—- Modelle—-

Computing Modell

Michael Felke Martina Mostert

Revision 2010-03-16

In Zusammenarbeit mit

Marc Horst, Christian Janßen

Weiterhin beigetragen haben:

Oliver Höfken

Lizenz

Verwendung des Modells

Der Inhalt dieses Dokuments führt ein owl's-Modell gemäss den nachstehenden Richtlinien, Konditionen und Hinweisen aus. Dieses Dokument stellt keine Verpflichtung dar irgendeinen Teil dieser Spezifikationen in irgendein Produkt einer Firma zu implementieren. An den in diesem Dokument enthaltenen Informationen sind Änderungen vorbehalten.

Lizenzbedingungen

In Abhängigkeit mit allen nachstehenden Richtlinien und Bedingungen gewährt Ihnen das owl's-Team als Inhaber der Urheberrechte an diesem owls's-Modell hiermit eine unentgeltliche, nicht exklusive, nicht übertragbare, unbefristete, weltweite Lizenz (ohne das Recht Unterlizenzen zu erteilen), dieses owl's-Modell zu nutzen, um Software und spezialisierte Modelle zu erstellen und vertreiben, die auf diesem owl's-Modell basieren, sowie dieses owl's-Modell zu nutzen, kopieren und zu verteilen gemäss dem Urheberrechtsgesetz; vorrausgesetzt dass:
(1) beide, der Urheberrechtshinweis wie oben kenntlich gemacht und dieser Lizenztext auf allen Kopien dieses owl's-Modells erscheinen;
(2) dieses owl's-Modell Informationszwecken dient und nicht zu kommerziellen Zwecken über einen Netzwerk-Rechner kopiert oder verbreitet oder über irgendein anderes Medium gesendet oder anderweitig weiterveräussert oder weitergegeben wird; und
(3) keine Änderungen an diesem owl's-Modell vorgenommen werden.
Diese eingeschränkte Lizensierung endet fristlos, falls gegen irgendeine dieser Richtlinien oder Bedingungen verstossen wird. Nach Beendigung sind unverzüglich alle im Besitz befindlichen Kopien dieses owl's-Modells zu vernichten.

Patente

Das Augenmerk von Anwendern wird auf die Möglichkeit gelenkt, dass die Einhaltung oder Übernahme von owl's-Modellen die Verwendung von urheberrechtlich geschützten Erfindungen erfordern kann. Das owl's-Team ist nicht verantwortlich dafür, Patente zu kennzeichnen, für die bei Beachtung eines owl's-Modells möglicherweise die Pflicht zur Lizensierung besteht, oder die Einziehung von Erkundigungen über die Rechtsgültigkeit oder den Geltungsbereich von solchen Patenten, die ihm zur Kenntnis gebracht wurden, durchzuführen. owl's-Modelle stellen lediglich eine Richtline bzw. technische Norm dar. Interessierte Anwender sind selbst dafür verantwortlich sich gegen Haftung bei Patentverletzungen zu schützen.

Allgemeine Verwendungseinschränkungen

Jede unberechtigte Verwendung dieses owl's-Modells kann Urheberrechte, Markenrechte, Bestimmungen und Satzungen des Nachrichtenwesens verletzen. Dieses Dokument enthält Informationen die urheberrechtlich geschützt sind. Alle Rechte vorbehalten. Kein in diesem Werk enthaltener, durch das Urheberrecht geschützter Teil darf nachgemacht oder in irgendeiner Form oder auf irgendeine Art verwendet werden - grafisch, elektronisch oder mechanisch, einschlieslich fotokopieren, aufzeichenen, aufnehmen oder Informationsaufzeichnungssysteme und Informationswiedergabesystemen - ohne die Zustimmung des Urheberrechte Inhabers.

Haftungsausschluss

Obwohl davon ausgegangen wird, dass diese Veröffentlichung fehlerfrei ist, wird sie ohne Mängelgewähr zur Verfügung gestellt und kann daher Fehler oder Druckfehler enthalten. Das owl's-Team gewährt keinerlei Garantie in irgendeiner Form, ausdrücklich oder stillschweigend, bezüglich dieser Veröffentlichung, einschlieslich, aber nicht beschränkt auf, jegliche Gewährleistung wegen Rechtsmängel oder Eigentum, einschliesslich Zusicherung allgemeiner Gebrauchstauglichkeit oder Gewährleistung der Eignung für bestimmte Zwecke oder Nutzung. In keinem Fall ist das owl's-Team haftbar zu machen für hierin enthaltene Fehler oder für direkte, indirekte, zufällige, konkrete, Folge-, Vertrauens- oder Sachschäden, einschliesslich entgangener Gewinne, Einnahmen, Daten oder Nutzen, zugezogen durch jeden Benuzter oder Dritte in Zusammenhang mit der Einrichtung, Durchführung oder Nutzung dieses Materials, selbst wenn auf die Möglichkeit solcher Schäden hingewiesen wurde. Das gesamte Risiko bezüglich Qualität und Leistungsfähigkeit der mit dieser Spezifikation entwickelten Software wird von Ihnen getragen. Dieser Haftungsausschluss stellt einen wesentlichen Teil der gewährten Lizenz zur Nutzung dieses owl's-Modells dar.

Markenzeichen

Alle Produkt- und Firmennamen dienen lediglich Bezeichnungs-Zwecken und können Markenzeichen ihrer jeweiligen Inhaber sein.

Modellkonformität

Das owl's-Team legt fest, dass es die einzige Instanz ist und sein wird, die Entwickler, Lieferanten und Verkäufer von Computer Software dazu berechtigen kann, Zertifizierungsmarken, Markenzeichen oder andere spezielle Bezeichnungen zu verwenden, um die Übereinstimmung mit diesen Materialien zu kennzeichnen. Für Software, die in Übereinstimmung mit dieser Lizenz entwickelt wurde, kann dann und nur dann Konformität mit diesem owl's-Modell erklärt werden, wenn die Konformität der Software darin besteht, dass sie vollständig in allen Punkten mit den Konformitätsanforderungen übereinstimmt, so wie diese in dem owl's-Modell festgelegt sind. Für Software, die nur teilweise die angegebenen Konformitätsanforderungen einhält, kann lediglich geltend gemacht

werden, dass die Software auf diesen owl's-Modellen basiert, kann jedoch nicht die Konformität mit diesen owl's-Modellen erklären. Im Falle, dass Test-Suits für dieses owl's-Modell vom owl's-Team implementiert oder anerkannt werden, kann für eine Software, die dieses owl's-Modell benutzt, nur dann die Konformität mit diesen owl's-Modellen erklären, wenn die Software die Test-Suits einwandfrei durchläuft.

Problemmeldung

Alle owl's-Modelle werden fortlaufend überprüft und verbessert. Als Teil dieses Prozesses möchten wir die Leser dazu auffordern, evtl. gefundene Zweideutigkeiten, Widersprüche und Ungenauigkeiten an die Adresse mail@owl-s.org zu schicken.

Inhaltsverzeichnis

Abbildungsverzeichnis

1 Einführung

Im Folgenden wird das Modell eines Computersystems aus der Sicht eines sich in Ausführung befindenden Maschinenprogramms beschrieben. Es wird jedoch nicht der physikalische Aufbau eines Computersystems erklärt. D.h. die aufgeführten Komponenten entsprechen nicht realen Komponenten eines Computersystems, sie stellen vielmehr eine logische Zergliederung dar, wie sie für die Ausführung und Entwicklung von Programmen von Bedeutung ist. Es handelt sich hier also um einen imaginäres Computersystem aber keinesfalls um eine virtuelle Maschine.
Es ist ein Modell der Umgebung, in der sich das Maschinenprogramm während der Ausführung befindet. Somit stellt es kein vollständiges Modell für alle auf einem Computer oder in einer verteilten Computerumgebung (DCE) ablaufenden Programme dar. Viele Betriebssysteme sind aus Sicherheitsgründen in abgeschottete Bereiche unterteilt. Jeder dieser Bereiche stellt eine eigenständige Umgebung für Programme bereit und wird gemäss diesem Computing-Modell separat behandelt. Gleiches gilt für Programme, die im Rahmen eines grösseren Programmpakets (z.B. Notes, Oracel) ausgeführt werden.

1.1 Aufbau des Modells

Das Modell gliedert sich in die drei Bereiche Software, Run-Time und Executory, welche auf drei Ebenen betrachtet werden. Im Zentrum des Modells steht das sich in Ausführung befindliche Maschinenprogramm - der Prozess!

	Software	*Run-Time*	*Executory*
obere Ebene	**Programmsammlung** (engl. software collection) Menge aller zur Verfügung stehenden Programme innerhalb einer Ausführungssphäre.	**Ausführungssphäre** (engl. execution sphere) Bereich, dem ein Prozess angehört und mit dem er während der Ausführung interagiert.	**Computeraggregat** (engl. computer aggregate) Gesamtheit aller internen und externen Geräte eines Computersystems oder Computersystemnetzes
mittlere Ebene	**Maschinenprogramm** (engl. program) Nach den Regeln einer Programmiersprache erstellte ausführbare Einheit aus Anweisungen und Vereinbarungen.	**Prozess** In Ausführung befindliches Programm inkl. Reservierungen auf Ressourcen, die von diesem Programm benötigt werden.	**Maschine** (engl. machine) Gerät zur Ausführung von Programmen.
untere Ebene	**Prozedur** (engl. procedure) Reihe von Anweisungen und Vereinbarungen, die eine funktionale Einheit bilden	**Thread** Prozedur, die sich bei einem Prozessor in Ausführung befindet.	**Prozessor** (engl. processor) Einheit, die Programmcode einliest, die Anweisungen auswertet und die daraus resultierenden Operationen ausführt.

