

Mobile Code

**Georg Lukas
2003-01-29**

Seminar "Ubiquitous Computing"

Gliederung

- Wozu Mobile Code?
 - ◆ Motivation + Einführung
- Paradigmen
 - ◆ und Kuchen backen!
- Mobile Code Systems
- Sicherheitsaspekte
- ausgewählte Programmiersprachen
 - ◆ JAVA, bei Zeit auch Telescript und Safe-Tcl
- Zusammenfassung
- Fragen / Diskussion

Einführung

Wozu Mobile Code?

Motivation

- rasantes Wachstum von Computernetzen
 - ◆ (Internet, Firmen-Intranets)
 - ◆ trotz Krisenstimmung
- günstige Geräte mit Netzanbindung
 - ◆ APs, DSL-Router, SetTop-Boxen
- drahtlose Netze
 - ◆ schlanke Endsysteme
- Skalierbarkeit von Client-/Server begrenzt
- Konfigurierbarkeit für unterschiedliche Ansprüche erforderlich
 - ◆ Remote-Wartung von N devices vs. Status-Übersicht für DAU
- Kombination bestehender Methoden und Paradigmen
 - ◆ CORBA: RPC und OOP
 - ◆ Flexibilität nicht hinreichend
 - ◆ kein Einfluss auf Ausführungsort
 - ◆ mehr Vert. Syst. als MCS
- Bedarf an Plattform- und Architektur-überschreitenden Technologien
- -> Code Mobility

Code Mobility

- "Code-Mobilität ist die Fähigkeit, die Bindung zwischen Code-Fragmenten und dem Ort der Ausführung dynamisch zu verändern" - *Carzaniga, Picco und Vigna*
 - ◆ Code auf Host, Ausführung auf anderem Host
- höhere Flexibilität im Software-Design
- einfachere Administration durch zentrale Speicherung

- Anwendungen existieren bereits
 - ◆ älteste davon:
 - ◆ PostScript (1984)
 - ◆ SQL (1989)
 - ◆ ...

- Beschränkungen auf Sprach- und Systemebene
 - ◆ später MCS-Strukturierung, meist nur Teile implementiert
- wissenschaftliche Forschung nur wenig ausgeprägt
- Basis-Terminologie fehlt / ist ungenau
 - ◆ unklare / überlagerte Bezeichnungen
 - ◆ *mobile agent*
 - letzter Vortrag: Agenten auf Marktplätzen mit Reputation
 - Thread / Code-Fragment / KI: "intelligenter" Agent
 - ◆ *code mobility / mobile code / mobile computations / mobile object systems / program mobility*
 - Code-Mobilität
 - eigene Verwendung: MC=Code CM=eigenschaft MCS=systeme

- Präsentationsziel: objektive analytische Betrachtung des Gebiets

- -> Abgrenzung VS vs. MCS

Typische Verteilte Systeme

- Abstraktion autonomer Rechner zu einem System
- Migration von Prozessen oder Objekten
 - ◆ meist transparent
 - ◆ Load-Balancing
- Fehlertoleranz
 - ◆ ...durch Redundanz
- Netze mit kleinem Umfang
 - ◆ Homogenität
 - ◆ hohe Bandbreite
 - ◆ (bekannte durchschnittliche Latenzzeiten)
 - ◆ geschützte Umgebung

Mobile Code-Einsatzgebiete

- *code mobility* im internet-Umfang
 - ◆ internet != Internet, ausgedehnte Netze
 - ◆ heterogene Hosts
 - ◆ unterschiedliche Autoritäten
 - Host-Besitzer
 - ◆ unterschiedliche Vertrauensebenen
 - Vertrauen ggü. Gegenstelle != Vertrauen ggü. MitM
 - ◆ unbekannte Bandbreiten
 - ◆ Verlust der Kommunikation möglich
- *location awareness*
 - ◆ Abstraktion des Ausführungsorts
 - ◆ starker Einfluss auf Design und Implementierung
- Code-Mobilität unter Programmierer-Kontrolle
 - ◆ Mechanismen zum Verschicken und Anfordern von Code
 - ◆ Unterstützung durch Laufzeitumgebung
- vor Editor, MCS-Auswahl:
- Software-Design
- -> Paradigmen-Auswahl

