Zeitschrift: Bulletin des Schweizerischen Elektrotechnischen Vereins, des Verbandes Schweizerischer Elektrizitätsunternehmen = Bulletin de l'Association suisse des électriciens, de l'Association des entreprises électriques suisses

Herausgeber: Schweizerischer Elektrotechnischer Verein ; Verband Schweizerischer Elektrizitätsunternehmen

Band: 81 (1990)

Heft: 17

Artikel: Lokale und verteilte Echtzeit-Applikationen : Entwurf und Implementation

Autor: Vonlanthen, Claude

DOI: https://doi.org/10.5169/seals-903152

Lokale und verteilte Echtzeit-Applikationen

Entwurf und Implementation

Claude Vonlanthen

Die Entwicklung von Echtzeit-Applikationen und deren Implementation ins vorgegebene Betriebssystem ist eine komplexe Aufgabe, die nur durch den Einsatz effizienter Methoden zu erreichen ist. Bei der Entwicklung des Betriebssystems SB-RTDS wurde eine Methode gefunden, die unabhängig vom Betriebssystem eingesetzt werden kann. Der Beitrag gibt einen Überblick über das SB-RTDS-Betriebssystem und die Soom-Methode.

La mise en œuvre d'une application temps-réel locale ou distribuée est une tâche délicate, qui nécessite l'utilisation d'une méthode de conception efficace. Parallèlement au développement du noyau temps-réel SB-RTDS est née la méthode de conception baptisée Soom. Cette méthode fut conçue pour être utilisée de manière indépendente du noyau. L'article introduit la méthode de conception ainsi que le noyau temps-réel.

Adresse des Autors

Claude Vonlanthen, Dipl. El.-Ing. ETH, Ingenieur-Büro Vonlanthen, Burgweg 34, 4600 Olten, Tel. 062/32 34 42

Echtzeitanwendungen sind komplex. Der Entwurf, die Implementation und der Test sind Aufgaben, die nur dann mit Erfolg gelöst werden können, wenn vom Projektbeginn bis zur Abnahme des fertigen Produktes eine konsequente Methodik verwendet wird. Viele der heute bekannten Methoden decken den Entwicklungsprozess nur bis zum Beginn der Realisierungsphase ab. Diese Lücke wird durch die Kombination der Entwurfsmethode Soom (Service and Object Oriented Method) mit dem Echtzeitbetriebssystem SB-RTDS (Service Based Real Time Distributed System) geschlossen. Soom und SB-RTDS sind Produkte der Firma Hiware AG in Basel.

Im Laufe der Entwicklung des Betriebssystems SB-RTDS kam die Idee auf, dem Kunden auch eine Anleitung zur Entwicklung von Applikationen mit diesem Betriebssystem mitzugeben. Im Rahmen von Anwenderschulungen bewährte sich diese Unterstützung, worauf die Anleitung zur Methode weiterentwickelt wurde. Obwohl ein enger Zusammenhang zwischen Betriebssystem und Methode besteht, könnten beide unabhängig voneinander eingesetzt werden.

Ziele von Soom

Folgenden Punkten wurde bei der Entwicklung von Soom besonderes Augenmerk geschenkt:

- In Echtzeit-Anwendungen kommen Datenflüsse wie Kontrollflüsse vor. Für das Zusammenspiel der Systemteile ist jedoch der Kontrollfluss von besonderer Bedeutung. Soom sollte die Möglichkeit bieten, beide Flüsse zu kombinieren.
- Die Beschreibung soll aus dem Gefühl kommen und einfach sein.
- Einfache Handbarkeit anhand einer Checkliste (Automatisierung)

Die Methode

Das Design und die Realisierung von Echtzeit-Anwendungen werden erst möglich, wenn die Aufgabe eines Gesamtsystems in Teilaufgaben und Teilsysteme aufgeteilt wird. Dabei wird die Aufgabe eines Systems so lange aufgegliedert, bis sie und die mit ihr beschäftigten Subsysteme leicht überschaubar und somit realisierbar sind (Tab. I).

Der fundamentale Unterschied zwischen der Top-Down-Methode Soom und anderen Top-Down-Methoden besteht in der Art und Weise der Aufgliederung. Sie erfolgt bei Soom unter dem Gesichtpunkt der Dienstleistung (Service).