1.1.1 Beziehungen zwischen den Modell-Komponenten

Beziehungen der Komponenten einer Ebene

untere Ebene Prozedur – Thread – Prozessor

- Eine Prozedur kann von mehreren Threads benutzt werden, die ggf. jeweils unterschiedliche Teile ausführen.
- Einem Thread ist eine Prozedur zugeordnet, welche wechseln kann, wobei die Speicherbereiche bis zum Ende des gesamten Threads erhalten bleiben.
- Ein Prozessor kann mehrere Threads haben und ggf. auch gleichzeitig ausführen.
- Einem Thread ist ein Prozessor zugeordnet.

mittlere Ebene Maschinenprogramm – Prozess – Maschine

- Ein Maschinenprogramm kann für mehrere Prozesse verwendet werden.
- Einem Prozess ist ein Maschinenprogramm zugeordnet, welches wechseln kann (siehe UNIX-Systeme).
- Eine Maschine kann von mehreren Prozessen verwendet werden.
- Einem Prozess ist ein Maschine zugeordnet.

obere Ebene Programmsammlung – Ausführungssphäre – Computeraggregat

- Eine Programmsammlung kann in mehreren Ausführungssphären verwendet werden.
- Einer Ausführungssphäre ist eine Programmsammlung zugeordnet, deren Zusammensetzung sich ändern kann.
- Ein Computeraggregat kann von mehreren Ausführungssphären verwendet werden.
- Einer Ausführungssphäre ist ein Computeraggregat zugeordnet.

Beziehungen der Komponenten eines Bereichs

Bereich Software Programmsammlung – Maschinenprogramm – Prozedur

- Eine Programmsammlung enthält ein bis mehrere Maschinenprogramme.
- Ein Maschinenprogramm kann in mehreren Programmsammlungen enthalten sein.
- Ein Maschinenprogramm enthält einen bis mehrere Prozeduren.
- Eine Prozedur kann in mehreren Maschinenprogrammen enthalten sein.

Bereich Run-Time Ausführungssphäre – Prozess – Thread

- Eine Ausführungssphäre enthält einen bis mehrere Prozesse.
- Ein Prozess ist nur in einer Ausführungssphäre enthalten.
- Ein Prozess enthält einen bis mehrere Threads
- Ein Thread ist nur in einem Prozess enthalten.

Bereich Executory Computeraggregat – Maschine – Prozessor

- Ein Computeraggregat enthält eine bis mehrere Maschinen.
- Eine Maschine ist nur in einem Computeraggregat enthalten.
- Eine Maschine enthält ein bis mehrere Prozessoren.
- Ein Prozessor ist nur in einer Maschine enthalten.

