
log4j.logger.edu.internet2.middleware.grouper.rules = DEBUG
```

Custom EL classes

You can configure custom EL classes to help with logic you need if not in the Grouper API. Here is an example:

```
# put in fully qualified classes to add to the EL context.  Note that they need a default constructor
# comma separated.  The alias will be the simple class name without a first cap.
# e.g. if the class is test.Test the alias is "test"
rules.customElClasses = edu.internet2.middleware.grouper.rules.MyRuleUtils
```

Make a class:

```
/**
 * @author mchyzer
 * $Id: MyRuleUtils.java 6947 2010-08-23 15:33:36Z mchyzer $
 */
package edu.internet2.middleware.grouper.rules;

import org.apache.commons.logging.Log;

import edu.internet2.middleware.grouper.Group;
import edu.internet2.middleware.grouper.GroupFinder;
import edu.internet2.middleware.grouper.GrouperSession;
import edu.internet2.middleware.grouper.Member;
import edu.internet2.middleware.grouper.MemberFinder;
import edu.internet2.middleware.grouper.util.GrouperUtil;


/**
 *
 */
public class MyRuleUtils {

  /**
   * remove a member of a group
   * @param groupId
   * @param memberId
   * @return true if removed, false if not
   */
  public static boolean removeMemberFromGroupId(String groupId, String memberId) {
    if (LOG.isDebugEnabled()) {
      LOG.debug("Removing member: " + memberId + ", from group: " + groupId);
    }
    Group group = GroupFinder.findByUuid(GrouperSession.staticGrouperSession(), groupId, true);
    Member member = MemberFinder.findByUuid(GrouperSession.startRootSession(), memberId, true);
    boolean result = group.deleteMember(member, false);
    if (LOG.isDebugEnabled()) {
      LOG.debug("Removing subject: " + member.getSubjectId()
          + ", from group: " + group.getName() + ", result: " + result);
    }
    return result;
  }
  /** logger */
  private static final Log LOG = GrouperUtil.getLog(MyRuleUtils.class);


}
```

Use that in an EL:

```
attributeAssign.getAttributeValueDelegate().assignValue(
        RuleUtils.ruleThenElName(),
        "${myRuleUtils.removeMemberFromGroupId(ownerGroupId, memberId)}");
```

Validate rules dependent groups/stems

Periodically a Grouper administrator should look at rules and see if dependent objects no longer exist. You should review the offending rules and probably delete them. Delete them by navigating to the owner object and removing the attribute assignment for that rule.

First look for invalid rules

```
select * from grouper_rules_v where rule_valid != 'T'
```

Look at the rule_valid column for the reason. You might want to run this query and see if any of the results are problems. Note, this will also show some of the invalid rules

```
select assigned_to_type, assigned_to_group_name, assigned_to_stem_name, assigned_to_attribute_def_name,
'memberSubjectIdNotFound' as reason, assigned_to_member_subject_id as the_data
from grouper_rules_v where assigned_to_member_subject_id is not null and
not exists (select 1 from grouper_members gm where gm.subject_id = assigned_to_member_subject_id)
union all
select assigned_to_type, assigned_to_group_name, assigned_to_stem_name, assigned_to_attribute_def_name,
'missingRuleCheckOwnerId' as reason, rule_check_owner_id as the_data
from grouper_rules_v where rule_check_owner_id  is not null
and not exists (select 1 from grouper_groups gg where gg.id = rule_check_owner_id)
and not exists (select 1 from grouper_stems gs where gs.id = rule_check_owner_id)
and not exists (select 1 from grouper_attribute_def gad where gad.id = rule_check_owner_id)
union all
select assigned_to_type, assigned_to_group_name, assigned_to_stem_name, assigned_to_attribute_def_name,
'missingRuleCheckOwnerName' as reason, rule_check_owner_name as the_data
from grouper_rules_v where rule_check_owner_name is not null
and not exists (select 1 from grouper_groups gg where gg.name = rule_check_owner_name)
and not exists (select 1 from grouper_stems gs where gs.name = rule_check_owner_name)
and not exists (select 1 from grouper_attribute_def gad where gad.name = rule_check_owner_name)
union all
select assigned_to_type, assigned_to_group_name, assigned_to_stem_name, assigned_to_attribute_def_name,
'missingRuleIfOwnerId' as reason, rule_if_owner_id as the_data
from grouper_rules_v where rule_if_owner_id is not null
and not exists (select 1 from grouper_groups gg where gg.id = rule_if_owner_id)
and not exists (select 1 from grouper_stems gs where gs.id = rule_if_owner_id)
and not exists (select 1 from grouper_attribute_def gad where gad.id = rule_if_owner_id)
union all
select assigned_to_type, assigned_to_group_name, assigned_to_stem_name, assigned_to_attribute_def_name,
'missingRuleIfOwnerName' as reason, rule_if_owner_name as the_data
from grouper_rules_v where rule_if_owner_name is not null
and not exists (select 1 from grouper_groups gg where gg.name = rule_if_owner_name)
and not exists (select 1 from grouper_stems gs where gs.name = rule_if_owner_name)
and not exists (select 1 from grouper_attribute_def gad where gad.name = rule_if_owner_name)
union all
select assigned_to_type, assigned_to_group_name, assigned_to_stem_name, assigned_to_attribute_def_name,
'missingRuleThenArg0' as reason, rule_then_enum_arg0 as the_data
from grouper_rules_v where rule_then_enum_arg0 like 'g:gsa :::: %' and not exists (select 1 from grouper_groups
gg where
rule_then_enum_arg0 = concat('g:gsa :::: ', gg.id ));
```

To find a missing group in PIT run this:

```
select * from grouper_pit_groups where source_id = 'f4ae2524dda34129b8b17abeebb7c8c9';
```

Troubleshooting rules

To troubleshoot rules, set the logging debug level, and check the grouper logs. Edit the log4j.properties:

```
log4j.logger.edu.internet2.middleware.grouper.rules = DEBUG
```

Make sure the rule is valid, print out the rule by owner, or check the DB under grouper_attribute_assign_value table

```
RuleApi.rulesToString(groupA)
```

Developers could also debug and set breakpoints in the RuleEngine class

To do:

- document the GSH parts on the GSH page (2.0+)
- invalidate duplicate rules (one of them?)
- group and stem move and copy should reflect in rule assignments (and clear rules engine?)
- member change subject should reflect in rules (and clear rules engine?)
- validate email template name with regex?
- add to image diagram the change log consumer? every minute
- rules engine externalized to work as PDP
- add a way to configure rules in an external file to work like the loader works

See also the Overview of Access Management Features page for guidelines of when to use rules, roles, permission limits and enabled/disabled dates.