ANFORDERUNGSPROFIL UND TOPOLOGIE EINES VERTEILTEN SOFTWARE-ENTWICKLUNGS-SYSTEMS

Thorsten Spitta

SCHERING AG Berlin/Bergkamen

Zusammenfassung

Der Beitrag entwickelt aus den Anforderungen professioneller Softwareentwicklung an Werkzeuge und Entwicklungsrechner eine Topologie für verteilte Software-Entwicklungssysteme im kommerziellen Bereich. Die Anforderungen an verteilte Werkzeuge und die Lösung des Portabilitätsproblems bei dedizierter Entwicklung bilden die Kernpunkte praktisch verwendbarer Systeme. Die Komplexität dieser beiden Fragestellungen wird im Zuge einer gewissen "PC-Euphorie" häufig unterschätzt. Kurz betrachtet werden "Sprachen der 4. Generation" in diesem Zusammenhang und ein Praxisbeispiel.

1 Einführung

1.1 Begriffe und Abgrenzung

Unter "verteiltes Software-Entwicklungs-System" (VSES) wird hier ein inhomogenes Rechnernetz mit verteilten Software - Werkzeugen verstanden, das verschiedene Rechnertypen von Entwicklungs- und Produktionsrechnern für verschiedene Benutzergruppen umfaßt.

"Verteilt" sind Software-Werkzeuge, wenn die von ihnen verwalteten Entwicklungsergebnisse oder Informationsbasen physisch auf mehrere Rechner aufgeteilt oder redundant sind und somit konsistent gehalten werden müssen. Die Notwendigkeit, ein Werkzeug verteilt zu halten, ergibt sich aus den Anforderungen der Benutzer an die Verfügbarkeit des Werkzeugs während des Entwicklungsprozesses.

Als Synonym für "Software-Entwicklungs-System" wird häufig der Begriff "Software-Produktions-Umgebung" benutzt (vgl. HAU 83). Uns erscheint der Begriff "Software-Entwicklungs-Umgebung" zutreffender, denn die Analogie zur industriellen Produktion ist nicht gegeben. Es besteht vielmehr eine starke Ähnlichkeit mit der ingenieursmäßigen Entwicklung von Produkten.

Da auch die Kennzeichnung "Umgebung" sprachlich nicht das kennzeichnet, um was es sich handelt, nämlich um ein Hardware-/Softwaresystem, benutzen wir den Begriff "Software - Entwicklungssystem" (SES).

Unter "Rechnertyp" wird insbesondere das Betriebssystem verstanden. Von unterschiedlichem Typ sind vor allem der/die Entwicklungsrechner (ER) und der/die Produktionsrech-

Die Projektgruppen müssen häufig räumlich voneinander getrennt in verschiedenen Gebäuden arbeiten.

Die Entwickler erstellen im Zuge des Entwicklungsprozesses Ergebnisse oder lassen sich Informationen über die Entwicklung vom ER erzeugen. Im letzten Fall "erstellen" sie Anfragen an den ER.

Mit Bezug auf die Bedienung eines ER unterscheiden sich die Benutzertypen durch

- Fertigkeiten
- Benutzungshäufigkeit
- bearbeiteten Ergebnistypen.

An Ergebnistypen werden unterschieden:

- TEXT : Texte und Grafik
- DATA : Daten in speicherbarer Form
- SOURCE : jede Art von formaler, maschinell verarbeitbarer Sprache
- QUERY : Eingaben zum Gewinnen von Informationen aus dem Rechner im Dialog
- ACC : Accounting-Daten, die maschinell auswertbar sind (z.B. zur Kostenverrechnung).

Es bestehen folgende Zusammenhänge:

Kategorie \ Benutzertyp	P M	P L	E W	
			A N	S E
Fertigkeiten	gering	mittel	mittel	hoch
Benutzungshäufigkeit	gering	mittel	mittel	ständig
Ergebnisstypen:				
TEXT		X	X	X
DATA		(X)	X	X
SOURCE		(X)	(X)	X
QUERY	X	X	X	X
ACC		X	X	X

Legende: X = ja, wird erzeugt
(X) = wird gelegentlich oder fallweise erzeugt

Die Erstellung von Datenbankabfragen durch den Anwender durch sog. "Endnutzersprachen" wird als Typ QUERY gesehen.

Für die Verteilung in einem Netz bedeutet die Tabelle:

Für den ständigen Benutzer (EW) ist die dezentrale komfortable Verfügbarkeit von Werkzeugen und Informationen am wichtigsten. Sie beeinflußt bekanntlich unmittelbar und nachhaltig seine Produktivität. Der Bedarf für PM und PL kann dem nachgeordnet werden, wenn es technisch oder wirtschaftlich Restriktionen gibt. Im Bereich QUERY muß in jedem Fall ständige Verfügbarkeit gefordert werden.

Accounting-Daten werden dezentral eingegeben und brauchen i.d.R. nicht online über QUERY abfragbar zu sein.

2.2 Werkzeugtypen auf Entwicklungsrechnern

Die in der Softwareentwicklung erforderlichen Werkzeuge werden im folgenden unter dem Gesichtspunkt ihrer "Verteilbarkeit" betrachtet und in 2.2.5 in einer Tabelle zugeordnet :

- nur dezentral benötigt.....: D
- nur zentral benötigt.....: Z
- verteilt benötigt.....: V.

Werden Ergebnisse von QUERY dezentral gespeichert, weil sie nicht im Dialog erzeugbar sind, so sind sie zueinander nicht konsistent. Dies fällt nicht unter den Begriff "Verteilung", wenn die Primärdaten nicht verteilt gehalten werden.

