NFS Organisation mit Amd

15. DECUS Symposium Karlsruhe

Frank Kardel

März 92 3/92

NFS Organisation mit Automounter

Frank Kardel
kardel@informatik.uni-erlangen.de

Friedrich-Alexander-Universität
Erlangen-Nürnberg
Martensstraße 1 • D-W8520 Erlangen

Durch den zunehmenden Einsatz von leistungsfähigen Workstations am Arbeitsplatz taucht immer mehr das Problem ihrer Verwaltung auf. Die meisten dieser Workstations werden von Ihren Nutzern betreut. Selten jedoch hat jeder der Nutzer umfassendes Wissen über die administrativen Belange “seiner” Workstation. Dadurch ergibt sich die Notwendigkeit abteilungsweite, zentrale Dienstleistungen bereitzustellen. Dazu gehört neben der Datensicherung auch das Bereitstellen von Softwarepaketen sowie die Verwaltung des verfügbaren Plattenplatzes. Diese Erfahrung wurde auch an der Universität Erlangen-Nürnberg in der Informatik gemacht. Zur Zeit werden mehr als 300 Rechner mit über 60 Softwarepaketen versorgt. Auf diesen Rechnern sind über 1800 Benutzer eingetragen und entsprechend vielfältig sind auch die Anforderungen an die Softwareinstallationen. Einzelne Softwarepakete erreichen eine Größe von mehr als 150Mb. Die ursprünglichen Verwaltungsmethoden (Einzelinstallation und Kopieren) haben sich als nicht mehr adäquat erwiesen, weil die Pakete einerseits von verschiedenen Personen betreut werden und zum anderen auf vielen unterschiedlichen Rechnern zur Verfügung stehen sollen. Der Beitrag erläutert die an der Universität Erlangen-Nürnberg entwickelte Lösung auf der Basis von NFS Dateisystemen und der Verwendung des Automounters (“*amd*”). Besonderes Augenmerk wurde dabei auf die Unterstützung verschiedener Rechnerarchitekturen gelegt.

1. Einführung

Im Fachbereich Informatik an der Universität Erlangen-Nürnberg werden zur Zeit über 300 Rechner von verschiedensten Herstellern betrieben. Diese Rechner werden nicht, wie an anderen Universitäten üblich, durch eine spezielle Rechnerbetriebsgruppe betreut, sondern durch Mitarbeiter der einzelnen Lehrstühle. Diese Organisationsform hatte zur Folge, daß, in Anbetracht der geringen zur Verfügung stehenden Zeit, Mechanismen zur effizienten Verwaltung dieser Rechner entwickelt wurden. Bei einer so großen Zahl eigenständiger Rechner und der mittlerweile großen Anzahl (mehr als 60) von Softwarepaketen, die von den verschiedenen Lehrstühlen eingesetzt werden, war es notwendig ein Schema zu entwickeln, welches erlaubt, mit einem Minimum an Aufwand, die Softwarepakete möglichst vielen Rechnern zugänglich zu machen. Alle Rechner sind über IP (Internet Protocol) miteinander vernetzt, und viele Rechnerhersteller bieten das von Sun entwickelte NFS Protokoll an. Mit Hilfe des NFS Dateisystems können viele Rechner auf Dateibäume eines Serverrechners zugreifen. Der "Nachteil" von NFS ist, daß es konfiguriert werden muß. Es ist ein großer Aufwand, auf über 300 Rechnern die entsprechenden Systemtabellen zu aktualisieren und die NFS Konfiguration anzupassen. Das Problem hat sich mit der Einführung eines zentralen Rechnerpools für die Studenten insofern verschärft, als daß Rechner des Rechnerpools auf die Softwarepakete aller Lehrstühle zugreifen können müssen.