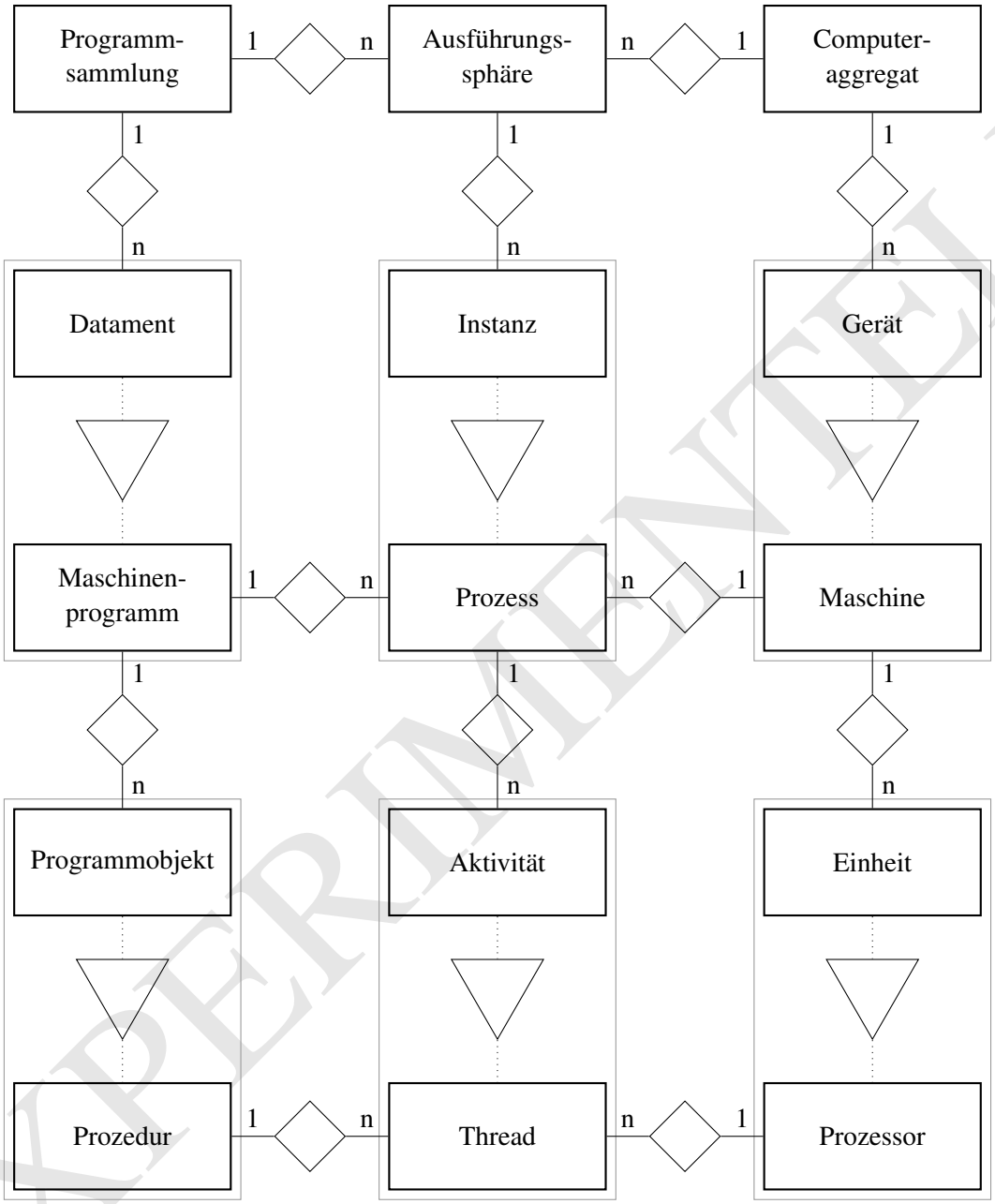


Abbildung 1.1: Beziehungen zwischen den Modell-Komponenten

2 Bereich Executory

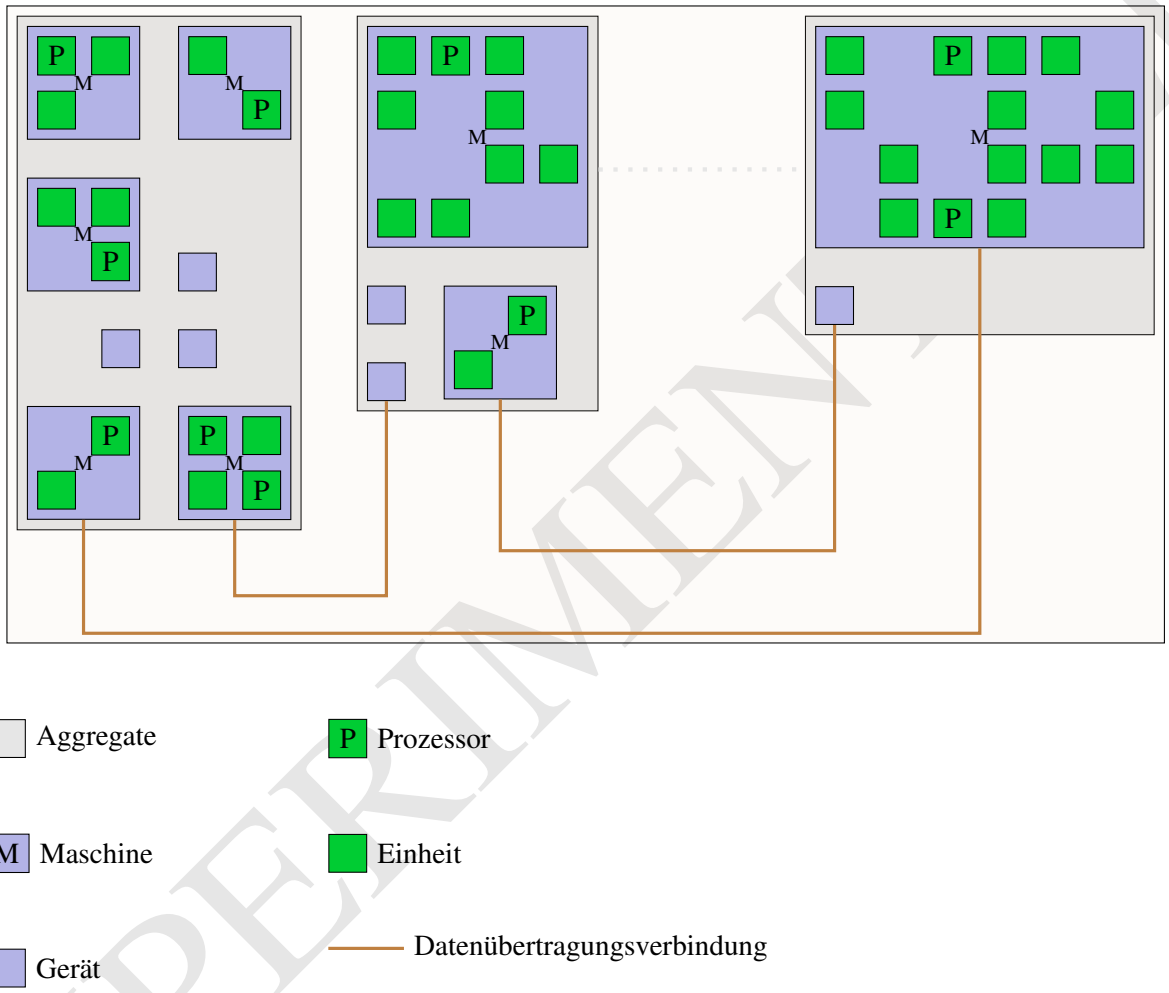


Abbildung 2.1: Executory

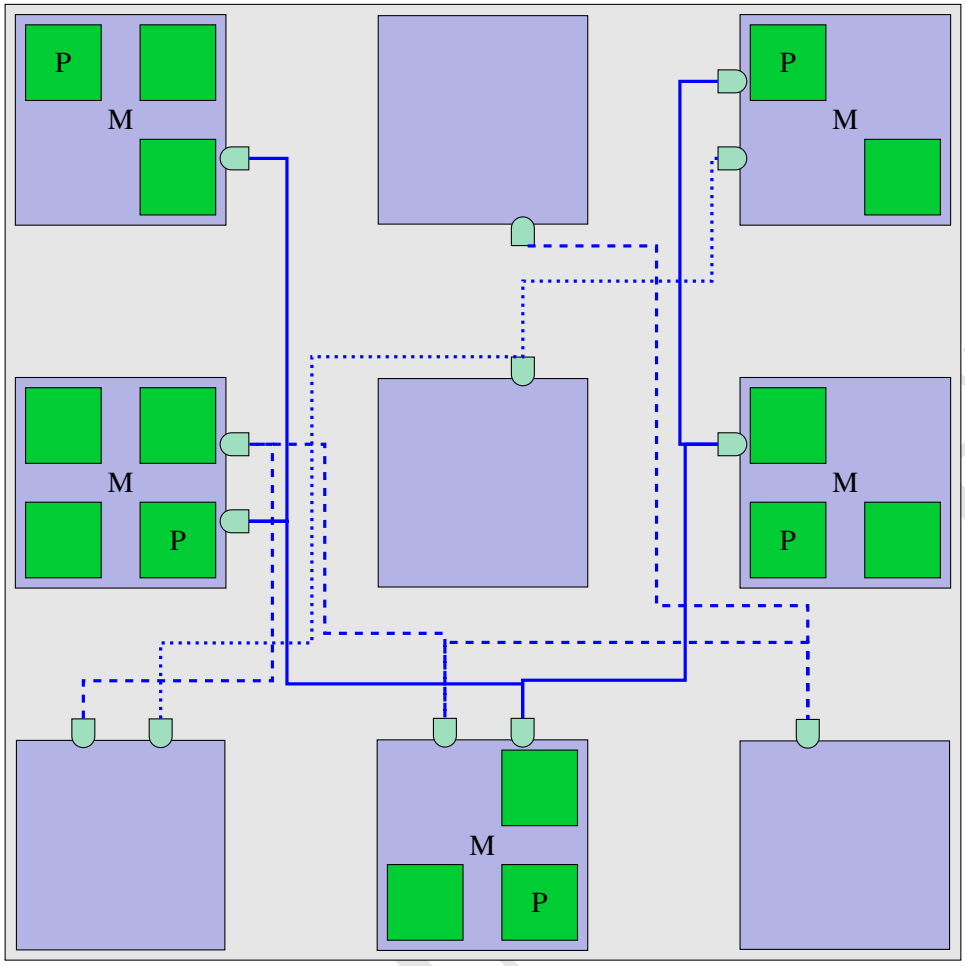
Das Executory besteht aus einem oder mehreren Computeraggregaten, welche über Datenübertragungs-Einheiten der Computeraggregate miteinander verbunden sind.

Jedes Computeraggregat enthält mindestens eine Maschine zur Ausführung von Programmen. [1] Die Maschine enthält neben mindestens einem Prozessor, zur Abarbeitung des Programmcodes auch weitere Einheiten, die über Kanäle miteinander verbunden sind.

2.1 Das Computeraggregat

Ein Computeraggregat bezeichnet die zu einer Gesamtkonfiguration zusammengefassten internen und externen Geräte eines Computersystems bzw. Computersystemnetzes. Neben einer oder mehreren universellen Maschinen beinhaltet das Computeraggregat auch “periphere” Geräte zur Datenerfassung, -speicherung, -ausgabe und -übertragung. Diese Geräte können ebenfalls spezialisierte Maschinen sein bzw. benötigen. Alle in einem Computeraggregat zusammengefassten Geräte und deren Einheiten besitzen eindeutige System-IDs.

[1] (Der Begriff Maschine wird verwendet, da der Begriff Computer bei vielen auch externe Speicher und periphere Geräte mit einschliesst.



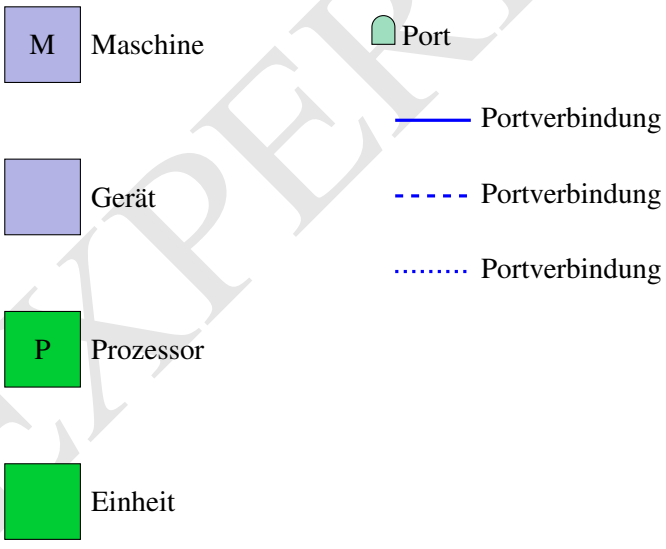


Abbildung 2.2: Computeraggregat

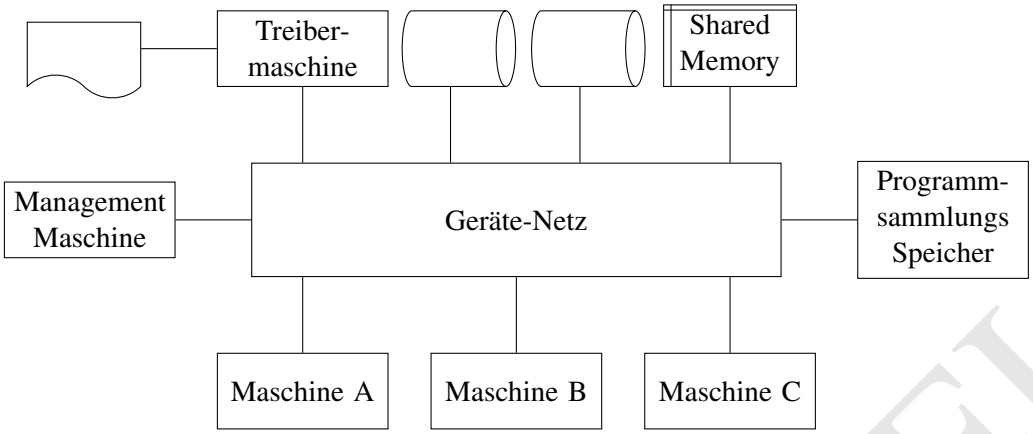


Abbildung 2.3: Computeraggregat

Die Darstellung von Datenarten bzw. die Abbildung auf Speicherzellen ist nur innerhalb einer Maschine, nicht aber innerhalb eines Computeraggregats gleich. Eine ggf. notwendige Umwandlung der Darstellung zum Austausch der Datenarten innerhalb eines Computeraggregats wird von den jeweiligen Ports wahrgenommen. Der Wertebereich aller nicht nur maschinenweit gültigen Datenarten jedoch ist innerhalb des Computeraggregats gleich.
Die transistente Datenart-Identifikation ist innerhalb eines Computeraggregats eindeutig. Desweiteren ist der Wertebereich der maschinen-optimierte Datenart Counter (für allgemeine Zählschleifen und Nummerierungen) abhängig vom jeweiligen Computeraggregat.

2.1.1 Ports

Ports sind standardisierte Schnittstellen zur Kommunikation und Datenübertragung zwischen Instanzen. Für Standardbussysteme zur Kommunikation zwischen Geräten (z.B. SCSI, IDE, USB) werden spezielle Ports zur Verfügung gestellt, diese erhalten Spezifikationen, welche es erlauben, die Funktionalitäten der Standardbusse hardwareunabhängig zu nutzen.

2.2 Die Maschine und andere Geräte

Eine Maschine ist ein Gerät zur Ausführung von Programmen. Sie besteht aus mehreren Einheiten, welche über Kanäle miteinander verbunden sind, und durch eine Executor-Einheit über den Admin-Kanal verwaltet, gesteuert und überwacht werden. Externe Prozeduraufrufe werden über eine Prozeduraufruf-Einheit zur Verfügung gestellt. Über diese werden auch externe Programmstarts abgewickelt. Ohne Prozeduraufruf-Einheit können nur Prozesse innerhalb der Maschine angelegt werden.