Erste Schritte

Das Aussehen und Verhalten des Systems wird vom Kunden durch ein Pflichtenheft vorgegeben. Solche Systeme sind zum Beispiel Produktionsstrassen, zu steuernde Maschinen und Prozesse. Im Pflichtenheft ist festgelegt, welche Aufgabe das System übernehmen muss. Ebenso sind die Randbedingungen formuliert.

Die Gesamtaufgabe, die vom System erbracht werden muss, wird als Dienstleistung betrachtet analysiert und beschrieben (Bild 1). Die Dienstleistung wird nun in überschaubare,

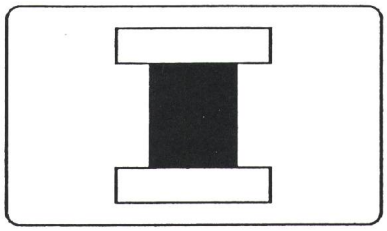


Bild 1 Ein System und seine zu erbringende Dienstleistung

Dienstleistung

System, Subsystem

Checkliste zur Systemzerlegung

Auf jeder Stufe in der Zerlegungshierarchie kann folgende Checkliste angewendet werden:

1. Analyse der Dienstleistung

Fragestellung: Welche Dienstleistung muss erbracht werden?
Ergebnis: (analysierte) Dienstleistung

2. Aufteilung des Subsystems in seine durch die Randbedingungen gegebenen Subsysteme

Fragestellung: Welche nachgeordneten Subsysteme sind vorgegeben?

3. Aufteilung der Dienstleistung in delegierbare Dienstleistungen

Fragestellung: Welche einfacheren Dienstleistungen müssen bekannte und noch festzulegende Subsysteme gesamthaft zur Verfügung stellen, damit die obige Dienstleistung erbracht werden kann?
Ergebnis: Menge von Dienstleistungen (kann auch leere Menge sein, wenn die zu erbringende Dienstleistung nicht weiter unterteilt werden soll)

4. Festlegen der zusätzlich nötigen Subsysteme

Fragestellung: Welche Subsysteme müssen zusätzlich zu den gegebenen (unter 2. gefundenen) noch festgelegt werden?
Ergebnis: Zweite Teilmenge von Subsystemen (kann auch leere Menge sein)

5. Zuordnung der gefundenen Dienstleistungen zu den Subsystemen

Fragestellung: Welche Dienstleistungen werden welchen Subsystemen zugeordnet? (Entfällt, falls die Menge der gefundenen Dienstleistungen unter 3. leer war.)
Ergebnis: Menge von Subsystemen mit zugehörigen Dienstleistungen. Mit diesen kann bei 1. weitergefahren werden.

Tabelle I

einfachere Teile aufgeteilt. Diese Teile sind wiederum Dienstleistungen, die an Subsysteme delegiert werden können. Häufig sind Subsysteme durch das Pflichtenheft schon vorgegeben (Sensoren, Maschinen mit eigenem Rechner usw.). Sie müssen daher als Randbedingungen bei der Zerlegung der Dienstleistung berücksichtigt werden. Sind alle Dienstleistungen gefunden, so werden sie den schon vorgegebenen oder möglicherweise noch festzulegenden Subsystemen zugeteilt. Ihre Ausführung kann von anderen Subsystemen angefordert werden.

Abhängig von der Betrachtungsweise können folgende Begriffe festgelegt werden:

Aus der Sicht von oben:
- Ein System besteht aus einer Menge kooperierender Subsysteme.
- Ein System implementiert eine Menge von Dienstleistungen.

Aus der Sicht von unten:
- Subsysteme sind kooperierende Teile eines Systems.
- Subsysteme erbringen Dienstleistungen und stellen sie anderen Subsystemen zur Verfügung.

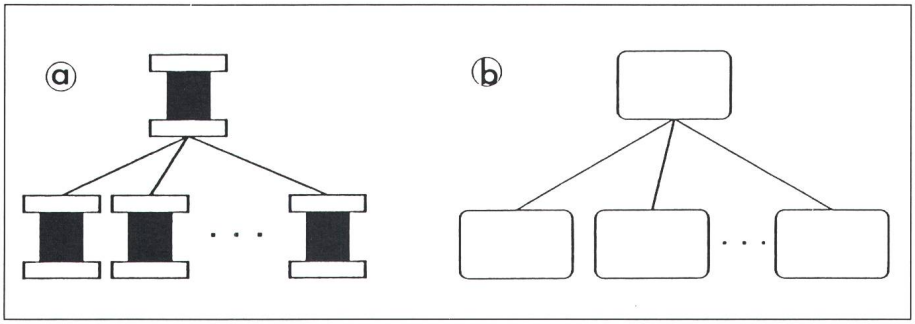


Bild 2 Erste Zerlegung der Dienstleistung und des Systems

a in untergeordnete Dienstleistungen

b in Subsysteme

Im Beispiel der Produktionsstrasse sind Maschinen, Computer, Software, Menschen usw. Subsysteme. Gemeinsam erbringen sie mit ihren individuellen Dienstleistungen die gesamte Dienstleistung «Produkt produzieren».

