Middleware für verteilte industrielle Umgebungen

Eigenschaften verteilter Systeme, Interprozesskommunikation

ifak

Inhalt des Kapitels

- “Verteilbare“ Ressourcen → Charakteristiken
- Resource Sharing
- Client-Server-Systeme
- Transparenz

Charakteristiken

- sechs Punkte zur Nützlichkeit verteilter Systeme:
  1. Betriebsmittelteilhaberschaften (resource sharing)
  2. Offenheit (openness)
  3. Nebenläufigkeit (concurrency)
  4. Skalierbarkeit (scalability)
  5. Fehlertoleranz (fault tolerance)
  6. Transparenz (transparency)
- keiner der Punkte impliziert jedoch automatisch die Verteiltheit
  - Anwendung und System sind sehr sorgfältig zu entwerfen . . .
  - . . . um diesen Punkten entsprechend Genüge zu tragen

ifak

Betriebsmittelteilhaberschaften

- gemeinsame Benutzung von Hardware-Komponenten . . .
  - Drucker
  - Platten
  - Netzwerkanschlüsse
  - ...
  - Prozessoren
- . . . aber auch von Software-Entitäten
  - Dateien
  - Fenster
  - Datenbanken
  - ...
  - Datenobjekte
- . . . durch "Subjekte“, d.h. Prozesse bzw. Kunden

ifak

Betriebsmittelteilhaberschaften (ff)

- erfährt besondere Bedeutung in vernetzten Systemen
  - computer supported cooperative work (CSCW) . . .
    - auch allgemein verstanden als groupware
  - . . . über Programme, ausgeführt auf verschiedenen Rechnern
- die Ressourcen sind physikalisch eingeschlossen
  - innerhalb des oder der Rechner
  - Zugriff ist nur möglich durch Kommunikation
- Ressourcen werden durch Programme verwaltet
  - Betriebsmittelverwalter bzw. Ressource-Manager
    - sind für einen bestimmten Typ von Ressource zuständig
    - pro Typ eigene Verwaltungsstrategien und -methoden
  - Kommunikationsschnittstelle gestattet Zugriff/Manipulation

ifak

Betriebsmittelteilhaberschaften (ff)

- Ressourcen besitzen systembedingte Gemeinsamkeiten
  - Namensschema zum individuellen (entfernten) Zugriff
  - Abbildung von Namen auf Kommunikationsadressen
  - Koordination nebenläufiger Zugriffe
- Ressourcen werden durch Programme genutzt
  - Betriebsmittelbenutzer bzw. Ressource-Nutzer

ifak

Betriebsmittelteilhaberschaften (ff)
Client versus Server

- der Client ist ein ressourcen-nutzendes Programm
  - er sendet eine Anforderung (request) zum Server . . .
  - . . . wann immer Zugriff auf die Ressource benötigt wird
- der Server ist ein ressourcen-verwaltendes Programm
  - er empfängt Anforderungen von Klienten . . .
  - . . . und führt den koordinierten Zugriff aus
- Programme können Ressourcen nutzen und verwalten
  - sie sind dann Client und Server in einer "Person“

ifak

Betriebsmittelteilhaberschaften (ff)
Client versus Server (ff)

- Client und Server sind jeweils als Prozess repräsentiert
  - der mindestens aus einem Faden (thread) besteht . . .
  - . . . und ggf. einen eigenen Adressraum besitzt
- mehrfädige Server gestatten (pseudo-) parallele Verarbeitungen
  - die Nebenläufigkeitsproblematik besteht aber auch ohne dem
- Client und Server können auf demselben Rechner ablaufen
  - die Hardware-Organisation ist nicht ausschlaggebend

ifak

Betriebsmittelteilhaberschaften (ff)
Server versus Dienst (service)

- der Server implementiert einen bestimmten Dienst
  - Adressenverzeichnis
  - Authentifikation
  - Ein-/Ausgabe
  - ...
  - netzwerkweit synchronisierte Uhren
- "Server“ ist jedoch nicht Synonym von "Dienst“
  - ein Dienst kann von mehreren Servern erbracht werden
    - verteilt, im "Ensemble“ zu erbringender Dienst
  - zur Diensterbringung kommunizieren die beteiligten Server

ifak

Betriebsmittelteilhaberschaften (ff)
Server versus Dienst (service) (ff)

