The K^2 System: Lisp at the Core of the ISP Business

Espen J. Vestre

Nextra AS

1 Introduction

The Nextra Group, a subsidiary of Telenor (the norwegian Telecom), is Norway's largest ISP, with over 400.000 single-user dialup accounts, a 70% market share in the norwegian dialup market. Besides Nextra Norway, there are subsidiaries in the Czech Republic, Slovakia, Hungary, Austria, Switzerland, Italy, Germany, Italy and Sweden.

For efficient and automated (self-service) customer service, an ISP needs smooth integration between the different service-implementing servers, customer service applications and billing. We have achieved this through what we call a *core system*, which is responsible for authorative customer information, username reservation, password and configuration distribution, and so on.

Such a core system not only needs to serve todays needs, but must be able to adjust to the extreme speed of the internet business, both in terms of volume growth and with respect to the introduction of new products and services.

After evaluating and rejecting several commercial systems, we decided to implement our own core system. The system, which we call K^2 (acronym for "Kjernesystem 2", meaning "Core System 2") is a 4-, or even 5- (counting web clients) layer client-server architecture, based on ORACLE, which has a set of servers implemented in almost pure Common Lisp as its most vital parts.

The decision to implement large parts of the system in Lisp was controversial, but the system has proved to be very stable, very flexible, and very scalable and efficient. The power of Common Lisp and CLOS reduced the implementation efforts and ensured flexibility.

2 System Overview

The system consists of the following layers[1]

2.1 Layer 1

At the base is the "Base Camp", an ORACLE database. There is also a log database, running on a different machine. K^2's architecture is easily extended to support several databases of several kinds, which all can reside on different machines.

2.2 Layer 2

The next layer is "Sherpa" which is a simple server implemented in C. Sherpa uses standard libraries to connect via TCP to Oracle. Sherpa provides simplified access to the database and returns values in Lisp-readable format. Each sherpa process is a single-threaded unix process. Each underlying database (main database and log database) needs its own sherpas.

2.3 Layer 3

Layer 3 is the most important one, and is implemented entirely in Common Lisp, using Allegro CL 5.0. At layer 3 there are 4 different multithreading Lisp servers (one of which currently runs on two machines for high availability). These servers share a large Lisp code base (and a common S-expression-based TCP protocol), but play somewhat different roles. They all communicate with sherpa through a pool of a configurable number of sherpa sessions.

Hushe (High-level USer Handling Engine) provides the main interface for entering and updating objects in the database. When fetching rows from the database, Hushe converts them into CLOS objects representing instances of classes corresponding to each table, e.g. "CUSTOMER" objects. Hushe provides low-level commands for finding, fetching, storing and updating such objects, and high-level commands for e.g. different typical ISP tasks like creating new accounts, changing the product-type for accounts, changing mail aliases and so on. Hushe provides a very fine-grained and configurable security system, giving access at all levels from ordinary customers who may

[1]The acronyms, mostly related to the mountain K^2 (Hushe is a valley close to K^2in the Karakoram), are inspired by the K^2 acronym itself and a workshop in the norwegian mountains...

only do simple operations on their own account, through local administrators in companies to our own customer care representatives and "super users".

Indus (INcremental Distribution of User Setup) provides read-only access to the actual configuration of the internet accounts for a variety of services, and keeps track of changes, enabling large-scale services to keep local password and configuration databases in sync[2] with K^2.

In addition there are two different servers for usage collection and real-time information (these are still under development).

Common to the layer 3 servers is the K^2 *product* modelling code. In K^2, each user account may be activated by one or more products. Each product gives access to a number of *services* (like dialup, mail, ftp, ...). For each product-service-pair, a *service-class* is assigned. Each service-class defines a set of attributes and default values for those attributes. Thus, different products may activate different attributes with different values. These attributes are used by the indus clients to implement the actual level of service and configuration for the given account. For example, our free mail service does not include the facility to forward messages. Thus, the forward-attribute does not appear in the mail configuration (i.e. the mail service-class) of the free mail product, but it does occur (with an empty default value) in all other products that include the mail service. The indus client for mail does not have to know *anything* about this underlying logic. It just configures the accounts with the set of attribute-value pairs that indus supplies.

