

# Design of Prototype Models for Some Problems Facing Large-Scale Configuration Management

*George Beckett* <[g.beckett@epcc.ed.ac.uk](mailto:g.beckett@epcc.ed.ac.uk)>  
*Kostas Kavoussanakis* <[k.kavoussanakis@epcc.ed.ac.uk](mailto:k.kavoussanakis@epcc.ed.ac.uk)>  
*Guillaume Mecheneau* <[guillaume.mecheneau@hp.com](mailto:guillaume.mecheneau@hp.com)>  
*Jim Paterson* <[jpaters2@inf.ed.ac.uk](mailto:jpaters2@inf.ed.ac.uk)>

## Abstract

This is the fourth report from the GridWeaver project. It describes a demonstrator that showcases a subset of the concepts explored in the previous report [[ABK<sup>+</sup>03a](#)]. The aim of the demonstrator is to illustrate that we can model, configure, and manage a complex, adaptive system, using our prototype LCFG/SmartFrog architecture as the deployment environment. To achieve our goal, we created a prototype, OGSA-compliant printing service, which we call *GPrint*.



**Revision 1.1 – September 29, 2003**



# Contents

<b>1</b>	<b>Introduction</b>	<b>5</b>
<b>2</b>	<b>Demonstrator scenarios</b>	<b>7</b>
2.1	Overview . . . . .	7
2.2	Composition of the demonstrator . . . . .	7
2.3	Story 1: Deployment of GPrint . . . . .	8
2.4	Story 2: Using the GPrint service . . . . .	11
2.5	Story 3: Failure recovery . . . . .	13
2.6	Story 4: Failure recovery II . . . . .	13
2.7	Story 5: Adding a new node and installing the Globus Toolkit software . . . . .	14
2.8	Story 6: Installation of a Grid service (Globus) . . . . .	15
2.9	Summary of the functionality of GPrint . . . . .	16
<b>3</b>	<b>Demonstrator design</b>	<b>17</b>
3.1	High level design . . . . .	17
3.2	Printing framework design . . . . .	17
3.2.1	The interface between Globus and the framework . . . . .	17
3.2.2	Description of existing LCFG/LPRng Printing Framework . . . . .	20
3.2.3	GPrint Printing Framework . . . . .	21
3.2.4	Configuration information . . . . .	25
3.3	Globus Toolkit installation . . . . .	26
<b>4</b>	<b>Experiences from the Demonstrator</b>	<b>29</b>



# Chapter 1

## Introduction

This report is the fourth in a series of deliverables from the *GridWeaver* project [Edi], a collaboration between Edinburgh University School of Informatics, HP Labs in Bristol, and EPCC, as part of the UK e-Science Core Programme. GridWeaver is focused on the challenges of managing the configuration of very large-scale computational infrastructures, especially those that form parts of computing Grids. Our goal is to develop approaches and technologies that are capable of representing and realising complex system configurations of hardware, software and services, across very large-scale infrastructures incorporating high degrees of heterogeneity, dynamism and devolved control.

In our first report, *Technologies for Large Scale System Configuration Management* [ABK<sup>+</sup>02], we discussed the basic approaches in use for configuring and managing large fabrics, and examined a number of specific system configuration technologies. We concluded that there is no complete tool to tackle all aspects of fabric management. The technologies available from the consortium were known to address parts of the problem, but they were found to be complementary and thus have a lot of potential.

In the second report, *Experiences and Challenges of Large-Scale System Configuration* [ABK<sup>+</sup>03b], we identified the fabric requirements of the emerging Grid world. We first presented a number of real world Case Studies, to illustrate how several different organizations manage large-scale system fabrics. We examined how well the technologies employed are currently meeting the requirements placed upon them, and looked for any limits being encountered now or in the future, extending to an investigation of the fabric managers' thinking. We also introduced some Projected Use Cases that aim to predict the way in which large-scale system configuration will need to evolve in order to meet the challenges of tomorrow's fabrics. We concluded that there is a paradigm shift that is taking place in the way we think about computing infrastructure, and indicated the research areas that the GridWeaver project will focus on.

In the third report, *Large-Scale System Configuration with LCFG and SmartFrog* [ABK<sup>+</sup>03a], we identified key concepts and idioms used in the modelling of system configuration, including an examination of possible representations for these concepts in the language of SmartFrog. We also provided three examples of complex system configuration description, designed to illustrate these concepts. Finally, we examined how our two complementary technologies – LCFG and SmartFrog – could be combined to provide a prototype

deployment environment to aid experiments and research into the concepts and idioms previously identified.

In this fourth document, we present a demonstrator for some of the concepts explored in the previous report [ABK<sup>+</sup>03a]. The aim of the demonstrator is to illustrate that we can model, configure, and manage a complex, adaptive system, using our prototype LCFG/SmartFrog architecture as the deployment environment. To achieve our goal, we created a prototype, OGSA-compliant printing service, which we call *GPrint*.

It is important to stress the fact that the focus of the present report is not on the design of this printing service itself; the GPrint example is merely a skeleton service. The intention is to outline clearly how we drive configuration of complex services, from basic aspects of the fabric up to complex recovery and adaptive mechanisms, through the combined architecture.

The structure of this report is as follows. In Chapter 2, we define the functional specification of the demonstrator, and explain it using six storyboard scenarios. A design for the component elements is presented in Chapter 3: this covers the architecture of the printing subsystem plus installation requirements of a basic Globus Toolkit environment. A video presentation prepared from the demonstrator may be viewed at [UoEEL].

## Acknowledgements

The report is a deliverable of the *GridWeaver* project, a collaboration between the University of Edinburgh School of Informatics, HP Labs in Bristol, and EPCC.

GridWeaver is part of the UK e-Science Core Programme, and we gratefully acknowledge the programme's assistance with this activity. We also wish to thank the other members of the GridWeaver team – Carwyn Edwards, Patrick Goldsack, John Hawkins – for their contributions to the project as well as Alexander Holt for his valuable input.

For administrative purposes, GridWeaver is also known under the project name *HPFab-Man*.

## Chapter 2

# Demonstrator scenarios

### 2.1 Overview

In this chapter, we describe the composition of the GPrint prototype. We also outline the functionality of the demonstrator with six individual storyboard scenarios. These scenarios explain:

- how GPrint facilitates the configuration and deployment of a printing system;
- a typical end-user interaction with the demonstrator through a Grid interface;
- the way that GPrint responds to two different types of hardware failure;
- the procedure for adding a new “bare metal” node into the fabric;
- how to deploy an OGSA service into the Globus installation running on the demonstrator.

The scope of the functionality is summarised at the end of the chapter, in Section 2.9.

### 2.2 Composition of the demonstrator

The demonstrator consists of four personal computer (PC) nodes and two Postscript printers.

The four nodes (with hostnames *server*, *client#1*, *client#2*, and *client#3* in this description) run an installation of the Red Hat Linux distribution configured using LCFG. Node *server* is pre-configured as an LCFG server using the LCFG Installation Package (described in [ABK<sup>+</sup>03a]). Nodes *client#1*, *client#2*, and *client#3* are configured as LCFG clients. The configuration of each client node is managed by the LCFG server.

The two printers are named *sesame* and *rainbow*. These are both network printers (i.e. exist within an IP network as autonomous nodes, configurable with DHCP) and accept Postscript print jobs. The printer *sesame* is a black and white (b/w) printer, while printer *rainbow* is a colour printer which is considered capable of printing large, intensive jobs.

For the purposes of the demonstration scenarios, *client#1*, *#2* and *#3* are designated as candidate print servers. The actual configuration of the printing service that runs on the fabric is established as part of the demonstration. Each printer is to be served by either *client#1*, *client#2*, or *client#3*, according to the state of the demonstrator at any particular instant. During normal operation, *client#2* and *client#3* are the preferred hosts for printing services, and *client#1* is a backup node.