Die Werkzeuge werden grob in der zeitlichen Folge ihrer Benutzung im Entwicklungsprozeß betrachtet.

Es werden nur die wichtigsten strategischen Werkzeuge detaillierter diskutiert, und zwar diejenigen, bei denen es eine Frage ist, ob sie zentral oder verteilt benötigt werden.

2.2.1 Dezentrale Werkzeuge

Dezentral zu halten sind folgende Werkzeuge:

- Mailbox-System
- Werkzeuge zur Texterstellung
  (vom Textsystem über Druckprozeduren bis zum syntaxgesteuerten Editor)
- Werkzeuge zur Grafikerstellung
- Produktmuster
  (vom kommentierten Inhaltsverzeichnis bis zum Programmskelett)
- Design-unterstützende Werkzeuge
  (z.B. für Spezifikation oder Modularisierung)
- die Codierung unterstützende Werkzeuge
  (Compiler, Interpreter, Generatoren, Prüf-Werkzeuge)
- Testwerkzeuge
  (Generatoren für Testtreiber, Stubs, Testdaten und Testfälle, Werkzeuge für Regressionstests)
- Technologie-Handbuch

Alle diese Werkzeuge müssen dezentral als Kopien existieren, die identische Ergebnisse bei gleichen Eingaben liefern.

Compiler und Interpreter werden dezentral -wenn verteilt Source entwickelt wird- und zentral auf dem PR benötigt (Wiederherstellung zerstörter Object-Module).

Unter "Design-unterstützende Werkzeuge" fallen im Bereich der Spezifikation sowohl einfache Formatier-Unterstützungen für Methoden als auch Spezifikatiossprachen wie etwa PROLOG.

Ein Online-Technologie-Handbuch, das wichtige Verfahren, Beispiele, Definitionen und Standards enthält, muß dezentral verfügbar sein und zentral gepflegt werden. Das Handbuch dient als zentrales Regelwerk innerhalb der Organisation und als Auskunftinstrument.

Die sonst nicht weiter diskutierten Werkzeuge erscheinen in der Tabelle von 2.2.5 . Sie werden dort z.T. zusammengefaßt.

2.2.2 Projektbibliothek

"Eine Projektbibliothek(PB) ist ein rechnergestütztes Bibliothekssystem für die Softwareentwicklung und -wartung, das

- Entwicklungsprodukte und ihre Beziehungen zueinander verwaltet,
- Dokumente durch einfache Abfragen gezielt und schnell zur Verfügung stellt,
- den Fortschritt einer Entwicklung jederzeit transparent macht." (SPI 84)

Eine Projektbibliothek wird dezentral für die Nutzung in Projekten benötigt. Sie muß aber zentral auskunftsfähig sein und den Zustand zentraler Produktionsbibliotheken kennen oder mitverwalten. Da sie zentral Kenntnisse über die dezentral gehaltenen projektspezifischen Ergebnisse benötigt,fällt sie unter die Kategorie "V". Sowohl Dokumentationen als auch Quellprogramme werden verteilt über die Entwicklungsrechner gehalten.

Die PB ist die Eingabeschnittstelle für Accounting-Daten. Sie liefert die Schnittstelle für QUERY über alle entwicklungsrelevanten Daten für alle Benutzergruppen. Nützlich ist es, wenn die Schnittstelle zu zentralen SOURCE-Bibliotheken in die PB integriert ist.

2.2.3 Source- und Object - Bibliotheken

Logisch sind SOURCE- und OBJECT - Bibliotheken Bestandteil der PB. Es sind jedoch historisch zwei Werkzeugtypen entstanden, die sich in der Praxis nur mit sehr großem Aufwand integrieren lassen.

Bei diesem Typ von Werkzeug müssen zwei Fälle unterschieden werden:

1. Programme "in Produktion" (Produktionsversion)
2. Programme "in Entwicklung", "in Wartung", "im Archiv".

zu 1:

Die Bibliothek enthält Source und Object der Programme und darf nur genau einmal auf dem Produktionsrechner existieren. Die Projektbibliothek muß mit der Produktionsbibliothek (PRB) kommunizieren können, da der PB die Versionsverwaltung unterliegt.

Die Erfahrung in unserem Hause zeigt, daß die PRB erst zum Werkzeug wird, wenn sie den eigentlichen Datenbestand der SOURCE-Bibliothek abschirmt und Operationen auf ihr anbietet. Hierzu gehört auch das Übergabeverfahren an Produktion bzw. die "Rückgabe" an die Wartung. Jede gezogene Kopie muß protokolliert werden.

zu 2.:
Alle übrigen Programm-Versionen werden von der Projektbibliothek zusammen mit den übrigen Dokumenten verwaltet. Sie werden sinnvollerweise dezentral in Verantwortung von Entwicklungs- und Wartungsgruppen gehalten.

Eine "Übergabe an Produktion" ist keine Verteilungsoperation, sondern ein Datentransfer (einer SOURCE). Das Original wird aus der PB in die PRB ausgelagert und in der PB gelöscht. Die PB kennt den Lagerort.

2.2.4 Datendesigner und Datenlexikon

"Datendesigner" ist eine Klasse von Werkzeugen, die der Analyse und Synthese von Datenmodellen dient, z.B. um relationale Datenmodelle zu erhalten. Sie werden langfristig mit den historisch älteren Data Dictionaries zusammenwachsen, da ihre Datenbasis dieselbe ist.