Die Lösung lag in der Verwendung des *amd*-Programms. Bei *amd* handelt es sich um ein Dämonprogramm das "logische" Verzeichnisse realisiert, die Verweise auf den aktuell gültigen Pfad für die gesuchten Pakete liefern. Sollte ein Dateibaum noch nicht in den rechnerlokalen Baum eingebunden sein, so wird dies transparent nachgeholt. Eine weiterer Vorteil des *amd* liegt darin, ständig die Dateiserver zu überwachen. Durch diese Monitorfunktion ist es möglich, nicht erreichbare Server zu erkennen und dementsprechende Fehlermeldungen zu liefern oder alternative Dateiserver zu nutzen. Für die Verwendung des *amd* im Gegensatz zu herstellerspezifischen Produkten wie z. B. der *automount* von Sun Microsystems, gab es im wesentlichen zwei Gründe. Zum einen erlaubt der *amd* eine wesentlich flexiblere Konfiguration und zum anderen ist die Quellversion des Programms verfügbar, womit ein Einsatz auf vielen Rechnern unterschiedlicher Architektur möglich ist (Laut Beschreibung unterstützt der *amd* mindestens 18 Maschinenarchitekturen). Der *amd* wird mittlerweile seit 1 1/2 Jahren mit Erfolg innerhalb der Informatik eingesetzt.

Zusätzlich zu den Softwarepaketen werden die Basisverzeichnisse ("homedi-

rectory"), sowie Projekt- und Quellverzeichnisse über den *amd* verwaltet. Dies erlaubt eine hohe Flexibilität in der Speicherung der entsprechenden Dateibäume auf den Dateiservern. Weil alle Teilbäume nur noch über logische Namen referenziert werden, ist es kein Problem, die Teilbäume zwischen den Dateiservern zu bewegen, vorausgesetzt, es wird während der Verlagerungsphase nicht auf dem entsprechenden Dateibaum gearbeitet.

Die folgenden Kapitel beschreiben den Aufbau von architekturunabhängigen Softwarepaketen und die Hilfsprogramme, die dafür entwickelt wurden.

2. Softwarepakete

Softwarepakete haben eine weitgehend einheitliche Struktur, die sich an einem Unix Dateisystem orientiert. Jedes Paket enthält folgende Inhaltsverzeichnisse:

- bin
  Ausführbare Dateien
- lib
  Bibliotheken, Hilfstexte
- etc
  administrative Kommandos, Konfigurationsdaten
- man
  Dokumentation

Alle Pakete werden mit Hilfe des *amd* über den Pfad **/local/***Paket***/...** referenziert. Der *amd* findet den entsprechenden Dateiserver und übernimmt die automatische Einbindung in das lokale Dateisystem (*NFS mount*). Danach liefert er den entsprechenden Verweis zu dem gesuchten Paket.
Weil alle Pakete gleichartig strukturiert sind, kann jeder Benutzer sie leicht in seine Arbeitsumgebung einbinden (Der Suchpfad muß um **/local/***Paket***/bin** erweitert werden, der Manualpfad um **/local/***Paket***/man**).

2.1. Architekturunabhängigkeit

Die obige Struktur ist in ihrer einfachen Form noch nicht architekturunabhängig. Um zu erreichen, daß verschiedene Architekturen unterstützt werden können, wird der Paketdateibaum um ein oder mehrere architekturspezifische Verzeichnisse erweitert:

- .arch
  Anbindepunkt und generischer Pfad für architekturspezifische Dateien
- .arch.*architektur*
  Teilbaum für die architekturspezifischen Dateien einer konkreten Architektur

Die zusätzlichen Verzeichnisse der Form .arch.*architektur* enthalten alle Dateien der Architektur "*architektur*". Hier werden die Binärdateien gespeichert. Das Verzeichnis ".arch" enthält auf einem Dateiserver Verweise auf die entsprechenden .arch.*architektur* Verzeichnisse. Dies erlaubt einem Dateiserver auch die Paketstruktur nutzen zu können, ohne den *amd*

verwenden zu müssen. Auf einem Rechner wird das *Paket*/**.arch** Verzeichnis als Anbindepunkt für das entsprechende *Paket*/**.arch.***architektur* Verzeichnis verwendet, wobei die "*architektur*" der des Clients entspricht.

