

Proseminar Ambient Intelligence

**Meeting the Computational Needs of Intelligent
Environments:
The Metagluue System**

20. Januar 2005

Martin Kaiser

martin@martin-kaiser.info



Inhalt

1. Projekt Oxygen
2. Anforderungen an Software für Intelligente Umgebungen
3. Metagluue
 - a) Voraussetzungen
 - b) Design



Projekt Oxygen – Vision

- Derzeitige Situation: Mensch (be-)dient Computer
- In Zukunft: Computer dient Mensch
⇒ daher sind Computer überall frei verfügbar, wie Sauerstoff (“Oxygen”) zum leben
- Benutzer braucht kein eigenes Gerät
- Stattdessen: Geräte integriert in die Umgebung
- Geräte lassen sich auf natürliche Weise bedienen (Gesten, Sprache,...)

Projekt Oxygen – Herausforderung und Realisierung

Herausforderung: das System muss

pervasive, embedded, “nomadisch”, leistungsstark und effizient, intentional, eternal (dt. ewig) sein.

Oxygen erlaubt pervasives, benutzerzentriertes arbeiten durch Kombination von Gerätearten:

- Enviro21s (E21s): eingebettete Geräte (Mikrophone, Kameras, Beamer, Drucker, Sensoren...)
- Handy21s (H21s): Handheld Geräte (eingebaute Kameras, Mikrophone, Displays,...)
- Sich selbst konfigurierende Netzwerke (N21s)
- Software, die sich an die verändernde Umgebung anpasst (O2s)

Software intelligenter Umgebungen

Infrastruktur intelligenter Umgebungen (IU):

- verteilt (Kameras, Mikrophone, Handhelds, Laptops, Displays,...)
- hohe Rechenleistung (Bewegungserkennung, Spracherkennung,...)

Schwierigkeit 1: hochgradige Dynamik der Umgebung:

- Benutzer bewegen sich (insbesondere: Betreten und Verlassen der IU)
- Komponenten verändern sich (z.B. Bewegung in der IU, Defekte, Upgrades,...)

⇒ ständige Rekonfiguration der IU und Ressourcenverwaltung “on the fly”

Schwierigkeit 2: multimodale Schnittstellen der IU:

- gleichzeitig zu bearbeitende Eingaben (z.B. Sprache und Gestik) erfordern hohes Maß an Parallelität

⇒ Schwierigkeit 3: Debugging einer IU:

- Formale Beschreibung der Zustände eines parallelen Systems

Metaglu

Metaglu:

- Programmiersprache um interaktive, verteilte Berechnungen durchzuführen
- Erweiterung für Java um linguistische Primitive
- kann große Anzahl unterschiedlicher Hard- und Softwarekomponenten verbinden
- kann kommunizierende Softwareagenten kontrollieren
- arbeitet in Echtzeit
- Systemkomponenten ein- und aushängen ohne das System anzuhalten
- Systemkomponenten upgraden ohne das System anzuhalten
- kontrollieren der Ressourcen - Verteilung
- permanente Möglichkeit zur Abfrage von Statusinformationen

Metaglu als eigene Sprache?

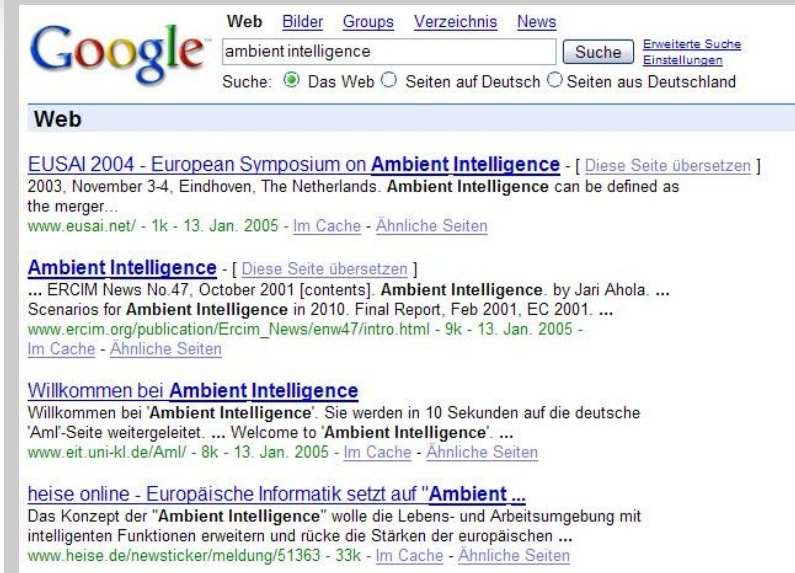
In traditionellen Sprachen (C, C++, Java, Lisp...) sind diese Aspekte nicht berücksichtigt
Metaglu bietet die Möglichkeit der High-Level Programmierung von Softwareagenten