Bezüglich einer Verteilung stellen sich dieselben Anforderungen wie an eine PB, jedoch verschärft, da ein ständiger Abgleich dezentraler Ergebnisse mit zentralen erforderlich ist. Ein verteiltes Datenlexikon (DLX) ist technisch eine verteilte Datenbank mit allen Konsequenzen bezüglich Datensicherheit, Konsistenz, Transparenz und Synchronisation bei updates (vgl. BAY 84).

Ein DLX ist inhaltlich die zentrale Meta-Datenbank der Informationsressourcen einer Organisation. Bei dieser Verwendung genügt die dezentrale Auskunftfähigkeit eines nicht verteilten zentralen DLX.

Häufig wird das DLX auch als formaler Strukturspeicher benutzt, vor allem zur Speicherung der Beziehungen zwischen Programmen (Aufrufbeziehungen) und zwischen Programmen und Daten (Schnittstellen). Dann genügt die dezentrale Auskunftfähigkeit nicht. Vielmehr muß es dezentrale Sublexika geben, die dazu beitragen, unnötig viele Abstimmungen zwischen Projektgruppen zu vermeiden. Erfahrungsgemäß sind während der frühen Phasen Beziehungen zwischen Daten und ihren Inhalten ständig im Fluß, so daß das zentrale DLX zu schwerfällig ist.

Die Sublexika bieten neben der Speicherung von Datenbeschreibungen Generierungsleistungen an (Daten-INCLUDES für Programme), über die Dictionaries üblicherweise verfügen.

Die dezentralen Sublexika müssen zu einem wohldefinierten Zeitpunkt der Entwicklung in das zentrale DLX integriert werden. Dies muß vor der Entwicklung der Programme sein. Ab diesem Zeitpunkt dürfen Datenbeschreibungen der Sublexika nicht mehr ohne weiteres geändert werden können, da sie teilweise Kopien darstellen. Bis zur Integration sind die Sublexika dezentrale, weitgehend autonome Werkzeuge. Danach sind sie Bestandteil eines verteilten Gesamt-DLX.

Die Datenhaltung des zentralen DLX sollte ohne Redundanzen ausschließlich zentral erfolgen. Im Gegensatz zu den Datenbanken ist das Datenvolumen eines DLX so gering, daß es aus Gründen des Mengengerüstes keine Gründe für eine Verteilung der Daten selbst gibt. Dies gilt für die allgemein übliche Verwendung des DLX wie bisher beschrieben. Vereinzelt wird jedoch das DLX auch als Projektbibliothek benutzt, indem Dokumente in die Beschreibung der Knoten des DLX gestellt werden. Dann ist u.E. das DLX nur noch verteilt zu handhaben.

2.2.5 Überblick

Benutzer-typ / Werkzeug	P M	P L	E W		Lokalität	Anmerkung / vgl.
			A N	S E		
Projektbibl.	X	X	(X)	(X)	V	2.2.2
Mailbox-Syst.	X	X	X	X	V	
Editor, Grafik Textsystem, Produktmuster		X	X	X	D	
Datendesigner, Datenlexikon		X X	X	X X	V Z/V	2.2.4
Spezifikations-Hilfsmittel		(X)	X	X	D	2.2.1
Moduldesigner		X		X	D	u.W. nicht verfügbar
Source-Bibl.		(X)		X	Z	2.2.3
Test-Datenbank		(X)	(X)	X	D	2.3.4
Compiler/Int.		(X)		X	D	
Werkzeuge zur Codeerzeugung u. -prüfung		X		X	D	Typ SOURCE
Testwerkzeuge		(X)	(X)	X	D	
Übergabe-Software an Prod. / Wartung		X		X	Z	2.2.3
online-Handbuch Technologie u. Standards	X	X	X	X	D	2.2.1

2.2.6 Zusammenfassung zu den Software - Werkzeugen

Der überwiegende Teil der Werkzeuge kann dezentral gehalten werden, wenn ein geregeltes Verfahren zur Verteilung der Kopien bei neuen Releases von Werkzeugen existiert.

Der Projektbibliothek als verteilter PB kommt eine privilegierte Funktion in einem verteilten Software - Entwicklungssystem zu, da sie einen zentralen Katalog mit entsprechender Zustandsverwaltung über die dezentralen Dokumente und Sourcen, deren Versionen und die Beziehungen zu Programmen in der Produktion führen muß. Eine -allerdings an ein bestimmtes Betriebssystem gebundene- verteilte PB ist als neues Release einer am Markt vorhandenen angekündigt. Die einzige uns bekannte "verteilte PB" ist das Bibliotheks-Verwaltungssystem CONTROL, das auf TANDEM-Rechnern läuft.

Ein reines Datenlexikon kann zentral gehalten werden, wenn es dezentral auskunftfähig ist.

Ein DLX als allgemeiner Strukturspeicher muß verteilt (zentrales DLX + Sublexika) ausgelegt sein. In diesem Fall müssen die Strukturen mit der Projektbibliothek konsistent gehalten werden oder die PB benutzt das DLX als Strukturspeicher. (Bsp.: verteilte PB im Hause SCHERING.)

Das Datenlexikon muß zunächst zentral erfolgreich eingeführt sein. Danach kann und sollte seine Erweiterung zum verteilten DLX durch dezentrale Sublexika erfolgen. Ein homogenes verteiltes DLX ist derzeit am Markt nicht verfügbar. Der Anwender muß also selbst verschiedene Dictionaries integrieren.