Der Pfad **/local/***Paket*/**.arch/bin** (oder auch **.../etc**, **.../lib** und weitere) erreicht durch die oben angegebene Verzeichnisstruktur *immer* die korrekten Binärdateien. Die jeweilige Assoziierung des **.arch** Verzeichnisses mit dem richtigen **.arch**.*architektur* Verzeichnis geschieht über den *amd*. Das Ziel der Architekturunabhängigkeit wurde somit erreicht.

2.2. Gemischte Verzeichnisse

Häufig besteht der Wunsch architekturabhängige wie -unabhängige Dateien in einem gemeinsamen Verzeichnis (z. B. *Paket*/**bin**) zu hinterlegen. Dies ist leicht durch Verwendung von symbolischen Verweisen (*symbolic links*) möglich. Das entsprechende Verzeichnis enthält alle architekturunabhängigen Dateien, während die architekturabhängigen Dateien über symbolische Verweise der Art **bin**/*name* ⇒ /**local**/*Paket*/**.arch/bin**/*name* referenziert werden. Hierbei sind zwei Dinge zu beachten: Zum einen ist der Paketname innerhalb des Verweises gespeichert, dies bedeutet, daß das Paket nicht elementar umbenannt werden kann. Zum anderen wird ein absoluter Pfad angegeben. Der absolute Pfad erklärt sich aus der Tatsache, daß, um auf das korrekte, architekturabhängige Verzeichnis über das **.arch** Verzeichnis zu gelangen, das *amd* Programm genutzt werden muß. Wird dies unterlassen (z. B. durch Verwendung von relativen Verweisen) kann es passieren, daß die Verweise durch NFS innerhalb des Dateiservers aufgelöst werden und man dadurch Binärdateien der falschen Architektur erhält.

Heutzutage ist der zeitliche Mehraufwand für das Verfolgen von symbolischen Verweisen durch die hohe Geschwindigkeit der Rechner tolerierbar geworden und der Zeitverlust wird durch die Uniformität der Dateibäume, sowie die administrativen Vorteile mehr als aufgewogen.

2.3. Paketspeicherung

Auf Dateiservern werden die Pakete, obwohl es ohne Probleme möglich wäre, nicht in dem Dateisystem **/local** gelagert, sondern unter **/export/lexec**. Dies erlaubt die Einrichtung eines speziell für einen Dateiserver zugeschnittenen /local-Dateisystems. Im allgemeinen können die Pakete überall gelagert werden, weil die entsprechenden Übersetzungen mit Hilfe des *amd* Programms geschehen. Um allerdings die Tabellen für das *amd* Programm möglichst einfach zu gestalten, bietet sich ein Standardverzeichnis auf den Dateiservern an. Die Verwendung eines Standardverzeichnisses erlaubt auch eine zumindest teilweise automatische Generierung der *amd* Tabellen bzw. die Verwendung von generischen Einträgen.

2.4. Systemintegration von Paketen

Das Bündeln von Softwarepaketen zu einem einzigen Dateibaum hat den Vorteil, daß es leicht möglich ist, neue Betriebssystemversionen einzuspielen, ohne die gesamte Softwareinstallation neu generieren zu müssen, bzw. Konfigurationsdaten retten zu müssen. Bei den heutigen, kurzen Zeiträumen, in denen neue Software Ausgabestände erscheinen, ist dies ein nicht zu unterschätzender Vorteil. Trotzdem gibt es (leider immer noch) viele Softwareprodukte, die auf gewisse “Standard”-Pfade angewiesen sind, oder darauf vertrauen, auf ganz bestimmten Pfaden (z. B. **/usr/***Paket*) installiert zu sein. Diesem Mißstand wird mit symbolischen Verweisen begegnet, die von den Systemverzeichnissen in das entsprechende Paket zeigen. Bei einer Neuinstallation des Betriebssystemes sind nur die Verweise neu zu erstellen. Da dieses Erstellen der Verweise ein potentiell häufig zu wiederholender Vorgang ist (mindestens je einmal für jeden Rechner mit eigenem Systembereich), bietet sich eine Beschreibungsdatei (**/local/***Paket***/++Package++**) zur Dokumentation und zur automatischen Generierung der Verweise mittels eines Hilfsprogramms (*plinks* – ein *perl*-Script) an. Neben der Dokumentation der Verweise wird in der ++Package++ Datei noch die email-Adresse des Paketbetreuers, das Quellenverzeichnis, sowie eine Kurzinformation gehalten.

```
#
# NTP Time Service
#
INSTALLER: kardel@immd4
SOURCE: /src/xntp
INFO:
NTP time synchronisation protocol
 implementation Version 3
Service is provided for Germany/Europe at
Erlangen
END
#
# link information
# (put symbolic links from /local/etc/bin/* into
# this package)
LINKS:
bin/*      /local/etc/bin     file=error directory=error
END
```