The node *client#1* is pre-installed with Globus Toolkit Version 3 (GT3). This installation is performed using LCFG. It will be the access point for transactions between Grid clients and the GPrint services that are running on the cluster.

Before a printer can accept print jobs it must be associated with a print queue: this association is made dynamically by GPrint during operation. Two queues are statically defined: these are *colourps* and *bwps*. In the preferred configuration, *colourps* is associated to *rainbow* (the colour printer) and *bwps* is associated to *sesame* (the black and white printer).

## 2.3 Story 1: Deployment of GPrint

This story demonstrates how simple it is to maintain the configuration of a complex aspect of a fabric: in this case, the printing sub-system. For usability reasons, we developed a GUI that allows the administrator to interact with the demonstrator. However, this GUI is in direct correspondence with the underlying SmartFrog description of the system, and an administrator would be equally well-equipped to implement changes by editing the SmartFrog description.

In the story, an administrator interacts with the GPRINT\_CONSOLE GUI (see Figures 2.1 and 2.2) on *client#1*. This presents a small, but interesting subset of the configuration parameters from the printing service model, allowing the user to make changes to the configuration of the service.

In the “Status” pane (see Figure 2.1), we see an interface that allows the administrator to prescribe or review the current printing configuration implemented by GPrint. In the upper panel, one can see the current association of print servers to queues and queues to printers. This table is indexed by queue name – queues which are currently not administered by a print server, appear in red. Queue names which appear in black have been successfully linked to a print server (and printer).

In the lower panel, one may view a list of the nodes that are candidate print servers. Here, nodes which are currently not running a print service, appear in red. Nodes in this panel can be selected with the mouse and printing services can be started or stopped by pressing “Start” or “Stop”, respectively. When a print service is successfully running on a node, the text colour changes from red to black.

In the second pane (Figure 2.2), one can see an interface that allows the administrator to specify some simple permissions for each user defined within the demonstrator. Permission may be granted to allow a user to print: b/w, colour, small, and large documents. The definition of what is a large document is done in a SmartFrog description and is subject to the service level agreement of the site in question. For example, an administrator may

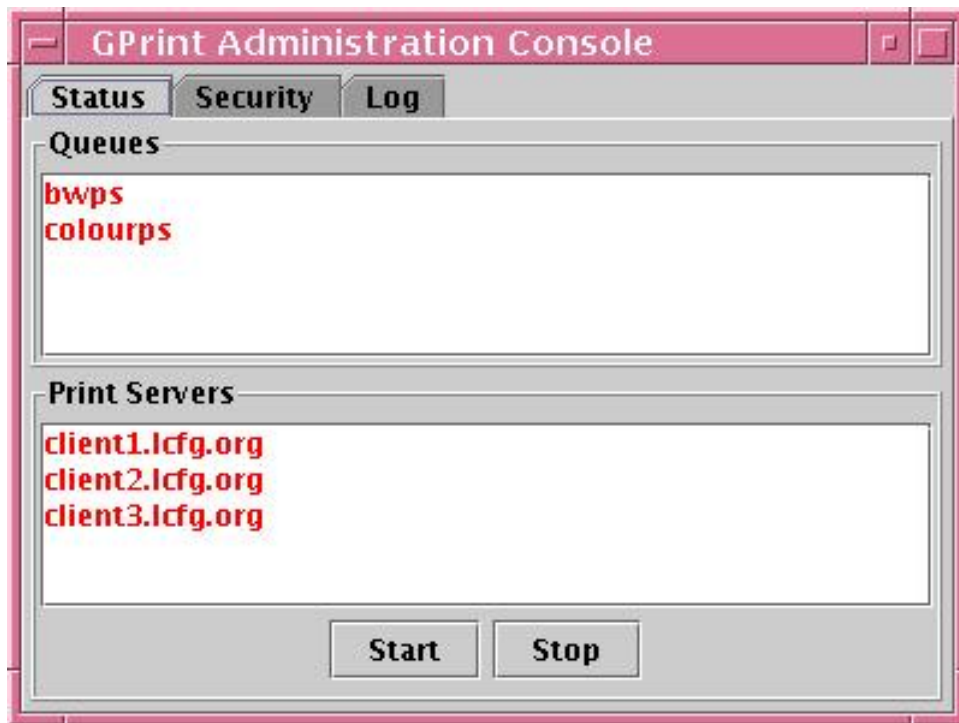


Figure 2.1: Prototype of the GPRINT\_CONSOLE GUI which is presented to the administrator on *client#1* during the demonstration. This first pane presents information regarding the status of the printing sub-system and allows the administrator to start and stop print servers on designated nodes.

define a large document to be a Postscript file greater than 2Mb in size. Once the administrator is satisfied with their parameter choices, they press the button labelled “Apply” which instructs GPrint to reconfigure the printing permissions in light of the configuration information input. The administrator can also add and remove users from this pane of the GUI.

In the third pane, labelled “Log”, a SmartFrog console is presented. This provides information regarding the resources (printers and print servers) which the GPrint service discovers/changes. A possible output might look like the following:

```
> Welcome to the GridWeaver GPrint demonstration.
> This is an example of configuring an OGSA-enabled
> Grid service with SmartFrog.
> GPrint is being reconfigured...please wait
> Looking for printers.
> Found printer: sesame@lcfg.org
> - this is a Postscript printer.
> Found printer: rainbow@lcfg.org
> - this is a colour Postscript printer.
> Checking print servers
> Print server client2.lcfg.org ...okay.
```

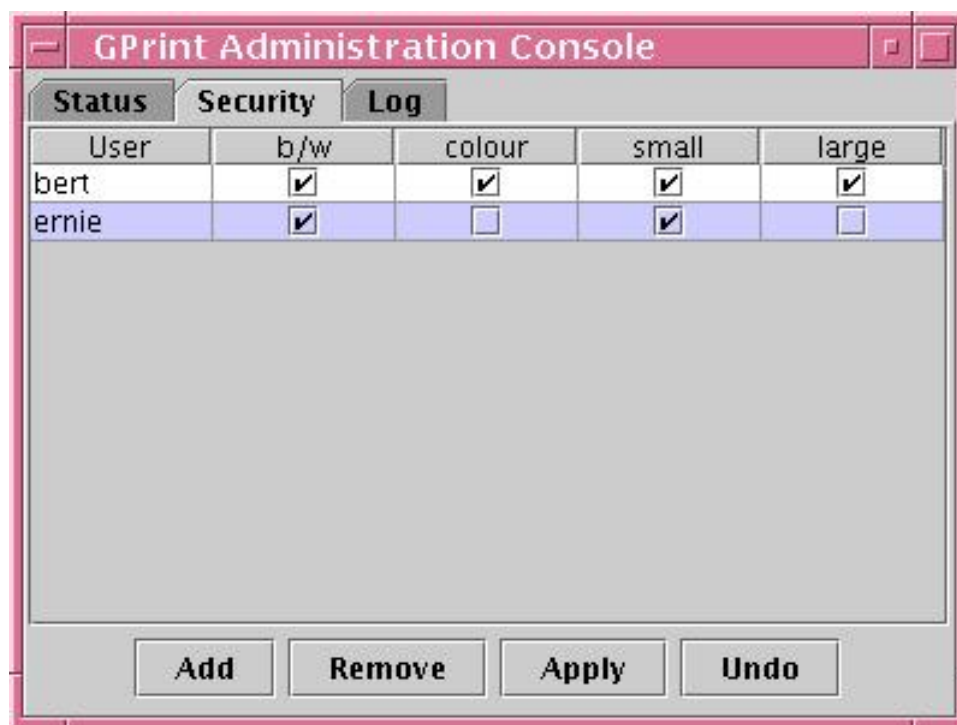


Figure 2.2: Prototype of the GPRINT\_CONSOLE GUI which is presented to the administrator on *client#1* during the demonstration. This pane presents information regarding user permissions which allows the administrator to control the privileges allocated to designated users.

```
> Print server client3.lcfg.org ...okay.
> Allocating printers/queues to print servers:
> rainbow@lcfg.org (colourps) allocated to print server
  client2.lcfg.org.
> sesame@lcfg.org (bwps) allocated to printer server
  client3.lcfg.org.
> Configuring LPD.
> Creating permissions.
> Starting LPD daemons.
> Starting print service monitor.
> The service is ready...
```

The story proceeds as follows. On the server, the administrator opens the GPRINT\_CONSOLE and selects the Security pane. From here, they create a new user account called *testuser*, allocating a selection of printing permissions to this new account. They then switch to the Status pane and start a print server on *client#2*. After a few moments, the Print Server entry for *client#2* changes colour from red to black (indicating that a printing service has successfully started on the node). Furthermore, the printing system automatically assigns the two queues to *client#2*. The administrator then starts another print server, this time on *client#3*. Again, after a few seconds the Print Server entry for *client#3* changes colour

from red to black. Furthermore, one of the print queues is migrated from *client#2* onto *client#3* to maintain a simple load balance of queues across print servers.