Durch die Verfeinerung ergibt sich unter der ursprünglichen Dienstleistung eine erste Schicht von Dienstleistungen (Bild 2a) und unter dem System eine erste Schicht von Subsystemen (Bild 2b). Werden die zwei Schichten übereinandergelegt, kann dargestellt werden, wie die Aufgabenteilung unter den Subsystemen erfolgt. Durch die Zerlegung sind Verbindungen entstanden, über welche die Subsysteme ihre gegenseitigen Dienstleistungsanforderungen stellen. Sie können nun als Dienstleistungskanäle eingezeichnet werden (Bild 3).

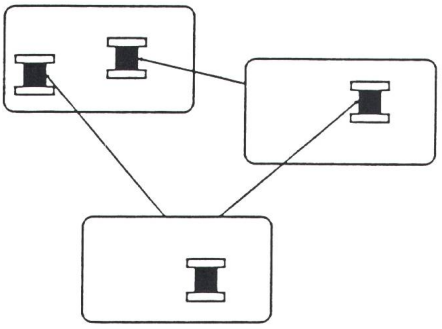


Bild 3 Subsysteme kommunizieren miteinander

Weitere Zerlegung

Nach jedem Zerlegungsschritt werden die resultierenden Teile Dienstleistungen, Dienstleistungskanäle und zugehörige Subsysteme analysiert, beschrieben und eventuell weiter zerlegt. Das Resultat der Analyse ist eine Beschreibung aller Dienstleistungen, welche die einzelnen Subsysteme zur Verfügung stellen, der Anforderungen, welche diese Dienstleistungen auslösen, sowie der jeweils nötigen Dienstleistungskanäle. Weiter müssen noch diejenigen Dienstleistungen vermerkt werden, welche die Subsysteme selbst von anderen Subsystemen benötigen.

Dienstleistungen und Subsysteme lassen sich also durch einen iterativen Prozess in immer einfachere Elemente zerlegen. Parallel und in enger Wechselwirkung zueinander entstehen so ein Dienstleistungsbaum und ein Subsystembaum.

Während dieses Vorgehens trifft man schon bald auf Dienstleistungen und Subsysteme, die unter den gegebenen Randbedingungen nicht weiter unterteilt werden können. Im System «Produktionsstrasse» sind dies die Maschinen, die Hardware und die Menschen, welche Dienstleistungen anfordern können (Terminalbedienung, Maschinenführung, Eingabe von Prozesswerten usw.).

Die weiteren Verfeinerungszyklen befassen sich deshalb immer mehr mit Dienstleistungen von Software-Subsystemen. Erfolgte die Beschreibung eines Subsystems in den oberen Zerlegungsschichten noch recht grob, so wird diese nun immer feiner. Dienstleistungen werden zusehends genauer durch die Beschreibung der damit verbundenen Tätigkeiten und der Dienstleistungskanäle definiert. Diese können neben der Dienstleistungsanforderung implizit auch Daten beinhalten und symbolisieren dann sowohl einen Kontroll- als auch einen Datenfluss.

Am Ende der Zerlegung

Nach einer von der Komplexität des Projektes abhängigen Anzahl Schritte werden Dienstleistungen und Subsysteme gefunden, die aus logischen oder physikalischen Gründen nicht weiter unterteilt werden können. Logische Gründe liegen vor, wenn die mit der Dienstleistung verbundene Aufgabe überschaubar oder unteilbar ist. Physikalische Gründe können durch die Abhängigkeit von Ressourcen (Hardware) oder durch Unteilbarkeit (Mensch, CPU-Karte, Rechnerknoten) gegeben sein. Die Analyse ist beendet, wenn nur noch solche Basis-Subsysteme bestehen. Möglicherweise wird auch eine zu weit gehende Zerlegung erkannt und muss rückgängig gemacht werden. In diesem Fall war das Subsystem der oberen Schicht schon im logischen Sinn unteilbar. Zwischen den Basissubsystemen bestehen Schnittstellen, über die Dienstleistungen angefordert und Daten ausgetauscht werden können. Die Applikation zeigt sich als Menge kooperierender Basis-Subsysteme (Bild 4). Auf dieser Stufe erfolgt jetzt die konkrete Beschreibung der Dienstleistungskanäle, wobei die Datenstrukturen im Innern der Kanäle definiert werden. Je nach der späteren Implementationssprache können bereits deren Möglichkeiten genutzt werden (z.B. Datenstruktur). Es können aber auch Pseudo-Daten-Strukturen verwendet werden.