2.4.1 Konfigurationsdateien

Viele Softwarepakete benötigen für den korrekten Ablauf auch Konfigurationsdateien. Es gibt mehrere Klassen von Konfigurationsdateien, die sich aus der Sicht der Paketverwaltung hauptsächlich durch ihren Gültigkeitsbereich unterscheiden. So gibt es Konfigurationen, die institutsweit gültig sind (z. B. Aufruf des Druckprogramms, Papierformat A4) oder auch Dateien, die einen kleineren Verwaltungsbereich haben (Arbeitsgruppe oder auch nur rechnerlokal). Es gibt prinzipiell zwei Möglichkeiten diesem Problem zu begegnen:

- symbolische Verweise
- *amd*

Die Lösung mit dem *amd* hat den Nachteil, daß für jedes Paket eine entsprechende Konfiguration in den *amd* Tabellen hinterlegt werden muß, was in jedem Falle vermieden werden sollte, da sich damit paketinterne Strukturen auf die *amd*-Tabellen auswirken und die Wartung unnötig schwierig wird.

Die symbolischen Verweise für die mehr oder weniger lokalen Konfigurationen oder Datenbereiche zeigen aus dem Paket in einen Bereich, der der entsprechenden Gruppe von Rechnern gemeinsam ist. Die derzeitige Lösung für rechnerlokale Dateien in der Informatik ist ein Verweis aus dem Paket in das Verzeichnis **/+private/local/***Paket/Konfiguration*. Hier würde eine entsprechende rechnerlokale Datei abgelegt sein. Für Dateien, die von mehreren Rechnern gleichermaßen gelten sollen, wurde **/+shared/***Paket/...* gewählt. Dieses Verzeichnis könnte auch als *amd* Verzeichnis ausgebildet sein, aber zur Zeit ist dies wegen der wenigen Pakete die eine solche Sonderbehandlung benötigen noch nicht realisiert worden. Auch wäre hier ein besonderes **.config.***Gruppe* Verzeichnis denkbar, was über den *amd* verwaltet wird.

2.5. Besondere Pakete

Um nicht jeden Nutzer mit den Feinheiten der Paketumgebung behelligen zu müssen, ist ein Spezialpaket eingeführt worden, das alle verwendete lokal installierte “Standard”-Software (wie z. B. Mail-Programm, WYSIWYG-Editor, lokale Dienstprogramme, ...) enthält. Diese Pakete sind unter **/local/++GENERIC++** abgelegt. Das ++GENERIC++ Paket ist genauso strukturiert wie ein normales Paket, wobei allerdings die meisten Dateien nur symbolische Verweise in die eigentlichen Pakete sind. Nur Dateien und Kommandos, für die es sich nicht lohnt, ein eigenes Paket zu installieren (z. B. eine Kommandodatei wie die *tcsh*), werden direkt im ++GENERIC++ Paket installiert. Zusätzlich zu dem ++GENERIC++ Paket gibt es in dem /local Verzeichnis die Namen lib, bin, man, include und etc, die auf **++GENERIC++/***name* verweisen, wodurch das Integrieren der lokalen Kommandos durch ein einfaches Erweitern der Pfadliste durch **/local/bin** zu erreichen ist. Hierdurch wird der Nutzer in die Lage versetzt, die Paketstruktur im Rahmen der standardmäßig installierten Software zu nutzen, ohne etwas von der Paketstrukturierung zu wissen. Durch den Einsatz weiterer Namen ist es leicht möglich, auch mehrere Versionen eines Softwarepakets zu unterstützen. Hier werden die eigentlichen Pakete unter ihrem Namen konkateniert mit der jeweiligen Softwareversion installiert. Ein zusätzlicher Name für das standardmäßig verwendete Paket verweist dann auf die entsprechende Version. So gibt es in der Informatik z. B. mehrere Versionen des Editors *Frame*, die alle über

/local/frame.*version* erreichbar sind. Zusätzlich ist die aktuelle *Frame* Version über **/local/frame** ansprechbar.