The administrator may continue to interact with the GUI, starting and stopping print servers as necessary.

## 2.4 Story 2: Using the GPrint service

This story demonstrates that we have successfully deployed the Globus Toolkit 3 and a custom Grid service onto the cluster. The end-user interacts with the LPRng[lpr] printing software running behind the Grid interface as if it were a locally installed application. While, in this example, the user interacts with a GUI, one could just as easily overload the Unix commands `lpr`, `lpq`, and `lprm`.

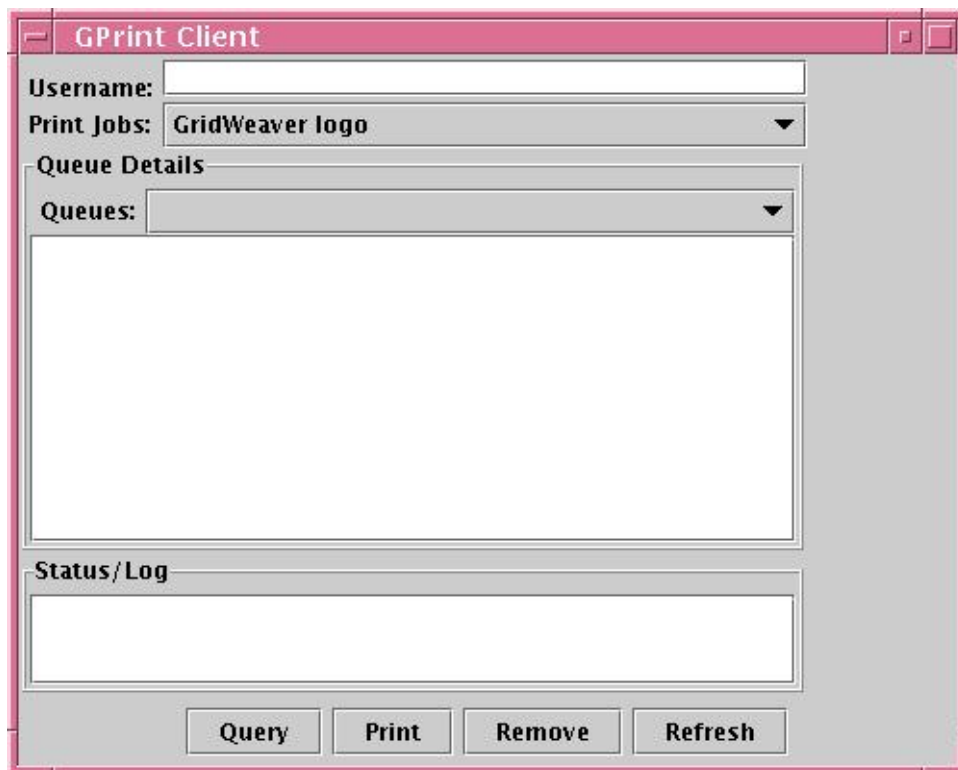


Figure 2.3: Prototype of the GUI (CLIENT\_CONSOLE) which is presented to the user on *Server* during the demonstration.

The CLIENT\_CONSOLE GUI running on *server* displays a simple interface that allows a user to work with the GPrint service. A sample of this GUI in a typical state is presented in Figure 2.3. As can be seen, the GUI is divided into four panels:

- At the top is a text box labelled “Username” in which the user may enter their username (the user should only enter a name which has previously been registered in the GPRINT\_CONSOLE window by the administrator). The username specified

in this dialogue is passed to the GPrint service with any requests issued. It is used to determine the privileges attributed to the job submitted by this user, based on the permissions specified in the GPRINT\_CONSOLE.

- Below the top dialogue is a pull-down list called “Print Jobs”. This provides a choice of two possible Postscript images for printing: (i) a colour Postscript document containing the GridWeaver logo<sup>1</sup>; (ii) a black and white Postscript document containing a photograph of the GridWeaver project team.

Once a print job has been selected, it can be submitted for printing by selecting the target queue in the “Queues” pull-down and then pressing the “Print” button. Depending on the permissions associated with the specified user, the job will either print successfully or be rejected.

- The third panel, labelled “Queue details”, provides status information on the queues which are being managed by GPrint. The information displayed in this pane is a snapshot of the state of the service at a point in time. It is refreshed by pressing the “Refresh” button.

The white text box within this panel will display information relating to the queue which is currently selected in the “Queues” pull-down list. Possible content is:

- a list of the print jobs currently awaiting printing.  
In this situation, it is possible to remove a job from a print queue by highlighting the job in the list with the mouse and pressing the “Remove” button;
- the message `<Queue empty>` in the event that no print jobs are waiting on a queue;
- the message `<Queue not responding>` in the event that an error occurred during the query. In this case, more information may be provided in the Status/log dialogue.

- The fourth panel, labelled “Status/Log”, displays a copy of the GPrint log file.

The CLIENT\_CONSOLE is started from the command-line. A typical invocation of the CLIENT\_CONSOLE is of the form:

```
gprint-client -Hhttp://server.lcfg.org:8080/
```

The address given is the server from which the GPrint service is accessible. Once CLIENT\_CONSOLE is running, the user enters their username (e.g. `testuser`) and selects the GridWeaver logo from the pull-down list of print jobs. The user clicks the “Refresh” button in order to query the GPrint service for available queues. The result of the query is displayed in the “Status/log” panel. Assuming any queues are available on the cluster, the user then selects a valid queue from the “Queues” pull-down menu and selects the “Print” button. The job is submitted to the queue and, assuming that the user has appropriate permissions, the Postscript document is printed. Otherwise, if the user does not have appropriate permissions, an error message is printed in the Status/Log panel.

---

<sup>1</sup>This option is shown in Figure 2.3.

## 2.5 Story 3: Failure recovery

This scenario illustrates the failure recovery functionality implemented within the prototype and demonstrates the implementation of a simple service level agreement. Specifically, it shows how the printing service responds to the failure of a print server.

To initiate the demonstration, the administrator disconnects one of the print server nodes. GPrint notes the loss of the print server and suspends any queues which are being served by the affected node. Then, GPrint re-deploys the orphaned queues on an alternative print server<sup>2</sup> (assuming any are left). If no servers remain, then all queues will be suspended until a print server is added to the system.

A notification message will be posted to the GPRINT\_CONSOLE. However, no messages are submitted to the CLIENT\_CONSOLE, since the failure should effectively be transparent to the user.

## 2.6 Story 4: Failure recovery II

This story demonstrates how the configuration environment can respond automatically to events generated outwith the framework. Specifically, we demonstrate how to monitor a network printer and implement contingency measures in the event of a hardware failure. The story also indicates how we tackle the concept of a convergent configuration: there is a preferred mapping of printers to queues which is only compromised in the event of a failure. Once the failure is resolved, the system returns to its preferred state.