Detailanalyse und Programmentwurf

Nun kann die Realisierung der Basis-Subsysteme in Angriff genommen werden. Durch die bisherigen Schritte sind ihr äusseres Verhalten, die angebotenen Dienstleistungen und ihre Schnittstellen definiert. Dazu wird jedes Basis-Subsystem einzeln betrachtet. Die Dienstleistungen müssen in seinem Innern realisiert werden, die Aussenwelt ist unwichtig. Die zugehörigen Datenstrukturen definieren seine Schnittstellen; es kann von verschiedenen Projektmitgliedern bearbeitet werden.

Das Basis-Subsystem wird als Menge kooperierender Prozesse (Tasks) entworfen (Bild 5). Diese Prozesse reagieren auf Anforderungen und führen darauf im Team die Dienstleistungen aus.

Die Beschreibung des Innern eines Prozesses, der Prozess-Code, kann nun prozedural mit einer der bekannten Methoden wie Nassi-Shneidermann, Jackson oder in einer Pseudosprache vorgenommen werden. Möglich ist auch die Verwendung der Hochsprache wie Modula 2 im Sinne des Stepwise Refinement.

SB-RTDS und seine Ziele

Bevor die Abbildung der Design-Resultate auf die Strukturen des Betriebssystems SB-RTDS betrachtet wird, sollen dessen Kommunikationsmechanismen kurz vorgestellt werden.

Bei der Entwicklung von SB-RTDS (Service Based Real Time Distributed

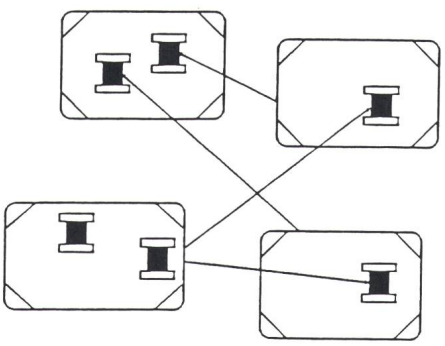


Bild 4 Der letzte Schritt der Zerlegung ist getan – es bestehen nur noch Dienstleistungen, die von Basis-Subsystemen erbracht werden

Zwischen den Subsystemen bestehen Dienstleistungskanäle, über welche die Dienstleistungsanforderungen gestellt werden.

Basis-Subsystem

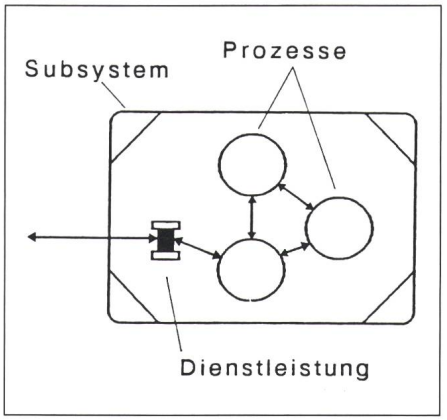


Bild 5 Kooperierende Prozesse bilden das Innere eines Basis-Subsystems. Sie übernehmen die Entgegennahme der Dienstleistungsanforderungen und erledigen die zugehörigen Tätigkeiten

System) standen folgende Ziele im Vordergrund:

- Zugriff aus der Hochsprache Modula 2 heraus. (Ein C-Interface ist in Kürze erhältlich).
- Unterstützung der Applikations-Entwicklung durch Strukturierungsmittel.
- Unterstützung für die hardwarenahen Software-Teile wie Device Driver.
- Effizienter Kern, d.h. schnelle Prozess-Umschaltzeiten.

SB-RTDS-Mechanismen

SB-RTDS stellt für die Kommunikation zwischen Prozessen zwei verschiedene Mechanismen, einen synchronen und einen asynchronen, zur Verfügung. Beiden gemeinsam ist, dass die Adressierung nicht durch eine Prozess-Identifikation, sondern über Dienstleistungsnamen erfolgt.