2.6. Redundante Pakete

Es ist häufig wünschenswert, Softwarepakete redundant auf mehreren Dateiservern zu speichern. Redundante Speicherung bietet sich bei häufig benutzten Softwarepaketen an, oder wenn die Netzinfrastruktur dies nahelegt (zu viele Gateways zwischen Client und Server). In diesem Fall wird ein Server als Primärserver ausgewählt und die Sekundärserver werden mit Mechanismen wie z. B. *rdist* aktualisiert. Diese Vorgehensweise wird auch durch das *amd* Programm insofern unterstützt, als daß für einen Dateisystemteilbaum mehrere Server angegeben werden können, aus denen der *amd* je nach Erreichbarkeit einen auswählt. Hierdurch wird die Verfügbarkeit einzelner Softwarepakete erhöht. Der Aufwand der redundanten Speicherung lohnt sich allerdings nur bei häufig genutzten Softwarepaketen und ist nur schwer durchführbar, wenn die Softwarepakete unterschiedliche lokale Datenbestände enthalten (z. B. in Form einer Statusdatei oder einer festen Konfigurationsdatei).

2.7. Dokumentationsdateien

Wie schon in den vorherigen Kapiteln angedeutet, verwendet das Erlanger Paketkonzept eine nicht zu unterschätzende Anzahl symbolischer Verweise. Um insbesondere in den Systemverzeichnissen lokale Erweiterungen von Originaldateien unterscheiden und den einzelnen Paketen zuordnen zu können, sind spezielle Dokumentationsdateien in den Systemverzeichnissen eingeführt worden, die die lokalen Erweiterungen beschreiben. In diesen Dateien sind die Namen der hinzugefügten Dateien, ihr Installationsdatum, der Ansprechpartner und das Quellenverzeichnis vermerkt. Da erfahrungsgemäß diese Dateien nie auf aktuellem Stand sind, werden die Einträge automatisch aus den ++Package++ Dateien generiert und täglich die Installationsverzeichnisse auf fehlende Einträge untersucht. Wird ein Softwarepaket über längere Zeit nicht eingetragen, so wird dies gelöscht. Diese Vorgehensweise garantiert (spätestens nach der zweiten Neuinstallation) ein korrektes Eintragen der Softwarepakete.

2.8. Amd Tabellen

Im Bereich der Informatik werden mehrere Verzeichnisse über das *amd* Programm realisiert. Neben dem Verzeichnis **/local** für die installierten Softwarepakete gibt es noch die Verzeichnisse **/src**, **/pro**j, **/home** und **/net**. Das **/src** Verzeichnis übernimmt die Anbindung von Quelldateibäumen. Projekte, wie zum Beispiel Kurse, werden über **/proj** erreicht. Über **/home** werden die Basisverzeichnisse der Benutzer abgebildet. Eine Besonderheit ist **/net**. Alle von einem Rechner montierbaren Dateibäume sind über **/net/***Rechnername* verfügbar.