While the GPrint service is running on the cluster, one of the managed printers will be disconnected from the network (for example, by unplugging the ethernet cable). The system (which is monitoring the state of the service) will note the disconnection and display an appropriate warning in the GPRINT\_CONSOLE. It will then suspend the queue associated with the disconnected printer.

Any subsequent print jobs which are submitted to the service and are determined to be destined for the affected printer will be rejected and the user will receive an error message of the form:

```
> Print request failed: printer 'colourps' could not be  
> contacted.
```

To complete the story, the printer is reconnected and, as previously, SmartFrog responds to the re-connection and reconfigures the GPrint service to its original state.

---

<sup>2</sup>Given the purposes of the demonstrator, it was not necessary for us to ensure jobs were preserved and migrate the content of a queue's spool directory from one server to another. Hence, in the event of a print server failing, any print jobs which are spooled for printing will be lost.

## 2.7 Story 5: Adding a new node and installing the Globus Toolkit software

This story demonstrates how easy it is to add a new node to the cluster with a few simple operations. It also illustrates how easily a complex software packages (in this case, the Globus Toolkit) can be installed on to a node. This is done by augmenting<sup>3</sup> the description of a node with a template software configuration.

During this demonstration, a fifth node called *client#4* will be added to the cluster and the Globus Toolkit will be installed on it. To start the demonstration, in a terminal on the server node, the administrator (with root privileges) creates a configuration description for the new node (by copying an existing LCFG node profile). This description is opened in an editor. The following shell commands could be used:

```
server> cd /var/obj/conf/server/source
server> cp client1 client4
server> emacs -nw client4
```

A typical client profile looks like:

```
/* Node client1 - GridWeaver client machine */
#include <mutate.h>
#include <gw_client.h>
#include <locations/gw_inf.h>

/* Add node-specific hardware requirements here */
#include <hardware/gw_dell_optiplex_gx260.h>
#include <hardware/video_radeon.h>

/* Add node-specific information here */
```

The administrator edits this file to confirm that the hardware settings are correct for the target machine (and possibly modifies the first comment line to read “Node client4 ...”).

These modifications to the configuration are sufficient to create a generic client machine. However, we would also like to install the Globus Toolkit Version 3 on this machine. To do this, the administrator adds an additional line to the profile<sup>4</sup>:

```
#include <gw_globus.h>
```

---

<sup>3</sup>Augmentation is a functionality provided by the provisional SmartFrog 2 language specification. It allows separate configuration descriptions to be combined into a single description. A detailed description of augmentation can be found in [ABK<sup>+</sup>03a].

<sup>4</sup>It was the original intent of the project team that node configurations would be written using the SmartFrog 2 (SF2) language. In the absence of SF2, we have instead utilised the LCFG language. The SF2 augmentation operator & may be expressed by the use of the preprocessor #include directive.

Having done this, the administrator: saves and closes the client profile; inserts the LCFG Installation CD-ROM into the drive on the *client#4*; and reboots the machine<sup>5</sup>. Installation and configuration of the new node proceeds automatically. Comprehensive details of the LCFG installation process can be found in [ABK<sup>+</sup>03a].

Prior to the commencement of the demonstration, the node is physically connected to the subnet and its MAC address and IP address are registered with the naming service running on the network.

## 2.8 Story 6: Installation of a Grid service (Globus)

This story demonstrates how simple it is to deploy a Grid service within the Globus environment running on the demonstration cluster. The service that we will deploy is GPrint.

Firstly, the service GAR file must be published on the web server (running on node *server*). This requires that the GPrint RPM file be added to the package list on *server*. To do this, the administrator opens the `/var/obj/conf/server/include/gw_ogsa.h` file and adds the line

```
!profile.packages mADD(gprint-ogsa-*-* )
```

Once saved, LCFG ensures that this change propagates through the configuration, automatically.

Then, to deploy the GPrint service within the Globus container, the administrator opens the server profile for editing (on *server*):

```
server> cd /var/obj/conf/server/source
server> emacs -nw server
```

and adds the GPrint service description by including the line

```
#include gw_gprint.h
```

to the profile (this should be placed after the GT3 `#include` directive). The file `gw_gprint.h` contains the following GPrint service description:

```
/* Add the GPrint OGSA portal to the list of OGSA services */
!globus.ogsa_services mADD(gprint)
globus.name_gprint gprint
globus.gar_url_gprint http://server/gprint.gar
```

Having completed these steps, the GPrint service is ready for use. This can be confirmed by starting the service from one of the client nodes.

---

<sup>5</sup>It is merely a matter of engineering to all of these steps. For example, using PXE-enabled nodes instead of a bootable CD-ROM, etc.

## 2.9 Summary of the functionality of GPrint

The functional content of the six stories presented in this chapter can be summarised in the following points. The demonstrator illustrates:

- configuration and maintenance of a printing system (Section 2.4).
- Dynamic discovery of (printing) services using SmartFrog (Section 2.3).
- Automatic, dynamic reconfiguration of a service in the event of a hardware failure (Sections 2.5 and 2.6).
- Simple and automated installation and configuration of GT3 (Section 2.7), by augmenting a node description with a template GT3 configuration description.
- Configuration of an OGSA-enabled Grid service using SmartFrog (Section 2.8).
- Installation of an LCFG client onto the network (Section 2.7).

The demonstrator does not illustrate:

- Installation of an LCFG server onto the network. This procedure takes approximately half an hour to complete and duplicates the functionality already demonstrated by adding a client node to the fabric: this is done in Section 2.7. The procedure for installing an LCFG server is described in [ABK<sup>+</sup>03a].
- The SmartFrog 2 language and compiler in action. The SmartFrog 2 language and compiler are not available at the time of writing. Parts of the anticipated functionality of SF2 has been expressed using the SmartFrog 1 language and LCFG.

## Chapter 3

# Demonstrator design

### 3.1 High level design

The high-level, component breakdown of the design is illustrated in Figure 3.1. The components are:

- *The Front End Printing Application* This is a Java application allowing print jobs to be sent to GPrint. It has a Swing-based GUI interface that corresponds to the GUI prototype CLIENT\_CONSOLE presented in Figure 2.3.
- *GPrint OGSA Interface* This is an OGSA-compliant interface to the printing subsystem. It acts as a portal onto the Printing Framework (that is, an adaptor receiving the OGSA calls and forwarding them to the framework).
- *SmartFrog/LCFG Printing Framework* This is a printing environment with strong analogues to the existing environment in the University of Edinburgh, School of Informatics (part of the DICE system [dic]). It has been enhanced using SmartFrog to provide examples of autonomic configuration, in response to either print server or printer failure.

Most of the complexity in the design lies in the SmartFrog/LCFG Printing Framework component. The other two components are relatively simple and are not discussed further in this report, except that the installation and configuration of the Globus Toolkit is described in Section 3.3.

### 3.2 Printing framework design

#### 3.2.1 The interface between Globus and the framework

The interface between the Globus Toolkit and the Printing Framework is devised so as to present a subset of the command-line interfaces available to the end-user in LPRng. These are `lpr`, `lpq`, `lprm`, and `lpstat`.

