

# H Ice

---



## 1 Motivation

---

- CORBA
  - ◆ komplexe Mechanismen
    - durch Standardisierungsprozedur Trend zu multiplen Lösungsansätzen
  - ◆ schleppende Integration von Verbesserungen
    - Kompatibilität muss erhalten bleiben
  - ◆ problematische Vorgaben des Standards
    - z.B. IIOP/CDR-Kodierung und Padding, IIOP/Zeichensatzangaben zum Transfer von Strings
- DCOM, .Net
  - ◆ beschränkt auf Microsoft-Systeme
- Web-Services
  - ◆ miserable Performanz
  - ◆ kaum standardisierte Infrastruktur



# 1 Motivation (2)

---

- ★ Gründung von ZeroC Inc. durch einige Middleware-Veteranen
  - ◆ z.B. Michi Henning
  
- ★ Entwicklung der *Internet Communication Engine (Ice)*
  - ◆ objektbasierte Middleware
  - ◆ geeignet für heterogene Plattformen
  - ◆ effiziente Kommunikation
  - ◆ Vermeidung unnötiger Komplexität
  - ◆ vollständige Implementierung von Werkzeugen und Diensten zur Unterstützung gängiger Problemlösungen



# 2 Architekturüberblick

---

- Client-Server-basiertes Modell
  - ◆ monolithisches Modell: Objekt immer an einem Ort
  
- CORBA-nahe Architektur
  - ◆ viele Konzepte übernommen
  
- Ice-Objekt
  - ◆ Instanz beantwortet Aufrufanfragen von Clients
    - lokale oder entfernte Aufrufe von Operationen (*Operations*)
  - ◆ ein oder mehrere Schnittstellen
    - genannt *Facets*
  - ◆ eindeutiger Identifikator
    - z.B. UUID



## 2.1 Proxy

---

- Client-seitiger Stellvertreter: Proxy
  - ◆ Lokalisierung des Ice-Objekts
  - ◆ Aktivierung des Ice-Objekts, falls erforderlich
  - ◆ Transport der Parameter
  - ◆ Warten auf Ergebnisse
  - ◆ Rückgabe der Ergebnisse an Aufrufer
  
- Proxy kapselt Zugangsinformation zum Ice-Objekt
  - ◆ Kommunikationsadresse
  - ◆ Objektidentifikator
  - ◆ Facet-Name zur Identifikation der Objektschnittstelle



## 2.1 Proxy (2)

---

- Objektreferenz als String
  - ◆ Darstellung der Zugangsinformation als String (*Stringified Proxy*)
    - z.B. **MyPrinter:tcp -p 4711 -h server.hurz.de**
  - ◆ Umwandlung des Strings in Proxy
  - ◆ Umwandlung einer Proxy-Referenz in String
  
- Direkte Proxies
  - ◆ Zugangsinformation bekannt
  - ◆ wird direkt verwendet
  
- Indirekte Proxies
  - ◆ Zugangsinformation nicht bekannt
    - lediglich Identifikator vorhanden
  - ◆ Ermittlung der Zugangsinformation durch Ortsdienst



## 2.2 Aufrufe und Aufrufsemantik

---

- At-most-once-Semantik
  - ◆ ... für den Normalfall
- Agressivere/einfachere Fehlerbehandlung im Falle von
  - ◆ Operationen, die den Objektzustand nicht verändern (*nonmutable*)
  - ◆ Operationen, die idempotent sind (*idempotent*)
- ◆ Effekt: weniger Fehler für den Aufrufer, aber abgeschwächte Aufrufsemantik



## 2.2 Aufrufe und Aufrufsemantik (2)

---

- One-Way-Aufrufe
  - ◆ keine Antwort für den Aufrufer (vgl. CORBA)
    - auch nicht im Fehlerfall
- Datagram-Aufrufe
  - ◆ wie One-Way
  - ◆ zusätzlich Verwendung von UDP-Transport
    - Nachrichtenverlust möglich
- Batched-One-Way, Batched-Datagram
  - ◆ Zusammenfassung mehrerer One-Way- oder Datagram-Aufrufe
  - ◆ nur eine Transportnachricht
  - ◆ sequentielle Ausführung auf Server-Seite