Paradigmen

Konzept

- Auswahl von Paradigmen
 - ◆ Vor- und Nachteile für eigene Anwendung abwägen
 - ◆ Betrachtung der Skalierbarkeit
 - ◆ Berücksichtigung der Interaktionskosten
- Komponenten
 - ◆ *code components* (enthalten Know-How für Berechnungen)
 - ◆ *resource components* (repräsentieren Daten oder Geräte)
 - ◆ *computational components* (können Code ausführen)
- Interaktionen
 - ◆ Ereignisse zwischen zwei oder mehr Komponenten
- Sites
 - ◆ beherbergen Komponenten
 - ◆ unterstützen Ausführung von *computational components*
 - ◆ intuitiver Begriff für Ausführungsort
 - ◆ site-lokale Interaktionen "günstiger" als zwischen sites
- Ausführung erfordert Anwesenheit auf einer Site:
 - ◆ Code-Komponente mit dem Know-How
 - ◆ alle benötigten Ressourcen-Komponenten
 - ◆ zuständige Rechenkomponente
- --> Vorstellung typischer Paradigmen
- A und B sind computational components
- A braucht Ergebnis einer Berechnung
- Alice und Bob / computational comp.
 - ◆ Rezept=Code, Herd, Zutaten=Ressourcen

Client-Server / Code on Demand

- Client-Server (CS)
 - ◆ keine Code-Mobility, aber verbreitet und bekannt
 - ◆ Alice möchte Kuchen, Bob hat Rezept, Zutaten und Herd
 - ◆ Client A benötigt Ergebnis einer Berechnung
 - ◆ Server B besitzt Code und Ressourcen
 - ◆ A fordert bei B Ergebnis an
 - ◆ B führt Berechnung aus
 - ◆ B liefert Ergebnis an A zurück

 - ◆ Server-Schnittstelle statisch
 - ◆ Engpässe beim Server möglich
- Code On Demand (COD)
 - ◆ Alice hat Zutaten und Herd, Bob hat Rezept
 - ◆ A hat Ressourcen- und Rechen-Komponente, B den Code
 - ◆ A fordert von B den Code an
 - ◆ B schickt Code zu A
 - ◆ A führt Code aus und erhält das Ergebnis

 - ◆ zentrale Speicherung von Code
 - ◆ einfache Wartung

Remote Evaluation / Mobile Agent

- Remote Evaluation (REV)
 - ◆ Alice hat Rezept, Bob hat Zutaten und Herd
 - ◆ A hat Code-Komponente, Ressourcen befinden sich bei B
 - ◆ A schickt Code zu B
 - ◆ B führt Code aus
 - ◆ B liefert Ergebnis an A zurück

 - ◆ angelehnt an Client-Server
 - ◆ flexiblere Verarbeitung der Daten möglich
 - ◆ höhere Belastung für B

- Mobile Agent (MA)
 - ◆ Alice hat Rezept und Teil d. Zutaten, Bob hat Herd
 - ◆ A hat Code-Komponente, benötigt aber eine Ressource von B
 - ◆ Ressource kann nicht zu A transportiert werden
 - ◆ kommt der Berg nicht zum Prophet...
 - ◆ A bewegt sich mit benötigten eigenen Komponenten zu B
 - ◆ A führt bei B den eigenen Code aus und benutzt B's Komponente