Computationale Eigenschaften von IU

Verknüpfung verteilter, modularer Systeme

- IUs bestehen aus vielen verschiedenen Subsystemen, z.B. Hardwareverbindungen, Softwareanwendungen, Mechanismen zur internen Kontrolle
⇒ diese müssen miteinander verbunden werden (mit “Kleber”, engl. Glue)
- Die Komponenten von IUs können nicht auf einem einzigen PC laufen (Hardwarebeschränkungen, Echtzeitanwendungen,...)
Auch in Zukunft: Computervision oder Spracherkennung sehr rechenintensiv!
- Verbindung einander fremder Software durch Protokolle nicht ausreichend

Beispiel: Verbindung einander fremder Software



Steuerung eines Browsers durch Spracherkennung (Spracherkennung \Leftrightarrow Browser)

- Nutzer möchte die Links mittels Sprache ansteuern
- Browser muss die zu erkennende Grammatik der Spracherkennungssoftware für jede neue Seite ändern
- Spracherkennung aktiviert Link in Browser über API
- Nötig: API in beiden Applikationen
- Ausreichend: Nein! \Rightarrow Beide APIs müssen verbunden werden!

Ressourcen Verwaltung

Problem: Ressourcenknappheit (z.B. Displays, PCs,...)

- Konflikte, wenn mehrere Anwendungen ein Gerät gleichzeitig benutzen wollen
- selbst einfache Dinge (z.B. Abspielen von Videos) können große Auswirkungen auf andere Systemteile haben
- Ressourcenkonflikte können auch auftreten, wenn mehrere Benutzer gleichzeitig zugreifen

Dynamische Konfiguration

- Benutzer mit eigenen PDAs
- Hardware/Software wird eingebaut, möglichst ohne das System anzuhalten
- Benutzer möchten die IU in einem anderen Kontext benutzen (Videokonferenz vs. Informationssystem)

Zustandsspeicherung

Software intelligenter Umgebungen muss dynamisch sein, d.h.

- Anhalten einzelner Komponenten
- Modifizieren einzelner Komponente
- Reload einzelner Komponente

muss möglich sein, um

- ein System mit Hunderten von Komponenten nicht komplett anhalten zu müssen
- neue Komponenten zu testen/debuggen

Dabei müssen Zustände gespeichert werden, da

- IUs Benutzerpräferenzen und Benutzeraktivitäten lernen
- “rückwärts lernen” nicht möglich

Parallelismus

- IU müssen die verschiedenen Benutzereingaben behandeln
- Menschen sprechen, bewegen sich, benutzen Gesten, zeigen Emotionen gleichzeitig
- IU muß nur einen kleinen Teil verstehen
- Alle Eingaben müssen parallel verarbeitet werden

Echtzeitverhalten

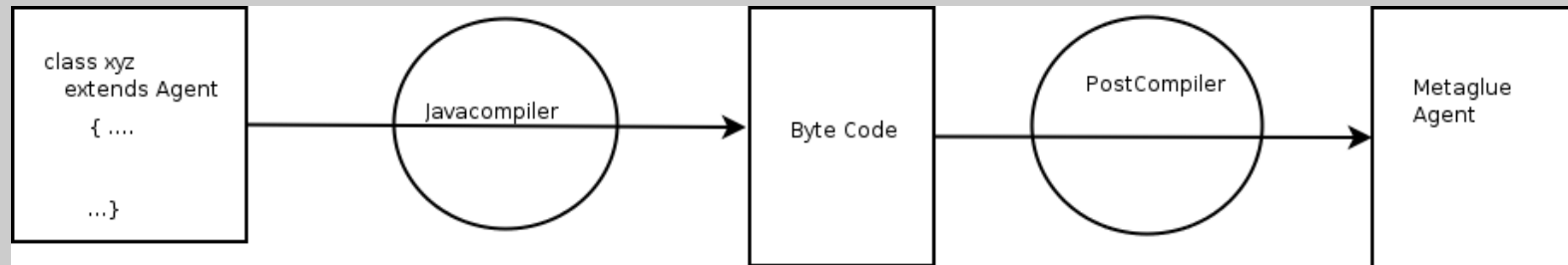
- Benutzer erhält keine Rückmeldung über Auslastung des Systems
- Komponenten produzieren Daten, die schnell Verarbeitet werden müssen

⇒ unbedingte Echtzeitverarbeitung von Ereignissen

Debugging

Schwierig, da Zustände nicht immer reproduzierbar

Metaglu Design



- Erweiterung von Java um eine *Agent* class
- Post-compiler: erzeugt aus Java Bytecode einen Metaglu Agenten
- Metaglu Runtime Engine: Metaglu Virtual Machine
- Nahezu jeder Java Code kann zu Metaglu Agent gemacht werden
- Alle Prozeduraufrufe, ob lokal oder entfernt sehen aus wie lokale Prozeduraufrufe
- 50% der Entwicklungszeit steckt in der Semantik des Systems, der Rest in der Implementierung
- Voraussetzung für alle Agenten: Metaglu Virtual Machine läuft permanent auf dem System