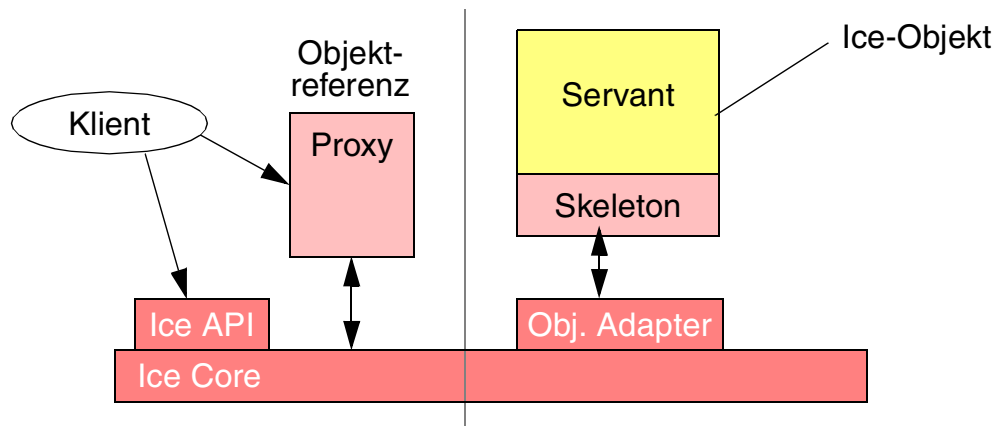
## 2.3 Servant

- Instanz zur Implementierung einer Objektschnittstelle
  - ◆ ein Servant pro Ice-Objekt (Ice-Facet)
    - Normalfall (vgl. CORBA)
  - ◆ mehrere Servants pro Ice-Objekt (Ice-Facet)
    - mehrere Zugangspunkte für einen Proxy
    - randomisierter Zugang (Lastbalancierung)
    - Nutzung genau eines Zugangspunktes
  - ◆ mehrere Ice-Objekte (Ice-Facets) pro Servant
    - Servant schlüpft in jeweilige Objektrolle
    - Kontextinformation erlaubt Unterscheidung



## 2.4 Architekturüberblick

- Überblick über die Ice-Komponenten



Bereitstellung durch: ■ Middleware ■ Tools ■ Objektentwickler

- ◆ starke Ähnlichkeit mit CORBA
  - einfacherer Aufbau



## 2.5 Schnittstellenbeschreibung

---

- Plattformunabhängige Beschreibung der Schnittstellen
  - ◆ Slice – Specification Language for Ice
  - ◆ Abbildung von Slice-Definitionen für jeweilige Implementierungssprache
- Sprachabbildung (Language Mapping)
  - ◆ für C++, Java, C# .Net (*Icicle*), VB .Net, Python, PHP (nur Client-Seite)
  - ◆ unterschiedliche Implementierungssprachen auf Client- und Server-Seite möglich



## 2.6 Dienste

---

- IcePack
  - ◆ Ortsdienst
    - Abbildung von Object-Adapter-Namen auf konkrete Zugangsdaten
  - ◆ Aktivierungsdienst
    - Erzeugung von Serverprozessen und Servants bei Bedarf
    - Deployment-Deskriptoren für Anwendungen (vgl. EJB)
  - ◆ Namensdienst
- IceBox
  - ◆ dynamisches Laden von Anwendungskomponenten
- IceStorm
  - ◆ Ereignisdienst zur Weiterleitung von Ereignissen an Menge von Abonnenten



## 2.6 Dienste (2)

---

- IcePatch
  - ◆ Dienst zur automatischen Aktualisierung von Anwendungssoftware
- Glacier
  - ◆ Firewall und Verschlüsselungsdienst für Ice-Anwendungen



## 3 Slice

---

- Schnittstellenbeschreibungssprache
  - ◆ angelehnt an Java und CORBA IDL
  - ◆ reduzierte Komplexität im Vergleich zu IDL
  - ◆ Module (*module*), Schnittstellen (*interface*) und Exceptions (*exception*) sind vergleichbar mit CORBA IDL
  - ◆ Operationen, **in**-, **out**-Parameter wie in CORBA
    - **in**-Parameter immer ohne **in**-Schlüsselwort
    - keine **inout**-Parameter
    - **out**-Parameter müssen hinter **in**-Parametern angeordnet werden
  - ◆ keine Attribute
    - stattdessen explizite Zugriffsoperationen



## 3 Slice (2)

---

### ■ Beispiel: Kontoschnittstelle

```
exception WrongAccountNumber { string errorMsg; };

interface Account
{
    nonmutating double getBalance();

    double withdraw( double amount )
        throws WrongAccountNumber;
    double deposit( double amount );
        throws WrongAccountNumber;
};
```

- ◆ verteilte Ice-Objekte können **Account**-Schnittstelle implementieren
- ◆ Unterschiede zu CORBA markiert



## 3.1 Bezeichner

---

### ■ Regeln für Bezeichner

- ◆ kein Unterstrich erlaubt
  - wird stattdessen in Language-Mappings für besondere Bezeichner verwendet
- ◆ Schlüsselworte können als Bezeichner auftreten
  - Backslash muss vorangestellt werden, z.B. **\interface**
- ◆ Bezeichner darf nur einmal pro Bereich genutzt werden
  - Groß-, Kleinschreibung wird unterschieden
  - alternative Schreibungen sind jedoch verboten (wie bei CORBA)