- der Server als zentraler Anbieter zu verwaltender Ressourcen
  - die zentrale Diensterbringung ist nicht wünschenswert . . .
    - sie ist kontraproduktiv zur dezentralen Systemstruktur
  - . . . kann aber auch nicht immer vollständig vermieden werden
- der Dienst ist ein abstraktes Gebilde
  - der kooperativ von mehreren Servern erbracht werden kann . . .
  - . . . deren Ausführung dann auf vernetzten Rechnern erfolgt
- der Dienst wird dem Client durch Server angeboten

ifak

Betriebsmittelteilhaberschaften (ff)
Objekt versus Ressource

- das Objekt als gemeinsam benutztes Betriebsmittel
- Objekte besitzen eine eindeutige Identifikation
  1. Adresse des Rechners
  2. Adresse des Adressraumes innerhalb des Rechners
  3. Adresse des Segmentes innerhalb des Adressraums
- Objekte können im Netzwerk frei bewegt werden
  - ohne ihre Identifikation ändern zu müssen
  - Objektadressen (im Netz) sind logische Adressen

ifak

Betriebsmittelteilhaberschaften (ff)
Objekt versus Ressource (ff)

- ressource-nutzende Programme versenden Nachrichten . . .
  - die Anforderungswünsche an Objekte enthalten
- . . . um den Zugriff auf eine Ressource zu begehren
- Nachrichten werden Prozeduren bzw. Prozessen zugestellt
  - um die angeforderte Operation durchzuführen
- der angeforderte Prozess antwortet ggf. dem anfordernden Prozess
  - er sendet eine Antwortnachricht (reply message) zurück

- Objekte gestatten eine einheitliche Sicht auf Ressourcen

ifak

Betriebsmittelteilhaberschaften (ff)
Objekt versus Client/Server

- Client und Server sind (aktive) Objekte
  - objekt- und/oder klassenbasierte Sicht . . .
  - . . . gekoppelt mit einem Prozesskonzept
- (aktive) Objekte können Ressourcen-Nutzer und -Verwalter sein
  - wie Clients Server sein können, und umgekehrt

- der Objektverwalter ist Server von Objekten einer Klasse
  - Kapselung von Prozeduren und Datenwerten
- Objektverwalter und seine Objekte bilden eine Einheit
  - die Objekte liegen im Adressraum des Objektverwalters . . .
  - . . . sie residieren dort, wo ihr Verwalter residiert

ifak

Betriebsmittelteilhaberschaften (ff)
Objekt versus Client/Server (ff)

- mobile Objekte migrieren zwischen Objektverwaltern
  - die Verwalter müssen dazu repliziert im System vorliegen
    - sie bilden ein Ensemble von Servern desselben Typs . . .
    - . . . die über das vernetzte System verteilt sind
  - die replizierten Verwalter kooperieren untereinander

- Objekte stellen die zu verarbeitenden Einheiten dar . . .
  - sie kennen implizit den ihnen jeweils zugeordneten Verwalter
- . . . ein Server ist Mechanismus zum entfernten Objektzugriff

ifak

Offenheit

- bezieht sich allgemein auf zwei Betrachtungsebenen:
  1. offen hinsichtlich Hardware-Erweiterungen
     - Peripherie, Speicher, Netzwerkzugänge
  2. offen hinsichtlich Software-Erweiterungen
     - Betriebssystemfunktionen, Protokolle, Dienste
- bedeutet auch Offenlegung von Schnittstellen
  - Spezifikation und Dokumentation

ifak

Offenheit (ff)

- offene Systeme sind herstellerunabhängig . . .
  - nach innen sind sie meist von heterogener Struktur
    - hinsichtlich Hardware und Software
  - nach außen zeigen sie eine standardisierte Schnittstelle
- . . . und basieren auf einheitlicher Interprozesskommunikation
  - Server laufen auf beliebigen Betriebssystemen und Rechnern . . .
    - eine wünschenswerte, sehr idealistische Sicht
  - . . . sie ermöglichen dadurch den globalen Ressourcen-Zugriff
    - abstrahieren vom jeweils darunter liegenden System
- Anwendungen werden unabhängig vom gegebenen Zielsystem

ifak

Nebenläufigkeit

- zwei Gründe für die Gelegenheit paralleler Verarbeitung:
  1. mehrere Benutzer interagieren "gleichzeitig“ mit dem System
     - Anwendungsprozesse laufen unter Benutzerkontrolle . . .
     - . . . verteilt über mehrere Rechner
  2. mehrere Server reagieren "gleichzeitig“ auf Anforderungen
     - die Anforderungen kommen von Anwendungsprozessen
     - die Server sind über mehrere Rechner verteilt
- einfädige Server bedienen Klienten sequentiell . . .
  - auch im Falle von Mehrprozessorsystemen
- . . . mehrfädige Server bedienen Klienten nicht-sequentiell
  - parallel im Falle von Mehrprozessorsystemen
- Zugriffe auf gemeinsame Ressourcen sind zu synchronisieren

