Mobile Code

Georg Lukas
2003-01-29

Seminar "Ubiquitous Computing"

Gliederung

- Wozu Mobile Code?
- Paradigmen
- Mobile Code Systems
- Sicherheitsaspekte
- ausgewählte Programmiersprachen
- Zusammenfassung
- Fragen / Diskussion

Einführung

Wozu Mobile Code?

Motivation

- rasantes Wachstum von Computernetzen
- günstige Geräte mit Netzanbindung
- drahtlose Netze
- Skalierbarkeit von Client-/Server begrenzt
- Konfigurierbarkeit für unterschiedliche Ansprüche erforderlich

- Kombination bestehender Methoden und Paradigmen
  - CORBA: RPC und OOP
  - Flexibilität nicht hinreichend
- Bedarf an Plattform- und Architektur-überschreitenden Technologien

Code Mobility

- "**Code-Mobilität** ist die Fähigkeit, die Bindung zwischen Code-Fragmenten und dem Ort der Ausführung dynamisch zu verändern" - *Carzaniga, Picco und Vigna*

- Anwendungen existieren bereits
  - PostScript (1984)
  - SQL (1989)
  - ...

- Beschränkungen auf Sprach- und Systemebene
- wissenschaftliche Forschung nur wenig ausgeprägt
- Basis-Terminologie fehlt / ist ungenau
  - *mobile agent*
  - *code mobility / mobile code / mobile computations / mobile object systems / program mobility*

Typische Verteilte Systeme

- Abstraktion autonomer Rechner zu einem System
- Migration von Prozessen oder Objekten
  - meist transparent
  - Load-Balancing
- Fehlertoleranz
- Netze mit kleinem Umfang
  - Homogenität
  - hohe Bandbreite
  - geschützte Umgebung

Mobile Code-Einsatzgebiete

- *code mobility* im internet-Umfang
  - heterogene Hosts
  - unterschiedliche Autoritäten
  - unterschiedliche Vertrauensebenen
  - 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
- vor Editor, MCS-Auswahl:
- Software-Design
- -> Paradigmen-Auswahl

Paradigmen

Konzept

- Auswahl von Paradigmen
  - Vor- und Nachteile für eigene Anwendung abwägen
  - Betrachtung der Skalierbarkeit
- 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

Client-Server / Code on Demand

- Client-Server (CS)
  - 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

- Code On Demand (COD)
  - 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

Remote Evaluation / Mobile Agent

- Remote Evaluation (REV)
  - A hat Code-Komponente, Ressourcen befinden sich bei B
  - A schickt Code zu B
  - B führt Code aus
  - B liefert Ergebnis an A zurück

- Mobile Agent (MA)
  - A hat Code-Komponente, benötigt aber eine Ressource von B
  - Ressource kann nicht zu A transportiert werden
  - A bewegt sich mit benötigten eigenen Komponenten zu B
  - A führt bei B den eigenen Code aus und benutzt B's Komponente

