Sharing Programs and Data in the DICE Environment

by Paul Anderson <dcspaul@inf.ed.ac.uk>

School of Informatics
University of Edinburgh

1 Introduction

The DICE environment provides a standard set of software which is managed in a sustainable way, and distributed to all DICE workstations. However, groups, or individuals, may have requirements to share other software, or data, which it is not appropriate to include in the core DICE distribution. This note outlines the different approaches available, and explains the advantages and disadvantages – this represents general system administration experience, and is not specific to DICE in any way.

2 Local vs Remote

In the early days of local networks, most data and application code was stored on central servers, and mounted directly by the clients using a remote filesystem. This was mainly due to the cost of disk space. Now that disk space is much cheaper, there is a tendency to distribute application code (at least) to the local workstation disks, rather than mounting remote filesystems. The technology for this is well established and it has the following advantages:

- ❑ It avoids the possible performance bottlenecks, single-point-of-failure, and problems such as stale filehandles, associated with a central server (if the server fails, the clients will continue to operate).
- ❑ The data is available, when the remote filesystem is not - for example, to disconnected laptops (on that long flight), and (currently) to self-managed DICE-machines.

This may be less appropriate for non-code data, especially for large datasets, or where the data changes frequently, in which case a remote filesystem may be necessary. Other solutions may also be appropriate, depending on how the data is used – for example, CVS is a good choice for collaborative development of source code or documents – keeping a local "checked out" copy of relevant modules avoids the above problems except at commit time, and provides the additional benefits of version control.

3 Managed Repositories vs Ad-hoc Filesystems

Again, in the early days of local networks, shared software was simply installed directly into a common directory. However, over time, it has become clear that this is not a sustainable solution – software directories fill with "rotting" files, and quickly become unmanageable, especially when multiple people are involved in their maintenance. We have seen many examples of this, including huge repositories where it is impossible to tell exactly what is installed, or what any individual file is for, and impossible to remove or upgrade any packages with any degree of confidence.

Initially , most sites developed their own ad-hoc solutions to this problem (including us – see [And91]). However, this is now largely addressed by standard package management software, of which RPM (see [Bai97]) is a good example (this is the standard for Redhat Linux, as used by DICE). The package manage system provides:

- ❑ The ability to know which packages are installed at any time, and to know which files belong to which packages.
- ❑ The ability to add or remove new packages at any time.
- ❑ Tracking of dependencies, so that packages are not accidentally removed when some other package depends on them.
- ❑ A convenient way of specifying and sharing packages between installations.
- ❑ And other benefits ...

Package management tools are most appropriate for distributing software packages, or small, read-only datasets. Large or frequently changing datasets will probably require different treatment, and CVS may be useful in some cases.

Repository management is a particularly insidious problem, because the simple directory solution has a very low startup cost (compared with the learning curve required for package management), but the negative consequences may not be apparent for several years, by which time re-organisation becomes a serious difficulty.

If simple software repositories are created without any use of tools, then it essential to have good manual procedures and strong discipline in their management.

4 Access Control

It is important to consider access control to any shared repository.

Access control to shared NFS filesystems will probably be handled by the simple Unix group mechanism - this is very crude, and the limitations on group membership restrict the number of groups to which users can belong.

CVS provides better access control, and it is planned to implement a simple RPM repository with fine-grained access control, as part of the DICE developments mentioned below.

5 DICE Restrictions

Mainly due to security issues with the existing NFS filesystem, there are currently restrictions on the software that can be installed on DICE clients. However:

- We hope that it will soon be possible to install user-created RPM packages on standard DICE clients, providing that the RPMS only install files into a specific part of the filesystem hierarchy (such as `/contrib`).
- The new "DIY DICE" proposals will allow users to have "partially managed" DICE machine which will have some limitations, but will allow software to be installed anywhere on the client (automatically, if the standard RPM distribution mechanism is used).

6 Producing Software for Export

If groups, or individuals, are creating software for export – either for use simply within Informatics, or for wider distribution – we would very much like to encourage consideration of the distribution phase. Poor understanding of modern packaging requirements can make it very difficult for other people to install and use the software and often creates a bad impression of the quality of the software itself.

Modern, industry-standard tools make this process relatively straightforward, and we are happy to offer advice on these. Making software widely available in the external community will also provide valuable feedback.

7 Conclusions

There are several different ways of providing shared software and data. Most of these are available in the DICE environment, but they have different characteristics, and it is worthwhile considering how the software/data will be used, and choosing appropriate technologies. Note that this may involve different processes for different types of data.

By way of example, the source code for local DICE software is stored in a remotely-accessible CVS repository. The build process for these modules automatically creates RPM packages of the program binaries. These binaries are stored on central repository, and the individual clients download the required packages (over HTTP) and install them on the local filesystem. This represents current "best-practise" and is appropriate for large amounts of core software which must be sustained long-term on a production service.

In many cases, such an engineered solution will not be appropriate for more restricted collections of shared data. However, we would strongly encourage consideration of the various options, and adoption of good management practises where possible – this will help to contribute to the overall sustainability and usability of the Informatics computing environment.

References

[And91] Paul Anderson. Managing program binaries in a heterogeneous Unix network. In *Proceedings of the 5th Large Installations Systems Administration (LISA) Conference*, pages 1–9, Berkeley, CA, 1991. Usenix Association.
`http:// ...`
`homepages.inf.ed.ac.uk/ ...`
`dcspaul/publications/ ...`
`LISA5_Paper.pdf`.

[Bai97] Edward C Bailey. *Maximum RPM*. Redhat Software Inc., 1997.
`http://www.rpmdp.org/ ...`
`rpmbook/`.