 - ◆ "fire and forget"-Strategie
 - ◆ effizientere Bandbreitennutzung

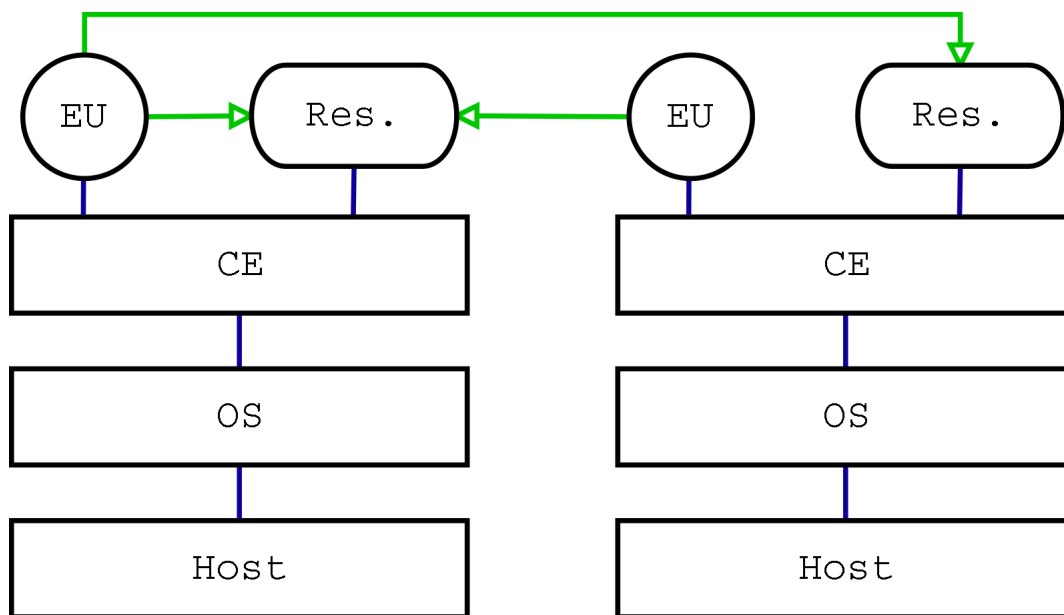
- gute Technologie nicht hinreichend für gutes Design
- Verbindung zwischen Technologie und Design
- alle Sprachen benutzbar
 - ◆ starke Mobility in C -> eigener Interpreter, Sicherung und Übertragung des Zustands

Mobile-Code-Systeme

MCS-Grundlagen

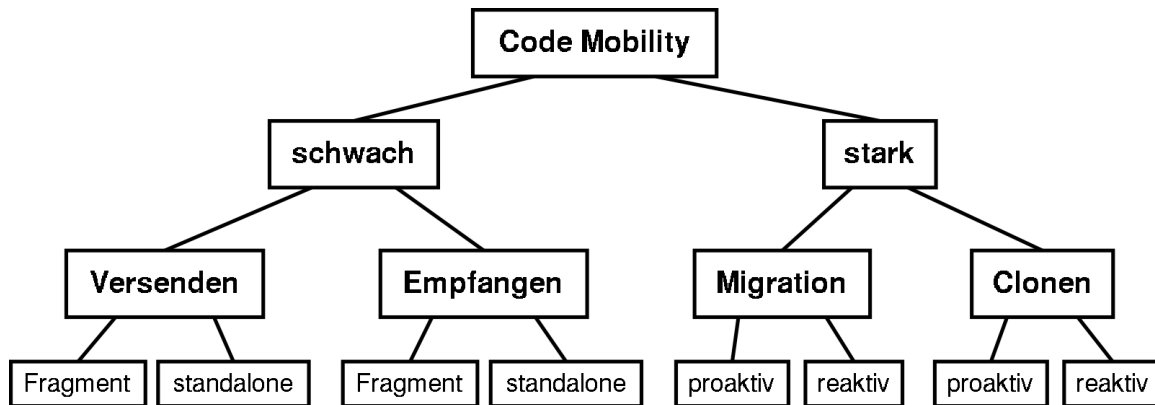
- "Mobile Code-Systeme sind Sprachen und Umgebungen, die Mechanismen zum Unterstützen von Code-Mobilität anbieten"
- **CE: Computational Environment**
 - ◆ "Rechenumgebung"
 - ◆ Plattform für EUs und Ressourcen
 - ◆ bildet *location*
 - Code muss Ausführungsort kennen (location awareness)
 - ◆ Abstraktion von Netzwerkprotokollen, Migrationsmechanismen
- **EU: Executing Unit**
 - ◆ Prozess oder Thread
 - ◆ läuft in einem CE
 - ◆ Zusammensetzung:
 - *code segment*
 - *data space* (Ressourcen-Referenzen)
 - lokale Variablen, Referenzen
 - *execution state* (Kontrolldaten, Stack, ...)
 - VM-Daten über Prozess
- **Ressourcen**
 - ◆ Elemente, die von mehreren EUs gemeinsam verwendet werden können
 - ◆ Dateien, Peripherie-Geräte
 - ◆ alles was shared werden muss
 - ◆ Bestandteil eines CEs
 - ◆ nicht zwingend auf selbem CE