Alle dezentralen Werkzeuge müssen in einem priviligierten Netzknoten gepflegt und von dort als Kopien bereitgestellt werden. Alle zentralen Werkzeuge können in einem beliebigen Netzknoten angesiedelt sein, wenn ein effizienter Zugriff von den Stationen gewährleistet ist.

Es kann also scheinbar (!) möglichst viel Intelligenz auf dezentrale Arbeitsplatzcomputer verlegt werden, wenn es ein zentrales Instrument (verteilte PB) gibt, das Kenntnis vom Geschehen auf den dezentralen ER hat.

Die Lösung ist für alle Ergebnistypen, die im weitesten Sinne "Dokumentation" sind (TEXT, DATA, QUERY,ACC), in der Tat nur noch von der technischen Mächtigkeit der dezentralen ER und entsprechender Netzknoten abhängig.

Für den Typ SOURCE trifft dies nicht zu, da sich hier das Problem der Rechnertypen von Entwicklung und Produktion stellt, und zwar bezüglich der Portabilität der Quellprogramme. Dies wird im folgenden näher beleuchtet.

2.3 Rechnertypen

Während bei den Software-Werkzeugen der heutige Istzustand nicht explizit betrachtet wurde, erscheint dies bei den Basissystemen angebracht, da sich konkrete Umwälzungen auf diesem Gebiet abzeichnen, die realisierbare verteilte Entwicklungssysteme in der kommerziellen Welt prägen. Aus diesem Grund werden auch Systemnamen genannt, die im Bereich unserer Zielgruppe dominieren.

2.3.1 Anforderungen

Softwareentwickler brauchen ein leistungsfähiges Teilnehmer - Betriebssystem mit

1. guter Text- und Druckaufbereitungssoftware
2. leistungfähigem, einfach zu handhabendem File-System mit Schutzmechanismen
3. Programmiersprachen zur Erstellung von Software-Werkzeugen mit guten Eigenschaften zur String-Manipulation
4. den "Produktionssprachen" der Zielmaschine (=PR)
5. Zugriffsmechanismen auf den/die DB-Typen der Zielmaschine, zumindest eine Simulation derselben
6. Möglichkeit des Testens der entwickelten Software unter dem Teilhaber - Betriebssystem der Produktionsrechners oder eine entsprechende Simulation (sog. Dialogmonitore)
7. einfacher Datenübertragung vom und zum Produktionsrechner.

2.3.2 Das Problem der Portabilität im kommerziellen Bereich

"Portabilität" ist die Eigenschaft eines Quellprogramms, mit möglichst geringen Änderungen von einem Rechnertyp auf einen anderen übertragbar zu sein, indem man auf der Zielmaschine lediglich neu compiliert. Das Programm muß dann dieselben Ergebnisse erzeugen wie auf der Ausgangsmaschine.

Genau der letzte Punkt macht diese Materie so schwierig und gefährlich.

In kommerziellen Umgebungen portabel sind heute nur COBOL und FORTRAN; dies auch nur für Batch-Programme.
Wenn man portieren will, muß in diesen Sprachen mit äußerst rigiden und empirisch überprüften Programmierkonventionen gearbeitet werden, sonst läuft man Gefahr, unerwartete Ergebnisse oder gar versteckte Fehler zu erzeugen.

Bei dedizierter Softwareentwicklung auf Rechnertypen, die sich von dem des PR unterscheiden, ist die Lösung des Portabilitätsproblems entscheidend für die entstehenden Kosten.

Wenn man sich die enormen Anstrengungen ansieht, die unter dem Aspekt der Portabilität bei ADA notwendig sind, so ist die Auffassung, Portabilität auf der Ebene einer Sprache sei nicht der richtige Weg, zumindest berechtigt.

Bei UNIX ist das Betriebssystem -wenn auch mit Einschränkungen- portabel und damit die Compiler selbst. Dieser Ansatz ist u.E. der zukunftsträchtigere,um Software-Investitionen langfristig zu sichern.

2.3.3 Istzustand

Ausgangssituation in der Praxis betrieblicher Rechenzentren ist heute überwiegend:

Entwicklung und Produktion auf demselben Rechner
Entwicklungsarbeitsplätze sind Terminals an diesem Rechner.

Mindestens die Eigenschaften 1. und 2. (vgl. 2.3.1), meist auch 3. sind auf üblichen PR nur eingeschänkt gegeben. Dazu herrscht in der Produktion Teilhaberbetrieb vor, der zusätzliche Teilnehmersysteme auf derselben Maschine stark benachteiligt, zumindest aber von der Belastung durch den Produktionsbetrieb abhängig macht.

Es spricht also vieles für dedizierte Entwicklungsrechner. Drei Wege sind dafür möglich und werden bereits praktiziert:

1. Dediziertes Text- und Filesystem: (TFS)

   Es erfordert für die Entwickler das Arbeiten mit zwei Schnittstellen, denn die Entwicklung läßt sich nur bis zur Quellcode-Editierung vorantreiben.
   Dafür vermeiden diese Systeme Portabilitätsprobleme.

   Vertreter: MAESTRO; Micro-Computer mit Netzknoten (leistungsfähiger File-Server).

2. Universalrechner mit portablem Betriebssystem: UNIX

   UNIX-Systeme ermöglichen eine geschlossene Entwicklungsumgebung, zugeschnitten auf Software-Entwickler, die es erlaubt, Software bis zum Test fertigzustellen (vgl. IVIE 77 und MAR 83 bzgl. Details). Inzwischen sind die ersten Mehrplatzsysteme mit PC's unter UNIX verfügbar.