It consists of four components:

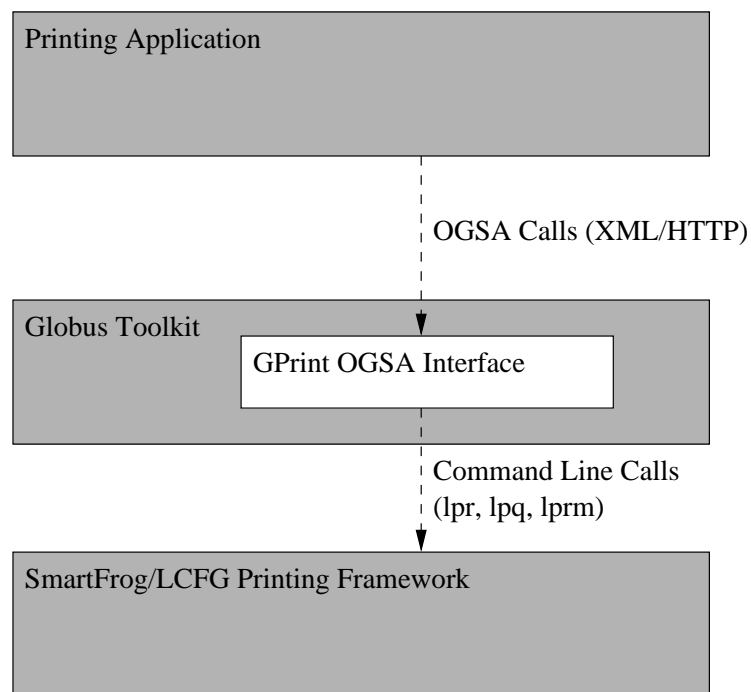


Figure 3.1: The high-level, component breakdown of the demonstrator.

- a print job submission function;
- a print queue query function;
- a print job cancel function;
- a function which returns the set of all available print queues.

The API for these functions is as follows:

---

```
String printJob(fileToPrint, userName, colour  
[, printQueue]);
```

**Description:** Submits a file for printing.

**Parameters:**

<code>fileToPrint</code>	The data to print (passed over SOAP).
<code>userName</code>	A string identifying the user.
<code>colour</code>	A boolean indication whether or not this is a colour print job.
<code>printQueue</code>	An optional print queue to send the job to. If this is not specified the printing framework will select a queue based on username and the colour flag.

**Returns:** A string containing the print job ID.

---

```
String queryJob(userName, queueName);
```

**Description:** Queries the status of previously submitted jobs.

**Parameters:**

userName	A string identifying the user.
queueName	The print queue to query.

**Returns:** A string summarising the user's jobs on this queue.

---

```
void cancelJob(userName, jobId);
```

**Description:** Cancels a previously submitted job.

**Parameters:**

userName	A string identifying the user.
jobId	Identifier for the job to be cancelled.

**Returns:** Void.

---

```
List getAvailableQueues(userName);
```

**Description:** Queries for the list of available print queues.

**Parameters:**

userName	A string identifying the user.
----------	--------------------------------

**Returns:** A list of the available print queues and a summary of their capabilities (support, colour, etc.).

---

In addition to the functionality highlighted above, the framework implements the following non-functional requirements:

- Automatic discovery and monitoring of candidate print servers.
  - Automatic discovery and monitoring of candidate printers.
  - Automatic allocation of print queues to printers (in order to fulfill a simple service level agreement between the queue and the printing capabilities).
  - Automatic re-assignment of a print queue to a different print server in the event that the original server fails.
  - Automatic load balancing of print queues between print servers.
-

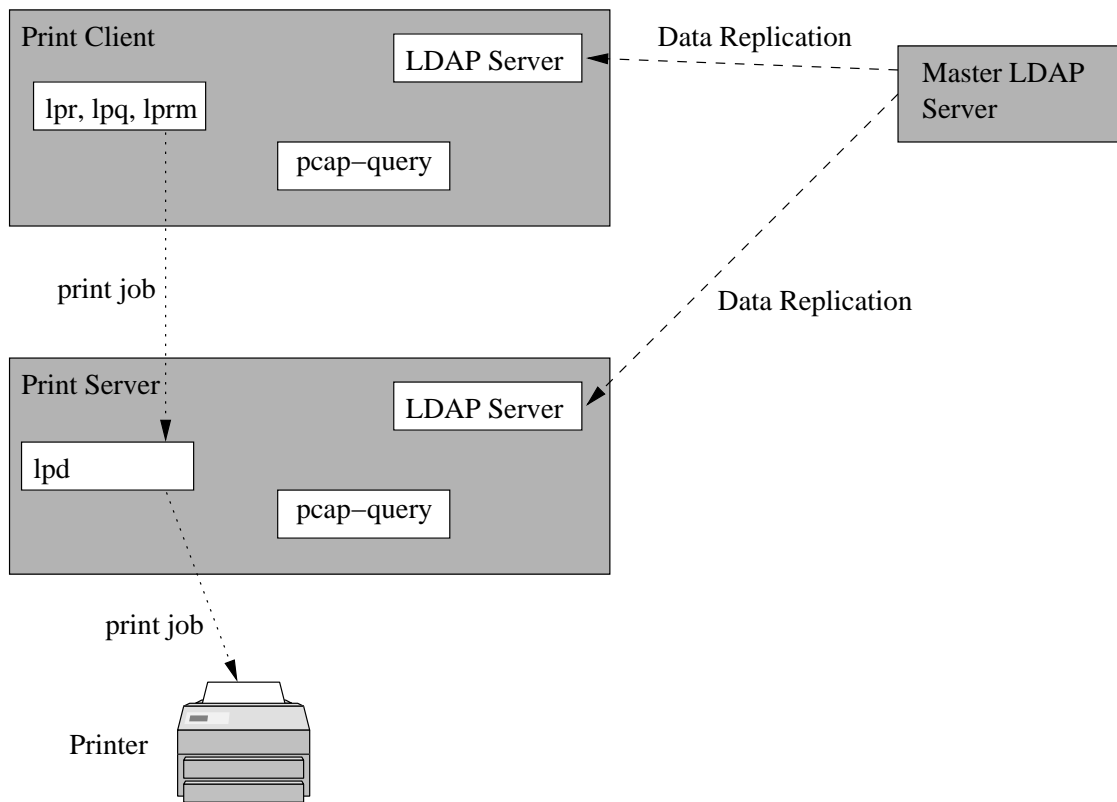


Figure 3.2: The components in the LCFG/LPRng Printing Framework

### 3.2.2 Description of existing LCFG/LPRng Printing Framework

As noted above, the printing framework is based on an existing printing installation (DICE [dic]) which is, in turn, based on the LPRng printing software. Figure 3.2 shows the relevant components of the DICE/LPRng framework.

The following sequence details the steps executed when a file is printed using the LCFG/LPRng framework and explains the role of the various components in Figure 3.2:

1. The print job is initiated by running the `lpr` command. This command takes a file (typically in Postscript format) and the name of the target print queue.
2. `lpr` executes the `pcap-query` script to obtain the details of the specified print queue: this data is referred to within LPRng as the *printcap* information. Primarily, it contains the host name of the print server servicing the queue plus other optional information regarding functionality and authorisation restrictions associated to the queue (see [lpr] for more details).

LPRng can obtain the *printcap* data from many different sources. In the LCFG environment, LPRng is configured to call the `pcap-query` script for this data. This is a Perl script developed as part of LCFG.

3. The `pcap-query` script queries an LDAP server running on the local machine. It extracts the *printcap* information for the specified queue and formats it according to

the LPRng conventions (the local LDAP server is a read-only replicate of the data that is maintained in a master LDAP server). This configuration information is not maintained within the LCFG source files.

4. Having obtained the printcap information, `lpr` sends the print job to the appropriate print server.
5. The `lpd` daemon<sup>1</sup> checks a local permissions file, `lpd.perms` and confirms that the job is authorised for printing on this queue.
6. The LCFG `lprng` component generates the print permissions file from information stored in the LDAP database. LCFG configures LPRng to use Kerberos for authentication of the user.
7. The `lpd` daemon makes a second call to `pcap-query` in order to obtain the information it needs to handle the print job. This again leads to a query on the LCFG LDAP database where the information is stored<sup>2</sup>.

The printcap data returned to `lpd` might instruct the daemon to forward the job to a different print queue (possibly on a different server). In this case, the procedure is repeated for the new queue. By this mechanism, the print job will eventually reach the print server/queue that is actually going to print the job.