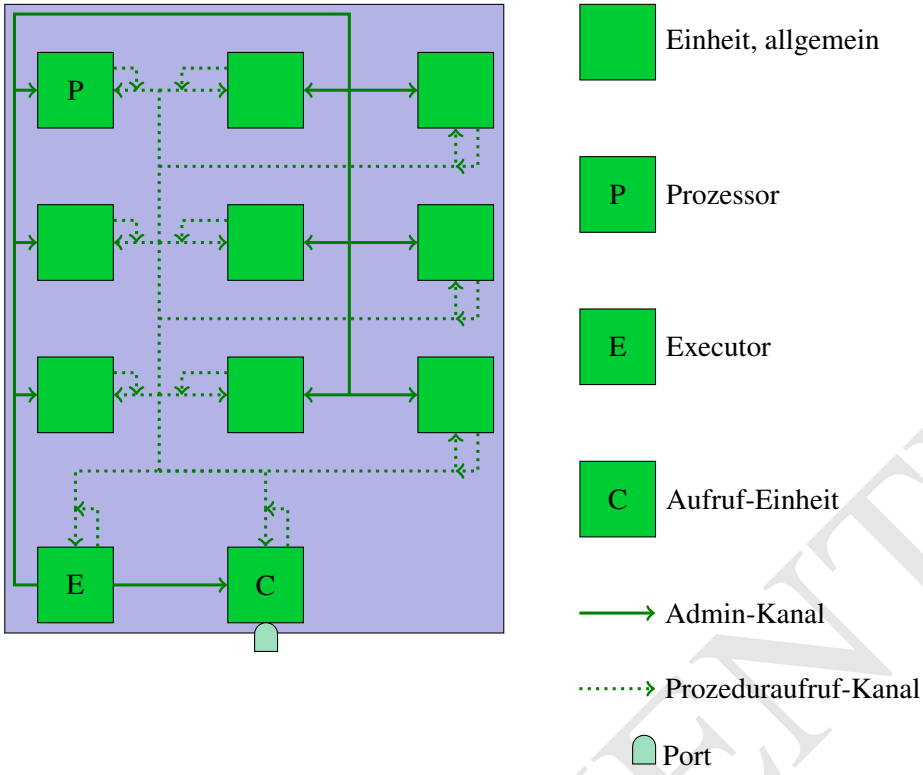


Abbildung 2.4: Maschine

Der Executor führt, wenn vorhanden, ein Boot-Programm aus. Er besitzt eine externe Schnittstelle zum Verwalten der Ausführungssphäre.

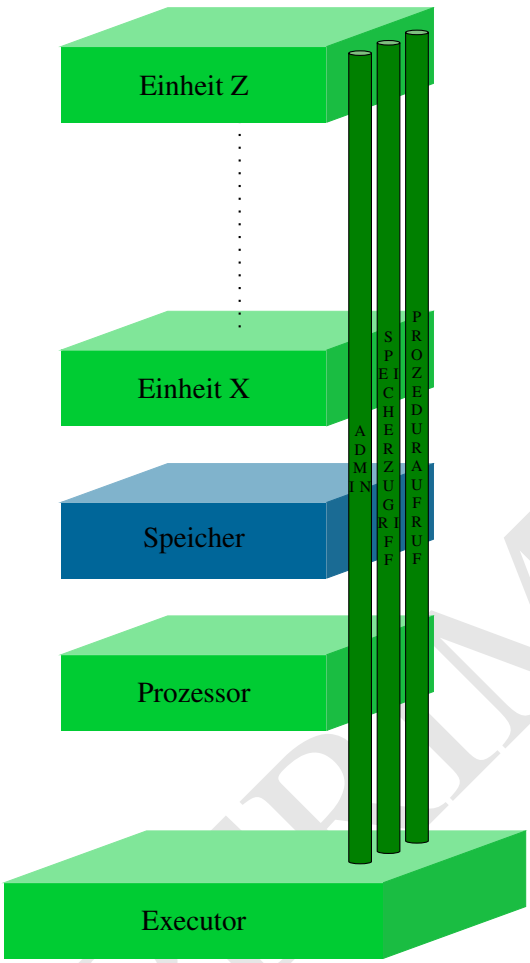


Abbildung 2.5: Maschine

2.2.1 Kanäle

Bei einem Kanal handelt es sich um eine für eine bestimmte Funktion konstruierte Verbindung zwischen zwei oder mehreren Einheiten. Für eine gleiche Funktion konstruierte Kanäle bilden einen bestimmten Kanaltyp, unabhängig von ihrem internen Aufbau. Kanäle gleichen Typs verfügen immer über die gleiche Funktionalität.
Spezielle Kanäle erlauben es, durch ihre Spezifikationen die Funktionalität von Standarbussen hardwareunabhängig zu nutzen.

Kanaltypen

- Admin-Kanal
- Datenzugriffskanal
- Prozeduraufrufkanal
- Coprozessor-Kanal
- E/A-Kanal
- DMA-Kanal
- Fehler-/Warn-Kanal
- Debug-Kanal
- Interrupt-Kanal
- Standardbus-Kanäle (PCI, ISA etc.)

Die Systembausteine werden über den E/A-Kanal angesprochen. E/A-Controller können entweder über einen E/A-Kanal an andere Einheiten gebunden sein oder über einen Datenzugriffskanal. Im letzteren Fall handelt es sich um die Abbildung von Memory-Mapped-I/O ins Computing-Modell.

Admin-Kanal

Aktionen, die über den Admin-Kanal durchgeführt werden:

- Prozess anlegen und damit Programme instantiieren.
- Prozess entfernen/beenden
- Prozess ”gewaltsam” terminieren ohne weiter Ausführung irgendeiner Prozedur
- Thread anlegen, Prozedur instantiieren
  - Verwendungszähler für mitbenutzte allgemeine Speicherbereiche raufzählen falls verwendet

- Debug-fähige Threads anlegen
  (wie Thread anlegen nur mit zusätzlicher Verwaltungsstruktur für Debug-Funktionen)
- Debug-fähigen Thread zurücksetzen
- Thread entfernen
- Systeminformationsdienst zur Verfügung stellen
- Prozess-"Fork" durchführen
- Programm eines Prozesses wechseln
- Prozedur eines Threads wechseln

Prozeduraufrufkanal

Aktionen, die über den Prozeduraufrufkanal durchgeführt werden:

- Prozedur: einen Thread starten
- Prozedur: "Abarbeitung" unterbrechen
- Prozedur: Bearbeitung eines Threads wieder aufnehmen
- Parameter eines Threads schreiben
- Parameter eines Threads lesen
- Thread zum nächst möglichen Zeitpunkt zu einem definierten Stopp bringen

Debug-Kanal

Aktionen, die über den Debug-Kanal durchgeführt werden:

- Prozedur schrittweise durchführen. Anzahl der Schritte wird übergeben (evtl. Startwert)
- Debug-Daten auslesen
  - Registerinhalte
  - etc.
- Debug-Daten schreiben (s.o)
- Haltepunkt setzen
- Trigger
  [2] setzen, um Prozedur anzuhalten

[2] überprüft, ob bestimmte Bedingungen bei Ausführung der Prozedur aufgetreten sind

DMA-Kanal

Der DMA-Transfer von Daten wird direkt vom jeweiligen Controller durchgeführt, wenn dieser dazu in der Lage ist. Hierzu werden beim Controller die für den DMA-Transfer zu verwendenden Speicherbereiche registriert. Die Steuerung der DMA-Transfers erfolgt über den so genannten DMA-Kanal.

Ablauf von DMA:

1. Speicher reservieren, auf den der E/A-Controller zugreifen kann
2. Speicherbereich(e) über DMA-Kanal beim E/A-Controller eintragen
3. ggf. Speicher mit Daten vorbereiten
4. DMA-Auftrag über DMA-Kanall erteilen
5. E/A-Controller schreibt/liest Daten über Datenkanal in den Speicher
6. Speicherbereich deregistrieren

Datenzugriffskanal

Der Datenzugriffskanal ermöglicht den wahlfreien Zugriff auf Daten. Er dient im wesentlichen dem Zugriff von Prozessoren auf die in Speichereinheiten abgelegten Daten. Und ist daher vergleichbar einem Datenbus klassischer Zentraleinheiten. Allerdings können auch andere Einheiten, wie z.B. Port-Controller, über den Datenzugriffskanal angebunden sein. Im Gegensatz zu einem reinen Datenbus, beschränkt sich die Funktionalität eines Datenzugriffskanals nicht nur auf das Übertragen von Daten. Es gibt auch Operationen zum anlegen, kopieren, tauschen und vergleichen von Daten. Abhängig von der Datenart ist auch einfügen, löschen und leeren möglich. Ferner bietet der Datenzugriffskanal Methoden um Adresse und Speichergrösse von Daten zu ermitteln.

Welche Datenarten verwendet werden können hängt von der Art des Speicherbereichs in dem die Daten abzulegen sind. Grundsätzlich können immer alle Datenarten verwendet werden, die auch in einem Register abgelegt werden können. Andere Daten fester und variabler Größe, erfordern zumindest einen beliebig einteilbaren Speicherbereich. Und nur auf dynamisch unterorganisierbaren Speicherbereichen sind auch dynamische Datenarten verwendbar.

2.2.2 Einheitentypen

- Prozessor
- Speichereinheiten
- Coprozessor
  für Operationen innerhalb der Maschine
- E/A-Controller
  für Operationen an externen Geräten, z.B. Festplatten, Monitor, Tastatur etc.

- Port-Controller
  für Operationen zur Kommunikation mit anderen Geräten, insbesondere anderen Maschinen, innerhalb des Computeraggregats, z.B. Prozeduraufruf-Einheit
- Interrupt-Controller

2.2.3 Interrupts

Ein Interrupt ist Unterbrechungssituation für bestimmte Meldungen über einen Signal-Kanal. Der Interrupt ordnet diesen Meldungen eine auszuführende Prozedur zu. In Folge eine Meldung tritt dann die Interruptbehandlung durchgeführt. Im Rahmen der Interruptbehandlung wird dann die zugeordnet Prozedur mit evtl. hinterlegten Parametern ausgeführt. Um die dem Signal zugeordnete Interrupt-Prozedur in erforderlicher Antwortzeit ausführen zu können, werden ggf. laufende Threads unterbrochen. Die Interrupt-Prozeduren sind an Interrupt-Controllern hinterlegt.