Bei der *synchronen Kommunikation* handelt es sich um ein Rendez-vous by Name. Der Senderprozess, der die Initiative ergreift (Initiator) wird so lange blockiert, bis der Empfänger (Server) ihn wieder freigibt. Der Sender kann seinem Aufruf Daten beifügen. Die Datenübergabe erfolgt dabei über eine *Region*, die vom Initiator-Prozess vorgängig kreiert wird. Der Server seinerseits hat die Möglichkeit, eine Antwort *Response* zurückzugeben. Während des Rendez-vous hat nur der Server Zugriff auf die Daten (Mutual Exclusion). Auf diese Weise lässt sich ein Verwalter eines Datenpools realisieren (Administrator).

In einem Rendez-vous ist der Aufbau weiterer Rendez-vous mit anderen

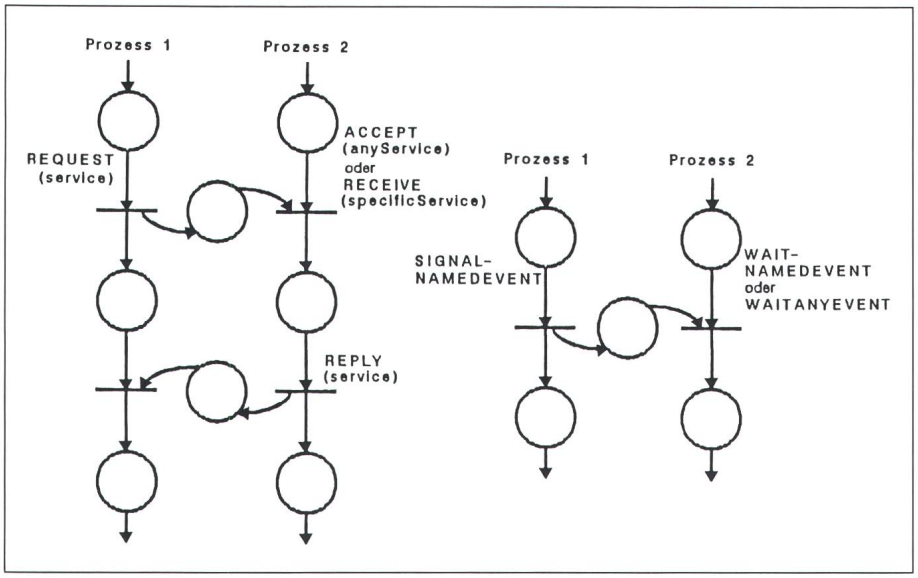


Bild 6 Synchrone und asynchrone Kommunikationsmechanismen des Betriebssystems
Die Petri-Netze zeigen die direkte Wechselwirkung zwischen den Prozessen bei Benutzung der Systemaufrufe.

Prozessen möglich. Die Beendigung der Rendez-vous kann in beliebiger Reihenfolge geschehen. Damit kann der Server den Ablauf anderer Prozesse steuern (Koordinator).

Die *asynchrone Kommunikation* kann als Ereignismeldung betrachtet werden. Mit ihr ist keine Datenübergabe möglich. Zwischen den Prozessen fliesst nur ein Kontrollfluss. Der Initiator-Prozess wird nicht blockiert, auch nicht wenn der Server-Prozess im Augenblick nicht in der Lage ist, die Ereignismeldung entgegenzunehmen. Die Meldung wird jedoch gespeichert. Der Server-Prozess kann auf die Meldung eines speziellen oder beliebigen Ereignisses warten.

Die erläuterten Kommunikations-Mechanismen lassen sich in einem Petri-Netz darstellen (Bild 6).

Strukturen und Möglichkeiten mit SB-RTDS

Die in der Methode Soom gefundenen Begriffe und Zusammenhänge werden im folgenden auf die Ebene des Betriebssystems SB-RTDS übertragen.

Soom-Objekte in SB-RTDS

Basis-Subsystem: In SB-RTDS-Terminologie wird ein Basis-Subsystem *Team* genannt. Ein Team ist eine Menge kooperierender Prozesse. Prozesse nehmen Dienstleistungsanforderungen entgegen und verrichten die damit verbundenen Tätigkeiten im Team. Als Basis-Subsystem ist ein Team unteilbar und wird deshalb auf einem Rechnerknoten als Ganzes implementiert.