- verteilte Programmierung $\neq$ parallele Programmierung

ifak

Skalierbarkeit

- die Software arbeitet unabhängig von der Rechneranzahl

$$2 \leq \text{N(Rechner)} < \infty$$

- keine einzige Ressource ist nur in begrenzter Menge verfügbar
  - sehr idealisierte aber allgemeine Entwurfsphilosophie
  - betrifft Hardware- und Software-Ressourcen
- besondere Probleme bereitet dabei die Netzstruktur (Topologie)
  - sie ist irregulär und i.A. nicht flaschenhalsfrei . . .
    - sehr zum Unterschied verteilter paralleler Systeme
  - . . . und besteht aus leistungsinhomogenen Komponenten
    - Netztechnologie, Knotenrechner, Endsysteme

ifak

Skalierbarkeit (ff)

- der Entwurf muss die "Unbegrenztheit“ explizit berücksichtigen
- im wesentlichen stechen drei Maßnahmen hervor . . .
  1. Replikation von Daten
     - typischerweise Dateien
  2. Geheimlagerung (caching) von Daten
     - insbesondere Namen und Adressen
  3. Vervielfachung von Servern
     - auch funktionale Replikation
- . . . die jeweils Zusatzaufwand zur Konsistenzerhaltung bedingen

ifak

Fehlertoleranz

- Rechner fallen verschiedentlich aus und produzieren Fehler
  - eine Tatsache, die Hardware und Software betrifft
- Fehler können durch folgende Maßnahmen geduldet werden:
  1. Redundanz (redundancy) von Hardware
     - eine nahezu natürliche Erscheinung vernetzter Rechner
       - Spezielle "hot standby“-Lösungen sind nicht nötig
     - Rechner übernehmen Funktionen ausgefallener Rechner
       - "andauernde“ Verfügbarkeit von Dienstleistungen
  2. Wiederaufsetzen (recovery) von Software
     - bedeutet die Wiederherstellung des alten Arbeitszustands
       - nachdem ein Fehler festgestellt worden ist
     - geht einher mit der zyklischen Zustandssicherung
       - check pointing . . . check pointing . . . check pointing

ifak

Fehlertoleranz (ff)

- der (Software-) Entwurf muss explizit darauf ausgerichtet sein
  - roll back gesicherter Daten nach erkanntem Fehler
    - Fehlererkennung, permanenter Speicher
  - error recovery : Zurücksetzen zum letzten check point

ifak

Verfügbarkeit

- Hardware-Ausfälle müssen nicht zum Totalausfall führen
  - redundante Komponenten können ersatzweise einspringen
- Dienstleistungen bleiben trotz Fehlerfall weiterhin abrufbar
  - schlimmstenfalls besteht verminderte Leistungsfähigkeit
  - "taktvolle Degradierung“ (graceful degradation)
- dezentrale, verteilte Hardware-Strukturen sind "nur“ die Basis
  - die noch notwendigen Software-Maßnahmen sind beachtlich

ifak

Verfügbarkeit (ff)

- Aussagen über Ausfälle basieren auf Wahrscheinlichkeiten . . .
  - proportionales Maß effektiver Nutzungszeiten
- . . . die die Zuverlässigkeit des Systems bestimmen
  - dependability

ifak

Zuverlässigkeitsmaße

1. mittlere Zeit bis zum Fehler
   - mean time to failure (MTTF)
   - Zuverlässigkeit (reliability) des Systems
   - besonders hohe Zuverlässigkeit (ultrahigh reliability):

$$MTTF \leq 10^{-9} \text{ Fehler/h}$$

2. mittlere Zeit bis zur Fehlerbehebung
   - mean time to repair (MTTR)
   - Wartbarkeit (maintainability) des Systems

ifak

Zuverlässigkeitsmaße (ff)

3. mittlere Zeit zwischen zwei Fehlern
   - mean time between failure (MTBF)