Tritt während der Behandlung eines Signals durch eine Interrupt-Prozedur ein weiterer Interrupt auf, so wird er analog behandelt. Um dies zu unterbinden, können nachfolgende Interrupts vorübergehend am Interrupt-Controller blockiert werden, die dazugehörigen Signale werden jedoch registriert. Bei Aufruf einer Interrupt-Prozedur wird das auslösende Signal automatisch zurückgesetzt. Interrupts können durch das Computersystem nach Prioritäten geordnet sein, so dass ein Interrupt niedrigerer Priorität durch einen Interrupt höherer Priorität unterbrochen und ggf. blockiert werden kann.

Der Interrupt-Controller

- registriert für Interrupts die aufzurufenden Prozeduren
- hinterlegt bei einem Interrupt eventuelle Parameter für die Prozedur.
- initiiert einen Thread für die Interrupt-Prozedur nach Erhaltung des entsprechenden Signals durch die auslösende Komponente
- ermöglicht ggf. die Zuordnung von Signalen zu einem Interrupt

Der Interruptkanal dient der Kommunikation mit einem Interrupt-Controller. Über den Interruptkanal wird die aus zuführende Prozedur einschließlich Parametern gesetzt und abgefragt, Software-Interrupts ausgelöst, sowie Interrupts blockiert und freigegeben. Unterumständen ist es auch möglich die Zurodung eines Signals zu einem Interrupt zu ändern. Eine Maschine kann mehrere Interrupt-Controller beinhalten. Der Executor ermöglicht die Reservierung und Freigabe von Interrupts durch die anderen Komponenten der Maschine (s.u.). Je nach Maschine können bestimmte Interrupts auch fest mit Komponenten verbunden sein. Um die Interrupts einer Komponente zu benutzen, muss man den zuständigen Interrupthandle bei der Komponente abfragen. Ein Interrupt-Handle besteht aus der Interrupt-Controller-ID und der Interrupt-ID.

Ablauf von PIO:

1. den benötigten Interrupt-Handle am E/A-Controller über E/A-Kanal oder Datenzugriffskanal abfragen

2. Prozedur für die Datenübertragung mit dem Interrupt-Handle am Interrupt-Controller registrieren
3. Datenübertragung am E/A-Controller initialisieren
4. durch die Interrupt-Prozedur Daten über E/A-Kanal oder Datenzugriffskanal vom oder zum E/A-Controller transferieren
5. nach Abschluss der Datenübertragungen die Prozedur beim Interrupt-Handler deregistrieren

2.3 Der Prozessor und andere Einheiten

Als Prozessor bezeichnet man eine Einheit, die den Programmcode von Prozeduren einliest, die Anweisungen auswertet und die daraus resultierenden Operationen ausführt bzw. ausführen lässt.

Ein Prozessor besteht aus:

- Threadmanager
- Adminadapter
- Befehlsinterpreter(n)
- Registerbänk(en)
- Operatoreneinheit(en)
- Kanaladapter
  - Codeleseadapter
  - Speicheradapter
  - Signaladapter
  - Prozeduradapter
  - optional Coprozessoradapter
  - optional E/A-Adapter

Über einen Adapter können mehrere Kanäle angesprochen werden. Ein Prozessor kann mehrere Adapter gleichen Typs, wie z.B. Speicheradapter, haben. Diese sind immer dann nötig, wenn für den Zugriff auf die Kanäle unterschiedliche Mechanismen oder Zusatzoperationen zu verwenden sind, z.B. Bankswitching beim C64er. Für Memory-Mapped-I/O wird ein spezieller E/A-Adapter verwendet. Ein Kanal kann höchstens über einen Speicheradapter an den Prozessor angeschlossen werden.

3 Bereich Software

Der Bereich Software umfasst in diesem Modell Programmsammlungen und die darin enthaltenen Programme bzw. Datamente (engl. datament) sowie deren Datamentsteckbriefe. Datamente sind z.B. Bilder, Programme, Texte etc. Jeder Ausführungssphäre steht eine Programmsammlung zur Verfügung.

3.1 Die Programmsammlung

Eine Programmsammlung besteht aus Programmen und anderen Datamenten, Programmsteckbriefen und weiteren Informationen zu den Programmen, wie z.B. einer Programmdokumentation.

Alle Programme, die innerhalb einer bzw. mehreren Ausführungssphären zur Verfügung stehen, bilden eine Programmsammlung. Die Programme werden unabhängig von ihrem Typ (Betriebssystem, Büroanwendungen, Treiber etc.) in die Programmsammlung aufgenommen. Dies bedeutet, dass eine Sammlung von Programmen, welche einer Ausführungssphäre zusätzlich bereit gestellt wird, die vorhandene Programmsammlung ergänzt.

Um ein Programm in einer Programmsammlung zu suchen, gibt es zu den Programmen einen systemabhängigen Suchpfad. Die Reihenfolge, in der die Programmsteckbriefe bei gleichem Namen gefunden werden, ist systemabhängig [1]. Das eigene Progset [2] gehört immer mit dazu, auch ohne dass es im Suchpfad eingetragen wurde. Der Suchpfad wird als Programmsammlungsbaum systemweit geführt. Es gibt mehrere Suchpfade je nach Programmtyp.

Eine Programmsammlung kann auch von mehreren Ausführungssphären gleichzeitig benutzt werden. Wird von einer der beteiligten Ausführungssphären eine Änderung - Löschen oder Einspielen von Programmen bzw. von Infos zu den Programmen - an der Programmsammlung vorgenommen, so wirkt sich dieses auch auf alle anderen Ausführungssphären aus. Es ist sichergestellt, dass die Ausführungssphären immer mit der aktuellen Version arbeiten.

Zugriffskonflikte werden verhindert. Sollte dieser Unterlaufen werden, ist eine entsprechende Änderung unter Umständen möglich.

Programme gehören zu einem oder mehreren Programmtypen, z.B. Link-Library, Laufzeitbibliothek, Treiber, Shell-Kommando, GUI-Programm etc. Ein Programmtyp legt fest:

- welche Prozeduren mindestens exportiert sein müssen
- ob und welche Initialisierungen durchzuführen sind

[1] z.B. durch Systemverwalter und/oder Benutzer eingestellt

[2] ähnlich Programmpaket, weiteres dazu siehe Modell Progset

- ob und welche Start-Prozeduren ausgeführt werden
- in welcher Umgebung das Programm verwendet werden kann

3.1.1 Der Programm-/Datamentsteckbrief

Ein Programmsteckbrief verweist auf das zugehörige Programm bzw. Datament - es wird über den Programm- bzw. Datamentsteckbrief gestartet.

Zum Programmsteckbrief gehören:

- Datament-Kennung (statisch)
  Identifikation des Programms
- Steckbriefname (statisch)
- Datamenttyp (statisch)
  Jeder Datamentsteckbrief enthält die Angabe genau zu einem Datamenttyp. Gehört ein Datament zu mehreren Typen so sind separate Datamentsteckbriefe erforderlich. Der Datamenttyp muss angegeben sein (Mimetype mit Untergruppe z.B. Programmtyp).
- Checksummen
- Berechtigungssets
- statische Parameter (änderbar)
- Icons

Zu jedem Datament gibt es einen ”Basis-Datamentsteckbrief”, der den Basisnamen enthält. Weiter Namen zu Datamenten werden durch zusätzliche Datamentsteckbriefe ermöglicht, die es ermöglichen dasselbe Datament mit alternativen Parametern unter anderem Namen zu benutzen. Ein Datamentsteckbrief wird nur einem Verzeichnis zugeordnet.
Zu einem Programm können ausserdem mehrere Programmsteckbriefe existieren z.B. aufgrund von verschiedenen Benutzereinstellungen oder verschiedenen Programmtypen. Je Programmsteckbrief ist genau ein gültiger Programmtyp einzutragen (siehe **Datamenttyp** unter 3.1.1 auf Seite 24). Programmsteckbriefe sind vererbbar. Der in einer Programmsammlung zu jedem Programm enthaltene Basis-Steckbrief, wird verwendet, wenn das Programm direkt und nicht über einen Steckbrief gestartet wird.

3.2 Das Maschinenprogramm und andere Datamente

Ein ausführbares Datament aus Anweisungen und Vereinbarungen, die nach den Regeln einer Programmiersprache erstellt wurde, stellt ein Programm dar. Es enthält 1 bis n Prozeduren. Im Sinne des Computing-Modells sind auch Laufzeitbibliotheken, Treiber und ähnliches Programme.

Programme werden anhand ihrer System-ID eindeutig innerhalb einer Programmsammlung identifiziert. Deren interner Aufbau ist systemabhängig[3]. Lediglich der Aufbau der Programm-Kennungen innerhalb von Progsets[4], welche in die jeweiligen Programmsammlungen importiert werden, ist eindeutig festgelegt und universell eindeutig.

Programme können in verschiedenen Versionen und Varianten vorliegen. Daher ist für die vollständige Identifikation in der Kennung Version und Variante enthalten.

Im Unterschied zu den Programmvarianten, bei denen das Verhalten des Programmes für den Benutzer variiert, gibt es Implementationsvarianten, bei denen die Implementation eines Programms variiert. Implementationsvarianten sind für die Benutzung des Programms unerheblich, dienen aber der Anpassung an unterschiedliche Systembedingungen.