Prozess: Prozesse sind die ausführenden Elemente eines Teams. Sie enthalten den Code und laufen innerhalb des Teams quasi-parallel ab, da ein Team nicht auf verschiedene Rechnerknoten aufgeteilt werden kann. Echte Parallelität ist zwischen Prozessen von Teams vorhanden, die sich auf verschiedenen Knoten befinden.

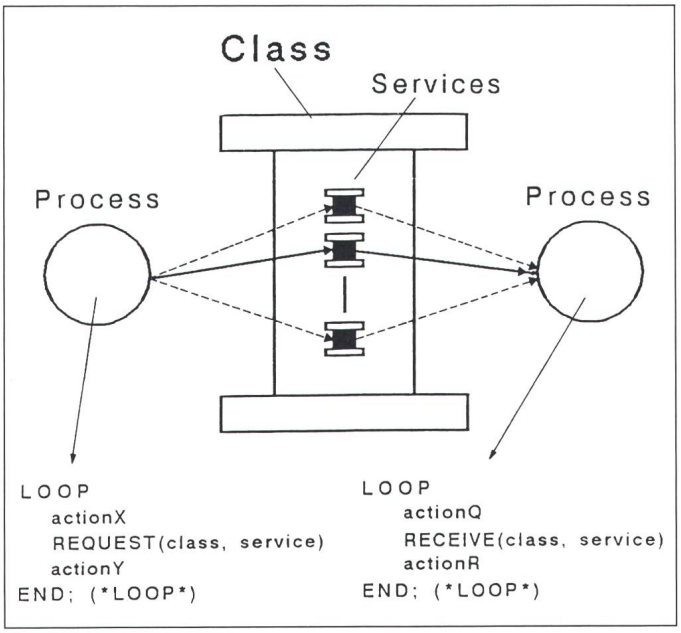
In beiden Fällen sind die Prozesse miteinander durch die beschriebenen Betriebssystem-Mechanismen verbunden. Sie stellen Dienstleistungsanforderungen (Request) oder nehmen sie entgegen (Accept, Receive) ohne jedoch den Empfänger bzw. den Sender zu kennen. Die Verbindungen werden über die Dienstleistungsnamen vom Betriebssystem aufgebaut.

Dienstleistung: Die Dienstleistung ist ein fundamentales Design-Element von Soom. Mit SB-RTDS ist sie als Service verfügbar. Ihre Ausführung kann nur über ihren Servicenamen verlangt werden (s. weiter unten). Mit dem synchronen System-Call Request (...,service,...) wird der Dienstleistungskanal geöffnet und die Dienstleistungsanforderung gestellt. Der asynchrone Aufruf Signalnamedevent(...service...) kann dazu verwendet werden, eine Ereignismeldung abzusetzen und dadurch auch eine Dienstleistung erbringen zu lassen.

Klasse: Mehrere Services, die vom selben Team erbracht werden, lassen sich zu einer Klasse zusammenfassen. Die Class bildet so eine Menge von Services. Die Anforderungen zur Erbringung dieser Services können dann von demselben Prozess im Team entgegengenommen werden. Dies ergibt eine zusätzliche Strukturierungsmöglichkeit und vermindert zugleich die Anzahl der Dienstleistungskanäle. Bild 7 illustriert die Benutzung der Servicenamen und Klassen.

Familie: Die Dienstleistungsfamilie (Family) ist eine Menge von Klassen (Class). Sie kann mehrere Mitglieder (FamilyMember) besitzen. Diese Familienmitglieder bieten alle die gleiche Menge von Dienstleistungen an; sind

Bild 7
Die Zusammenfassung von mehreren Dienstleistungen (Services) zu einer Klasse (Class) schematisch oben und als Code in Prozessen
Die Teams sind hier nicht eingezeichnet.



Family	Class	Service
FileSystem	SaveRestoreOps	Save Restore
	ReadWriteOps	Read Write
	Filemanagement	Open Close

Tabelle II

also Inkarnationen derselben Dienstleistungsfamilie. Auf diese Weise lassen sich gleiche Dienstleistungen, die an verschiedenen Orten von verschiedenen Teams erbracht werden, unterscheiden (z.B. Filesystem auf verschiedenen Knoten).

Eine einzelne Dienstleistung ist durch Familienmitglied, Klasse und ihren Namen eindeutig bestimmt. Family und Class können als Strukturierungsmittel betrachtet werden.

