History, motivations

- Started in 1993 for providing support for horizontal software development at LAL
- After an evaluation of the *autoconf* world, decision was taken to create a *new* package
  - basic requirements were :
    - emphasis on simplicity to use and to understand by non software experts
    - should be based on a real conceptual model
  - study of works
    - SEI : definitions of Configuration Management (IEEE Std-729-1983)
    - Inspiration from the CMM (freely interpreted)
- Promote Human Oriented Software

The configuration management definition

- **IEEE Std-729-1983**

  “Configuration is the process of identifying and defining the items in the system, controlling the change of these items throughout their lifecycle, recording and reporting the status of items and change requests, and verifying the completeness and correctness of items”

- **Identification**

  describes the system structure, the nature of its elements, their identity, and gives access to each item version

- **Control**

  organises versions and changes to system items while keeping coherency and consistency on the complete system.

Principles of the package

- **Based on some scenarios**
  - **Projects or sub-projects promote part of their software base as reusable**
  - **Projects are divided into small groups of closely related developers**
  - **Integration phases are iterative and must remain under the control of the package managers**
  - **Configuration parameters should be defined/specified independently to their implementation**
  - **Configuration parameters should be queried at any time outside of any effective software operation**
  - **Maintain several concurrently visible versions of each package**
  - **...**

Principles of the package

- …and on some definitions and conventions
  - the package
    - » the smallest autonomous entity in the software base
    - » support use relationships to other packages
    - » the physical organisation of packages is independent of their logical structure
  - the configuration parameters
    - » identify and describe the constituents of a package
      - applications, libraries, …
    - » set up the environment needed to develop the package
      - environment variables, make macros, include paths, …
    - » set up the environment needed to operate the package
      - environment variables, aliases, setup scripts, …

Principles of the package

- …and on some definitions and conventions
  - physical organisation

```
<root1>/<package>/<version tag>/src
                                /mgr
                                /<conf. tag1>
                                /<conf. tag2>
<root2> >/<package>/<version tag>/src
                                /mgr
                                /<conf. tag1>
                                /<conf. tag2>
                                /...
```

Principles of the package

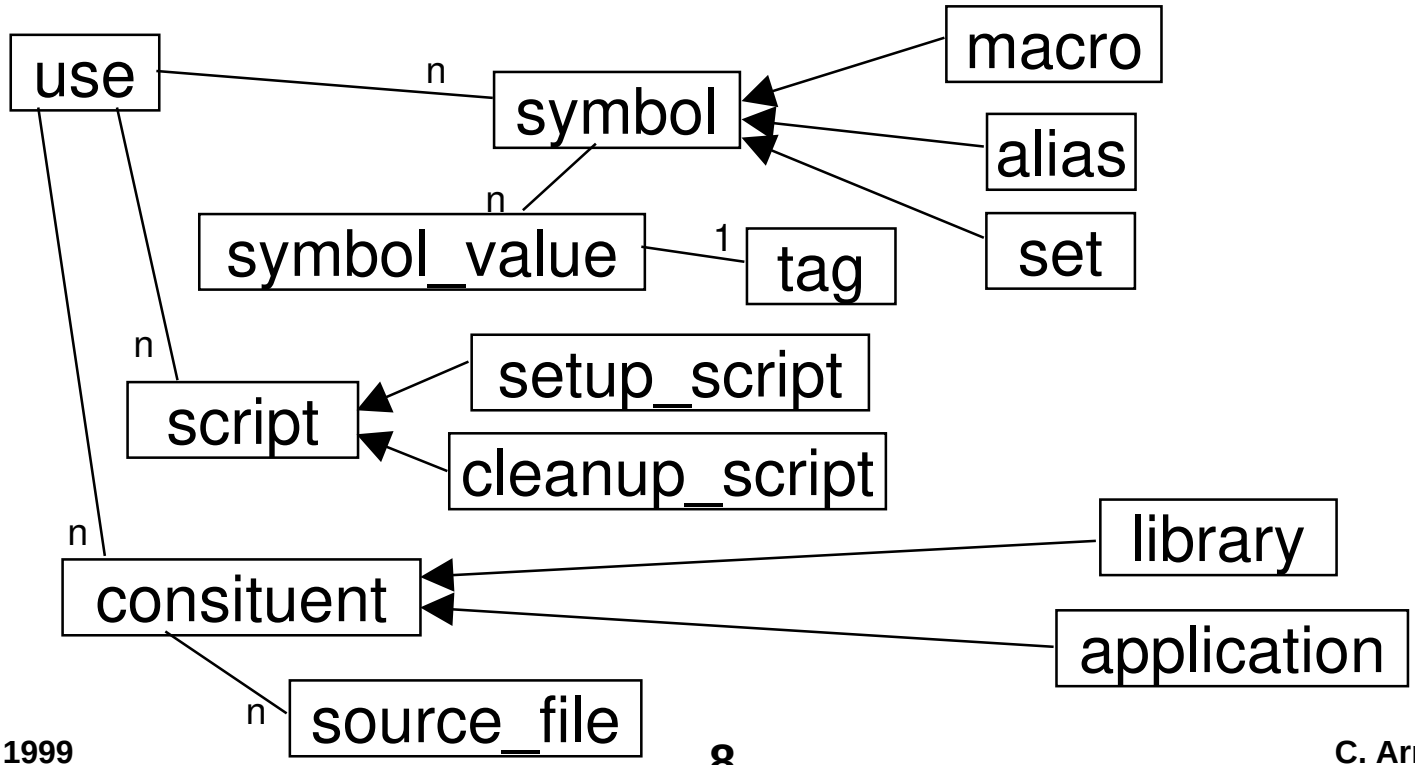
- **...conventions**
  - **configuration tag**
    - » **reflects the machine type, the operating system, the context (*debug*, *gnu*, etc…)**
    - » **may be either automatically from system features (*uname*, *fs sysname*, …) or freely defined**