     = MTTF + MTTR
   - Verfügbarkeit (availability) des Systems
   - hohe Verfügbarkeit:
     - lange MTTF und/oder kurze MTTR
   - MTBF = 0 entspricht nicht der Realität

ifak

Transparenz

- die Existenz bewusst separierter Komponenten verbergen
  - das System (weiterhin) als Gesamtheit auffassen
- verschiedenste Formen der "Durchsichtigkeit“ bestehen . . .
  1. Zugriffstransparenz (access transparency)
  2. Ortstransparenz (location transparency)
  3. Nebenläufigkeitstransparenz (concurrency transparency)
  4. Replikationstransparenz (replication transparency)
  5. Ausfalltransparenz (failure transparency)
  6. Migrationstransparenz (migration transparency)
  7. Leistungstransparenz (performance transparency)
  8. Skalierungstransparenz (scaling transparency)
- Netzwerktransparenz (network transparency) (Varianten 1 und 2)

ifak

Transparenz (ff)
Wichtige Eigenschaften

- Zugriffstransparenz – auf alle Objekte wird in gleicher Weise zugegriffen
- Ortstransparenz – der Ort, an dem sich ein Objekt befindet, ist für den Benutzer transparent; zwischen lokalen und im Netz verteilten Objekten wird nicht unterschieden.
- Nebenläufigkeitstransparenz – mehrere Anwender oder Anwendungsprogramme greifen gleichzeitig auf gemeinsame Objekte (z.B. Daten) zu

ifak

Transparenz (ff)
Netzwerktransparenz

- sorgt für die Anonymität der Betriebsmittel
  - vergleichbar zu den zentralisierten Systemen
- bezieht sich vornehmlich auf den Ressourcen-Zugriff:
  1. lokale und entfernte Zugriffe mit identischen Operationen
  2. Zugriffe erfolgen ohne Wissen der Lokalität
- Unix bietet i.A. keine (netzwerk-) transparente Sicht . . .
- . . . wie kein einziges kommerzielles Betriebssystem überhaupt

ifak

Transparenz (ff)

- Transparenz muss optional und nicht obligatorisch sein
  - es ist nicht immer wichtig, wo ein Prozess abläuft
    - wichtig ist, er läuft überhaupt (irgendwo) ab
  - es ist oft wichtig, wo ein Dokument gedruckt werden soll
    - dann ist aber auch wichtig, wo der Druckprozess abläuft
  - es kann wichtig sein, die Prozessoranzahl zu wissen . . .
    - oder die Verbindungsstruktur des Netzwerkes zu kennen
  - . . . um ein paralleles Programm effizient ablaufen zu lassen
- vereinzelte “Nicht-Transparenz“ ist erforderlich

ifak

Zusammenfassung Eigenschaften

- das Konzept des "resource sharing“ ist fundamental
  - es unterlegt alle anderen betrachteten Eigenschaften
  - es beeinflusst maßgeblich die Software-Architektur
- "resource“ steht für Software-/Hardware-Komponenten . . .
  - auf die Prozesse gemeinsam/koordiniert zugreifen können
- . . ."sharing“ impliziert eine globale Verwaltung

- Client/Server-Systeme spielen z.Z. die dominierende Rolle
  - Obgleich "Server“ für die zentrale Verwaltung steht . . .
  - . . . damit eigentlich nicht zu dezentralen Systemen passt
- objektbasierte Systeme sind flexibler und problemgerechter

- Transparenz ist wünschenswert, darf aber nicht zwanghaft sein

ifak

IPC – Interprocess Communication

- Client/Server- und Gruppen(mehrteilnehmer)kommunikation
- eine "Bedienungsanleitung“ für Netzwerkprotokolle
  - high-level Protokolle zur Prozessinteraktion
  - Grundlage bilden Transportprotokolle, z.B. TCP und UDP
  - kommunizierte Informationen stellen Datenströme dar
    - die keine Nachrichtengrenzen aufweisen
    - deren Transfer (i.A.) gepuffert erfolgt
  - Abstraktionen zum message passing
- die zwei zentralen Fragestellungen beziehen sich dabei auf . . .
  - 1. die Repräsentation zu übertragender Datenstrukturen
    - d.h. der Datentypen der Anwenderprogramme
  - 2. die Kommunikation zwischen entfernten Prozessen
    - problemorientierte, anwendungsbezogene Protokolle
- . . . End-to-End Arguments in System Design
  - Vermeidung redundanter Nachrichten und Funktionen

