Gjallar Original Requirements And Design

Author: Göran Krampe

Note:

This document first describes general goals, motives and requirements for the Gjallar project and a basic terminology is defined. The remaining bulk of the document describes the model of Gjallar, how it is meant to work and the major concepts. All these objects and mechanisms are not implemented at this time (2006-08-02).

This document has been rewritten and refreshed for public publication.

Introduction

The design is described primarily as an object model with concepts and functions that Gjallar needs in order to match the perceived requirements in a flexible and efficient way.

The user interface is meant to be implemented separately from the domain code (as always in modern systems development) and is not described in this document other than when needed from a functional viewpoint.

Main goals for Gjallar

- Offers **a single coherent system** improving integration/interoperation of case processes without sacrificing functionality.
- Fulfils **advanced requirements**, like handling responses through email, more advanced filtering and statistics, offline operations, a security model etc.
- Is **adaptable to changes** in the business processes through configuration instead of programming. The processes change over time and so must the system be able to do within limitations.
- Supports a **secure way of partially sharing data** with other installations.
- Can more easily evolve with future requirements through **a better design and architecture**.
- Works as a **platform for integration** with other systems.
- **Scales in complexity** from very simple processes to quite advanced processes.

General overall requirements for Gjallar

Three very important requirements is driving the initial development of Gjallar:

- Simple to use
- Customizable and flexible
- Scalable in simplicity

There are many factors making a system simple to use:

- A good user interface
- Information hiding, not overwhelming the user
- Function hiding, not offering functions in the user interface that the user does not need
- Good terminology and concrete concepts
- As few blocking situations as possible

Customizable typically means that Gjallar should be easily adaptable firstmost by configuration but also through programming. This requires a rich meta object model and a clean system making it easily extendable. Flexible implies that the object model is not only configurable up front, but also reconfigurable through time.

Scalable in simplicity (or "in complexity") means that it should be easy to handle simple case processes, both for users and administrators. But it should still scale to handle much more complex processes.

Terminology

The terms are taken from both the area of issue tracker software, workflow systems and other domains. The explanations of the terms are very brief but they are described in detail later on.

- **Case** – In the industry the term "issue" is perhaps more pervasive but at Micronic the term "Case" is being used today. During the feasibility study we also noted that "issue" is harder to say mixed into Swedish.
- **Workflow Graph** – A directed graph of Stages in which Cases flow from stage to stage. The edges connecting the stages are called Transitions.
- **Transition –** An edge in the Workflow graph depicting an allowed path from one stage to another in the case lifecycle.
- **Stage** – A node in the Workflow graph representing a logical step in the handling of a case. There are many other possible names for this concept; place, state, station, activity, node etc. It is very similar to a state in a finite state machine.
- **Link** – A bidirectional relation that can be created between two cases. Links can also be classified by having a named type.
- **Attachment** – An uploaded file attached to a case. Attachments could possibly be attached to other objects in the system in order to describe various procedures to be used or to complement the online help.

- **Process** – A process is a defined procedure in Gjallar supporting a specific business process. The Process has a workflow graph, its own administrator(s), permission model and other local policies making it essentially a separate case tracking system "within the system".
- **User** – A person with a user account in Gjallar.
- **User Group** – A group of users for easy multiple specification of users.
- **User Alias** – A virtual user that can rotate among real users. Used for dynamic roles.
- **Contact –** A person that does not have a user account in Gjallar but that still needs to be referred to.
- **Custom Object –** A custom defined object representing a concept from the business domain, like for example Department, Project, System, Priority or Release.
- **Category –** A tag that can be attached to many different kinds of objects in Gjallar.
- **Form –** A group of fields (of various types) with labels. Forms can be attached to cases (and possibly other objects) dynamically in order to add more structured information.
- **Sub Form –** A Form that is embedded inside another Form.

Objects and their relations

Cases

All cases in Gjallar have some fundamental information regardless of the specific Process they belong to:

Name	A simple increasing integer with or without an alphabetic Prefix (see below). This is used a lot in communication about cases in email or voice etc. Never changes and is always created incrementally from a single global counter per installation regardless of Process.
Prefix	A prefix name using letters to denote the Gjallar installation in which the case was created. Each offline installation (laptop) or mirror server has its own identifying prefix.
Roles	The users or contacts related to the case. When the case is created one role called "Reporter" is automatically added.
Subject	A one line textual description of the case.
Description	A full textual description.
Stage	The current lifecycle stage in the workflow graph. See Workflow Graph.
Notes	A chronological collection of notes on the case. See Case Notes.
Categories	A case can belong to multiple categories. See Categories.
Forms	A collection of field groups with information about the case. See Forms.
Links	A collection of links to other cases. See Case Linking.
Attachments	A collection of uploaded documents. See Attachments.

Transactions	A chronological collection of Transactions fully describing how the case has changed over time. See Transactions.

The name is the case identifier generated from an integer counter. Cases created in the master Gjallar server do not have a prefix. Other Gjallar copies (see Partial Mirrors) like synchronized partial mirror servers or especially personal offline mirrors are all assigned a unique Prefix when created. The prefix + the integer is then guaranteed to uniquely identify the case. The name never changes.