   Für eine professionelle Entwicklung unserer Zielgruppe entsteht dann aber das Portabilitätsproblem, das nicht nur die Programmiersprache, sondern ganz besonders auch Dialogmonitor und DB-System betrifft.

3. Microcomputer, angeschlossen an den Produktions- oder einen Entwicklungsrechner

   Diese Lösung ist einerseits sehr flexibel bzgl. des Einsatzes dezentraler Intelligenz, Zahl und räumlicher Verteilung der Stationen. Andererseits muß die gesamte Integration der Entwicklungsarbeit auf dem zentralen Netzknoten vom Typ "PR" geleistet werden.
   Das Portabilitätsproblem stellt sich wie bei 2. oder es wird umgegangen wie bei 1. Probleme bei der Portierung von Quellprogrammen treten verschärft auf, da z.T. unzuverlässige und lückenhafte Compiler am Markt sind (vgl. CW 84).

Ein funktionsfähiges Netz dedizierter Entwicklungsrechner mit verteilt gehaltenen Werkzeugen ist im Bereich unserer Zielgruppe nicht bekannt.

2.3.4 Sollkonzept

UNIX wird u.E. die klassischen "Produktionssysteme" DOS, MVS, BS2000 u.a. in absehbarer Zeit nicht verdrängen können. So bleibt nur die Lösung, einen weiteren Typ von ER vorzusehen, der die Fertigstellung der Software unter einem dem PR gleichen oder ähnlichen System erlaubt.
Es soll bei der Prämisse bleiben, daß Entwicklung und Produktion zumindest logisch streng getrennt sind.

Einerseits ist UNIX als Entwicklungsrechner unter dem Aspekt von Portabilität und Verfügbarkeit von Software-Werkzeugen das Gegebene (vgl. MAR 83), andererseits wird ein "Hypervisor" benötigt, der UNIX und das Produktionssystem auf einer Maschine zusammenführt. Dies erleichtert die notwendigen Tests erheblich. Ein solches System ist z.B. VM.
Im Falle BS2000 als Betriebssystem kann nur vom UNIX-Rechner auf einen BS2000-Rechner portiert werden.

Auf von-Neumann-Rechnern ist u.E. für die nächsten 10 Jahre kein neues Betriebssystem zu erwarten, das allgemeine Verbreitung finden wird, wenn man sich vor Augen hält, daß

UNIX ca. 15 Jahre (vgl. THO 82)
VM ca. 20 Jahre (dem Autor als Vorläufer CP/67 seit 1969 bekannt)

alt sind.

Bezüglich der Verteilung von Rechnertypen bzw. Hardware ergibt sich daraus:

I. Kurzfristig:

- Dezentrale Entwicklungsrechner nutzen die Vorteile von UNIX, Werkzeugen von Arbeitsplatzrechnern oder eines spezialisierten TFS wie z.B. MAESTRO.
- Logisch gibt es einen zentralen ER-Z, der unter einem System wie VM die Betriebssysteme UNIX und z.B. MVS auf einer Maschine zusammenführt.
  Dies reduziert die Portabilitätsprobleme UNIX -->MVS, da es gemeinsame Datenbestände auf einer Maschine gibt. Portiert wird jedoch wie folgt:
  ER-D (UNIX) ===> ER-Z (UNIX) ===> ER-Z (MVS)
- Physisch kann ER-Z als separater Rechner existieren oder als "partition"-ähnliches Subsystem auf einem sehr mächtigen (Mehrprozessor-)Produktionsrechner.
- Arbeitsplätze der dezentralen ER sind "Personalcomputer", die die meisten dezentralen Werkzeuge aufnehmen und über einen leistungsfähigen Durchgriff auf ER-Z verfügen.

II. Mittelfristig werden folgende ER einsetzbar sein:

ER-D ist ein Netzknoten für PCs mit zentralem File-System, ein mittelgroßer "klassischer" Rechner oder ein TFS-System. ER-D läuft unter UNIX.

ER-Z kennt UNIX und das Betriebssystem des PR. Die beiden Betriebssysteme laufen unter VM und können gut miteinander kommunizieren. Insbesondere verhalten sich Zugriffe auf Datenbestände für den Programmierer unter UNIX genauso wie unter dem Betriebssystem des PR. ER-Z kann Programme und Daten problemlos "in Produktion" übergeben oder "für Wartung" empfangen. ER-Z kann auch eine "virtuelle Maschine" auf dem PR sein.

III. Langfristig wird die Bedeutung von ER-D als Netzknoten immer mehr abnehmen zugunsten von leistungsfähigen PC's. Irgendein privilegierter PC wird dann die Rolle des Netzknotens in einem Teilnetz übernehmen.

3 4.-Generations - "Sprachen" als Alternative ?

Die sog. "Sprachen" der 4. Generation (Bsp. ADR, FOCUS, MANTIS, MAPPER, NATURAL) enthalten zwar als Kern eine Programmiersprache, sind aber als Ganzes zentralisierte, homogene Software - Entwicklungssysteme.

Ist dies nicht eine einfachere Alternative zu einem VSES ?

Die schnellen Erfolge dieser Systeme beruhen auf der starken Spezialisierung für

- kommerzielle Anwendungen
- bevorzugt DB-Manipulation und online-Schnittstellen
- strukturell gleichartige Zielsysteme als PR
- interpretative Arbeitsweise
- Informationssysteme.

Alle Systeme werden jeweils von einem Hersteller gewartet und weiterentwickelt und unterliegen keiner internationalen Normung.