Der *amd* benötigt für alle, mit Ausnahme von **/net**, oben genannten Verzeichnisse Tabellen für die korrekte Umsetzung von den gesuchten Namen in einen Verweis auf den entsprechen Teildateibaum. Für die Erstellung der entsprechenden Tabellen wurde der *m4* Macroprozessor als Hilfsmittel eingesetzt. Dies war insbesondere deswegen möglich, weil die Pakete, wie auch die Quell- und Projektverzeichnisse, eine regelmässige Struktur haben. Eine *amd* Tabelle für die Pakete, die unter **/local** ansprechbar sein sollen, könnte wie folgt aussehen:.

```
#
# amd Tabelle für /local
# package(<Paket>, <Präfix-Pfad>, <Rechner>)
package(NTP, /export/lexec, faui45)
package(elm, /export/lexec, faui01)
package(frame, /export/lexec, faui01)
```

Ähnliche Macros sind auch für die Verzeichnisse **/src**, **/proj** und **/home** vorhanden.

```
#
# amd Tabelle für /proj
# project(<Projekt>, <Präfix-Pfad>, <Rechner>)
project(FrameKurs, /proj.stand, faui43)
project(Pascal, /proj.stand, faui09)
#
# amd Tabelle für /src
# source(<Quelle>, <Präfix-Pfad>, <Rechner>)
source(emacs, /src.stand, faui43)
source(elm, /src.stand, faui43)
```

Die Verbreitung der *amd* Tabellen geschieht über den NIS (*Network Information Service*) von Sun. Andere Mechanismen wie z. B. Dateien, Hesiod oder DBM werden vom *amd* unterstützt, aber nicht innerhalb der Informatik verwendet.

3. Erfahrungen

Das Paketkonzept, sowie der *amd* zum automatischen Einbinden von NFS Dateisystemen, sind seit ca. 1 1/2 Jahren im Einsatz, wobei am Anfang die korrekte Konfiguration des *amd* noch Schwierigkeiten bereitet hat. Inzwischen ist die Installation stabil und die Arbeitsersparnis bei der Softwareinstallation und Wartung beträchtlich. Das Eintragen eines neuen Pakets erfordert nur noch das Einfügen einer einzigen Zeile. Viele Produkte von Softwareherstellern lassen sich leicht auf die Paketstruktur anpassen. Einige Produkte sind schon bei Auslieferung so strukturiert, daß man sie wie ein Paket einsetzen kann.

3.1. Verfügbarkeit

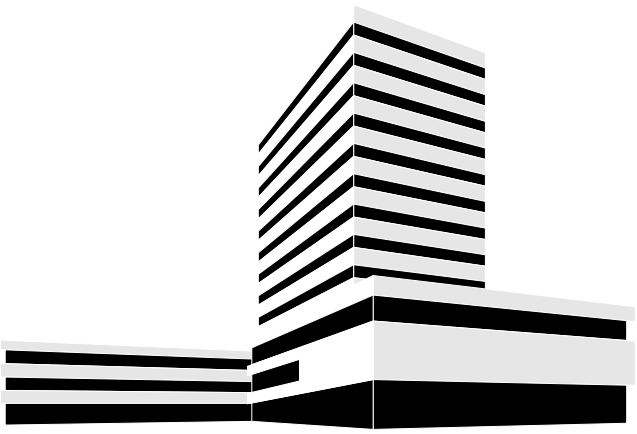
amd ist über anonymes FTP (File Transfer Protocol - Internet) von *usc.edu* (128.125.1.45) in der Version 5.3-beta (23. 2. 1992) erhältlich. Kopien sind auch auf deutschen FTP Servern vorhanden. Der Script-Interpreter *perl* ist auf fast allen FTP Servern erhältlich.

15. DECUS München Symposium

NFS Organisation mit Automounter

Frank Kardel

kardel@informatik.uni-erlangen.de



IMMD IV • Martensstraße 1 • D-W8520 Erlangen

1. Motivation

- **immer mehr, immer leistungsfähigere Rechner**
- **Vernetzung dieser Rechner notwendig (Fileserver, Zentrale Dienste)**
- **Verschiedenste Hersteller**
- **Softwarepakete**
  - **Unterstützung mehrerer Hersteller**
  - **immer komplexer (damit aufwendige Pflege)**

2. Lösungsansätze

- **Einheitliche Installation für alle Rechner (Benutzerfreundlichkeit)**
- **Dateiserver (Konsistenz, und damit leichtere Administration)**
  - **AFS (Andrew File System) gute Caching Eigenschaften, aber relativ teuer**
  - **NFS (Network File System) preiswert, da von fast jedem Hersteller angeboten**
  - **andere... (LOCUS, SPRITE, ...)**