ifak

Abbildung der Datenelemente

- die Datenelemente bilden sich aus Datenstrukturen
  - d.h. sie stellen strukturierte Daten dar
  - dabei handelt es sich um einfache, elementare Datentypen . . .
    - char, int, unsigned, double
  - . . . aber auch um komplexe, dynamische Datenstrukturen
    - struct, union, class
    - Listen, Bäume, Graphen
- Nachrichten sind nichts weiter als linearisierte Daten
  - Datenstrukturen sind in eine „flache Form“ zu bringen . . .
    - sie werden vor dem Versand in einen Puffer kopiert
  - . . . und in ihren Originalzustand wieder hergestellt
  - sie werden nach dem Empfang aus einen Puffer kopiert
- eine externe Datenrepräsentation ist erforderlich
  - Nachrichten enthalten Daten beliebiger Typen
  - Typen sind nicht überall gleich repräsentiert

ifak

Datenrepräsentation

- Möglichkeiten, ein einheitliches Verständnis zu erlangen:
  1. Konvertierung der Daten in eine externe Darstellung
     - Sender wandelt lokale Daten um in die externe Form
     - Empfänger wandelt externe Daten um in die lokale Form
  2. verbindungsorientierte Kommunikation und Aushandlung
     - eine Umwandlung ist nur bedingt erforderlich . . .
       - wenn z.B. die Prozessoren inkompatibel zueinander sind
     - . . . ihre Notwendigkeit wird ausgehandelt
     - beim Verbindungsaufbau
  3. Attributierung der Nachrichten mit dem Prozessortyp
     - der Empfänger wandelt empfangene Daten um . . .
     - . . . wenn seine CPU zum angegebenen Prozessortyp passt
- in allen Fällen liegt eine externe Datenrepräsentation vor

ifak

Externe Datenrepräsentation

- zwei Kategorien werden unterschieden . . .
  1. getypte Datendarstellung in den Nachrichten
     - ASN.1 (abstract syntax notation) (CCITT-Standard)
     - XML
     - der Typ der Datenelemente ist genau spezifiziert
  2. ungetypte Datendarstellung in den Nachrichten
     - Sun XDR (external data representation), Xerox Courier
       - Firmenstandards
     - der Typ der Datenelemente ist nicht spezifiziert
       - spezifiziert wird die Struktur der Elemente
     - die Daten werden als Bytestrom übermittelt
- . . . die beide zu Lasten der Bandbreite gehen
  - wegen der zusätzlich zu transferierenden Informationen

ifak

XDR Nachrichten

- Nachrichten stellen "Quartett“-Sequenzen dar
  - Objekte fester Länge (4 Bytes)
  - folgende Konventionen werden eingeführt:
    1. Zahlen belegen ein Objekt (32 Bits)
    2. Zeichenfolgen von 4 Bytes Länge belegen ein Objekt
    3. Felder, Strukturen, Zeichenketten sind eine Bytefolge
       - mit einem anführenden Längenfeld ("offene Felder“)
    4. Zeichen liegen im ASCII vor

5	Sequenzlänge
„Ried“	Riedl
„l “	
7	Sequenzlänge
„Wahl“	Wahlitz
„itz“	
2010	unsigned

XDR Nachrichten (ff)

- weitere Konventionen legen fest:
  5. das höchstwertigste Ende eines Objektes
  6. ob Zeichen gepackt vorliegen
  7. welches von den 4 Bytes zuerst übertragen wird

❒ die feste Objektgröße senkt Verarbeitungskosten

- allerdings auf Kosten der Übertragungsbandbreite

ifak

Marshalling

- Eine "Datenkollektion“ zu einer Nachricht verpacken
  - umfasst zwei grundlegende Schritte . . .
    1. Linearisierung der Datenstrukturen
       - einfach bei statischen Datenstrukturen
       - aufwendig bei dynamischen Datenstrukturen
    2. Umwandlung in die externe Repräsentation
  - . . . die manuell oder automatisch vorgenommen werden
    - von Hand z.B. mit `sprintf()`:

```
const MAX = 1024;
char buf[MAX];
char* name = "Riedl", ort = "Wahlitz";
sprintf(buf, "%u%s%u%s%u",
        strlen(name), name,
        strlen(ort), ort,
        2010);
```

ifak

Marshalling (ff)

- automatisch i.A. nur im Zusammenhang mit Fernaufrufen

- der umgekehrte Vorgang bezeichnet sich mit "unmarshalling“
  - findet auf der Empfängerseite statt

- am Beispiel von XDR:

1. Spezifikation des Nachrichtenformates

```
const MAX = 8;
struct String {
    unsigned size;
    char data[MAX];
};
struct Message {
    String name;
    String place;
    unsigned year;
};
```

ifak

Marshalling (ff)