Die Sprachen selbst sind eine Mischung aus

- hochkomfortablen Lösungen
  Bsp.: Unabhängigkeit von NATURAL-Quellprogrammen vom Dialogmonitor, teilweise der Datenbasis.
  Hierdurch ist das Portabilitätsproblem im Bereich Großrechner exzellent gelöst.
- Sprachkonzepten auf niedrigstem Niveau mit hohen Unsicherheiten
  Bsp.: Tabellenverarbeitung in NATURAL.

Alle Sprachen bieten CALL-Schnittstellen zu allgemeinen Programmiersprachen an. Hierdurch können Mängel z.T. ausgeglichen werden.

Derzeit entwickeln sich die 4. Generations-Systeme einerseits in Richtung auf

- Verbesserung der Programmiersprache
- mehr Entwicklungskomfort
- Zugriffe (lesend) auf beliebige Datenbasen
- Versionen auf Microcomputern, die mehr oder weniger portabel zum weiterhin zentralen Werkzeug auf dem Produktionsrechner sind.

Die Sprachen als Kern dieser SES werden u.E. nicht den Reifegrad erreichen, der es ermöglicht, die konventionellen Programmiersprachen gänzlich zu ersetzen. Die Hersteller geraten nämlich zunehmend in den Bereich überproportional steigender Komplexität der Entwicklung einer Programiersprache. Sie werden "Lösungen der ersten Stunde" aufgeben

müssen, dann aber Migrationsprobleme bekommen. Alles dies übersteigt die Finanzkraft eines einzelnen Herstellers.

Weiterhin gehen die Hersteller selbst auf dedizierte Entwicklung mit portablen Sourcen über. Die Systeme entwickeln sich also selbst in Richtung auf verteilte SES.

Der Vorteil für den Anwender von 4. Generations-Systemen liegt darin, daß die Hersteller das Portabilitätsproblem im eigenen Interesse lösen werden und nicht den Anwender damit alleine lassen, wie dies bei den allgemeinen Programmiersprachen der Fall ist.

In einem VSES wird man die großen Vorzüge dieser Systeme im Verbund mit konventionellen Sprachen nutzen, bis völlig neue Sprachen alles dies wiederum ablösen. Die 4. Generationssysteme sind also eine Ergänzung zu einem VSES.

4 Topologie eines verteilten SES

Die Topologie wird in mehreren Alternativen angegeben, wobei sich die bildliche Darstellung für Alternative 3 erübrigt.

1. mehrere physische Entwicklungsrechner
   - Ring von dezentralen ER-D
   - Stern zum zentralen ER-Z

   Zeithorizont: heute bis in ca. 2 Jahren

2. PC-Ring mit Netzknoten
   - Ring von PC´s
   - Stern zum Zentralrechner
     (ER-Z und PR liegen physisch auf derselben Maschine)

   Zeithorizont: ca. 2 Jahre

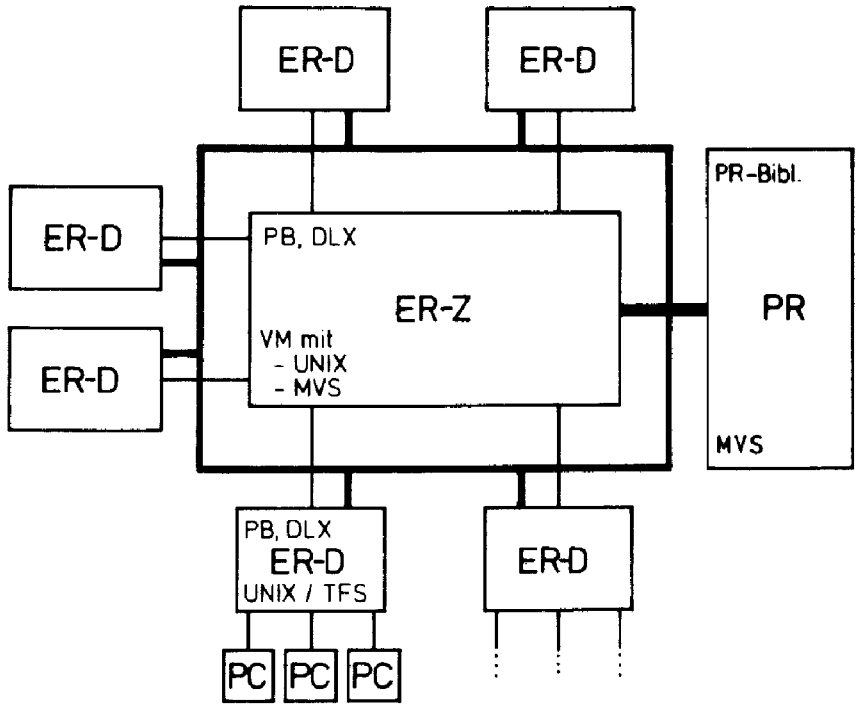
3. Vollständiges Netz mit Workstations

   Eine "Topologie" erübrigt sich:

   Jeder ER-D kann mit jedem alles.

   Zeithorizont: ca. 5-7 Jahre.

4.1 Mehrere physische ER



Legende: ER-D : dezentrale Entwicklungsrechner, die als Netzknoten, File-Server und "Verstärker" von Prozessorleistung dienen
ER-Z : zentraler Entwicklungsrechner
PR : Produktionsrechner
PC : Personal-Computer, angeschlossen an ER-D mit Durchgriff zu ER-Z

Ein physisch separater ER-Z wird für kleine Organisationen nicht wirtschaftlich sein. Dann ist er "logisch" - d.h. abgesichert vom Produktionsbetrieb - auf dem PR einzurichten.