Als Beispiel zu den Begriffen Family (Member), Class und Service diene das Filesystem in Tabelle II.

Ein solches Filesystem werde nun auf zwei Knoten angeboten. Die Dienstleistungen sind dann auf beiden Knoten identisch und bilden zusammengefasst jeweils ein Mitglied derselben Dienstleistungsfamilie. Das konsumierende Team gibt an, von welchem Familienmitglied die Dienstleistung erbracht werden soll. Beispielsweise sind FileSystem1 und FileSystem2 zwei Mitglieder der Dienstleistungsfamilie FileSystem.

Interaktionsnamen

Teams stellen sich gegenseitig Dienstleistungen zur Verfügung. Will nun ein Team den Service eines anderen Teams beanspruchen, muss es nur den Servicenamen kennen. Über diesen erfolgen alle Interaktionen zum Team und seinen Prozessen, welche selber alle anonym bleiben. Die Namen der Dienstleistungen und die innere Form der Dienstleistungskanäle (Datenstrukturen) müssen deshalb vom erbringenden Team bekanntgegeben werden. Dies geschieht mittels zweier Libraries in Form von Modula-2-Modulen. Zusätzlich kann der Zugriff auf eine Dienstleistung in einer Access-Library zur Verfügung gestellt werden (Bild 8). Mit diesen Libraries ist eine saubere Schnittstelle zum Innern des Teams vorhanden. Damit ist auch auf der Systemebene das Prinzip der Verkapselung (Information Hiding) angewendet.

Prozessunabhängigkeit

Für die Anforderung einer Dienstleistung muss also nur deren Name (Interaktionsname) bekannt sein; es werden kein Team oder gar dessen Teile – die Prozesse – adressiert. Das auffordernde Team muss keine Kenntnis über Ort und Art der Dienstleistungserbringung besitzen. SB-RTDS übernimmt den Verbindungsaufbau. Umgekehrt muss aus der Sicht des Teams und seiner Prozesse nicht bekannt sein, von wem und woher die Dienstleistungsanforderung stammt, nur dass die Anforderung erfolgt ist und die Dienstleistungen damit erbracht werden müssen. Die Kommunikation ist also prozessunabhängig.

Lokale und verteilte Applikationen

Dass keine Adressaten vorkommen, bedeutet weiter, dass eine angeforderte Dienstleistung auf einem anderen Knoten erbracht werden kann als demjenigen, auf dem sich das auffordernde Team befindet. Auch dies muss vom konsumierenden Team nicht beachtet werden (Bild 9). Mit SB-RTDS ist auch bis zum letzten Moment offen, wie die Verteilung auf die Knoten erfolgt.

Werden auf einem Knoten die Dienstleistungen eines anderen Knotens konsumiert, müssen lediglich die beschriebenen Team-Libraries kopiert werden. Eine Applikation lässt sich damit *knotenunabhängig* entwickeln. Die Teams werden erst am Schluss auf die einzelnen Rechnerknoten verteilt.

Prozess- und Knotenunabhängigkeit haben zur Folge, dass beim Entwurf der inneren Teamstruktur die Umgebung des Teams nicht mehr zu interessieren braucht. Der Entwickler eines Teams muss sich keine Gedanken darüber machen, ob ein oder mehrere andere Teams die Dienstleistungen anfordern und ob diese Teams sich auf demselben Knoten befinden. Es muss nur die Schnittstelle mit Beschreibung der Dienstleistungsnamen und den zugehörigen Datenstrukturen zur Verfügung gestellt werden.

Die Entwicklung und Realisierung der einzelnen Teams kann von mehreren Projektmitgliedern übernommen