3. Probleme beim Einsatz von NFS

- **Konfiguration**
  statische NFS Dateianbindungen sind nicht skalierbar

- **Pflege**
  Änderungen der Tabellen sollten möglich sein, auch ohne, daß alle beteiligten Rechner erreichbar sind

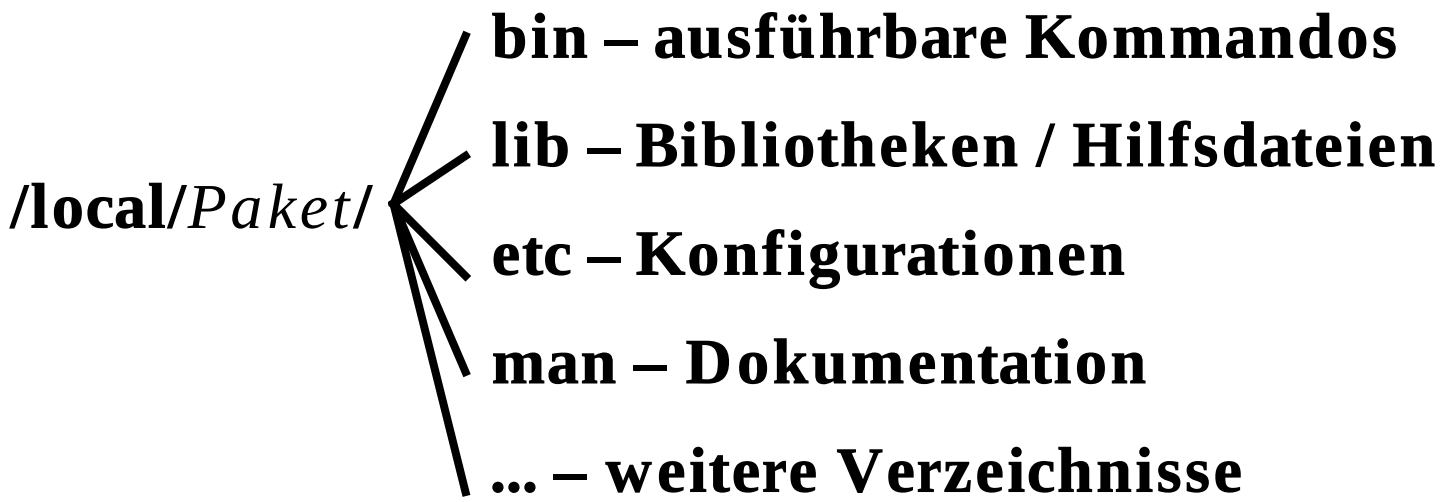
- **Integration**
  möglichst viele Hersteller sollten unterstützt werden

4. *AMD* - ein “Automounter”

- **für viele Hersteller erhältlich (d. h. übersetzbar)**
- **Konfigurationstabellen können über Netz verteilt werden**
- **Es wird nur “montiert”, was gebraucht wird**
- **Netzwerkweit gültige Pfade**
- **Tolerierung defekter Dateiserver**

