Message Passing

Message passing: Ein Beispiel der Kommunikation bei der Nachrichten vom Sender zu einem oder mehreren Empfängern gesendet werden → invocation, signals, data packets

Eigenschaften der Versandsarten:

- verlässliche Übertragung
- garantierte Übertragung
- unicast, multi-/broadcast, client-server, all-to-all
- synchron, asynchron

Persistente Kommunikation:
Nachricht solange gespeichert bis Empfänger bereit (z.B. E-Mail).

→ asynchron oder synchron

Transiente Kommunikation:
Nachricht gespeichert, solange sendende und empfangene Applikation ausgeführt werden (z.B. Router, Socket).

→ asynchron oder synchron

Asynchrone Kommunikation:
Sender wird fortgesetzt nach dem er seine Nachricht zur Übertragung weitergegeben hat (z.B. E-Mail, Diskussionsforum).

Synchrone Kommunikation:
Sender wird blockiert bis Nachricht an Empfänger übermittelt wurde oder Puffer des Empfängers die Nachricht übernommen hat (z.B. Telefonie, Instant Messaging, Internet-Videokonferenz).

Weitere transiente Kommunikationsformen:
Auslieferungsbasierte und antwortbasierte transiente synchrone Kommunikation.

Message Passing Interface (MPI): Standard, der Nachrichtenaustausch bei parallelen Berechnungen auf verteilten Computersystemen beschreibt → kein konkretes Protokoll

Nachrichtenverarbeitung

Nachricht enthält:

- **ID**,
  identifiziert Nachricht (z.B. GET, POST) → nicht immer benötigt
- **Argumente**,
  können Datentypen sein (Int, String, etc.) → Vorteil ist Argumente können direkt über `DataInputStream` und `DataOutputStream` gelesen und geschrieben werden → können zusätzliche Infos enthalten

---

Prinzipien der Nachrichtenverarbeitung:

- Trennung zw. Kommunikations- und Applikationsdetail
- Verknüpfung von Nachrichten mit Methoden der Applikationsobjekte

---

Fabrikmethode,

> **Zweck**: Erzeugung von Objekten an Unterklasse delegieren
> **Nachrichtenverarbeitung mit Fabrikmethode**: Basic Message definiert Protokoll (Klassenstruktur) und sorgt für Senden und Empfangen. Basic Message Handler erzeugt Nachricht → Alle Netzwerkteilnehmer benötigen diese Struktur

Basic Message,

- besitzt Methoden: `messageId`, `argList`, `readArgs`, `writeArgs`
- hat Getter/Setter für ID und Argumente
- beinhaltet abstrakte Methode `operate`

Basic Message Handler,

- besitzt Methoden: `sendMsg`, `readMsg`, `buildMessage`
- sendet/empfängt Nachrichten an/von Netzteilnehmern
- stellt das globale Message-Handler Objekt `current` zur Verfügung

Message,

- besitzt Methoden: `messageId`, `argList`, `readArgs`, `writeArgs`
- hat Getter/Setter für ID und Argumente
- beinhaltet abstrakte Methode `operate`, `handles` und `newCopy`

Protokolle

Fixe Protokolle:
Menge der möglichen Kennung (IDs), die möglichen Argument (Anzahl + Typ) **vor** einer

Kommunikationssitzung bekannt → keine Änderungen während Kommunikation

Adaptive Protokolle:
Nachrichten können während Laufzeit ändern.

- Änderung Länge der Argumentliste
- Änderung der Argumenttypen
- Änderung des Nachrichtentyps

→ häufiger als fixe Protokolle

Prototyp

Durch `Prototype` werden eine Menge von Objekten zur Verfügung gestellt, die durch Aufruf von `clone` wieder zur Objekterzeugung herangezogen werden können.

- Message definiert das Protokoll
- Message Handler erzeugt die Nachricht anhand einer Liste der Nachrichtentypen