Zusammenhang zwischen CEs, EUs und Ressourcen



- Schichten: CE auf OS auf Host
- Execution Units und Ressourcen auf CE
- Res.-Referenzen (grün) über CE-Grenzen hinweg
- Res.-Sharing

Code-Mobility-Mechanismen



- **Gliederungskriterien für MCSs**
 - ◆ Unterstützung für welche der Mechanismen?
- **schwache Mobilität: Übertragung von Code**
 - ◆ Versenden / Empfangen: aktive Seite verschickt/führt aus
 - ◆ standalone: ganze Programme
 - ◆ Fragmente: Objekte/Befehlsfolgen
- **starke Mobilität: Migration laufender Prozesse**
 - ◆ Migration / Clonen
 - ◆ proaktiv: selbst bewegen; reaktiv: bewegt werden
 - ◆ Bindungen nach Migration ungültig
 - ◆ Ressourcen-Management erforderlich
- -> Ressourcen-Management

Ressourcen-Management

- Aufgabe: Ressourcen-Verfügbarkeit nach Migration
- System muss entscheiden, abhängig von:
- Anwendungsseitige Ressourcen-Bindung
 - ◆ über Typ (durch gleichwertige Ressource ersetzbar)
 - durch beliebige Ressource gleichen Typs ersetzbar
 - sinnvoll für Ressourcen, die auf jedem CE vorhanden sind
 - lokale Konfiguration, Temporärspeicher
 - ◆ über Wert (durch identische Ressource ersetzbar)
 - bei Migration durch Ressource mit gleichem Wert und Typ ersetzbar
 - Duplikation anwendbar
 - Programm-Konfiguration
 - ◆ über Namen (Ressource eindeutig bestimmt)
 - eindeutig bestimmte Ressource
 - bei Migration nicht durch gleichwertige Ressource ersetzbar
 - Datei, Nutzer-Interface
- Ressourcen mit eingeschränkter "Bewegungsfreiheit"
 - ◆ *transferrable / not transferrable*
 - techn. Möglich (Datei vs. Drucker)
 - ◆ *transferrable: free / fixed*
 - fixed = nicht sinnvoll / wünschenswert (vertraulich?)
- daraus resultierend:
- Verfahrensweisen bei Migration:
 - ◆ verwerfen
 - ◆ wiederherstellen (als Netzwerkbindung)
 - ◆ Re-Binding
 - mit Res. gleichen Typs
 - ◆ duplizieren / verschieben
 - Ressourcen zusammen mit Code bewegen
- Art von Ressourcen und Bindungen oft durch Sprache / Implementierung begrenzt oder vorgegeben
- Klassifizierung von MCS danach
- -> bei Auswahl auch zu beachten: Sicherheit

Sicherheit von Mobile Code

Sicherheitsanforderungen an MCSs

- Mobile Code nicht immer aus zuverlässigen Quellen
- Einschränkung der Ausführung
- *safety*
 - ◆ Bugs in Anwendungen dürfen unabhängige Software nicht beeinflussen
 - Betriebssystemebene
 - Einschränkungen durch Programmiersprache und Objektcode
 - Laufzeitüberprüfung (z.b. Array-Grenzen)
- *security*
 - ◆ Sicherheit gegenüber Mißbrauch
 - ◆ *safety* notwendig, aber nicht hinreichend
 - ◆ vier Eigenschaften nach Russel und Gangemi:
 - Vertraulichkeit
 - Integrität
 - Verfügbarkeit
 - Authentizität