Conceptual and user requirements

- Packages are autonomous entities
- Packages are related to each other by use relationships
- Package evolution and history is specific to each package
- Configuration parameters are any information required to
  - identify the constituents of the package
  - set up the environment needed to develop and operate the package
  - describe the use relationships between package
- Version tag semantics reflects backward compatibility or incompatibility
- Configuration parameters are propagated through the chain of *use* relationships.

Design

- **Based on the modelling of the configuration management parameters**
  - **one object model (used to build tools)**
  - **one syntax (used to make conf. Parameters persistent)**



Implementation

- Parameters are stored and maintained within a textual file named ***requirements***
- Use a simple (easy to read) and homogeneous syntax
- One basic parser application implements the object model from the textual representation (***cmt_parser.exe***)
- One main user interface (***cmt***) to the parser provides the command oriented (à la CVS) interface used for
  - querying the configuration parameters
  - generate the effective environments
    - for package development
    - for package usage

Some scenarios

- **Creating a package**

```
> cmt config
>  edit requirements
        use A v1
        library mylib mylib1.cxx
        application myapp myapp1.cxx
>  edit source files
> gmake
>  run
```

...Some scenarios

- **Selecting cooperating projects**

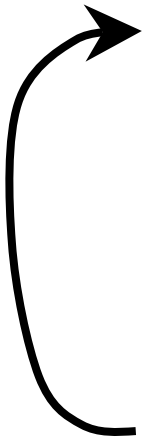
```
> setenv CMTPATH projectA:projectB:projectC
```

or

```
> cat >~/.cmtrc
CMTPATH=projectA:projectB:projectC
```

...Some scenarios

- **Iterative integration**



```
>  edit use statements within the requirements
                    use A v2
> gmake
>   run
```

...Some scenarios

- **Package evolution**

```
> cd dev
> cmt checkout …
>     select new version tag
> cmt config
> gmake
>     change source files
>     test
> cvs commit
> cvs rtag
```

...Some scenarios

- **Building *domain* packages**
  - **This is an *interface* package only containing a set of *use* statements towards a selection of versions for the packages belonging to a given conceptual domain**
    - » **simulation**
    - » **reconstruction**
    - » **visualisation**
  - **A user of the *simulation* domain will simply use one given version of the simulation interface package, which automatically provides by *transitivity* (or inheritance) the appropriate selection of versions of all the simulation-related packages.**
  - **All configuration parameters defined in these packages are therefore inherited through this domain package.**

The services

- **Parameter monitoring**

`cmt show macro xxx`	show a particular macro
`cmt show macros`	show all macros
`cmt show constituents`	show all constituents

- **Environment generation**

`cmt config`	(re-)install a package
`cmt build setup`	compiles the setup script
`cmt checkout`	checkout a package from CVS

The requirements file

- **General syntax**

```
package <name>
set <name>  “default value” [ <tag> “value” ] …
alias <name>  “default value” [ <tag> “value” ] …
macro <name> “default value” [ <tag> “value” ] …
application <name> <source file> …
library <name> <source file> …
tag <name> <tag name> …
include_dirs <path> …
setup_script <name> …
cleanup_script <name> …
make_fragment <name>
```

The requirements file

- Some examples
  - Virgo
    - » Cm, CmTimer, Virgo (domain package)
  - LHCb
    - » task (release package)
  - Atlas
    - » atlas (link to SRT), AMDB

Status

- **Ported to**
  - **all Unix flavours (Dec, AIX, SGI, SunOS, Linux, HP-UX)**
  - **Windows NT / 95 / 98 (interfaced with VisualC++)**
  - **LynxOS (Cetia, CES)**
- **Used by**
  - **the development team at LAL**
  - **the Virgo experiment (`http://www.virgo.infn.it`) (*Lapp Firenze Frascati IPN-Lyon Napoli LAL ESPCI Perugia Pisa Roma*)**
  - **the LHCb experiment**
  - **the NEMO experiment (`http://www.lal.in2p3.fr/NEMO/nemo.html`) (*France, Russia, INEL MHC Finland Ukraine Praha*)**
  - **the AUGER experiment (`http://wwwlpnhep.in2p3.fr/auger/welcome.html`)**