Cases are entered by individuals (disregarding plausible automatic submission of cases from software) and typically that individual is a user in the Gjallar system. Sometimes cases are received by email and the correct user is looked up or a contact object is created automatically matching the sender email address. This user or contact is added as the role "Reporter".

Other important roles in Gjallar are "Source", "Responsible" and "Assigned". Source is the person from where the case is originating – it is normally the same as Reporter and is then left out. There can only be one Reporter and one Responsible (at a given time). Several people can be Assigned to the case. There is always a Responsible person and unless it is given from the start it will be calculated given different rules.

All cases have a one line textual description and a full textual description.

A case is always in a certain stage, from the beginning it is in the start stage in the workflow graph of the process.

All cases can be categorized using categories.

All cases maintain a chronological and threaded collection of notes about the case.

All cases can have file attachments and bidirectional links to other cases.

The above attributes are the only mandatory attributes that, regardless of Process in Gjallar, all cases have. It is worth to note that a single email will then be enough to create a basic case.

All cases can further be extended, at any point during their lifetime, with additional so called Forms and all cases have a full transaction log modelling all changes to it as a sequence of Transactions.

Workflow Graphs

There is an important higher concept often missing in simpler issue trackers – **Workflow graphs**. A Workflow graph is a description of the case lifecycle consisting of a set of **Stages** interconnected with **Transitions** in a directed graph with **one designated start stage** and **at least one or more designated end stages**.

When cases are created they always start at the designated start stage. Various users normally observe different stages and typically someone makes a first rough categorization and assessment and ensures the case fulfils the required criteria in order to make a transition to another stage.

A workflow graph is created for every kind of case process that the system should handle. In organizations there are typically many different cases processes like product customer

support, internal IT support, software development processes (possibly one for each independent software development group) etc. Workflow graphs are not hard coded in the system.

Stages

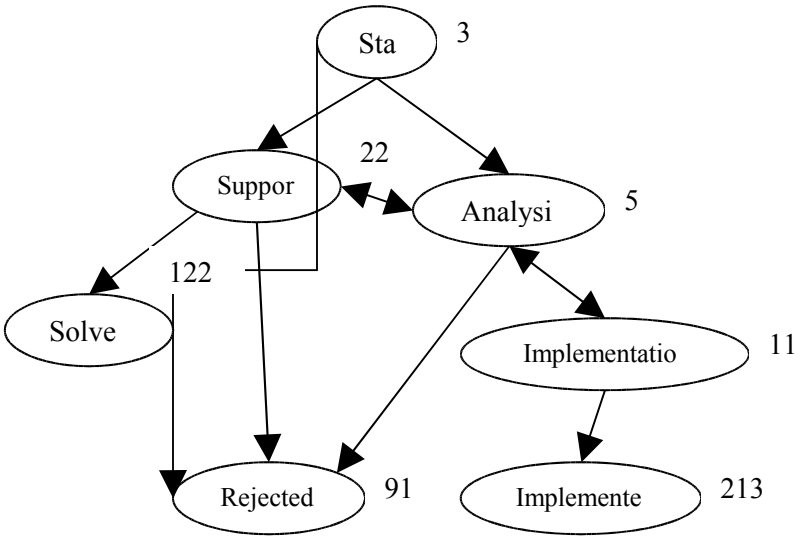
A **Stage** can be viewed as the current step for the handling of the case. A case can only be in one stage at a time, so you can ask *"What stage is case 841 in?"* and the answer could be *"It is still in the Analysis stage."* or something similar.

In each workflow graph there is **a single designated start stage** where new cases enter the graph. There can be **multiple stages marked as end stages** like "Closed","Trash" or "Rejected" but there is nothing intrinsically different with those stages compared to other stages, the only difference in practice is that cases do not typically move further.

Having Gjallar know about the start and end stages gives it an implicit distinction between **"new", "open"** and **"closed"** cases. A case sitting in the start stage is considered new (and not yet open) and cases sitting in any of the end stages can be considered closed. All other cases are open.

For each stage there are rules and logic associated with it. A case must fulfil a set of criteria in order to enter a stage and other criteria to leave it. Typically a criterion can be that more specific information about the case needs to be entered before it can move to the next stage. Criteria can also be associated with the actual directed transitions in the graph of stages.

This creates a mental image of a graph in which cases are poured into the start stage and then travels from stage to stage until they reach one of possibly multiple end stages. So at any given time a stage in the graph will hold a number of cases.



In this **imagined example** workflow graph (ignore the messed up graphics) we can see that there are 3 cases in the start stage waiting for an initial assessment. They can then either be immediately rejected (following the transition on the left all the way down to the Rejected stage) or considered to be a support case or a more complex case in need of deeper analysis (following one of the other two transitions).

The analysis and support stages can move cases between them typically representing two groups of responsible users. There are no transitions back to the start stage in this graph so all cases appearing there are certain to be newly created.

Analysis is the front stage when deciding if a case should result in any kind of new implementation and if so decided, the case is moved to implementation. Some transitions only go in one direction, others are bidirectional. In this example the case can be moved back to Analysis from Implementation if it is decided that the case need more analysis.