Programmteile:

- Export-Deklarationen
- Import-Deklarationen inkl. Versionsnummer, Funktionsnamen usw. – je Implementationsvariante
- über ID's ladbare Ressourcen (Daten wie Text, Zahlen, Grafiken, Sound)
- Prozessorbeschreibungen
- Daten-Deklarations-Blöcke – Verwendung durch Implementationsvariante
- Codeblöcke – Verwendung durch Implementationsvariante
- Datenblöcke – Verwendung durch Implementationsvariante
- Konstantenblöcke – Verwendung durch Implementationsvariante
- Definitionslisten – Verwendung durch Implementationsvariante
- Liste der globalen Speicherbereiche - je Implementationsvariante (mit Blockzuordung, Zugriffsklasse: private, shared und vererbbar)
- Prozedur-Definitionen
- Liste der Implementationsvarianten
- Default Exception-Handler - je Implementationsvariante
- Zuordnung von Interrupts/Signalen zu Prozeduren - je Implementationsvariante
- Reservierung von Ports - je Implementationsvariante

[3] Dies erlaubt einfachere Anpassung an das Zielsystem.
[4] siehe Modell für Progsets

Semaphoren können direkt bei der Definition der Speicherbereiche festgelegt werden. Semaphoren für Shared-Memory werden im Export-Bereich deklariert.
Neben den im Programmcode vorgegebenen Blöcken kann ein Programm zusätzlich dynamisch Blöcke allokieren. Das Programm ist selbst dafür verantwortlich den dynamisch allokierten Speicher wieder freizugeben (Zur Prozedur zur dynamischen Speicherreservierung siehe unter 4.3.1 auf Seite 37.

Ein Programm kann sich die Bearbeitung von Ressourcen exklusiv reservieren.

3.3 Die Prozedur und andere Programmobjekte

Eine Reihe von Anweisungen und Vereinbarungen, die eine funktionale Einheit innerhalb eines Programms bilden, nennt man eine Prozedur (in manchen Programmiersprachen auch Funktion, Methode oder Routine). Die Verallgemeinerung einer Prozedur ist ein Programmobjekt.

Aufbau einer Prozedur:

- je Implementationsvariante einen Verweis auf einen Codeblock
- je Implementationsvariante eine Liste der Speicherbereiche
- Inputdefinition (formale Parameter)
- Outputdefinition (formale Rückgabeparameter)
- Referenzdefinition
- Flags (als Rückgabewerte)

Eine Prozedur enthält in ihrem Codeblock mindestens eine Anweisung.
Eintrittspunkt einer Prozedur ist immer der Anfang des Codeblocks. Und der Austrittspunkt ist das Ende des Codeblocks. Ein Rücksprung wird als Sprung ans Ende des Codeblocks abgebildet. Der Codeblock enthält einen Verweis auf die Prozessorbeschreibung des für seine Ausführung benötigten Prozessors.
Jede beliebige Prozedur eines Programms kann mit einer Implementationsvarianten-abhängigen Zeitspanne in Pseudo-Takten (TimeOut) versehen werden (siehe 4.3 auf Seite 35), nach der diese gewaltsam beendet werden soll. Für Prozeduren mit einer Mindestausführungsgeschwindigkeit in Echtzeit kann die benötigte Mindestanzahl Pseudo-Takte pro Sekunde angegeben werden.
Die Speicherbereiche einer Prozedur werden analog zu den Speicherbereichen eines Programms über eine Liste deklariert.
Die Inputdefinition enthält eine Beschreibung der Daten, die der Prozedur vor ihrer Ausführung zur Verfügung zu stellen sind. Sie werden vor dem Start der Prozedur in den Übergabebereich des ausführenden Prozessors übertragen. Die in der Outputdefinition beschriebenen Daten sowie die Flags werden nach Beenden der Prozedur in den Übergabebereich des aufrufenden Prozessors

übertragen. Referenzdefinitionen werden wie In-/Outputdefinitionen behandelt. Allen ist gemeinsam, dass sie neben Definitionen von festen Argumentlisten auch Bereiche für variable Argumentlisten und eine Liste von übergebenen Speicherbereichen enthalten können.

4 Bereich Run-Time

Der Bereich Run-Time des Modells beschreibt das dynamische Verhalten von Ausführungssphären mit den darin enthalten Prozessen und deren ausgeführten Threads.

4.1 Die Ausführungssphäre

Die Ausführungssphäre stellt den Bereich dar, dem ein Prozess angehört und mit dem er während der Ausführung interagiert. Sie wird von einer ”Logischen”-Instanz, der Verwaltungsinstanz verwaltet.
Da die Administrations-Aktivitäten einer Maschine (z.B. Speicher reservieren) nicht Teil des jeweiligen Prozesses sind, sondern der Verwaltungsinstanz, erstreckt sich diese daher über alle Maschinen.

Der Ausführungssphäre sind zugeordnet:

- Prozesse
- Instanzen (engl. instances)
  Stellt die Abstraktion eines Prozesses dar, der optional als Arbeitspartner für einen Prozess zur Verfügung steht. Sie ist nur durch ihre Schnittstelle nach aussen definiert - Interna sind unbekannt.
- Verwaltungsinstanz
  - Kommunikationsdienste
  - Shared-Memory mit Manager
  - Umgebungsvariablen
    je Computeraggregat, Maschine, Benutzer, Programm oder Prozess etc.
  - Loader
    Der Loader wird zum Laden der Programme verwendet.
    * zugeordnete Programmsammlung
  - Ressource Manager
    für die Betriebsmittel des Computeraggregats
  - Berechtigungsverwaltung

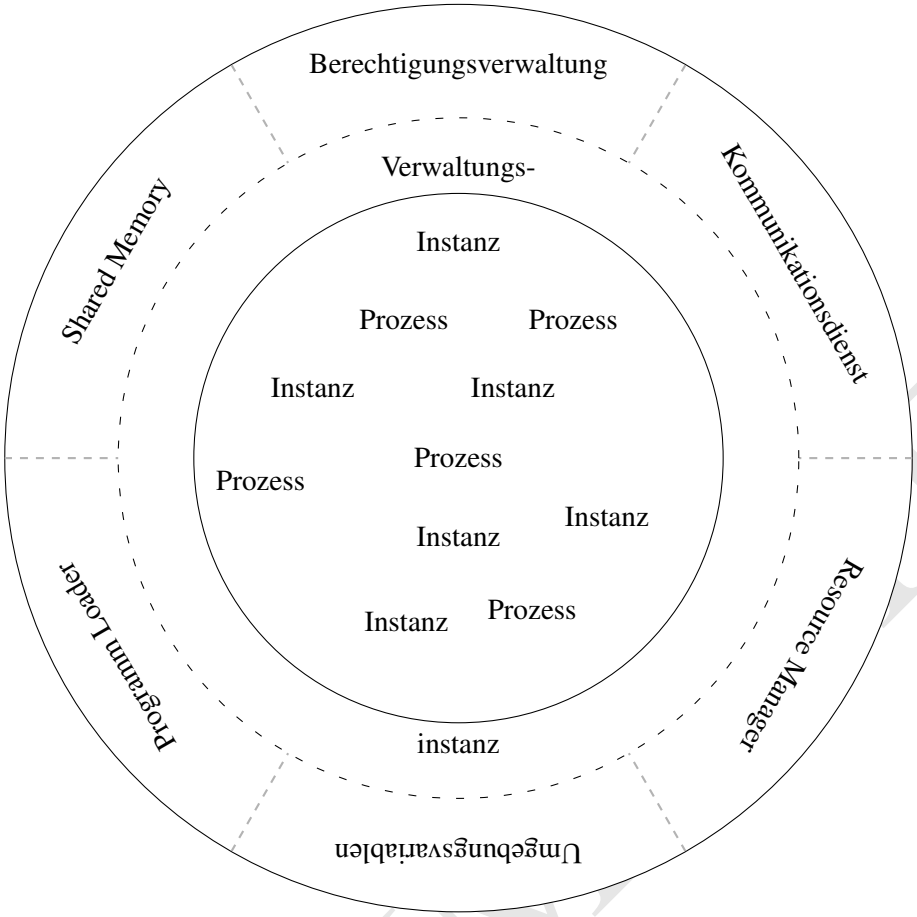


Abbildung 4.1: Die Ausführungssphäre

4.1.1 Programmstart/Bootvorgang

Beim Starten eines Programms wird ein neuer Prozess angelegt und die Hauptprozedur des Programms entsprechend des Programmtyps ausgeführt. Ebenfalls abhängig vom Programmtyp wird der Prozess ggf. am Ende der Hauptprozedur beendet.

Für jedes Programm wird in dem(den) dazugehörigen Programmsteckbrief(en) eingetragen, ob dieses beim Booten initialisiert bzw. gestartet wird. Gestartet wird das Programm nach Initialisierung dann, wenn eine Start-Prozedur exportiert wurde. Die Reihenfolge der Initialisierung der Programme ist beliebig und erfolgt ggf. parallel. Jedes beim Booten initialisierte bzw. gestartete Programm läuft in einer eigenen Maschine. Damit können maximal so viele Programme gebootet werden, wie Maschinen vorhanden sind. Das Verhalten des Systems bei mehr Boot-Programmen als Maschinen ist nicht näher definiert, aber im Zweifel wird der Bootvorgang gestoppt.

4.1.2 System-IDs und ungeordnete Mengen / Kennungen

Die Ausführungssphäre vergibt für die im Computing-Modell beschriebenen Komponenten System-IDs.
Werden dynamisch vergebene System-IDs benötigt, so erfolgt die Vergabe dieser System-IDs durch ein für das ganze Computeraggregat einheitliches Verfahren. Dies vereinfacht das Referenzieren von Geräten im gesamten Computeraggregat, z.B. bei Benutzung von Shared-Memory.
Für die System-IDs werden ungeordnete Mengen verwendet (siehe Modell Datenarten). Dabei bilden die Prozesse, Threads, Kanäle etc. jeweils eine eigene ungeordnete Menge. Die Elemente dieser ungeordneten Mengen (System-ID-Mengen) sind durch eindeutige Kennungen definiert, welche vom jeweiligen System abhängen und automatisch vergeben werden.
Ausnahme: Bereich Software - dort werden die extern vergebenen Kennungen verwendet.