Schichten der Sicherheit 1

- Sicherheit betrifft alle Schichten eines Systems
 - ◆ entwickler-aufwand vs. cracker-aufwand

- Kommunikation
 - ◆ Vernetzte Rechner
 - ◆ safety: robustes Protokoll, sicher gegen Fehler
 - ◆ security:
 - Datenverbindungen über nicht vertrauenswürdige Hosts
 - Abhören / Manipulation möglich
 - Benutzung kryptographischer Protokolle (SSL, IPSec)
 - Verfügbarkeit kann selten garantiert werden

- Betriebssystem-Ebene
 - ◆ Hardware-Speicherschutz
 - Isolation unterschiedlicher Prozesse voneinander
 - Systemaufrufe als einziges Interface
 - stark Plattform-abhängig
 - für unterschiedliche Sprachen nutzbar
 - auf embedded systems/PDAs evtl. nicht verfügbar

Schichten der Sicherheit 2

- abstrakte Maschine
 - ◆ Ersatz für Hardware-Speicherschutz
 - ◆ unabhängig von Betriebssystem/Hardware
 - ◆ Verwendung von Interpreter / Zwischencode
 - ◆ erhöhter Verarbeitungsaufwand
- Programmiersprache
 - ◆ Festlegung auf eine sichere Sprache
 - ◆ Verlust der Sprachunabhängigkeit
 - ◆ Einschränkung der an Zeigern möglichen Operationen
 - ◆ automatisches Speichermanagement
 - memory-leaks, zugriff auf free()d, double-free-exploits
 - ◆ Code-Überprüfung zur Compile-Zeit
 - Manipulation im Nachhinein möglich
 - kryptographische Signaturen zur Absicherung
 - Compiler-signiert? -> manipulierte Compiler
 - Programmierer-signiert? -> Vertrauen
 - Benutzung von sicherem Zwischencode
 - zwischencode darf nicht mehr als source
- nach MCS und Sicherheit: -> Sprachbeispiele, JAVA

Programmiersprachen

JAVA

- klassenbasierte objektorientierte Sprache
- seit 1995 von Sun Microsystems entwickelt
- Einsatz von Zwischencode (Bytecode)
 - ◆ Interpretation durch CE (JVM)
- *class loader*: dynamisches Laden und Linken von Klassen
 - ◆ automatisches oder explizites Laden
 - ◆ automatisch = nicht im Namespace
 - ◆ explizit = Programmanforderung
- keine Unterstützung für starke Code-Mobilität
 - ◆ Management verteilter Ressourcen nicht erforderlich
- Integration in Webseiten: Empfang von Code-Fragmenten (Klassen)
- JavaOS - Betriebssystem in JAVA
 - ◆ JavaStation - s. Abbildung
 - ◆ wenig effizient
- (J)VM-Einsatz in embedded systems



JAVA (2)

- safety:
 - ◆ keine unsicheren Sprachelemente (Zeigerarithmetik, uneingeschränkte Typumwandlungen)
 - ◆ Integration von Exceptions
 - ◆ Klassen-Verwendung kontrollierbar (*abstract*, *final*)
 - ◆ Sichtbarkeit von Attributen (*private*, *default*, *protected*, *public*)
 - ◆ Bytecode-Überprüfung durch die JVM beim Laden
 - ◆ automatisches Speichermanagement
 - ◆ *range checks* zur Laufzeit
 - Arrays, Strings
- security:
 - ◆ Durchsetzung auf Sprachebene und durch JVM
 - ◆ SecurityManager zwischen Applets und lokalem System
 - ◆ *class loader* verbietet Ableitung systemkritischer Klassen
 - u.a. *java.** und *sun.**
 - ◆ Sicherheitssystem auf Applets ausgelegt
 - Unterschied zwischen lokalen und über Netz bezogenen Klassen
 - Unterstützung signierter Bytecode-Archive
- --> Telescript + Safe-Tcl oder --> Zusammenfassung