Typically users, given the appropriate access level, could override the workflow graph and move a case arbitrarily if needed, but that would normally indicate that the workflow graph is incorrect.

Global Scope

In the global Gjallar scope users marked explicitly as Gjallar administrators can create/define:

- Users
- Processes
- Global categories
- Global custom objects
- Global forms

This scope is not directly accessible by regular users, only indirectly through the

Processes that they are allowed access to, see Processes.

There are also categories and custom objects local to the Process.

A User is explicitly given access to one or more processes and each Process explicitly imports a selection of the global categories, global custom objects and users (selection of users specified by global categories) that should be visible and accessible inside that Process.

This means that on the global scope there is a very simple and clear access model and each Process can be totally isolated from the others while still allowing controlled sharing of global objects through imports; users, categories, custom objects and forms.

Processes

Each case **Process** is typically defined and mainly used by a specific group of users. The support crew mainly works within their support process and the IT personnel mainly works within the IT support process etc.

One or more users are designated by the Gjallar administrator(s) to be the Process administrators (owners). Those users alone have access to the Process administrative functions and can also add or remove access to this Process (not other Processes) for a user.

The processes are almost totally isolated from each other so that the users do not need to be exposed to all the functionality of the different processes nor all the cases being handled in Gjallar.

For example, the process of customer support is highly interconnected with the development process of the products. But the process dealing with organization personnel, like a new employee being taken care of when he or she starts working, is most probably isolated from the other processes.

Sharing of information is done **only** via:

- The imports from the global scope (categories, custom objects, users)
- Case linking between Processes

Linking of cases is subjected to the user level access – if a user does not have access to the other Process (in which the linked case live) the link will not be visible etc.

Case Linking

A case can (normally) not move between Processes – or in other words, there are no transitions between the separate workflow graphs of the different Processes. This may sound like a grave limitation, but there is another mechanism for dealing with inter-Process dependencies – case linking.

When a case in for example a support process causes actions to be taken in product development a new case (or several) is created in the development process and is then linked to the original support case triggering it. Cases do not change “type” nor are they cloned when these needs occur, instead new cases are created in the other Processes (or the same) and then they are **linked to each other**.

This mechanism makes sure that:

1. The original support case is not “lost”. It still exists in the support process and it is still perfectly valid in that context, regardless of any linked cases in other processes.
2. The new cases in the development process are linked to the original support case and the users can easily see the full **interleaved history** of all linked cases, including all notes from the support case.
3. The two Processes are still fully independent of each other. Some of the development cases may be solved quickly and enters ending stages, but the support case may still be active because it is a compound problem needing several development cases to be fully solved and a new release might also need to be installed at the customer site before the actual support case can reach an end stage.
4. Users working in the different processes can still maintain a mental picture of the local process and ignore other processes if they want to. Developers do not have to see the support cases triggering the development cases, but they can if they need to, and vice versa.

It may be tempting to try to create logic about the lifecycle of a case depending on linked cases, but we feel that the benefits do not outweigh the loss of isolation. It is also important that the users feel in control of how cases move in the lifecycle, that is why moving a case to a new stage in Gjallar is **always a manual decision** by a user and not automated.

Case Notes

Just as in most issue trackers notes can be added to a case. A Note in Gjallar has an optional relation to a previous note (showing it is a reply to it) making it possible to display a threaded view of the note exchange. Notes can also be added to existing cases by simply sending email to the system.

Categories

In many issue trackers issues have a certain “type”. Cases in Gjallar do not have types but are instead categorized which extends the “type” concept in two important ways:

1. A case can be in **multiple** categories.
2. The categories **are not fixed**.

Categories can be defined globally or Process wide. The global categories are used to establish company wide standard categorizations and are selectively imported into each Process. Categories are simple “tags” and do not form a hierarchy themselves.

Most objects in Gjallar (users, cases, stages, etc) can be categorized and the categorization can be used for filtering or easy reference to specific groups of objects, like certain user groups etc. It is primarily the basis of the permission system inside a Process.

The categories are maintained by the Gjallar system administrators or the Process administrators respectively and a category basically has a name and a description.

Custom Objects

Apart from the objects mentioned so far each Process (or globally) has a range of custom objects that the cases (or other objects) can refer to. Examples of concepts typically

modelled in Gjallar as custom objects are:

- Project
- Product
- System
- Release
- Customer
- Department
- Company

Custom objects have a few basic attributes like name and description. As the administrator in a Process, or global administrator, you can create new **custom object types** and:

- Give it a name and a description.
- Add forms to allow entering more information about it.

Custom objects are also used to model simple enumeration attributes, like for example the allowed values for probability, severity, priority which are then presented in drop down lists.

Custom objects are often mirrored into Gjallar from an external system using for example ODBC or some other mirroring mechanism and the updating is done using a regularly running service in the scheduler.

Contacts

Often individuals who do not have user accounts in Gjallar needs to be referenced, for example a customer contact or perhaps a person reporting a case using email or by phone.

Such individuals are represented in a different way than regular users and can be added on the fly with very little information about them, like for example just the name and email. They can later be upgraded to be regular users with a password if needed.