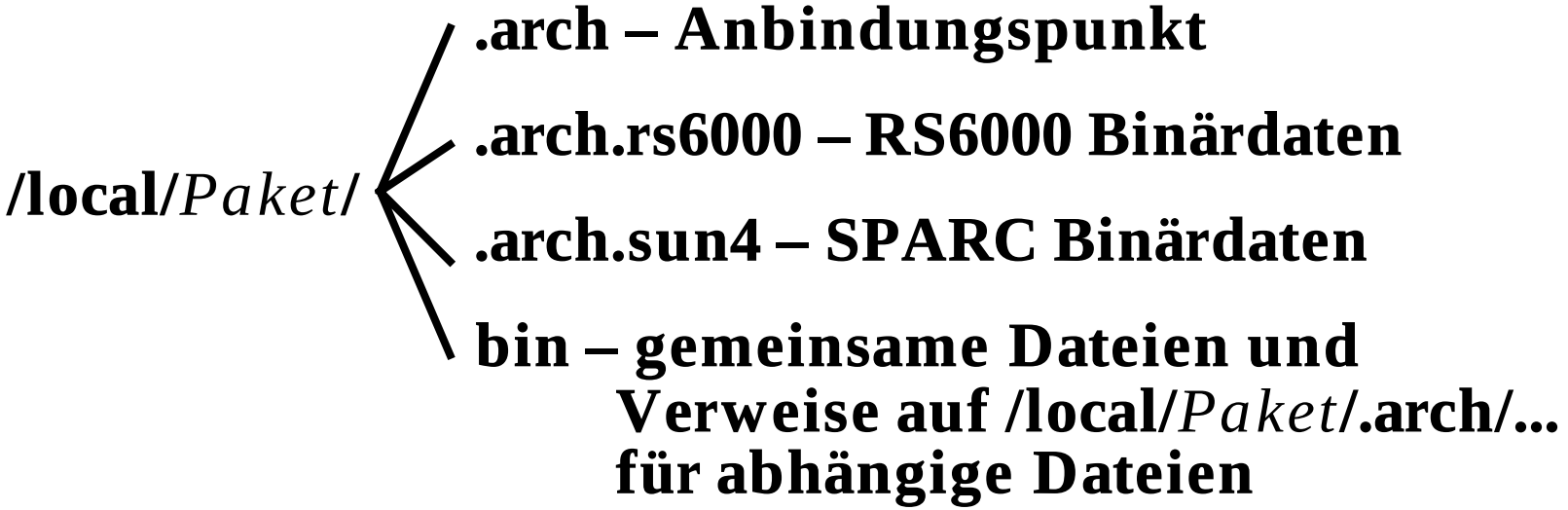
5. Softwarepakete

- **einheitliche Namensgebung (/local/*Paket* - per *AMD*)**
- **Pakete sind in einem Verzeichnis abgelegt**
- **einheitliche Struktur**



6. Architekturunabhängige Pakete

– **Trennung von architekturabhängigen und unabhängigen Dateien**



7. Systemintegration von Paketen

- **Pakete werden mit Hilfe von symbolischen Verweisen in das System integriert**

- **Die Datei** *++Package++* **beschreibt, welche Verweise erstellt werden müssen**

```
#
# Time Synchronization - Service for Germany and Europe
#
# link information
# (put symbolic links from /local/etc/bin/* into this package)
LINKS:
bin/*           /local/etc/bin          file=error directory=error
END
```

8. Besondere Pakete

- *++GENERIC++* **enthält:**
  - **Verweise auf allgemein gebräuchliche Pakete (mail, Texteditoren, ...)**
  - **allgemein verwendete Software, die zu klein wäre, ein Paket zu rechtfertigen (Dienstprogramme, Kommandointerpreter)**

9. Redundanz

- **Häufig genutzte Pakete können repliziert werden**
- **Konsistenzwahrung**
  - *rdist*
    Aktualisierung der Paketkopien von einer Referenzinstallation

10. *AMD* Tabellen

- */local* – **Softwarepakete**
- */src* – **Quelldateibäume**
- */proj* – **Projektverzeichnisse**
- */home* – **Benutzerverzeichnisse**
- */net* – **exportierte Dateisysteme anderer Rechner**

11. Generierung von *AMD* - Tabellen

– Nutzung des *m4* Makroprozessors

```
#
# amd Tabelle für /local
# package(<Paket>, <Präfix-Pfad>, <Rechner>)
package(NTP, /export/lexec, faui45)
package(elm, /export/lexec, faui01)
package(frame, /export/lexec, faui01)
```

```
#
# amd Tabelle für /proj
# project(<Projekt>, <Präfix-Pfad>,<Rechner>)
project(FrameKurs, /proj.stand, faui43)
project(Pascal, /proj.stand, faui09)
#
# amd Tabelle für /src
# source(<Quelle>, <Präfix-Pfad>, <Rechner>)
source(emacs, /src.stand, faui43)
source(elm, /src.stand, faui43)
```