2. Zusammenstellen der Nachricht

```
char* name = "Riedl", ort = "Wahlitz";
Message buf;
buf.name.size = strlen(name);
strcpy(buf.name.data, name);
buf.place.size = strlen(ort);
strcpy(buf.place.data, ort);
buf.year = 2010;
```

- der Programmieraufwand ist noch recht erheblich

ifak

Kommunikationsprimitiven

- `send()` und `receive()` von (beliebigen) Nachrichten
  - in zwei möglichen Ausführungen . . .
    1. synchrone Kommunikation
       - Datentransfer erfolgt abhängig von Prozesszuständen
       - Sende- und Empfangsprozess blockieren dazu ggf.
       - Ende-zu-Ende Semantik
    2. asynchrone Kommunikation
       - Datentransfer erfolgt unabhängig von Prozesszuständen
       - Sende- und Empfangsprozess blockieren (i.d.R.) nicht
       - keine Ende-zu-Ende Semantik
  - . . . die Nachrichtenwarteschlangen bedingen
    - aber nicht unbedingt Nachrichtenpuffer

ifak

Kommunikationsprimitiven (ff)

- gepufferte Kommunikation impliziert blockierende Operationen
  - wenn Nachrichtenpuffer voll oder leer sind
  - wenn keine wiederverwendbaren Betriebsmittel frei sind . . .
  - . . . um konsumierbare Betriebsmittel zu verwalten
  - Eine "Auszeit“ (timeout) kann die Blockade aufheben
- "synchron“ muss nicht immer "blockieren“ bedeuten

ifak

Kommunikationsadressen

- repräsentieren den Bestimmungsort von Nachrichten
  - müssen im `send()/receive()` spezifiziert werden
- der Bezeichner (identifier) des Bestimmungsortes . . .
  - muss den Sendeprozessen bekannt sein
  - kann den Empfangsprozessen bekannt sein
- . . . identifiziert den Empfangsprozess direkt oder indirekt

ifak

Kommunikationsadressen (ff)

- drei Varianten sind gebräuchlich:
  1. direkte Kommunikation
     - von Prozess zu Prozess
  2. indirekte Kommunikation
     - von Prozess über Port zu Prozess
  3. verbindungsorientierte Kommunikation
     - von Prozess über Verbindung zu Prozess
     - die Verbindung besteht zwischen Ports
- trade-off zwischen Performanz und Flexibilität/Transparenz

ifak

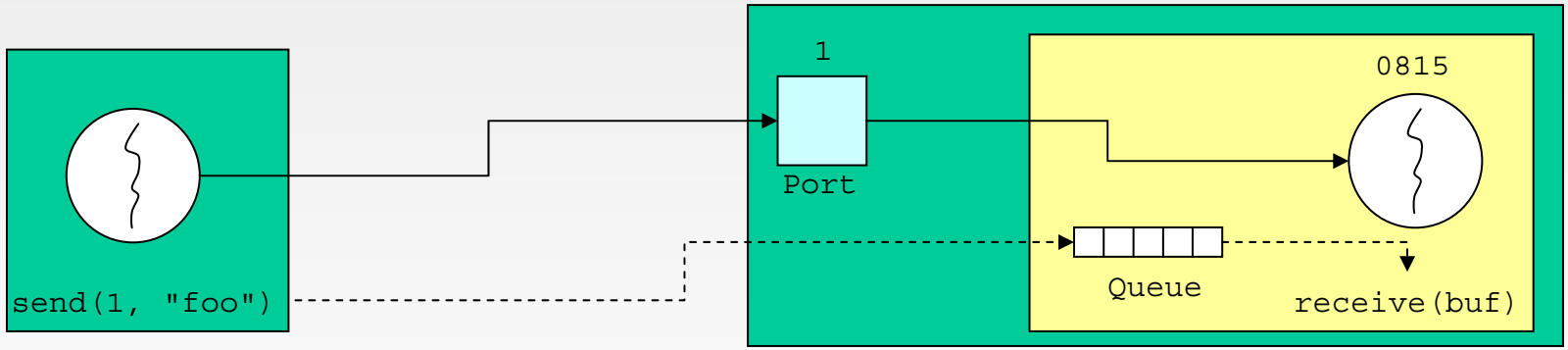
Ortsunabhängige Bezeichner

- repräsentieren (Ziel-) Adressen für zu sendende Nachrichten
  - formaler Parameter der `send()`-Primitive
- sind abzubilden auf ortsabhängige Bezeichner . . .
  - Aufgabe der Software zur Wegewahl (routing)
- . . . um Nachrichten ihrem Bestimmungsort zuzuführen
  - einem Prozess oder Briefkasten (mailbox) zuzustellen
  - ggf. durch ein Tor (port) weiterzuleiten
- die bezeichneten Objekte werden dadurch "umsetzbar“
  - Server können z.B. ihre Lokalität verändern . . .
  - . . . ohne Klienten den neuen Ort mitteilen zu müssen
- Prozessmigration unter Beibehaltung der Ortstransparenz

ifak

Port versus Mailbox

- der Port dient der Nachrichtenweiterleitung
  - mit ihm ist keine Nachrichtenwarteschlange verbunden