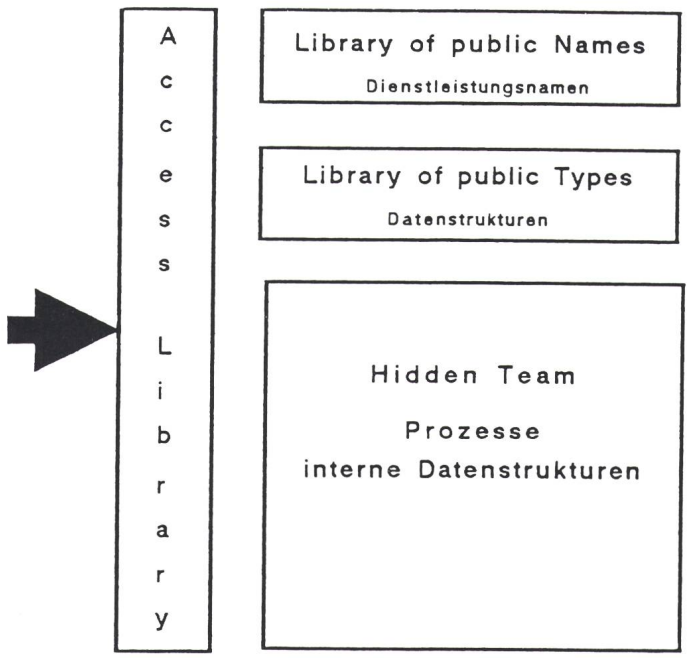


Bild 8
Information Hiding: Vom Team sind nur die Dienstleistungsnamen und die zugehörigen Strukturen bekannt; das Innere des Teams bleibt verborgen. In der Access-Library können die Zugriffe in Prozeduren verpackt zur Verfügung gestellt werden.

werden. Das Verkapselungsprinzip (Information Hiding) unterstützt die getrennte Implementation einer komplexen Applikation, da eine Aufteilung mit sauber definierten Schnittstellen möglich ist. Die Denkweise in verteilten Systemen führt auch bei kleinen Applikationen, die nie auf verschiedenen Rechnerknoten zu laufen kommen, zum selben Resultat: Die Software wird leichter wartbar.

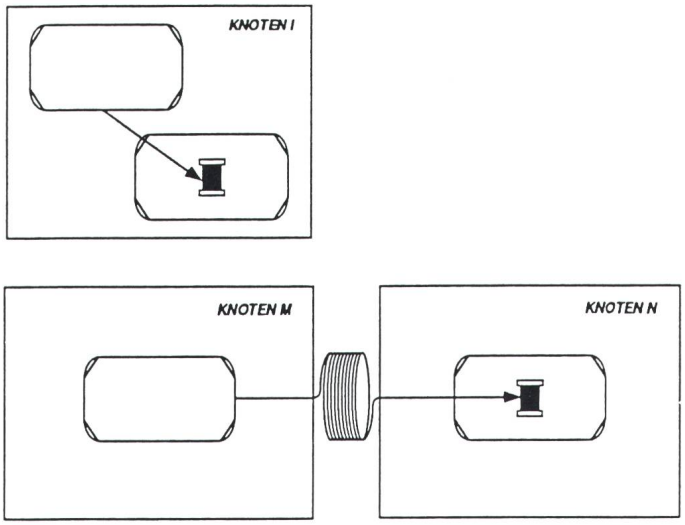
Entwicklungsumgebung und Software-Pakete

Zum Entwickeln von Applikationen (auch ohne SB-RTDS) steht ein komplettes Werkzeug für verschiedene Entwicklungs- und Zielsysteme zur Verfügung (MCDS Modula 2 Cross Development System). In der Testphase kommt der Source Level Debugger zum Einsatz. Mit dessen Hilfe können die Daten von Teams, Dienstleistungskanälen usw. manipuliert werden. Wichtig ist auch die Möglichkeit, auf die Prozesszustände zugreifen zu können.

Als Standard-Software unter SB-RTDS sind verschiedene Pakete erhältlich:

- Filesystem
- Windowsystem

Bild 9
Knotenunabhängigkeit. Zwei Teams einmal auf demselben Knoten (oben) und auf zwei verschiedenen Knoten (unten).



- Maskengenerator zum Windowsystem
- Kommunikations-SW für ISO-OSI-Protokolle auf Ethernet
- Verschiedene Treiber-Software zu I/O-Hardware

Schlussfolgerung

Die vorgestellte Methode ermöglicht den effizienten Entwurf von Real-Time-Applikationen. Insbesondere eignet sie sich für den Entwurf von verteilten Systemen. Das Prinzip der Aufteilung in Dienstleistungen und Basis-Subsysteme und des damit verbundenen Information Hiding verhilft schon in einer frühen Phase zu einem effizienten Einsatz der Projektmitglieder. Als Nebenprodukt entsteht dabei leicht zu wartende Software.

Obwohl die Methode die Implementierung mit allen gängigen Echtzeit-Betriebssystemen erlaubt, können bei Verwendung von SB-RTDS die Objekte schon auf dem Niveau der Basis-Subsysteme 1:1 auf das Betriebssystem abgebildet werden.