

# Writing LCFG Components

Paul Anderson  
Division of Informatics  
University of Edinburgh  
<paul@dcs.ed.ac.uk>

# Summary

- Overview
  - What is an LCFG "component"
  - The component API
  - Methods
  - Libraries
- The Generic Component
  - Writing methods
  - The configure method
  - The template processor
  - Start and stop methods
  - Input / Output
  - Using dice-buildtools
- The Future

# What is an LCFG Component?

- A component is a script that reads configuration parameters from the LCFG "profile" and uses them to configure some subsystem
- Components are often associated with daemons, because they must be restarted when the configuration changes.
- Current components are mostly written in shell script, but there is now native Perl support as well.
- Simple example templates are available:
  - `lcfg-example` (shell)
  - `lcfg-perlex` (Perl)

# The Component API

- Components interface to the LCFG framework in two directions:
- The component is called by other processes which specify the required action (or "method") as an argument:
  - `mailng configure`
- The component makes library calls on the LCFG framework to retrieve configuration parameters:
  - `qxprof mailng.relay`
- A major aim of DICE stage1 is to ensure that components conform to a standard API, so that the framework itself can be changed without future changes to the components

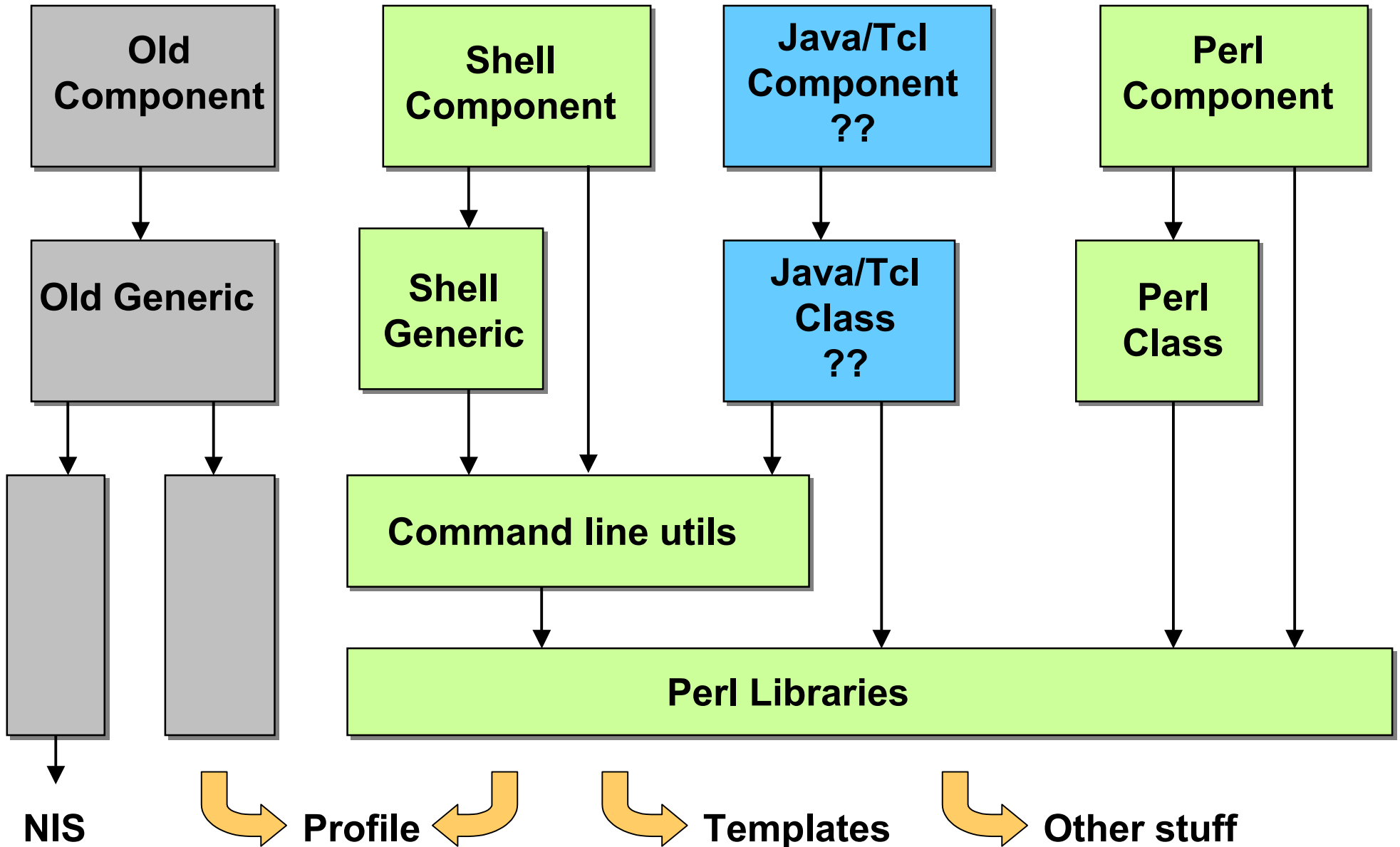
# Methods

- The set of standard methods are defined, together with the required behaviour. These include:
  - `configure`
  - `stop / start / restart`
  - `suspend / resume`
  - `logrotate`
  - `run`
  - `monitor`
- The specification of the behaviour includes aspects such as locking.
- Ad-hoc methods are supported, but discouraged