Users

Users in Gjallar are registered on a global level so you do not have to register multiple times for different Processes. Each user has all the ordinary fields like:

- Login name
- Full name
- Title
- Password
- Email (primary and secondary)
- Phone, cell phone, short phone
- Signature (for email replies)
- Comment
- Date format, time zone adjustment, time format

A user also has:

- Allowed Processes (not editable by user)

- Categories (not editable by user)
- Filters
- Subscriptions
- User Lists
- Case Lists

Users are given access to Processes by one of the Gjallar administrators.

Users are categorized by either the Gjallar administrator (only global categories) or by the Process owner (Process wide or imported global categories). The categorisation is the basis for Permissions and for specifying User Groups.

The user then has private personal data – filters, subscriptions and user and group lists. The lists are simple named collections of users or list of special interest. Such cooperatively maintained lists are also available on the Process level.

There is also a Gjallar global super user called “admin” normally only used for specific system maintenance.

User Lists and Aliases

User Lists are process wide (or personal) and are not used for defining permissions but are a simple way to reference teams, work groups or other loose groups of users.

User Lists can be created, modified and deleted ad hoc by the users of the process and can be shared in the Process given the proper Permission.

A user Alias is a named personal role in Gjallar and they are also defined process wide (not global). A user alias always refers to a specific user but the user can vary over time. This makes it possible to for example assign cases to a role which has a rolling schedule of actual users. Aliases are maintained by the Process owner.

In various places in the system where you need to **specify a single person** you can select a user, an alias or a contact (if that is allowed in the context – contacts are not users).

In other contexts you are allowed to **specify multiple persons** and then you can select one or more users, user lists (just a comfortable way of selecting multiple users), aliases or contacts (if that is allowed in the context – contacts are not users).

Forms

A case does not contain much structured information from the start. Each Process defines additional named forms of information with a set of fields and labels with corresponding validation logic etc. A Form can dynamically be “attached” to a case, either as an explicit operation of the user (at case creation or afterwards when editing) or automatically as the case moves between stages.

A stage describes what forms (or fields within a form) are mandatory before a case can enter it and what forms (or fields within a form) are mandatory before a case can leave it. If the start stage has mandatory forms upon entry, then those forms will be presented in the case submission form since the information then needs to be entered by the case reporter.

If a stage has mandatory forms before exit, then those forms will automatically be added (but empty) to the case when the case enters the stage. Then it will be up to the users to fill in those forms, and make sure they validate, before it can be moved to another stage. The system always shows what information is needed before a move to a new stage can be made.

A stage can also have mandatory forms associated **with a specific transition** and those will not be automatically created upon entry, since it is not known at that time if that specific transition will later be made. It will still be enforced if the users try to move the case using that transition. This combined mechanism ensures that all relevant information is entered at all stages in the lifecycle of a case.

Forms can also be defined globally so that they can be reused between Processes. Changes to forms can be made during use; added fields will be given default values, removed fields will cause entered values (in the cases having that form) to be removed and modifications of fields can also be made given a few restrictions. Another option is to copy the form and modify the copy in order to let existing cases still use the old form.

The different field types available to choose from when defining forms are: Selection – a choice from a list, single or multiple.

1. Boolean – true or false.
2. Text – one or multiple lines.
3. Integer – an integer value with simple validation rules and unit.
4. Float – a float value with simple validation rules and unit.
5. Date – a certain day, with simple validation rules.
6. Time – an amount of time with simple validation rules.
7. Association – a combination of two items from two different lists.
8. Subform – single or multiple embedded forms.

Selection

Selections can be from either:

1. Built in domain objects like cases, users, contacts, processes, categories or stages.
2. Custom objects.

The entries eligible for selection can be filtered, like only showing users that fulfil a certain criterion. It can also be configured to allow additions on the fly for maintaining contacts or custom objects.

Association

An Association is a dual Selection. Instead of selecting a single object from one list the user effectively selects **one object each from two different lists**, a key and a value, creating a pair/assocation. Typically the lists contain custom objects or built in domain objects.

The key list may contain just a single object, in which case only the second list – the choice of values – needs to be shown to the user. In that case it will “look” like an ordinary selection but in the model the first object is used as a “type” for the relation.

An example of using associations is modelling a “role” where you want to associate a

user object (value) with a role object (key) and store that association in the field. First you define a custom object type called “Role” and add objects to it like “responsible for documentation”, “budget controller” etc. Then you add a field with type Association in a form where you select the key list to be the Role custom object and users (for example the subset of users allowed in the current process and form) to be the values.

Note: This is the same mechanism used for the built in Roles attribute of Cases.

If the first list (keys) holds multiple values the field is presented as two drop down menus giving the user an option to refine the kind of association made.

Associations make it possible for Gjallar to distinguish between different kinds of relations to objects like for example users.

Subform

A Subform is a form embedded in another form. It can be useful for simple “form reuse” but since subforms can also be configured to be multi valued, it effectively turns it into a mechanism to have a “custom table of records” inside a form.

When a subform is embedded you choose if it should allow multiple values. Gjallar will then automatically add a table below the form and buttons for creating, selecting and deleting rows in it. The columns of the table are the fields of the subform.