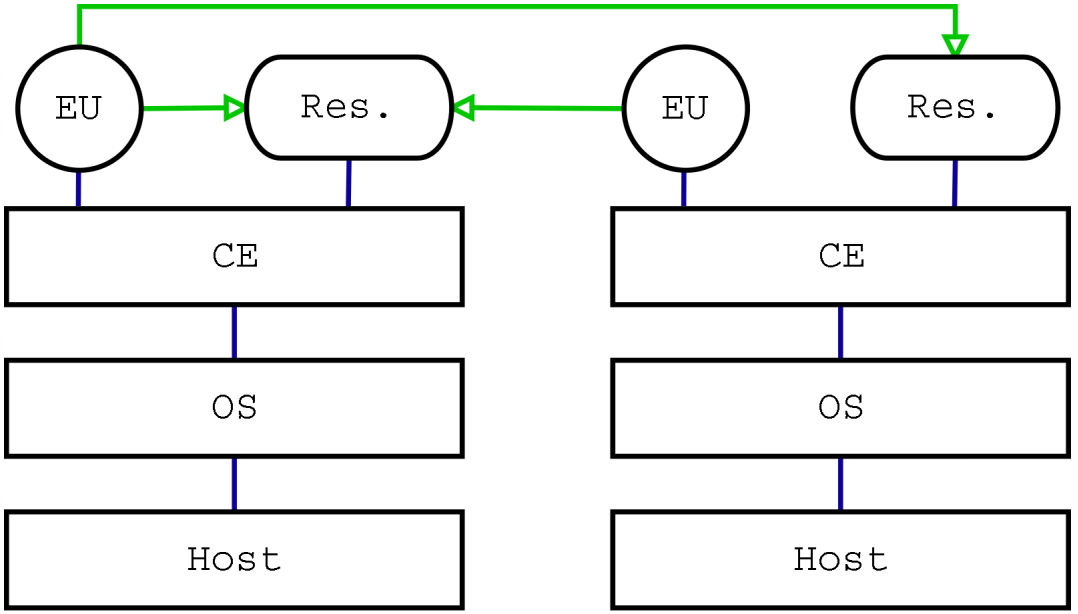
Mobile-Code-Systeme

MCS-Grundlagen

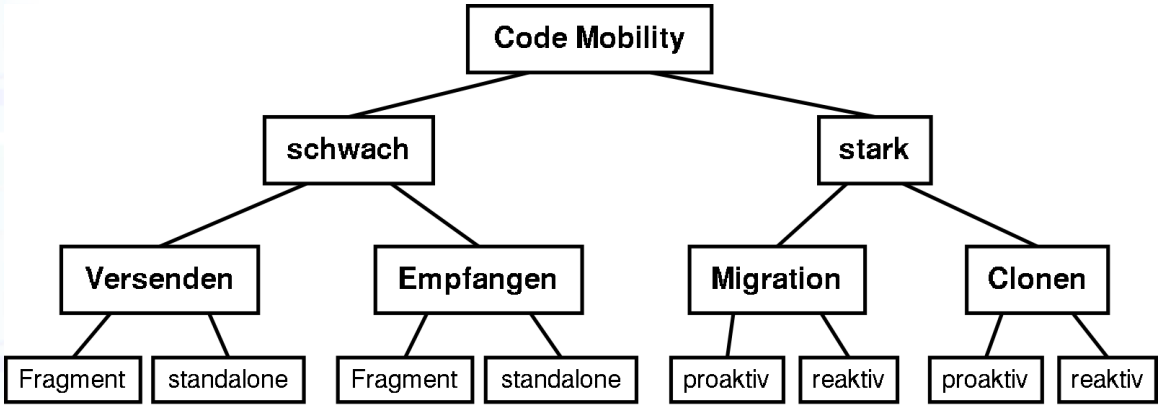
- "*Mobile Code*-Systeme sind Sprachen und Umgebungen, die Mechanismen zum Unterstützen von Code-Mobilität anbieten"

- **CE**: Computational Environment
  - Plattform für EUs und Ressourcen
  - bildet *location*
- **EU**: Executing Unit
  - Prozess oder Thread
  - Zusammensetzung:
    - *code segment*
    - *data space* (Ressourcen-Referenzen)
    - *execution state* (Kontrolldaten, Stack, ...)
- Ressourcen
  - Elemente, die von mehreren EUs gemeinsam verwendet werden können

Zusammenhang zwischen CEs, EUs und Ressourcen



Code-Mobility-Mechanismen



- schwache Mobilität: Übertragung von Code
- starke Mobilität: Migration laufender Prozesse

Ressourcen-Management

- Aufgabe: Ressourcen-Verfügbarkeit nach Migration
- Anwendungsseitige Ressourcen-Bindung
  - über Typ (durch gleichwertige Ressource ersetzbar)
  - über Wert (durch identische Ressource ersetzbar)
  - über Namen (Ressource eindeutig bestimmt)
- Ressourcen mit eingeschränkter "Bewegungsfreiheit"
  - *transferrable / not transferrable*
  - *transferrable: free / fixed*
- Verfahrensweisen bei Migration:
  - verwerfen
  - wiederherstellen (als Netzwerkbindung)
  - Re-Binding
  - duplizieren / verschieben

Sicherheit von Mobile Code

Sicherheitsanforderungen an MCSs

- Mobile Code nicht immer aus zuverlässigen Quellen
- Einschränkbarkeit 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
- 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
  - Code-Überprüfung zur Compile-Zeit
    - Manipulation im Nachhinein möglich
    - kryptographische Signaturen zur Absicherung
    - Benutzung von sicherem Zwischencode

Programmiersprachen

JAVA

- klassenbasierte objektorientierte Sprache
- Einsatz von Zwischencode (Bytecode)
- *class loader*: dynamisches Laden und Linken von Klassen
  - automatisches oder explizites Laden
- keine Unterstützung für starke Code-Mobilität
- Integration in Webseiten: Empfang von Code-Fragmenten (Klassen)
- JavaOS - Betriebssystem in JAVA
- (J)VM-Einsatz in embedded systems

JAVA (2)

- safety:
  - keine unsicheren Sprachelemente (Zeigerarithmetik, uneingeschränkte Typumwandlungen)
  - Integration von Exceptions
  - Klassen-Verwendung kontrollierbar (*abstract, final*)
  - Bytecode-Überprüfung durch die JVM beim Laden
  - automatisches Speichermanagement
  - *range checks* zur Laufzeit
- security:
  - Durchsetzung auf Sprachebene und durch JVM
  - SecurityManager zwischen Applets und lokalem System
  - *class loader* verbietet Ableitung systemkritischer Klassen
  - Sicherheitssystem auf Applets ausgelegt
    - Unterschied zwischen lokalen und über Netz bezogenen Klassen
    - Unterstützung signierter Bytecode-Archive

Telescript

- klassenbasierte objektorientierte Sprache
- 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, "Aufenthaltsort" 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

- Bedarf an Code Mobility
- Überblick über Paradigmen
- Analyse von Mobile Code Systems
- Aspekte der Sicherheit
- Beispielanalyse

- Herangehensweise bei Entwicklung mobiler Applikationen:
  - Design
  - Bewertung und Auswahl von Programmiersprachen
  - Umsetzung

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/