# Libraries

- Old components use a very simple program to retrieve resource values from a DBM file.
- A new set of libraries provides a more powerful interface based on a Perl library, with simple wrapper programs for use in shell scripts:
  - `LCFG::Resources` is the library
  - `qxprof` is a shell command based on the library
- Components using the new interface can be isolated from future changes to the framework

# Component Framework



# Generic Components

- It is possible to write components from scratch, using the library, and the specification for the method behaviour
- Normally, a "generic" component is used; this implements all the default behaviour, so that code is only needed for any component-specific behaviour
- The shell generic component is implemented as a file of shell functions:
  - `. /etc/obj/ngeneric`
- The Perl generic component is a Perl object which can be subclassed:
  - `Use LCFG::Component;`

# Minimal Components

- A minimal Shell component:

```
#!/bin/shell  
. /etc/obj/ngeneric  
Dispatch "$@"
```

- A minimal Perl component:

```
#!/usr/bin/perl  
package LCFG::Perlex;  
@ISA = qw(LCFG::Component);  
use LCFG::Component;  
new LCFG::Perlex() -> Dispatch();
```

# Writing Methods

- When a method FOO is called, then generic component calls the function Method\_FOO()
- Method\_FOO() performs some "generic" operations before calling FOO(). Eg:
  - Log the call
  - Lock method calls to the component
  - Process "generic" options
  - Load resources into environment
  - (or pass them as a parameter in Perl)
- The default FOO() function is normally NULL and can be redefined by the component to perform component-specific operations
- Method\_FOO() can also be redefined, but this is strongly discouraged

# The configure Method

- Old LCFG components only recognise configuration changes when they start at boot time. This is not really acceptable when machines can stay up for a long time.
- Configurations also change at other times, for example when a portable switches network
- The configure method is now the most important method. It is called whenever the LCFG configuration changes (as well as when the component starts)
- This method should use the configuration parameters to reconfigure the corresponding subsystem

# Scheduling Configuration Changes

- If there is some daemon running, then the configure method should also do whatever is necessary to make the daemon recognise the new configuration.
- It may not be appropriate to make drastic changes immediately, and scheduling these is an ongoing research area
- At present, the decision about whether to make the change immediately is up to the individual component. Eg:
  - Kdm defers changes until the user logs out

# An example configure Method

- The generic component provides a function to determine if the component is started
- If so, the component needs to signal the daemon when the configuration has changed

```
Configure() {  
    echo $LCFG_foo_key >config.new  
    If ! cmp -s config.new config ; then  
        mv config.new config  
        IsStarted || return  
        kill -HUP `cat pidfile`  
    fi  
}
```

# The Template Processor

- The template processor (sxprof) greatly simplifies the process of creating configuration files from LCFG resources
- It is a macro processor which uses LCFG resource variables directly
- It supports substitution, conditionals and loops over lists
- It returns an exit status indicating whether the configuration file has changed:

```
sxprof TEMPLATE FILE
```

```
if [ $? = 2 ] ; then configuration changed
```

# An Example Template

```
Server <%server%> {
```

```
<%for: m=<%clients%>%>
```

```
  <%client_<%m%>%>
```

```
<%end:%>
```

```
}
```

```
<%if: <%port%>%>PORT=<%port%><%end:%>
```

- Notice that new resources can be added without changing any code

# Command Line Arguments

- The template processor can even be used for handling changes to daemon command line parameters:

```
sxprof -- - foo.args <<EOF
<%if: <%optiona%>%>-a<%end:%>
<%if: <%optionb%>%>-b<%end:%>
EOF
If [ $? = 2 ] ; then
    daemon `cat foo.args`
fi
```

# Start / Stop Methods

- Start and stop method are called when the system is booting or shutting down. These are primarily for starting and stopping daemons.
- The generic code calls "configure" before starting
- The start method is responsible for storing the PID of any daemon so that it can be located by the stop method
- The old mechanism of storing private information in the resource status file is no longer allowed, but `qxprof` can be used to store temporary status information in a separate file

# Input / Output

- Components should not generate output unless the behaviour of the method explicitly defines it (for example, status or log)
- If a component needs to generate (short!) informational messages, it should use the supplied functions:
  - `Info()`, `Debug()`, `Warn()`, ....
- Output from all commands should normally be routed to the logfile

# Using dice-buildtools

- There are some important rules on the type and location of files shipped with an LCFG component RPM
- See the shell and Perl example components for using dice-buildtools
- Dice-buildtools and the generic components support testing of component code without installing as root

# Miscellaneous

- Logrotate is now used to manage the logfiles and the logrotate method is called to notify a daemon when the file has been rotated
- The generic components also provide a number of miscellaneous functions for things like debugging support
- The generic component parses some generic options to enable flags such as debug and lock timeouts

# The Future

- Framework-level control of configuration change scheduling (“intrusion levels”)
- Richer resource structures generated from a new high-level language, without using the “tag” convention