8. At this stage, `lpd` passes the print job to an appropriate print filter. This filter is responsible for converting the print job into the appropriate format for the printer (e.g., Postscript or PCL).
9. Finally, once filtered, `lpd` sends the job to the printer<sup>3</sup>.

### 3.2.3 GPrint Printing Framework

In order to meet the requirements of GPrint, parts of the LCFG/LPRng framework have been modified. The main changes were introduced to support the automatic fail-over when either printers or print servers fail: this capability is not available in the LCFG implementation.

The assignment of printers and print servers to print queues is notified to LPRng through the printcap data. The design must therefore allow this data to be dynamically modified. To achieve this, GPrint does not use LDAP to store printcap information. Instead, a SmartFrog component is responsible for generating the printcap data on demand. This mechanism is incorporated into LCFG by substituting an alternative implementation of the `pcap-query` script: the modified script will make an Java Remote Method Invocation (RMI) call to the SmartFrog framework instead of an LDAP query.

---

<sup>1</sup>The application `lpd` is the print daemon that runs on every LPRng print server. It receives and processes the print job from `lpr`.

<sup>2</sup>Within LPRng, printcap entries for a single print queue can contain configuration information for both client and server applications. Applications such as `lpr` and `lpd` receive information relevant to their role within the process, thereby allowing a single master printcap database to be used for both clients and servers.

<sup>3</sup>In practise this is done via an intermediate print spool to allow concurrent processing of multiple jobs.

One other key change to the printing framework is the removal of the Kerberos authentication mechanism. This has been done to simplify the framework and reduce the work required to configure it. The LPRng printcap data controls the type of authentication to use. For the demonstrator the ‘no authentication’ option has been prescribed, instead of the ‘Kerberos5’ option used by LCFG. Simple user permissions are demonstrated by recording per user printing restrictions directly in an `lpd.perms` file: this is a valid method for specifying printing permissions within LPRng. Permissions within the file are validated against a username passed as a string through the OGSA interface. The validation will be implemented by LPRng: this can be achieved by using the `-U username` switch on LPRng commands<sup>4</sup>.

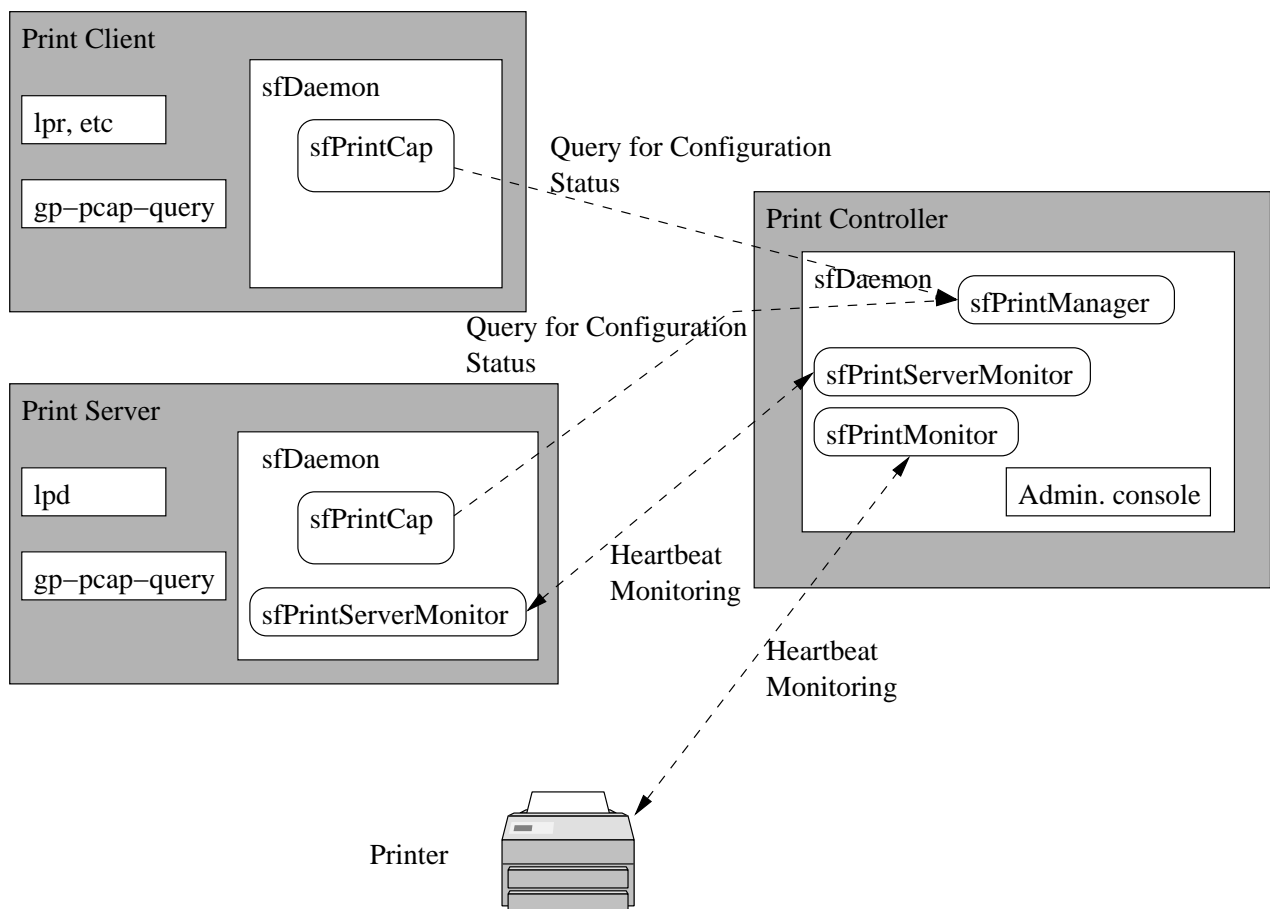


Figure 3.3: The components in the GPrint Printing Framework. A Box with round corners represents a SmartFrog component.

Figure 3.3 shows the key elements of the GPrint printing framework. Each grey box represents a component compound which runs on a node (multiple compounds may run on the same node). The Print Client and Print Server compounds handle requests for printing services: requests from the client interfaces (on the OGSA portal) are processed by the Print Client compound and requests from the `lpd` daemon are handled by the Print Server

<sup>4</sup>The queue query command `lpq` packaged with LPRng does not support user permission checking and cannot take a `-U` option.

compound. Both the Print Client and Print Server compound actually resolve queries by submitting them to the Print Controller compound. The Print Controller compound is centrally responsible for the maintenance and orchestration of the configuration for the printing system.

The remaining items in Figure 3.3 are explained below:

- *Administrative console*: This is a Java application that allows an administrator to configure aspects of the operation of the Print Service Manager. It has a Swing-based GUI interface and this corresponds to the GUI prototype GPRINT\_CONSOLE presented in Figures 2.1 and 2.2.
- *gp-pcap-query*: This is the alternative implementation of the LCFG `pcap-query` script. `gp-pcap-query` is called by the LPRng software to obtain the printcap information. It is a shell script wrapping a command-line Java application which makes an RMI call to the SmartFrog daemon running on the local host. This, in turn, establishes a route to the local instance of the *sfPrintCap* component.
- *sfPrintCap*: This is the SmartFrog component responsible for generating the printcap information. The information is derived from two sources:
  - The static SmartFrog configuration description of the printing framework.
  - The state of the print servers and printers at the time the query is made.