User Experience

The following sections capture the most important aspects of using the system as a user.

Entering and modifying cases

A case is entered or modified by:

1. Using the web user interface working online
2. Remote partial mirrors of Gjallar connecting and committing transactions (not online)
3. Sending an email to Gjallar (not online)
4. Programmatic online access (using for example SOAP etc)

The first option is to use the web interface online. This is the common interface to the system and offers all functionality. If the start stage has associated mandatory forms, then those will be presented in the submission form.

The second option is the same but is limited in functionality. Adding cases is allowed, but editing is typically restricted to a subset of operations due to the conflict resolution needed when synchronizing the transactions. Mirrors are described separately.

Sending email to the system is very useful for simple reporting of cases or participating in email correspondence about existing cases, adding further notes to them. It could also be used for other operations that are feasible without an online validating connection.

Creating a case using email is simple and the sender email address will be matched with existing users or contacts and if there is none a contact will automatically be created. The subject line and email content constitutes the corresponding fields in the case and attachments are handled properly.

Multiple receivers in to- and cc-fields can be set to automatically create specific

associations of roles in the case. Multiple inbox addresses can be configured in Gjallar and for each address a specific process is the receiver and a specific template case is cloned and filled with the data from the email. This makes it possible to associate different email addresses to different base data in the case, like for example certain categories or values of custom objects.

Cases without added forms (they only contain the base information that all cases have) that are still in the start stage can be moved to a different process if the user doing the move has access to those processes. In other situations the user will have to create a new case in the correct process – but can use the existing case as a template.

Roles

A case has a collection of roles and two role types are hard coded into Gjallar – Reporter and Responsible. There can only be one user (or contact) as Reporter and there can only be one user (at a given point in time) as Responsible. Other users or contacts can have other roles in the case.

Reporter is automatically set when a case is created. When using the web submission form you must be logged in so Gjallar knows who you are. When sending an email the sender address is used as key – this can be forged of course, but that risk is considered less important. If you are creating a case originating from someone else you are still the actual Reporter but you should add an additional optional role called “Source” for the other person.

The Responsible role is very central to the system. A case always has a single person as responsible. Most of the more advanced processes have people acting as “gate keepers” that monitor the start stage in the process making a first assessment of the reported case assigning someone as responsible and moving the case to another stage. In these processes the “gate keepers” are automatically assigned as responsible for new cases.

But there are also processes without such a “gate keeper” and in those processes the responsible must be either explicitly selected by the reporter or automatically chosen. If the case is reported through the web submission form then that form – which is a Form (see Forms) – can be set to include an Association field for the responsible role with an optionally filtered list of users to choose from. It could also include a mandatory category selection or custom object that is then mapped to a responsible user through a simple script (see Scripting).

For cases submitted through email the responsible can be automatically chosen based on the first address in the to:-field that is not a Gjallar inbox address.

Filters & Searching

Users in Gjallar can define their own filters for selecting a subset of available cases.

In Gjallar there is also a second level of filtering, namely filtering the actual transactions within the case. This means a filter in Gjallar can operate not only on the current state of a case but on how the case has changed over time.

Filters always belong to a user but can be shared to other users using user categories or user lists.

A filter is built by constructing a chain of conditions using boolean logic. The set of

available fields to use in each predicate are:

1. The built in single valued fields valid for any case in Gjallar, like process, id, prefix, stage, subject and description.
2. The built in multi valued fields, like Roles, Notes, Categories, Attachments, Forms and Links.
3. The union of all fields in all Forms available in the Process including imported Forms from the global scope.
4. Computed fields typically based on information in the Transaction model.
5. Transaction fields. Each transaction has a type, timestamp, user and key/value pairs for the modifications it made.

Number 5, the transaction fields, offers advanced filtering based on how the cases have changed over time. Exactly how advanced predicates we should offer using a normal user interface is questionable.

Gjallar should offer a good user interface for 1-4 above but could possibly be able to drop down to a textual interface for making even more advanced filters, typically only offered to certain privileged users.

For text fields a good "free text" engine like for example OpenFTS (note: Swish-e is now used) that can be used through a special predicate operator. This makes free text searching cheap since such engines are very fast and scalable.

When creating a filter you create a Boolean expression of predicates where each predicate has a single field from 1-5 above on the left side, an operator (operators available depends on type of field), and a value on the right side. Placing a predicate on a transaction field evaluates to true if there exists any matching transactions in the case history.

This means you can easily find cases which have visited a certain stage regardless of its current stage, or cases in which a certain field has been modified, regardless of its current value. You can also find all cases which you or someone else has modified in any way, regardles of value.

When loading a saved filter the values on the left side in the predicates have the values that were entered when creating the filter. The user can then easily edit the values to refine the filter before applying it.

Searching and filtering is the same thing in Gjallar.A specific saved filter can be loaded and then you can edit it and apply it (causing the list to be filtered) and optionally save it as a new filter.

Gjallar additionally offers to load multiple filters (combined with logical AND) on top of each other. A typical use case is showing all cases that I am responsible for using a predefined filter and then performing yet another ad hoc search **within that subset** of cases.