Kennungen werden für globale technische Benennungen verwendet. Es dürfen nur ”a” bis ”z”, ”0” bis ”9”, und ”-” verwendet werden.

Zur Verwaltung der ungeordneten Menge gibt es eine zentrale Schnittstelle. Diese ist im Datenartenmodell im Kapitel "Verwaltung von ungeordneten Mengen"beschrieben.

An der Schnittstelle kann jeder Prozess nur die Elemente einer ungeordneten Menge sehen (bearbeiten), die er auch selber registriert hat. In Hochsicherheitssytemen sollte die Vergabe der Datenwerte [1] für die Elemente der ungeordneten Mengen zufällig erfolgen. Ggf. sind noch mehrere physikalische Zustände [2] für einen Datenwert vorzusehen.

4.1.3 Berechtigungsverwaltung

Es gibt keine User oder Gruppen sondern Berechtigungssets [3]. Diese werden von der Ausführungssphäre verwaltet und innerhalb der Berechtigungsverwaltung gespeichert. Zu Prozesse und Berechtigungen siehe unter 4.2.1 auf Seite 33. Weitere Informationen zur Berechtigungsverwaltung und zu Berechtigungssets befinden sich im **Sicherheits Modell**.

4.2 Der Prozess und andere Instanzen

Ein Prozess ist ein in Ausführung befindliches Programm inkl. aller von dem Programm während der Ausführung reservierten Ressourcen. Er umfasst 1 bis n Thread(s).

Die Blöcke eines Programms sind wie folgt auf den Maschinenspeicher verteilt:

[1] auch Handlewerte oder kurz Handle
[2] Bit- bzw. Byte-Kombination
[3] entspricht den credential der GSS-API

Codespeicher	Datenspeicher
Codeblöcke	initialisierter Datenblock
Konstantenblöcke	nicht-initialisierte Datenblöcke
Definitionsblöcke	

Ein Prozess kann einen 'Sub-Prozess' oder einen neuen Prozess initiieren. Ein Sub-Prozess läuft auf derselben Maschine wie der Prozess, der ihn gestartet hat und teilt sich mit ihm vererbbare Ressourcen. Im Unterschied dazu läuft ein neuer Prozess auf einer beliebigen Maschine und teilt sich in Folge dessen keine Ressourcen mit seinem Parent-Prozess. Der Speicher eines Prozesses kann von den durch ihn gestarteten/geladenen Sub-Prozessen gesehen werden. D.h. die Sub-Prozesse können auf den Speicher ihres Ursprungs-Prozesses zugreifen. Sonstige Prozessübergreifende Speicherzugriffe sollten von einem Schutzmechanismus gesperrt werden.

4.2.1 Prozess erzeugen/entfernen

Zum Anlegen bzw. entfernen eines Prozesses müssen folgende Schritte durchgeführt werden:

- Prozess erzeugen
  - Prozess-Kontext erzeugen
  - Programmcode laden
  - Statische Ressourcen gemäss Programm reservieren und initialisieren
  - Ggf. allg. Initialisierungs-Prozedur ausführen. Diese hat keine Parameter und gibt zurück, ob sie erfolgreich durchgeführt wurde.
  - je nach Aufruf-Verfahren (verwendeter Programmtyp) entsprechende exportierte Prozedur(en) starten
- Prozess entfernen
  - Ggf. allg. Deinitialisierung-Prozedur ausführen. Diese besitzt weder Parameter noch Rückgabewerte. Mit TimeOut aus Programmkopf in Abhängigkeit von der Implementationsvariante.
  - Statische Ressourcen freigeben
  - Programmcode entladen
  - Prozess-Kontext zerstören

Ein Prozess muss mindestens eine Prozedur gemäss eines Programmtyps exportieren. Prozess erzeugen und entfernen kann auch eigenständig ohne Aufruf einer exportierten Prozedur durchgeführt werden.

Bei der Erstellung des neuen Prozesses gibt es zwei Möglichkeiten:

1. Ein Programm wird neu initialisiert.
   Dies kann entweder als Sub- oder als neu initiierter Prozess geschehen.

2. Der Prozess wird geklont und der neue Prozess läuft an der gleichen Stelle weiter, an der sich der alte Prozess zum Zeitpunkt des Klonens befunden hat.
   Auch der geklonte Prozess kann sowohl als Sub-Prozess unter Verwendung gemeinsamer Ressourcen (siehe BSD vfork) als auch als neuer Prozess (siehe UNIX fork) initiiert werden.

Eine häufige Form von Sub-Prozessen sind Laufzeitbibliotheken. Auch die Amiga-Shell führt Programme als Sub-Prozess aus.

Zu jedem Prozess sind zugeordnet:

- ID
- Programm in einer bestimmten Implementationsvariante
- Threads
- Aktivitäten (engl. activities)
  Stellt die Abstraktion eines Threads dar, die optional als Arbeitspartner für einen Thread zur Verfügung steht. Er ist nur durch seine Schnittstelle nach aussen definiert - Interna sind unbekannt. Teil einer Instanz. Eine Instanz kann mehrere Aktivitäten haben.
- Speicherbereiche
  - globale Speicherbereiche
    (für alle Threads verfügbar)
  - Speicherbereiche für die Threads
  - Globale Konstanten
  - vererbbarer Speicherbereich
    (z.B. für vfork)
- bei Bedarf Semaphoren
  (zur Prozess-internen Synchronisation)
- weitere verwendete Ressourcen der Maschine
- Signale und Interrupts
- Default Exception-Handler
- Ports (min. 1)
- mindestens ein Berechtigungsset

Parameter werden an Prozesse per Message übergeben.
Die Verzeichnis-Struktur der Programmsammlung stellt sich dem Prozess als aus mehreren Bäumen bestehend dar. Dies berührt nicht die interne Struktur der Programmsammlung.
Zum Zugriff auf die Programmsammlung stehen einem Prozess ggf. mehrere Handle für unterschiedliche Bäume bzw. Teilbäume zur Verfügung. Die in einem Baum enthaltenen Verzeichnisse können - aber müssen nicht - in anderen Bäumen enthalten sein.

Prozesse können, falls dazu berechtigt, von einem zu einem andern Berechtigungsset wechseln. Entsprechend können ggf. für Sub-Prozesse auch andere Berechtigungssets verwendet werden, sofern erlaubt.

4.3 Der Thread und andere Aktivitäten

Eine Prozedur, die sich bei einem Prozessor in Ausführung befindet, ist ein Thread. Threads sind die eigentlich ausgeführten Komponenten eines Prozesses.

Ein Thread kann einen 'Sub-Thread' oder einen neuen Thread initiieren. Ein Sub-Thread wird von demselben Prozessor ausgeführt wie der Thread, der ihn gestartet hat, und teilt sich mit ihm vererbbare Ressourcen. Im Unterschied dazu wird ein neuer Thread von einem beliebigen Prozessor ausgeführt und teilt sich in Folge dessen keine Prozessor-Ressourcen mit seinem Parent-Thread.

Zu jedem Thread sind zugeordnet:

- eine Prozedur
- ein Prozessor
- verwendete Register des Prozessors
- Exception-Handler
- statischer Speicherbereich für lokale Daten (optional)
- Übergabe-/Rückgabeparameter (optional)
- Datenstack (optional)

Ein Thread kann gestartet werden:

- durch andere Threads des Prozesses; es besteht die Möglichkeit einen Sub-Thread zu erzeugen
  - als Unterroutine - aufrufender Thread hält an
  - als parallele Unterroutine - parallel zum aufrufenden Thread
  - als Folgeroutine - aufrufender Thread ist beendet

  - als Parallelroutine - kehrt nicht zurück
  - per Fork - der aufrufende Thread wird geklont
- per Exception (erbt den Speicher automatisch)
- per Interrupt
- per Bootvorgang
- per externem Aufruf (z.B. Library-Call)
- über einen Port

Threads ausführen:
Bei der Ausführung eines Threads sind über die Kanäle verschiedene Aufgaben abzuwickeln.

- Thread anlegen durch Angabe des Thread-Typs und der Prozedur → Admin-Kanal
  1. interne Verwaltungsstruktur anlegen
  2. einen passenden Prozessor für Prozedur suchen
  3. Codeblock dem Prozessor zur Verfügung stellen, wenn er noch nicht zur Verfügung steht
  4. Parameterbereiche initialisieren
  5. Speicher (Daten) und andere statisch festgelegte Ressourcen (festgelegt im Prozedurkopf) reservieren.
  6. interne Verwaltungsstruktur vollständig initialisieren/vervollständigen
- Parameter schreiben → Call-Kanal
- Thread starten d.h. mit der Abarbeitung der Prozedur beginnen → Call-Kanal
  intern: Verwendungszähler der Speicherbereiche werden erhöht
- Thread anhalten, wenn gewünscht, d.h. Abarbeitung der Prozedur des Threads durch den Prozessor pausieren → Call-Kanal
- Thread fortsetzen, wenn gewünscht, d.h. Abarbeitung der Prozedur nach deren pausierter Anweisung fortsetzen → Call-Kanal
- Thread beendet, d.h. die Abarbeitung erreicht das Ende der Prozedur → Call-Kanal
  oder
  Thread abbrechen, d.h. die Abarbeitung wird innerhalb der Prozedur beendet → Call-Kanal
  intern: Verwendungszähler herunter zählen
- Rückgabeparameter lesen → Call-Kanal
- Thread freigeben → Admin-Kanal

Statische Thread Speichertypen:

- konstanter Speicher
  steht allen Threads der Prozedur als nur-lese Speicher zur Verfügung

- static Speicher[4]
  steht jedem Thread zur Verfügung, der die zugehörige Prozedur ausführt

- lokaler Speicher[5]
  steht nur während der Ausführung des Threads zur Verfügung

- vererbbarer Speicher[6]
  steht während der Ausführung des Threads zur Verfügung und solange noch Threads existieren, an die dieser Speicherbereich weiter vererbt wurde. Eine Maschine, die parallele Threads unterstützt, muss auch die Speichervererbung für diese unterstützen.