The *sfPrintCap* component maintains a log of the print queue to server mapping implemented by the Print Controller. The *sfPrintCap* component receives one input, a string containing the name of the queue for which printcap information is required. The *sfPrintManager* returns a string containing both the server and client printcap entries for the designated queue: if the queue is not associated with a print server, the *sfPrintManager* returns an empty string. An example of a typical printcap string is:

```
queue1:client:lp=%P@client2:auth=none
queue1:server:lp=rainbow.hpl.lcfg.org%9100: \
    sd=/var/spool/lpd/%P
```

- *sfPrintServerMonitor*: This is a SmartFrog component responsible for monitoring the health of a print server. It allows the *sfPrintManager* component to query whether the print server is running. This component makes use of partition monitor components implementing a group membership protocol.
- *sfPrintMonitor*: This is a SmartFrog component responsible for monitoring the health of a printer. Network ping is the monitoring protocol used to implement this.
- *sfPrintManager*: This is the SmartFrog component with overall responsibility for the printing framework. It holds the master configuration description for the system

and uses the monitor components to evaluate environmental changes and adapt the configuration to component failure events.

Only one instance of the `sfPrintManager` component runs within the demonstrator environment and this is responsible for the orchestration of all aspects of configuration for the LPRng printing system.

Within the SmartFrog description, a static list of print queues is declared to the `sfPrintManager`, along with a corresponding list of network printers. It dynamically establishes connections between print servers and print queues, attempting to: (i) ensure that every print queue is associated to a print server; (ii) maintain a balanced enumeration of queues to servers.

The relationship between print servers and queues is re-evaluated when either of the following events are detected:

- a new print server advertises itself to the Print Controller;
- a print server ceases to advertise itself to the Print Controller.

When a new print server advertises itself to the Print Controller, the Controller will:

- (i) Check if any print queues are currently not allocated to a print server. If so, it will allocate such queues to the newly discovered print server.
- (ii) Take action in order to maintain a load balancing, moving queues from servers which have too many, to servers which have too few.

In this way, the Print Controller implements a simple form of load balancing.

When a print server ceases to advertise itself to the Print Controller, each queue which was previously being managed by the lost server is, in turn, moved to a print server with the smallest number of queues. This is also conducive to load balancing.

Figure 3.3 does not show all the components required to implement the GPrint solution. The following components are also needed:

- *Print and Status Logs*: These record information which can then be used to explain the operation of the system. Logging to a file is done using the Java 1.4 Logging Framework. The log files can be examined on the fly by tailing the files, using the Unix utility `tail`.
- *sfGlobusManager*: This is a SmartFrog component responsible for managing the GT3 toolkit. It provides functionality to deploy OGSA services within GT3 (in particular the GPrint OGSA interface).
- *GPrint Client*: This is a simple GUI application that is used to submit print jobs to the GPrint OGSA interface.
- *GPrint OGSA Interface*: This is the OGSA interface to GPrint.

- *gp-lprng*: This is a modified version of the existing LCFG `lprng` component. The LCFG implementation performs an LDAP query to build the printing permissions file. This has been modified to furnish the information directly into the SmartFrog environment.
- *sfLpd*: This is a SmartFrog component responsible for controlling the `lpd` daemon. It will use the SF->LCFG adaptor (see [ABK<sup>+</sup>03a]) to leverage the modified LCFG `lprng` component described in the previous item.

### 3.2.4 Configuration information

The configuration of the printing system, GPrint, is composed from a combination of static and dynamic configuration information. The static information prescribes the initial resource set for the service and provides information to the Print Service Manager which will not be discovered at runtime. The Print Service Manager takes this static data and composes it with dynamically discovered information. Then, based on prescribed service-level constraints, a complete configuration of the service is formulated and deployed in SmartFrog.

A declarative description of the configuration of LPRng is held in SmartFrog and is composed from the following SF components:

**GPrint Service Manager:** The GPrint Service Manager holds the following information:

- `Locator` – component which listens to advertisements from print servers and queues;
- `psMappings` – the mapping of queues to print servers (populated dynamically);
- `staticQueues` – a list of the initially prescribed queues;
- `queuesMapping` – the mapping of printers to queues (populated dynamically);
- `queueDescription` – a template queue description suitable for prototyping additional queues in the event that the static queue resource is exhausted.

The relationship between queues, printers, and print services is resolved dynamically based on the service level constraints encoded into the demonstrator.

**Group Membership Advertiser:** A component which allows a SmartFrog daemon to monitor the health of objects within the framework. Within GPrint, the advertiser is used to listen for instances of queues, print servers, and printers.

**Print Queue:** This component holds the following attributes:

- `queueName` – the name of the print queue, e.g. *colourps*;
- `advertiser` – the Anubis advertiser used to publicise the existence of the queue within SmartFrog;

- `printServerHost` – the node from which the queue is being served (populated dynamically);
- `servedPrinter` – the printer to which the queue is connected (populated dynamically);
- `capabilities` – a list of the printing capabilities associated to the queue (derived from `servedPrinter`).

In this demonstrator, we make the simplifying assumption that there is a one-to-one correspondence between a print queue and a printer (except in the non-functional state, when a queue is not serving a printer).

A static list of initial print queues is provided to the demonstrator (via the GPrint Service Manager component). However, additional queues may appear during the demonstrator (in the event that more than two printers are present in the system).

**Print Server:** The Print Server component is simply an extension of a node component. Any node within the cluster can become a print server simply by starting the Print Server software: its presence is automatically recognised by the Print Service Manager.

**Printer:** The printer component is also an extension of a node component with an additional description called `capabilities`. This description has two attributes, `colour` and `job_size`, which describe the functionality of the printer. An example of a printer component is given by *rainbow*:

```
rainbow extends Printer {
  ipAddress "10.0.0.101";
  capabilities:colour true;
  capabilities:job_size "large";
}
```

The attributes of each printer must be specified in advance, since such information has not currently been implemented to be discovered dynamically within the testbed.

The SF language description is based on Modelling Example 1, presented in [ABK<sup>+</sup>03a].

### 3.3 Globus Toolkit installation

The specific release of the Globus Toolkit used within the demonstrator requires a number of additional software components. These are:

- the Sun Java Runtime Environment (JRE) Version 1.4.1\_01-b01;
- Apache Ant [[ant](#)] Version 1.5.1;

- Jakarta ORO [oro] Version 2.0.7.

These software packages<sup>5</sup> are installed using the Red Hat Package Manager via the standard LCFG distribution mechanism (see [ABK<sup>+</sup>03a]). The LCFG repository already provides RPMs for the JRE and Ant. For this demonstrator, Jakarta ORO has been packaged as an RPM and added to the base LCFG RPM repository.

The toolkit functions as a J2EE web application within Jakarta Tomcat Version 4.1.24. Since neither LCFG nor SmartFrog provide a configuration management component for Tomcat, an LCFG component has been developed to provide this functionality. The component developed includes:

- an RPM to install Tomcat.
- AN LCFG management component to:
  - start and stop Tomcat;
  - configure the minimum and maximum heap size for the Tomcat process;
  - configure the MIME types used by Tomcat.
- A SmartFrog component that wraps the LCFG Tomcat component using the SF1->LCFG adapter [ABK<sup>+</sup>03a].

The Globus Toolkit has also been wrapped as an RPM for distribution. The RPM implements the basic installation procedure that is currently provided by the Globus installation scripts. It also deploys the toolkit as a web application within Tomcat. A SmartFrog component has been developed to manage the toolkit. The component provides the following functionality:

- It deploys an OGSA Service within GT3 (the alpha release of GT3 does not provide any easy way to undeploy an OGSA service. Therefore, this functionality is not provided in the demonstrator).
- It redeploys an OGSA Service within GT3 — this provides a method to upgrade an already installed OGSA service.

The OGSA deployment mechanism provided by GT3 requires:

- the OGSA service to be packaged as a GAR file;
- this GAR file to be available locally on the machine(s) running GT3;
- an Ant build-target to be run on each GT3 node, unpacking the GAR file and copying the contents into the GT3 directory structure.

---

<sup>5</sup>The versions numbers quoted have been selected for compatibility with GT3 and, where possible, are currently available within the LCFG configuration system RPM repository.