Telescript

- klassenbasierte objektorientierte Sprache
- Entwickelt 1996 von General Magic
- spezialisiert auf Kommunikation
- Ziel: realisierung virtueller Marktplätze
- starke *code mobility*
- portable Zwischensprache *Low Telescript*
- verwendete Konstrukte:
 - ◆ *agent*: EU mit starker Code-Mobility
 - ◆ *place*: verschachtelbare EU, "Aufenthaltort" für agents
 - ◆ *engine*: CE, assoziiert mit einem EnginePlace
 - ◆ *TeleSphere*: Netz aus allen Telescript-engines
- Migration bzw. Clonen von agents mit **go / send**

Telescript (2)

- Ressourcennutzung immer einem User zugeordnet
- agent-Funktion nicht von Besitzer-Präsenz abhängig

- safety:
 - ◆ transparente Sicherung der Objekte durch engines
 - ◆ agents werden durch *permits* beschränkt:
 - *age, extent, priority*

- security:
 - ◆ Einsatz von *tickets* für agent-Migration
 - ◆ engine kann ankommende agents ablehnen
 - ◆ Festhalten der agents durch ein place nicht möglich
 - ◆ agent kann nur mit seiner engine und darüber mit anderen lokal präsenten agents kommunizieren

Safe-Tcl

- basiert auf Tcl
 - ◆ prozedurale Script-Sprache
 - ◆ Ziele: einfach, portabel, mächtig
 - ◆ Effizienz nebensächlich
- keine eigene Code-Mobility
- Einsatzgebiet: aktive E-Mail, Webseiten
- MIME-Erweiterung, Ausführung des Inhalts:
 - ◆ *delivery-time*
 - ◆ *receipt-time*
 - ◆ *activation-time*
 - ◆ (bei Webseiten unmittelbar nacheinander)

Safe-Tcl (2)

- safety:
 - ◆ alle Variablen haben den Typ String
 - ◆ keine Zeiger
 - ◆ nur zwei Gültigkeitsbereiche: lokal/global
- security:
 - ◆ Safe-Tcl Untermenge der Tcl-Befehle
 - ◆ keine unsicheren Funktionen
 - ◆ Ersatz zu generischer Fkt. durch spezielle
- --> Zusammenfassung

Zusammenfassung

- Bedarf an Code Mobility
- Überblick über Paradigmen
 - ◆ und Kuchen
- Analyse von Mobile Code Systems
- Aspekte der Sicherheit
 - ◆ safety und security auf allen Schichten
- Beispielanalyse

- Herangehensweise bei Entwicklung mobiler Applikationen:
 - ◆ Design
 - ◆ Bewertung und Auswahl von Programmiersprachen
 - abhängig von Bedarf, Sicherheitskriterien
 - ◆ Umsetzung
 - ◆ ...fertig ;-)

- ...das wars
- aufschlussreich und interessant?

Quellen

- A. Fugetta, G.P. Picco, G. Vigna, "Understanding Code Mobility", *IEEE Transactions on Software Engineering*, Vol. 24, No. 5, Mai 1998, S. 342ff.
- T. Thorn, "Programming Languages for Mobile Code", *ACM Computing Surveys*, Vol. 29, No. 3, Sept. 1997, S. 213ff.
- D. Russell, G.T. Gangemi Sr., "Computer Security Basics", 1991, O'Reilly & Associates Inc., Sebastopol, CA.
- A. Carzaniga, G.P. Picco, G. Vigna, "Designing Distributed Applications with Mobile Code Paradigms", *Proc. 19th Conf. Software Eng. (ICSE'97)*, R. Taylor, ed., ACM Press 1997, S. 22-32
- Präsentation (incl. Source): <http://op-co.de/mobilecode/>
- Fragen?