Note: In most system you can not load multiple filters, nor can you edit the predicates – the fields and operators are often hard coded in a specific sequence of AND predicates.

The current filters affect all operations that intuitively "should" be affected, like listings, reports etc.

In Gjallar there is always a user specified limit of the number of cases listed and you can

select to see all in a second step or the next 50 etc.

Notifications and Subscriptions

In Gjallar there are **no hard coded email notifications** and **all email notifications use the same subscription mechanism**. The only difference is that some subscriptions are defined centrally and tied to Roles of users (using Roles, Associations or user categories), and other are personal subscriptions.

When users or contacts are created they are cloned from prototype objects. Those prototypes have pre-defined subscription objects in them so that global or process wide policy can be maintained for notifications. The users themselves can normally not modify these base subscriptions.

The notification mechanism in Gjallar also covers alarms for specific situations, like when a case is stalled (things taking too long) or some kind of abnormal handling occurs like a case entering a specific stage a third time (obviously not moving along regularly) etc, see Scripting.

The general process of notifications looks like this: First a modification is made in the system, either by a user or an executed script resulting in an automatic modification.

1. The modified object signals a low level event. The low level events are mapped to one or more high level events predefined in Gjallar. Scripts can also trigger high level events directly and the administrators can add custom high level event types.
2. Subscriptions listen to a list of specific high level event types and will receive the event if it matches any of the types.
3. If the event refers to a case, the subscription has an optional filter (See Filters) that are checked against the case and if the case is validated by the filter – then the actual notification is made typically resulting in an email being sent.

The filtering in step 3 means that users can define more fine granular notifications and not drown in a notification flood. Examples of high level event types are:

- Case added
- Case modified
- Related case modified (only for related users/contacts)
- Case with id xxx modified
- Case moved to another stage
- Related case moved to another stage (only for related users/contacts)
- Case with id xxx moved to another stage
- User added
- Workflow graph modified

Some of these Event types are global (like “User added”) but most are Process scoped and relate to Cases. The subscription also indicates, if it refers to Process scoped events, what Processes it should listen to (among those the user has access to).

Finally the subscription can be marked for digest so that notifications are bundled in a

single email once per day (or another interval) and you do not get duplicate notifications for the same subscription per day.

An example, **tracking cases I am responsible for** (this could be a predefined subscription that the user can not change):

1. A subscription is added that listens to event type “Related case modified” in all Processes that the user has access to.
2. A filter is added that explicitly filters on Roles in the case with the “Responsible” Role object as key and with the value being “myself”.

Another example, **tracking cases in a specific process entering a specific stage with a specific checkbox checked** in a daily digest:

1. A subscription is added that listens to event type “Case moved to another stage” in a selected Process.
2. A filter is added that explicitly filters on the “Has XGF:” checkbox field set to true and Stage set to a selected value.
3. The subscription is marked as a daily digest, specifying a time of day when to get it.

The last example is a pre-defined subscription on the contact prototype object on the global level. This subscription will be used by all contacts in Gjallar and it makes sure **the contact receives an email when a reported case is resolved**.

1. The subscription listens to “Related case moved to another stage” in all Processes.
2. A filter is added that filters on the Stage field matching all stages categorized as “End stage” and the Reporter role matching the contact.

User interface

The web user interface of Gjallar should follow official standards like XHTML and CSS and not rely on web client specific coding or workarounds.

One user interface technique that probably would make Gjallar easier to operate is a tab based user interface hiding less used information in other non default tabs. This is a good way to hide information to lessen information overload on the user.

Note: An important feature in many issue trackers is the visual feedback given by colour coding connected to priorities or other values of interest like age or time since last action. Gjallar should of course offer the same thing, but not hard coded to a specific attribute.

Gjallar should aim to be fast as experienced by the user. This can be achieved by a good design, efficient HTML/CSS, a good user interface (avoiding very large tables to be generated) with a “my view”, good caching etc.

Reports & Export

The mechanism used to construct reports is basically the same as for constructing filters, the main difference is that you select fields to include instead of fields to place predicates on. A report can then be produced in various formats, like XHTML, XML, CSV or pdf. The filters are still used for the actual selection of data.

Exporting data can be done in two ways. The first way is a full dump of the database in

XML format. This is typically used when “all else fails” or when considering migration of the system or large scale offline transformations since the same format can be used as a full import. (XML dump not implemented as of 2006-08-03)

The other option is to selectively mirror data out into an ODBC source. The user uses the report tool to select subsets of data to export and then defines how often the ODBC source should be updated to reflect the contents in Gjallar. The update will only make incremental changes adding/deleting or updating records and only touch the fields used by Gjallar which means the user can add extra fields to the tables.

The user can then use Excel or other tools to operate on the mirrored data without having to worry about placing an extra load on Gjallar or making accidental changes to production data.

Help & Guides

Gjallar has a built in wiki so that the users can easily add and edit help directly in the system. Wiki pages are attached to Processes, Stages, Forms, Custom Objects, Fields and built in user interfaces. Wiki pages can also hold attached files which make it possible to easily add policy information directly available in the user interface of Gjallar.