2.4 Layer 4

At Layer 4 there is a Lisp-based webserver ("Climb") which is used by customer service for interaction with K^2. This webserver implements a dynamic, customizable and userfriendly GUI[3]. The webserver manages open TCP-sessions to Hushe (keeping one session open for each customer representative), ensuring very quick responses to most commands.

Other users (e.g. ordinary dialup customers) access Hushe through traditional web-servers with Perl CGI-scripts or java servlets.

Numerous scripts for different tasks (e.g. mass production of accounts) also operate at layer 4 (as Hushe clients).

[2]During normal operation, new accounts are activated within seconds.

[3]Colors, font sizes and even the language of the texts is customizable per user and "on the fly"

3 Lisp Advantages

Originally, our plan for K^2 was a very modest one. We wanted to build an intermediate system to serve our most immediate needs, and then later have a second evaluation of the commercial alternatives. But K^2 has been so successful that the original plans for a second evaluation round have been discarded.

We think that the choice of Common Lisp as the implementation language can account for a large part of the success of K^2. Some of the most interesting and useful features of K^2 are actually very dependent on unique features of modern Lisp environments:

3.1 Flexible Product Modelling With CLOS

K^2's flexible product and service modelling is probably its most unique feature compared to other similar system.

One problem with many conventional systems of this kind, is that support for new services has to be hardwired into the system, from database and upwards. K^2 takes a very object-oriented approach to product and service modelling, where products and service classes are represented throgh CLOS classes. These are stored (indirectly) in the Oracle database, and loaded into the Lisp servers on demand (generating the classes on the fly). Through multiple inheritance, we can very easily define new products as combinations of existing ones, dramatically reducing implementation time.

3.2 Incremental Development on Running Servers

K^2 has been in constant development since its release, but still the server processess have had runtimes of up to 3 months - during which there were numerous bug fixes and enhancements. The servers run as background processes, but we have integrated telnet servers into them, through which we can connect to a real Lisp listener and patch them with "fasl"-files while they are still "live".

The ability to modify running code is one of the features which makes Lisp extremely useful for "netcentric" computing: The advantage of server-based software is even greater when the servers can be patched without even having to notify the users in advance of scheduled disruption.

With Lisp we can have more frequent changes to the software while still keeping very high availability.

3.3 Lisp-based webserver

A multi-threading webserver without the overhead of old-fashioned CGI is a *must* for an application which has to do extensive session tracking like the K^2 webserver. The dynamic capabilities and built-in threads support of modern lisp environments make them ideal as webserver platforms.

3.4 S-expression-based TCP protocols

For interprocess communication we use S-expression-based TCP protocols. There are Lisp, java, perl and other libraries for talking to the servers, and through a uniform syntax with keyword-based command parameters and keyword-plist output, adaption to new commands is very easy.

3.5 Fast Development, High Quality

K^2 was developed with very limited resources (with only one Lisp programmer until 3 months before the initial launch) in a short time, and it is still under constant development (there are now 4 programmers working on K^2). The garbage collection and error handling of Lisp makes it easy to keep errors under control. Only very rarely do errors make the servers actually defunct, most of the time an error in some part of K^2 only affects a few users and the bug can be fixed quickly and the patch can then be loaded into the running servers.

3.6 Other Lisp Advantages

1. Keyword arguments to functions have proven to be extremely valuable, making it very easy to make backward-compatible changes to server commands.

2. We make essential use of *multi-methods* in K^2.

3. Modern Lisp environments have excellent support for *multi threading*.

4. *Macros* help making configurable parts of the code easy to read and write.

5. Efficiency: Lisp is not what comes to everyone's mind when considering efficiency, but we were surprised how efficiently our code runs, especially when compared to other internal systems, both traditional relational database applications and more modern java-based software.

4 Conclusion

Using Lisp has helped us to build a complex system with limited resources. It enables us to continuously develop the system while avoiding "kludgy" solutions and while keeping very high availability. Lisp has also made the system very adaptive to changes in the business model and the spectrum of products offered.

Encouraged by this, one of the next steps we will make, is to move even more of the surrounding systems into Lisp.