Als ungefährer Anhalt kann gelten, daß ein separater ER-Z ab ca. 40 Entwicklern wirtschaftlich ist. Dies hängt jedoch stark von den Gegebenheiten ab.

Die dezentralen Entwicklungsrechner ER-D (UNIX oder TFS, vgl. 2.3.4) sind an einen Ring (z.B. BUS oder Token Ring) angeschlossen und als Stern an den zentralen Entwicklungsrechner. Der Stern ist eine Dialogleitung. Für die Entwicklung wichtig ist eine hohe Ausfallsicherheit des Ringes.

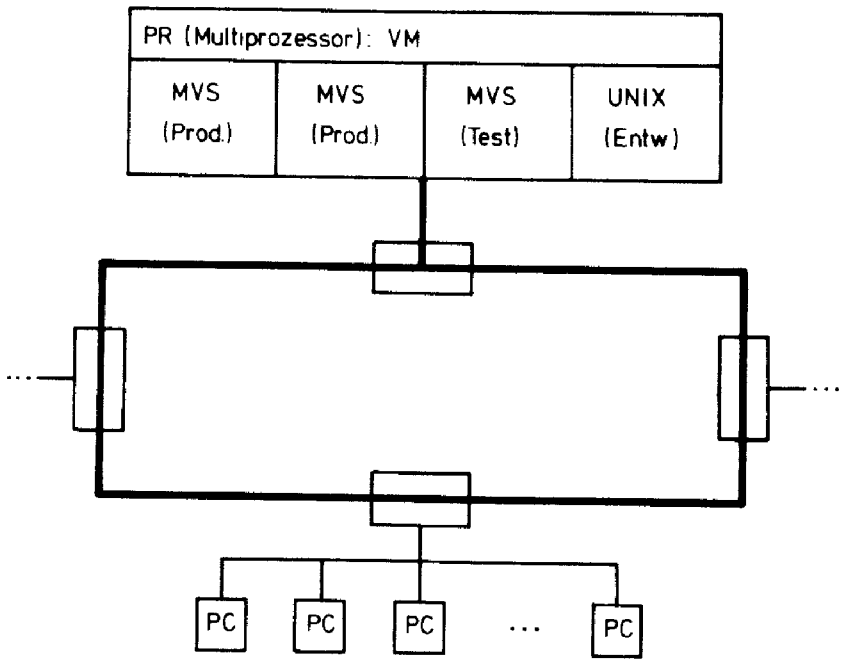
ER-Z und der Produktionsrechner benötigen lediglich eine direkte, ausfallsichere Verbindung in beiden Richtungen. Hier genügt RJE.

Die Nutzerschnittstelle für alle Nutzer liegt physisch an Stationen der ER-D (räumliche Verteilung!).Bei Benutzung von Werkzeugen des ER-Z wird im Dialog-Durchgriff gearbeitet.

Der Ring der ER-D untereinander dient außer dem Mailbox-System und dem Dateitransfer hauptsächlich dazu, jeden Arbeitsplatz (Bildschirm oder PC) an einem ER-D (i) die Arbeit auf jedem Datenbestand jedes beliebiegen ER-D (j) zu erlauben.

Nur diese Topologie stellt ein effizientes Arbeiten ständig neu zusammengesetzter Projektgruppen sicher, ohne daß die Mitarbeiter ständig umziehen, d.h. den Arbeitsplatz wechseln müssen, weil ihre Daten auf einem anderen ER liegen als auf dem bisherigen.

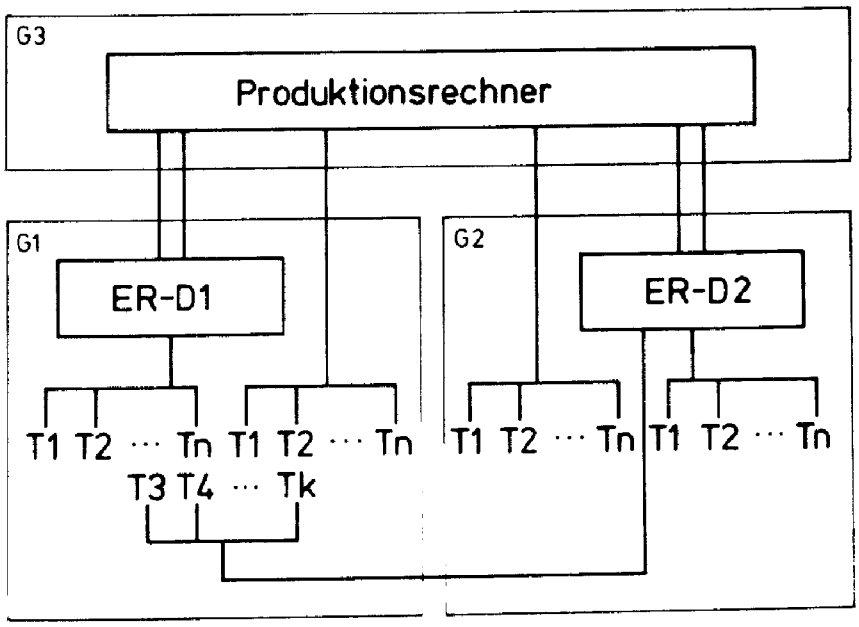
4.2 PC-Ring mit Netzknoten und Zentralrechner



Die dezentralen Entwicklungsrechner sind die PC's selbst. Jeder PC hängt am Ring der Netzknoten und hat über diese auch Durchgriff auf den Zentralrechner. Dieser vereinigt PR und ER-Z in einer Maschine, die ihre verschiedenen "partitions" gegeneinander weitgehend ausfallsicher betreiben kann. Die Ausfallwahrscheinlichkeit des gesamten Prozessorkomplexes wie auch des Ringes der ER-D ist äußerst gering. Privilegierte PC's fungieren als Netzrechner und File server.