For example, a Process administrator can write Word documents describing certain Forms or Stages in detail, or just make small helpful notes directly in the wiki pages helping the user.

The wiki pages can be disabled as a preference per user or on demand.

Architecture

Some concepts are fundamental for the architecture of the system and enable several mechanisms both in the short and the long term.

Transactions

Most case tracking systems have a simplified textual log showing how the case has been handled over time. They typically note when it was created, when and to whom responsibility was transferred and when important states of the case have been changed.

Unfortunately it is often “just a log”; incomplete and not implemented as central to the object model.

Gjallar models all changes in the system directly as **a chronological series of transactions**. This is especially apparent when looking at a case – the state of a case is at all times the accumulated state from applying all transactions that affects it.

The **current** state in the case (the fields described above including the dynamically added forms) is maintained in parallel to the transactions but it is logically redundant and those fields could be recalculated from the transactions alone.

The theoretical advantages of this internal model are numerous: There is **full traceability**, per definition. You can always see who did what when.

- You can view a case at a certain point **back in time**. This includes producing reports from the system as they would have looked at a certain point in time.
- You can **undo any change**, or recover data that has been changed.

- Each transaction groups the modifications of different fields and adds an optional transaction comment **making the changes more clear and cohesive**.
- Creating **online or offline mirrors is easy** since we implicitly get a model for incrementally replaying the transactions in the mirror to keep it synchronized.

There are also other advantages like being able to tie notifications to transactions creating fine granular notification and more advanced filtering and statistics. For example:

- *How many cases took more than x days before reaching a certain stage?*
- *How many cases were moved back to a previous stage it had already visited earlier?*
- *What cases have I edited in any way whatsoever during the last week?*
- *Show a graph of number of cases entering a specific stage over time throughout year 2004!*

Activity log

The transactional model fully tracks all modifications to the objects in Gjallar, but obviously does not track pure "read" operations. Tracking reads is important in order to allow activity visualization over time and to offer the ability of tracing information flow.

Internally Gjallar uses a granular low level event model and all events regardless if they are related to modifications or not, are logged in the activity log with enough context information to define the exact data read.

The activity log gives the system an ability to for example show the current activity or show what users have viewed a certain case over the last week or if there are cases with responsible persons that haven't yet had time to view their cases.

Mirrors & Offline operation

Gjallar has a mirroring mechanism enabling easy setup of a partial copy of the system on a different computer. A partial mirror can be used for:

1. Creating a partial copy of Gjallar to be used on a laptop computer for users temporarily offline, like being on an airplane or entering a secured area.
2. Offering a controlled subset of Gjallar to external parties (agents, customers etc)

The offline mechanism is a local full installation of Gjallar, typically on a laptop machine, but only with a personal subset of the Gjallar database. The local installation offers the same web user interface running locally and the user can work and record transactions, within a set of limitations, and can synchronize with Gjallar when connecting at a later time.

Such a personal mirror can only contain information that the user has access to since it would otherwise open up a possibility of "breaking" and reverse engineering data in the local installation.

Offline transactions made in a mirror, depending on their nature, may of course fail when being synchronized with Gjallar, for example, fields were edited in a case but the values were stale or objects have been removed since the copy was made.

Offering a controlled subset of Gjallar to external parties can be done in two ways, either a controlled subset is created and hosted locally, typically in an outer layer zone in the network to which access can be granted to the external parties in question. Such a mirror system is updated regularly online and is up to date within seconds, but only with partial content for increased security.

The second option is to offer such an installation hosted on an external machine on a network at a remote site (an agent or sub contractor etc) that typically also may have an unstable or slow Internet connection. The bandwidth needed for online synchronization is much less than the bandwidth needed to offer the online user interface. Again, such a system would only contain a partial mirror on a "need to know" basis.

Gjallar uses a series of chronological transactions as the internal model which is a design very suitable for creating mirrors. Essentially the mirror fetches the transactions performed in the master Gjallar system since the last update, and applies them locally. Then any local modifications are fed back to the master Gjallar in the same way.

In order to control the subset of information that the mirror should contain, the transactions are filtered based on the normal access model in Gjallar and on other filters created for the specific mirror before being sent.

A personal mirror is created in the following way: A user (with the proper access level to create mirrors) names and creates a mirror object in Gjallar intended for a specific user – typically him or herself. Or an administrator does it.

1. The user then downloads and installs the Gjallar software locally and starts it.
2. While still having network access to the master Gjallar the user opens the local installation and can then proceed to load the created mirror from Gjallar after authentication in master Gjallar.
3. Then the user can proceed to work locally offline.
4. At a later point in time the mirror can load updates from Gjallar. When all updates (=transactions that the mirror has missed while being offline) have been applied locally and any conflicts have been resolved the user can proceed with submitting the pending transactions he/she has made in the local installation and in turn resolving any conflicts with those.

An external party mirror differs from a personal mirror in the following ways: It automatically updates in both directions as soon as it can (given the connection) It may contain information that some of its users do not have access to (but others do). In other words, it is not "personal".

