Overview

# What is Heraldry?

Heraldry is a network application that accepts text input from users. These messages are stored in a database and distributed in real time to other users.

Heraldry was developed as a tool for an IT operations department. Changes needed to be logged for auditing purposes and the other staff must be informed of those changes. Heraldry combines these two requirements.

# Communication Workflow

Heraldry uses asynchronous messaging for exchanging information. The messages are transported via Microsoft Message Queuing (MSMQ). This allows a client to go off line and get the messages when it is back on line.

Storing messages in a database and distributing them among the clients is the responsibility of a server process. All clients send their messages to the server. The messages are stored in a database by the server and are then distributed to the other clients using a Publish/Subscribe pattern.

The general communication workflow is shown in figure 1. A user creates a message on a client and sends it to the central server (1). The server stores the message in a database to support searching these messages.

The message is then published to other users (2). Some users, for example the helpdesk, can only receive messages (2a). Users can also choose to receive only messages that belong to certain categories, for example only messages that are marked as important, messages that are related to the mail service, or that are related to a single server machine.

Figure

The latter filtering is also used on servers. These servers receive messages that are related to them and they add the messages to an audit log, for example a simple text file, or the Windows event log.

## Communication Details

Figure 2 shows the details of how the messages are sent between the different parts of the application:

Figure

1. The sender creates a message object and sends it to the central server. The message object contains the following fields:
   * The message’s content,
   * a timestamp reflecting the point of time at that the message was created and
   * the message’s categories or tags as an array of strings.
2. The server creates another object that can be saved to a database. It will add at least to other bits of information:
   * The time the message was received by the server and published to the other users.
   * The name of the user that created the initial message.
3. The newly created object is sent to the other users after some conversions took place.

It is important to state that neither the sender nor the recipient require access to the database to send or receive a message. This is reflected by the fact that no database related IDs are sent over the wire, but full message objects containing the actual data.

## Message Details

The objects that are used for messaging are not normalized in any way. Especially the tags/categories are not looked up in a database but added as strings. The reason for this is allowing a client to send a message or receive and display a message without having database access at all.

Categories can be handled strictly, meaning that it is only allowed to choose them from those already configured. This would be enforced by not using strings but the IDs of existing Category-objects. But we have to keep in mind that we are creating an asynchronous system. Many clients can be preconfigured to use only a few well-known categories. An example is a service on an application server that sends a notification when it has to reboot.

Other clients might only want to display or cache the message without going to a database. The before mentioned application server might have a service process that receives all messages about the application it hosts and write them to a text file that contains other log data not distributed over Heraldry.

# Which technology is used?

Heraldry uses three main technologies:

1. Messaging is implemented with NServiceBus, a configurable service bus for .Net that builds upon MSMQ and exposes a simple yet powerful API over it.
2. Database access is done with Castle ActiveRecord which is an ORM framework built upon the well-known NHibernate ORM.
3. GUIs are implemented in WPF, which allows rich, customizable graphical interfaces.