5 Praxisbeispiele

5.1. Beispiel Istzustand



Legende: T1, T2,... = Terminal

Beschreibung:

Softwareentwicklung findet in 3 Gebäuden (Gi) statt (Terminals in G3 sind nicht eingezeichnet). In zwei Gebäuden stehen dedizierte Entwicklungsrechner. G3 hat aus technischen Gründen keinen Zugang zu den ER-D.

Die ER-D verfügen über jeweils k Plätze. Sie sind derzeit nicht weiter ausbaubar. Infolge höheren Bedarfs in G1 und geringeren Bedarfs in G2 sind j Terminals in G1 an ER-D2 angeschlossen. Die Entwickler können z.B. nur in G2 drucken.

Es gibt die -allerdings unkomfortable- Möglichkeit, über Batch-Jobs via PR Datenbestände zwischen ER-D auszutauschen.

Die ER-D sind seit einigen Jahren installiert und verfügen über je eine Batch- und eine Dialogleitung zum PR.

Außer auf den ER-D wird auch direkt auf dem Produktionsrechner entwickelt. Die Werkzeuge auf PR und ER-D sind weitgehend unterschiedlich, die Benutzerschnittstelle für die

Entwickler ebenso. Wenn eine Gruppe mit einer "Mischung" von Terminals ausgestattet ist, kann sie SOURCE-Entwicklung nicht mit den Werkzeugen des ER-D durchführen, sondern ist gezwungen, auf dem PR im Dialog-Durchgriff zu arbeiten.

Hinzu kommt ein weiteres SES auf dem PR, eine 4.-Generations-"Sprache". Wenn Programme damit von ER-D entwickelt werden, geschieht dies im Dialog-Durchgriff. ER-D ist dann reiner Dokumentations-Verwalter.

Alle Entwickler-Terminals sind "dumme" Terminals .

Damit beim Leser kein Mißverständnis aufkommt:

Diese an einigen Stellen offensichtlich verbesserungsfähige Konfiguration ist -verglichen mit der großen Masse der Betriebe gleicher Größenordnung- eine sehr fortschrittliche Lösung.
Es gilt die Aussage zum Istzustand aus 2.3.3 !

5.2 Beispiel Sollzustand für ein Großunternehmen

Mengengerüste: ca. 90 EW
ca. 15 PRJ mit diversen Unterprojekten
möglicher Zeithorizont: stufenweise 1 - 3 Jahre.

Sollzustand kann eine der beiden Alternativen 4.1 oder 4.2 sein. Die konkrete Lösung wird wahrscheinlich dazwischen liegen. Entsprechende Skizzen kann sich der Leser aus 5.1 selbst ableiten.

Da es nicht primäre Aufgabe des DV-Bereiches eines Industriebetriebes ist, anspruchsvolle Software - Werkzeuge zu entwickeln, ist ein Sollkonzept mit folgenden Unsicherheiten behaftet:

Wir wissen nicht, ob:

- in 2 Jahren der entsprechende Multiprozessor als Zentralrechner verfügbar ist und ob er dann angeschafft wird,
- das angekündigte Release eines vernetzten Betriebssystems der ER-D den hier gestellten Anforderungen genügt,
- wie schnell sich die PC's bezüglich Leistungsfähigkeit und Verbindung zu vernetzten Systemen entwickeln.

Literatur

BAY 84 Bayer, R.; Elhardt, K.; Kießling, W.; Killer, D.:
Verteilte Datenbanksysteme
(Eine Übersicht über den heutigen Entwicklungsstand), in:
Informatik-Spektrum 7(1984) H. 1, S. 1-19

CW 84 Edition Computerwoche (Hrsg.):
Höhere Programmiersprachen auf Mainframe und PC,
CW-CSE, München 1984

IVIE 77 Ivie, Evan, L.:
The Programmer's Workbench - A Machine for Software Development, in:
CACM 20(1977) H. 10, pp. 746-753

HAU 83 Hausen, H.-L.:
Software-Produktionsumgebungen ("Das aktuelle Schlagwort"), in:
Informatik-Spektrum 6(1983) H. 1, S. 39-40

MAM 84 Mambrey, P.; Oppermann, R.:
Partizipation von Nutzern bei der Software-Entwicklung - Fortschritte und Rückschläge - : 5. Fachgespräch der GI-Fachgruppe Software Engineering Berlin Nov. 1984.
erscheint in: Softwaretechnik Trends.

MAR 83 Marty, R.:
UNIX - Eine Einführung für den professionellen Softwareentwickler, in:
Informatik-Spektrum 6(1983) H. 4, S. 191-204

SPI 84 Spitta, Th.:
Die Projektbibliothek als Zentrum der Softwareentwicklung, in:
Handbuch Moderne Datenverarbeitung (HDM), 21(1984), H. 116, S. 37 - 50

SUR 78 Surböck, E.K.:
Management von EDV-Projekten,
Berlin - New-York 1978

THO 82 Thomas, R.A.; Yates, J.:
A Guide to the UNIX System, New York 1982