A personal mirror only updates (in any direction) when the user takes the initiative to do that. This also means that the user can deal with potential conflicts arising when updating. An external party mirror can not do that and instead relies on offering a proper subset of operations that are conflict free. The advantage is that it can contiuously load updates in both directions.

An external party mirror needs to be hosted in a somewhat trusted environment (locally or at the remote site) so that users can not try to reverse engineer the database in order to access data they do not have access to.

This means that a personal mirror is as powerful as running online in Gjallar, but is

normally only used when working offline. An external party mirror offers a reasonable subset of information, mainly for read only purposes and can also handle a limited functionality enough to interact with Gjallar – reporting new cases, tracking existing cases and responding to further requests while handling existing cases.

Permissions

Globally in Gjallar there is a very simple access model described under heading Global Scope where each user is given access to one or more Processes. Then each Process defines a more advanced permission model internally.

The Process wide model is similar to the model used in Unix file systems, but slightly more flexible due to the ability to categorize objects. It follows the principle of **selectively allowing instead of denying** access. Access models based on denial are generally prone to mistakes.

The basis of the model is that users are categorized using the global or process wide categories. This allows arbitrary orthogonal grouping of users and association with permissions.

The Permission objects are owned by the Process, and are associated with the users by being appropriately categorized into the same categories as they are. This means a particular user, given the categories he/she belongs to, is associated with a set of Permission objects in those same categories. The union of those Permission objects describes the total access space for the user within the Process.

If no Permission objects are created by the Process owner the **default access level is none**.

A Permission object is a list of categories where each category is associated with an access level (read, read/write) for objects in that category. Since most objects in the system can be categorized this covers cases, notes, case links, stages, users, custom objects and so on.

The Permission object also has a default access level (none, read, read/write) for objects that do not match any category. This makes it possible to establish a base level and is principally a step away from selectively allowing access – but it makes life easier for the Process administrators.

The only object that is also restricted by another mechanism is the **links to cases in other Processes**. In order to see a link you **additionally** need access to the Process of the linked case.

The only user with unrestricted access is the Process administrator(s) who specifies Permission objects, imports from the Gjallar level and makes other Process wide configuration.

A common restriction is to only allow the user to see a specific subset of forms and notes, or at least limit some forms as read only. This means that a user may have access to all cases, but not all entered information on them.

The access model is used for two different things:

1. To maintain online restrictions in Gjallar, for all users with access.

2. To define the subset of information created in mirrors.

Three simple examples:

Roger has access to three Processes; the IT-support process, the HR process and a small private work group process called X24.

The IT-support process has one Permission object in the “All users” global category with default permission “none” since there may be information in that Process that only the IT-personnel should be able to see and if the objects don’t match specific allowed categories, then it is safer to not show it at all by default. Then the Permission object specifies read access for some object categories and read/write for others.

IT-support also has an additional Permission object in the local “IT Support staff” category (applying to all support staff users) with read/write as default and no categories of objects listed – there is no need since the default is read/write. This is the way a certain group of users is given full access.

The Permission object for the HR process for “All users” has default “read” since there is nothing in there that can not be shared to all users with access to the process. It also has a similar second Permission object allowing read/write access to staff.

The X24 process has a single Permission object for “All users” with a default access level of read/write. That gives all users with access to the Process all rights and is the simplest configuration making all users equal (disregarding Process administators).

The logic for the system to find the appropriate access level follows these steps:

1. Check if the user is one of the Process owners, if so give read/write access.
2. Check the allowed Processes for the user against the current Process. If the process is not allowed, give no access.
3. For the Permission objects of the current Process, find those that belong to a matching category as the user belongs to. Then for all categories of the object in question, find all matching categories in the Permission objects. If any are found, pick the highest access level among them. If none are found, pick the highest default access level of the Permission objects.
4. If no Permission objects were found in step 3, give no access.

Scripting

Normally subscriptions are used for notifications about when certain modifications are made to the objects. These modifications are mainly intentionally made by logged in users.

But in order to allow alarms or automatic “actions” Gjallar needs a generic way to react and optionally make transactions to automatically modify objects when certain criteria are met.

In Gjallar scripts can be registered by:

1. Subscribing it to a set of events
2. Scheduling it in time

When the script is triggered it can do much anything like: Signalling new custom events.

- Make arbitrary checks and validations in Gjallar.
- Perform arbitrary transactions to modify objects.
- Change its schedule or set of subscribed events.
- Amend the activity log entry with additional information.
- Write to the system log.
- Work with external files.
- Perform any function that can be performed in Gjallar using the UI.
- Maintain private persistent data between invocations.

This means that administrators with good knowledge of the system will be able to plug in any kind of automatic behaviour and when they are executed they get properly logged in the activity log.

System log

Gjallar has a system log which tracks everything happening in the system outside of the domain model. This log is only available to Gjallar administrators and is primarily intended to keep track of system level problems like various malfunctions, external integrations or backup routines etc.

Integration

In order to facilitate better integration between Gjallar and other systems the custom objects in Gjallar can be populated by external sources using ODBC queries that are executed on a scheduled basis. Possibly other ways of input are needed, like simple file input in XML or CSV format etc.

In order to make it easier for other systems to integrate online with Gjallar a SOAP web services API should be available.