- privater Speicher[7]
  steht als eigene Kopie in jedem Thread zu Verfügung, der die zugehörige Prozedur ausführt. Eine Maschine, die parallele Threads unterstützt, muss für diese auch privaten Speicher unterstützen.

Multithreading und Dead-Locks

Bei Multithreading-Programmen können Probleme bei der Zuteilung der Ressourcen auftreten. Um diesem begegnen zu können, wird eine Art Transaktion implementiert.
Der Transaktions-geschützte Teil darf nicht länger als eine definierte Anzahl Pseudo-Takte dauern. Wird dies nicht eingehalten, kann das Programm als fehlerhaft abgebrochen werden.
Alle Ressource-Operationen (Reservierung, Freigabe, Abfrage) laufen über den Admin-Kanal laufen. Zu Beginn der Ressource-Transaktion muss der Prozess mitteilen, auf welche Ressource sich diese Transaktion beziehen soll. Die Ressource-Verwaltung der Verwaltungsinstanz kann sich anders verhalten, je nachdem, ob mit Ressource-Transaktionen gearbeitet wird oder nicht. Z.B. kann die Anfrage nach zur Verfügung stehendem Speicher innerhalb einer Transaktionsumgebung niedrigere Werte liefern als ausserhalb.

Pseudo-Takt:

$o :=$ Operation im GMC
$P() :=$ Pseudo-Takte der Operation
$t_P :=$ Zeit eines Pseudo-Takts
$t_O :=$ tatsächliche Zeit der Operation

$t_O \leq t_p \cdot P(o)$ oder $\sum t_o \leq \sum t_p \cdot P(o)$

[4] OpenMP: global-lifetime memory
[5] OpenMP: private variable
[6] OpenMP: shared variable
[7] OpenMP: threadprivate memory

4.3.1 Speicherbereiche

Datenarten, die nur im Speicher gehalten werden können, werden beim Starten von Prozeduren in normalen Speicherbereichen, auf die die Prozeduren Zugriff haben, abgelegt. Zu diesen Speicherbereichen gibt es eine interne Verwendungsregistrierung, die es den beteiligten Prozeduren unabhängig voneinander ermöglicht, den Speicher freizugeben. Diese zeigt allerdings nur an, wie oft der Speicherbereich mitbenutzt wird.

Die Auswahl der Arbeitsmenge eines Programms an virtuellem Speicher wird durch Zugriffsattribute für Speicherbereiche festgelegt. Die Zugriffsattribute reichen von ”wird ständig benötigt” bis ”zur Zeit nicht benutzt”. Diese Speicherattribute dienen auch der Optimierung des Cache-Managements. Zur Unterstützung von Bankswitching, kooperativer Speicherverwaltung, Cache-Management und Remote-Memory können Speichereinheiten vorübergehend ”abgekoppelt” werden. Um auf diese wieder zugreifen zu können, müssen sie erst wieder ”angekoppelt” werden. Neben einfachen Speicherbereichen fester Länge werden auch Memory-Pools angeboten (nicht im Minimalumfang), die optional (mit weiterem Zusatzpaket) auch hierarchisch gegliedert werden können.
Die Speichereinheiten bieten neben einfachen Speicherbereichen und Memory-Pools auch Speicherbereiche zur zellenweisen Allokation und Freigabe an (nicht im Minimalumfang). Hierfür werden spezielle Speicherblöcke reserviert. Zur Unterstützung von virtuellem Speicher mittels Segmentierung werden Speichereinheiten verwendet. Dabei werden die Segmente der virtuellen Speicherverwaltung auf Speichereinheiten abgebildet. So ist sichergestellt, dass alle Daten eines Segmentes von Allen gelesen werden können, die Zugriff auf diese Speichereinheit haben.

Ein Prozess kann dynamisch Speicher allokieren und muss diesen selber wieder freigeben. Der Speichertyp wird beim Allokieren angegeben, wobei der Prozess auf die Auswahl des passenden Speichertyps selber achten muss. Die Grösse eines allokierten Speicherbereichs wird vermerkt. Am Ende der Prozedur des Programms, die ausgeführt wird, muss ein Clean-up durch den Prozess erfolgen, noch bevor dieser terminiert wird.

Wenn notwendig wird der von einem Thread allokierte Speicher mittels der Verwaltungsinstanz verwaltet.

Pointer sind immer relativ zur Basisadresse des Speicherbereichs, auf den sie sich beziehen, zu interpretieren.

Der Zugriff auf atomare Datenarten erfolgt immer atomar (→ Strings sind keine atomaren Datentypen!).

Der Wert eines Datenfeldes ist beim ersten Zugriff ohne vorherige Initialisierung undefiniert, d.h. zufällig.

Es gibt einen Universal-Pointer in den alle anderen Pointer übertragen werden können. Die Zurückübertragung von Universal-Pointer in spezielle Pointer ist nur dann fehlerfrei möglich, wenn vorher von diesen speziellen Pointern in den Universal-Pointer übertragen wurde. Der Universal-

Pointer muss eindeutig sein. D.h. die Übertragung von verschiedenen speziellen Pointern in den Universal-Pointer darf nicht zu gleichen Universal-Pointern für verschiedene Objekte führen. Spezielle Pointer werden angelegt, wenn es nicht möglich ist (gemäss Computing-Modell bzw. Schnittstellentechnik), schon vorhandene spezielle Pointer-Typen zu verwenden.

Dynamische Speicherreservierung

1. Der Thread schickt über den Admin-Kanal eine Speicheranfrage an die Verwaltungsinstanz. Die Speicheranfrage enthält die Mindestgrösse des Blocks und ggf. weitere Attribute (noch nicht geklärt).

2. Die Verwaltungsinstanz sucht nach einem passenden Speicherbereich.

3. a) passender Speicherbereich gefunden: Der Thread erhält über den Admin-Kanal eine Antwort mit Kanal-ID, Block-ID, zusätzlicher Blockgrösse

   b) kein passender Speicherbereich gefunden: Der Thread erhält einen negativen Bescheid mit der Angabe der noch maximal möglichen Blockgrösse. Auf Systemen, auf denen dies nicht möglich ist, wird immer eine Restblockgrösse von Null zurückgegeben.

Dynamische Speicherfreigabe

Das Programm meldet die Freigabe von Block-ID auf Kanal-ID über den Admin-Kanal an die Verwaltungsinstanz.

Aus Kanal-ID+Block-ID erhält man eine eindeutige Blockadresse und anders herum.

4.3.2 Exceptions

Exceptions sind Fehlersituationen, die in Folge der Abarbeitung eines Programmschritts an einem Prozessor auftreten, z.B. beim Zugriff auf falsch allokierte Speicherbereiche. Der die Exception verursachende Thread wird angehalten und stattdessen eine Behandlungsroutine ausgeführt. Die Behandlungsroutine(n) ist(sind) zusammen mit den Threads am Prozessor hinterlegt.

Im Falle einer auftretenden Exception gibt es folgende Möglichkeiten:

Fall 1 es gibt keine Behandlungsroutine für diese Exception oder die letzte Behandlungsroutine der Kette kann die Exception nicht behandeln
-> der Thread wird abgebrochen und der aufrufende Thread wird informiert. Wie der aufrufende Thread darauf reagiert, hängt vom Verfahren ab, mit dem der Thread aufgerufen wurde. Ggf. wird eine Exception am aufrufenden Thread erzeugt. Im Minimalumfang wird für einen abgebrochenen Subthread nur ein Flag gesetzt.

Fall 2 Ein Handler in der Kette kann die Exception korrigieren
-> der verursachende Thread läuft mit dem nächsten Befehl weiter.

Fall 3 Eine Behandlungsroutine in der Kette kann die Exception nicht behandeln
-> der nächste Handler in der Kette wird aufgerufen

Fall 4 Eine Behandlungsroutine kann die Exception zwar behandeln aber nicht korrigieren
-> der verursachende Thread wird mit ggf. von der Behandlungsroutine gesetzten Rückgabewerten beendet.

Fall 5 Ein Handler für eine Exception leitet eine andere Exception ein
-> die Kette der Exceptionhandler wird nicht weiter behandelt, stattdessen werden die Behandlungsroutinen der neuen Exception aufgerufen.

Fall 6 Ein Handler kann die Exception zwar behandeln, ist aber nicht in der Lage den Thread ordnungsgemäß zu beenden
-> der Thread wird abgebrochen und dem aufrufenden Thread wird dies mitgeteilt, der Grund des Abbruchs kann von der Exception-Behandlungsroutine gesetzt werden.

Tritt währen der Bearbeitung einer Exception durch einen Exception-Thread eine weitere Exception auf, so wird diese wiederum entsprechend behandelt, wobei allerdings jetzt die Behandlungsroutinen-Kette des Exceptions-Threads gilt.

Ein Thread kann sich einen Exception-Handler installieren, welcher mehrere Exceptions behandeln kann. Eine Installation von mehreren Exception-Handlern ist zulässig. Der installierte Exception-Handler wird für bestimmte Exceptions registriert und nur dann angesprungen, wenn eine dieser registrierten Exceptions auftritt. Alle anderen Exceptions gehen an den vorher (übergeordneten) registrierte Exception-Handler. Exceptions können vom Exception-Handler auch an den vorher registrierten Exception-Handler weitergeleitet werden. Ausserdem kann er auch andere (korrigierte) Exceptions weiterleiten.

Index