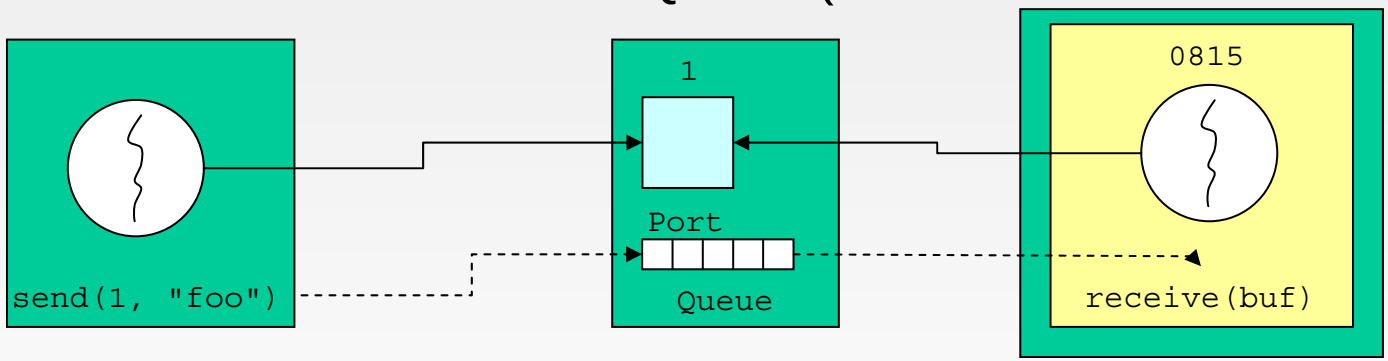


- die Nachrichten werden "unverzüglich“ dem Prozess zugestellt
  - der Prozesskontrollblock kapselt die Nachrichtenwarteschlange
- der Port verweist auf den ihm angebundenen Prozess
  - durch eine Adresse auf den Prozesskontrollblock . . .
  - . . . oder den (systemweit eindeutigen) Prozessbezeichner

ifak

Port versus Mailbox (ff)

- die Mailbox dient der Nachrichtenaufbewahrung
  - sie ist ein Port mit einer Queue (Nachrichtenwarteschlange)



- Prozesse müssen Nachrichten aus ihren Briefkästen abholen
  - dabei ist die Mailbox explizit zu spezifizieren

ifak

Nachrichtenbezeichner

- zur Erhöhung der Zuverlässigkeit der Kommunikation
  - zuverlässige Nachrichtenzustellung (reliable message delivery)
  - request-reply Kommunikation zwischen Client und Server
- die eindeutige Identifikation der Nachrichten ist zweiteilig:
  1. Kennung der Anforderung (request id)
     - bei jedem send() um 1 erhöhter Zählerwert
     - wird vom Sendeprozess generiert
  2. Kennung des Sendeprozesses (sender id)
     - Prozess- oder Portbezeichner
       - um eventuelle Rücknachrichten empfangen zu können
     - Adresse (d.h. Parameter) der reply() Primitive