The SmartFrog GT3 management component fulfills each of these requirements. The service to be deployed is passed to the component as the URL of the corresponding GAR file. This allows such GAR files to be published on a web server accessible from the cluster (for example, the web server which hosts the LCFG RPM repository).

**Notes:** The current implementation excludes the installation and configuration of standard grid services like the Job Control Manager. It also excludes the installation and configuration of the GT3 security framework.

## Chapter 4

# Experiences from the Demonstrator

In this report, we have documented how we used the combined LCFG/SmartFrog framework to describe, configure, and manage a complex, adaptive demonstrator system, which transforms the fabric from the bare-metal state to a fully functioning, Grid-enabled printing service. The demonstrator illustrates the following achievements of the project:

***Representing multi-level system configuration:*** We have described the configuration of a printing system, including (at a high level) abstract concepts such as the server-to-print queue-to-printer relationship and (at a low level) real entities, such as configuration of a node. The model presents a view of the system that contains only the elements of configuration which are of interest to the printing administrator, hiding the complexity attributed to other aspects of the fabric.

As an example of this, consider the complexity normally associated with the definition of a user (for example, disk quota, choice of shell, e-mail aliases, etc.). The majority of these elements are of little consequence to the printing administrator and thus are not present in the extract of the user description which is presented to them. They see only the attributes of a user (e.g. name/group and printing permissions) that have a bearing on the printing model. In a complete configuration solution, these attributes would be augmented to a richer description of a user during the deployment phase.

***Deploying system configuration:*** The combined LCFG/SmartFrog prototype is a first attempt to realise complex descriptions in a real fabric. The complementary nature of SmartFrog and LCFG means that it has been convenient, in this prototype, to use LCFG to perform low-level configuration actions (especially the basic configuration of nodes) and SmartFrog to configure pan-nodal contributions to the infrastructure. However, the demonstrator shows that we can control SmartFrog elements from within LCFG and vice-versa using adaptor technologies, thus allowing full flexibility in the decision as to how configuration actions are applied.

From an engineering perspective, a combined system based on LCFG and SmartFrog may not be an ideal solution. However, the separation of responsibility for different stages of the configuration process between the two systems – that is, LCFG deploying a static description for a basic cluster configuration, on top of

which SmartFrog establishes and maintains specific pan-nodal services – could be re-used in a production configuration system. This separation of roles is discussed further in [AT03].

**Weaving a description:** A separation exists between the definition of the component elements and the description of the fabric to which they contribute. As part of the demonstrator, we have created prototype descriptions for printers, queues, servers, and so on. These descriptions have been motivated by the requirements and functionality of the LPRng software. Based on these components, we have used the SmartFrog language to weave a reactive, event-driven fabric, which responds to demands for load balancing and reliability in an automatic manner.

In another context, given different demonstrator requirements, it would be equally easy to weave a completely different model but still using the same basic component elements.

**Configuring external agents:** The demonstrator illustrates how an external agent, such as a printer or a third-party software application, may be assimilated into the configuration environment, using a proxy component. This component encapsulates product-specific configuration attributes and caveats, presenting a simple and consistent interface to the administrator. The use of a proxy allows new technologies to be quickly and easily included in the fabric. For example, printer hardware encapsulates a diverse family of technologies. To manage the configuration of a printer as part of the demonstrator, only a small subset of the configuration parameter space needs to be represented (as determined by the functional requirements of the demonstrator and the LPRng software). Once a prototype description of a printer has been determined, it is a simple matter to instantiate descriptions of specific printers at runtime, based on core, static data enriched with dynamically discovered information.

**Collating information:** The demonstrator illustrates the dynamic collation of information from multiple descriptions. Such a construction is similar to the *spanning map* mechanism of LCFG. For example, the Print Service Manager (PSM) subscribes to the list of candidate print servers, monitoring which nodes are actually in use as servers at any particular instance.

Unlike LCFG, contributors to the spanning map do not need to explicitly publish their information – this is done by a listener component. Hence individual description authors need not be concerned with such issues.

There are a number of additional functionalities that could be developed as extensions to this demonstrator.

- One could implement dynamic constraints satisfaction, devising a framework that automatically generates a system description which satisfies a set of constraints. The framework propagates the necessary updates and changes in order to realise this configuration on the running fabric. Subsequent changes to these constraints would be reflected by corresponding changes to the configuration, again resolved automatically.

- 
- The system could be made more reactive by associating the deployment of print servers to the level of demand (that is, number and size of print jobs) in the system<sup>1</sup>.
  - In the current environment, the PSM represents a single point of failure from which the system cannot recover. However, as the PSM is itself enclosed in a description, one might use peer election to eliminate this weak point from the demonstrator. Simply put, each node in the fabric would have access to the description of the PSM and would be designated as a potential host for the component. In the case of a failure, the node pool (*peer*) would elect a replacement (*leader*) to deploy a new PSM. Since other key components within the description (for example, printers, print queues, and print servers) are established dynamically, state is never lost.
  - One could devise a more invasive example of composition from two very different aspects, such as the network infrastructure and the printing service. Within the current demonstrator, such a composition is not explicitly completed, since the network infrastructure is maintained by LCFG, while the printing structure is implemented by SmartFrog.
  - One might extend the functionality of the OGSA portal, allowing an administrator to submit a configuration description of the actual printing service through the Grid interface. Such a submission would trigger the necessary configuration changes on the fabric in order to deploy the prescribed configuration. This could be further enhanced, by allowing a user to submit a configuration description that affects a designated subset of the configuration of the system, in a secure manner. The configuration description submitted would be parsed and authenticated, before being augmented into the complete service description to be deployed. In this way, a fabric would constantly be reconfiguring itself, tailoring itself in response to individual job submissions.

---

<sup>1</sup>Print servers are currently started interactively by an administrator, using a GPRINT\_CONSOLE running on one of the nodes.



# Bibliography

- [ABK<sup>+</sup>02] Paul Anderson, George Beckett, Kostas Kavoussanakis, Guillaume Meche-  
neau, and Peter Toft. Technologies for large-scale configuration manage-  
ment. Technical report, The GridWeaver Project, December 2002.  
<http://www.gridweaver.org/WP1/report1.pdf>.
- [ABK<sup>+</sup>03a] Paul Anderson, George Beckett, Kostas Kavoussanakis, Guillaume Meche-  
neau, Jim Paterson, and Peter Toft. Large-scale system configuration with  
LCFG and SmartFrog. Technical report, The GridWeaver Project, July  
2003.  
<http://www.gridweaver.org/WP3/report3.1.pdf>.
- [ABK<sup>+</sup>03b] Paul Anderson, George Beckett, Kostas Kavoussanakis, Guillaume Meche-  
neau, Peter Toft, and Jim Paterson. Experiences and challenges of large-  
scale system configuration. Technical report, The GridWeaver Project,  
March 2003.  
<http://www.gridweaver.org/WP2/report2.pdf>.
- [ant] The Apache Ant Project. Web page.  
<http://ant.apache.org/>.
- [AT03] Paul Anderson and Peter Toft. The gridweaver project: Summary and con-  
clusions. Technical report, The GridWeaver Project, July 2003.  
<http://www.gridweaver.org/WP4/report4.2.pdf>.
- [dic] Distributed Informatics Computing Environment. Web page.  
<http://www.dice.informatics.ed.ac.uk/>.
- [Edi] Edinburgh School of Informatics, EPCC and HP Labs. The GridWeaver  
Project. Web page.  
<http://www.gridweaver.org>.
- [lpr] LPRng Printing Software. Web page.  
<http://www.lprng.org/>.
- [oro] Jakarta ORO regular expression package. Web page.  
<http://jakarta.apache.org/oro/>.
- [UoEEL] School of Informatics University of Edinburgh, EPCC, and HP Labs. Grid-  
weaver documents. Web page.  
<http://www.gridweaver.org/docs/>.