## 3.2 Basistypen

---

- Integer-Datentypen
  - ◆ **byte** (8 Bit), **short** ( $\geq 16$  Bit), **int** ( $\geq 32$  Bit), **long** ( $\geq 64$  Bit)
  - ◆ keine **unsigned**-Typen
  - ◆ keine Zeichentypen
- Strings
  - ◆ Stringtyp **string**
  - ◆ Unicode-Zeichensatz
  - ◆ keine Längenangabe, keinen Nullstring
- Fließkomma-Datentypen
  - ◆ **float**, **double**
- Boolescher Typ
  - ◆ **bool**



## 3.3 Komplexe Typen

---

- Komplexe Datentypen
  - ◆ Strukturen: **struct** *name* { *memberdef* }
    - *name* kann wie Typname benutzt werden
    - keine verschachtelten Strukturdefinitionen
  - ◆ Aufzählungstyp: **enum** *name* { *constantdef* }
    - Konstanten werden im gleichen Namensraum abgelegt
  - ◆ Sequenztyp: **sequence** < *basetype* >
    - tatsächliche Länge wird in sprachabhängiger Form gesetzt und abgefragt
    - **sequence**-Typ kann sich nicht selbst enthalten (im Gegensatz zu CORBA)



## 3.3 Komplexe Typen (2)

---

### ■ Neue Datentypen

#### ◆ **dictionary**-Typen

- repräsentieren Name-Wert-Paare, z.B. Personalnummer zu Personendatensatz
- Beispiel: **dictionary**<**long**, `Employee`> `EmployeeMap`;
- erster Typ: Typ des Namen
  - beliebiger integraler Typ oder **string** bzw. **sequence** oder **struct** aus lediglich integralen oder **string** Komponenten
- zweiter Typ: Typ des Werts
  - beliebiger Typ
- Repräsentation in einer Sprache abhängig vom Language-Mapping
- Typ definiert lediglich Übergabemöglichkeit der Daten als Parameter



## 3.3 Komplexe Typen (3)

---

- ▲ Wichtig: alle komplexen Datentypen müssen erst als Typdefinition deklariert werden bevor sie benutzt werden können!
- Unterschiede: *Slice* im Vergleich zu CORBA
  - ◆ keine **typedef**-Typen, keine **fixedwidth**-Typen
  - ◆ keine Array-Typen
  - ◆ keine **union**-Typen
    - stattdessen Einsatz so genannter **class**-Typen

## 3.4 Konstanten

---

### ■ Definition von Konstanten

- ◆ erlaubt für integrale Type, Fließkommatypen und **string**
- ◆ im Gegensatz zu CORBA keine Operatoren/Ausdrücke erlaubt



## 3.5 Schnittstellen

---

- **interface**-Konstrukt wie bei CORBA
  - ◆ nur Operationen, keine Attribute
  - ◆ **out**-Parameter hinter **in**-Parametern
    - **in**-Parameter ohne Schlüsselwort
  - ◆ innerhalb von **interface**-Definitionen keine Typdefinitionen möglich
  
- Markierung von Operationen
  - ◆ **nonmutating**: Operation ändert Zustand des Objekts nicht
  - ◆ **idempotent**: Mehrfachausführung der Operation hat selben Effekt
  - ◆ Wirkung:
    - Auswirkung auf Language-Mapping: z.B. in C++ wird **nonmutating** Operation zu **const**-Funktion
    - Auswirkung auf Erzielung der At-most-once-Semantik: einfachere Fehlerbehandlung



## 3.5 Schnittstellen (2)

---

- Exceptions
  - ◆ Deklaration wie in CORBA
    - **struct**-ähnlicher Typ mit Komponenten
    - **throws** statt **raises** bei der Deklaration einer Operation
  - ◆ über CORBA hinaus:
    - Vererbung zwischen Exceptions
    - z.B.

```
exception BaseException { string reason; };  
  
exception LogicException extends BaseException  
{ int errnr; };
```
  - ◆ aktuell geworfene Exception kann spezieller als bei Operation deklarierte sein



## 3.5 Schnittstellen (3)

---

### ■ Vererbung

#### ◆ (mehrfache) Vererbung zwischen Schnittstellen

- Beispiel:

```
interface LimitedAccount extends Account
{
    void setLimit( double newLimit );
};
```

#### ◆ kein „Overriding“, kein „Overloading“

#### ◆ in erbender Schnittstelle darf kein Name zweimal auftreten

- auch nicht aus zwei verschiedenen Basis-Schnittstellen geerbte Namen

#### ◆ Diamond-Shape-Inheritance möglich (vgl. CORBA)

#### ◆ implizites Erben von **Object**



## 3.5 Schnittstellen (4)

---

### ■ Aufruf an einer Schnittstelle

#### ◆ Übergabe von Basistypen und komplexen Typen: **Call-by-Value**

#### ◆ Übergabe von **interface**-Typen, Parameter deklariert mit „\*“ nach dem Typnamen:

- z.B. **interface** ifc { **void** meth(Account\* acc); };
- Call-by-Copy-of-Proxy; de facto: **Call-by-Reference**

#### ◆ entspricht CORBA, benötigt jedoch „\*“ hinter Typnamen

- „\*“ zeigt an, dass Proxy vorliegt