ifak

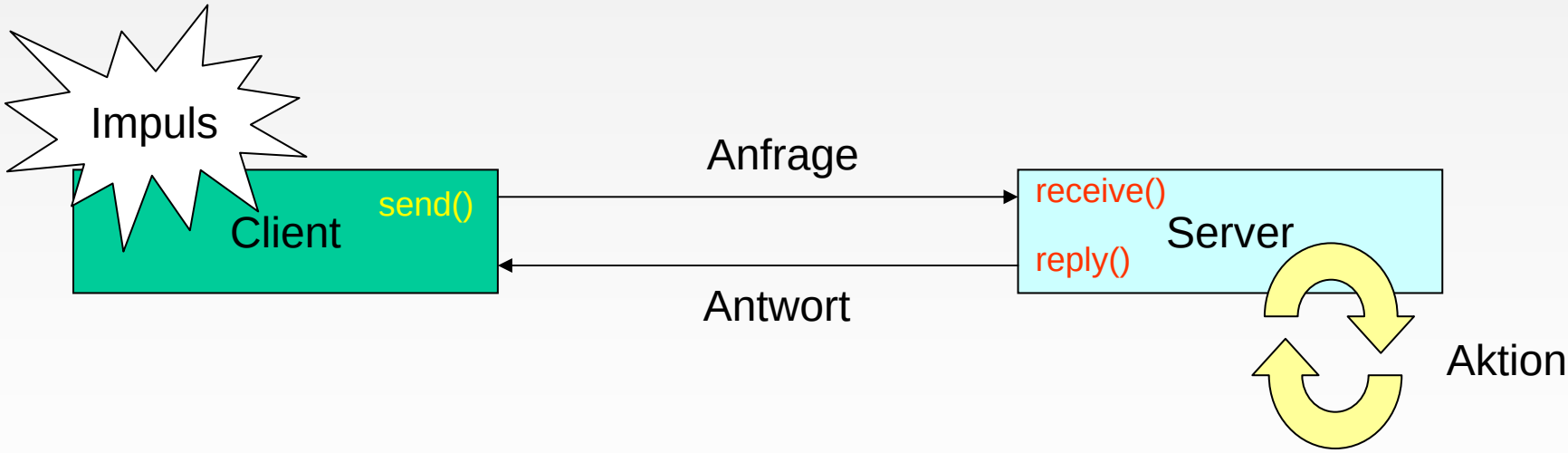
Nachrichtenbezeichner (ff)

- die request_id macht die Nachricht lokal eindeutig
  - allerdings nur für den jeweiligen Sendeprozess
- die sender_id macht die Nachricht global eindeutig
  - speziell für den jeweiligen Empfangsprozess
  - generell für das verteilte System

ifak

Client-Server-Kommunikation

- „Normalform“ ist synchrone Kommunikation
  - der Client fordert, blockiert und wartet auf eine Antwort
  - request-reply



Client-Server-Kommunikation (ff)

- request-reply Protokolle basieren auf drei Primitiven:
  1. `send()` (bzw. DoOperation())
     - nicht-blockierender Versand der request-Nachricht
     - blockierender Empfang der reply-Nachricht
  2. `receive()` (bzw. GetRequest())
     - blockierender Empfang der request-Nachricht
  3. `reply()` (bzw. SendReply())
     - nicht-blockierender Versand der reply-Nachricht

ifak

Aufbau einer request-reply-Nachricht

- unterscheidet typischerweise vier Einträge:
  1. Nachrichtentyp
     - request oder reply
  2. Nachrichtenbezeichner
     - sendeseitig generierte request id
  3. Operationskode
     - Kennung der empfangsseitig auszuführenden Funktion
  4. Parameter
     - linearisierte Datenstrukturen (aktuelle Ein-/Ausgabewerte)
- beschreibt einen Fernaufruf (*remote procedure call*, RPC)

ifak

Protokollvarianten

1. request (R) Protokoll
   - wird eingesetzt, wenn kein "Rückgabewert“ erwartet wird
   - der Client setzt seine Ausführung nach dem Versand fort
2. request-reply (RR) Protokoll
   - spezielle Bestätigungsnachrichten werden nicht benötigt
     - das reply() bestätigt implizit die request Nachricht
     - ein weiteres send() bestätigt die reply Nachricht
   - nur zwei Nachrichten werden ausgetauscht
3. request-reply-acknowledge reply (RRA) Protokoll
   - die reply Nachricht wird explizit bestätigt
     - die request id der reply Nachricht wird zurückgeschickt
     - Blockbestätigung für alle "niederwertigen“ replies
   - drei Nachrichten werden ausgetauscht

ifak

Mögliche Fehler

- duplizierte *request* Nachrichten erkennen und ignorieren
  - duplicate suppression protocol
  - Problem sind die möglichen "Auszeiten“ (timeouts) . . .
  - . . . d.h. die explizit wiederholten request Nachrichten
- verlorene *reply* Nachrichten behandeln
  - eine ggf. bereits gesendete Antwort nochmals anfordern
    - den *request* dabei nur bedingt noch einmal bearbeiten
  - idempotente Operationen bzw. Zustandsfreiheit . . .
    - . . . Buch über bereits geschickte *reply* Nachrichten führen
- alles Probleme mit Ende-zu-Ende-Relevanz

ifak