Konfigurationsmanagement

- Realisiert durch interne SQL Datenbank für modifizierbare Parameter (Attribute)
 - Attribute müssen nicht im Agent gespeichert werden (schwer modifizierbar)
 - Attribute sind z.B. Workstation auf der der Agent laufen muss
 - Attribute können per Webinterface im laufenden Betrieb geändert werden
- Beispiel:

```
Attribute location = new Attribute("location");  
System.out.println("I run on " + location.getValue());
```

Agenten Konfiguration

- Agenten können minimale Anforderungen an das Gesamtsystem stellen (z.B. Hardwareanforderung)
- Werden mit *tiedTo()* spezifiziert
Beispiel: *tiedTo(capability.FrameGrabber)*
- Metagluе verschiebt den Agenten auf einen Rechner der die Bedingung erfüllt

Agenten Status

- *freeze()* - Funktion eines Agenten friert dessen Status ein
- Standardmässig beim Herunterfahren eines Agenten
- *defrost()* - Funktion eines Agenten lädt dessen Status aus der SQL Datenbank, wird im Konstruktor ausgeführt
- Aspekte, wie z.B. Verbindungen zu anderen Agenten, brauchen nicht gespeichert zu werden

Agenten Verbindung

- Metagluе verbindet Agenten miteinander, egal auf welchem Rechner sie laufen
 - Prozeduraufruf *reliesOn()* verbindet Agenten mit Service
- Beispiel:

```
Agent speechSynthesizer = reliesOn(speech.Synthesizer);  
speechSynthesizer.say("Hello!");
```

- Agenten referenzieren sich über abstrakte Namen und nicht direkt
- ermöglicht einfaches Austauschen von Agenten
- läuft ein spezifizierter Agent nicht, versucht Metagluе ihn zu starten

Änderung des laufenden Systems

- Metagluе versucht, ein bestehendes System aus Agenten am laufen zu halten
- Agenten warten auf per Hand angehaltene Agenten per Default
- Weitere Möglichkeit: anderen Agenten suchen

Beispiel: Abgestürzter Agent

- Agent für Lichtsteuerung stürzt wegen Hardware oder Softwarefehler ab.
- Metagluе wird versuchen, den Agenten neu zu starten
- Neustart möglicherweise auf anderer MVM, die aber Agent Attribute einhält
- Statusinformationen können verloren sein
- Agent kann das Computer Vision System fragen, ob es etwas sieht
- Sieht es nichts, sind die Lichter aus

Metagluе ist selbst aus Agenten gebaut

Fehlermeldungen können über eingebaute Agenten ausgegeben werden (z.B. Speech Synthesizer)

Resourcen Management

- Applikationen treten in Konflikt mit anderen Applikationen
- kein simultanes Ausführen mehr möglich
- Geräte können aus der IU genommen werden
- Metagluue erlaubt den Agenten auf Ressourcen zuzugreifen
- Agenten brauchen kein Wissen über Ressourcenmanagement
- Ausnahme: *Dealer* Agenten, zuständig für gesamtes System

Dealer:

- Verschiedene Dealertypen mit eingebauten Verteilungsfunktionen
- Dealer garantieren Fairness und/oder Prioritäten
- Eigene Dealeragenten können hinzugefügt werden
- Dealer können Agenten Nutzungsrechte entziehen
- Agenten müssen Dealer benutzen um Geräte benutzen zu dürfen

Event Broadcasting

- Metagluе Agenten können Nachrichten austauschen, statt direkt auf Funktionen zuzugreifen
- Melden sich bei anderen Agenten an, um mitzuhören
- Nachrichten werden auch genutzt, um Agentengruppen über Kontextswitches in der IU zu informieren

Debugging

- Metagluе hat graphisches Interface um laufendes System zu untersuchen
- zeigt alle laufenden Agenten und erlaubt interaktives Aufrufen der Agentenmethoden
- Metagluе hat Logging facility, sammelt alle Textmeldungen der Agenten
- Leider nicht vorhanden: Source Level Debugging von Remote Agenten, Dynamic Object Browser, Möglichkeit Breakpoints über Agentengruppen simultan zu setzen

Literatur

- [1] Michael Coen, Brenton Phillips, Nimrod Warshawsky, Luke Weisman, Stephen Peters, and Peter Finin. Meeting the computational needs of intelligent environments: The metagluue system. In *Proceedings of MANSE'99*, Dublin, Ireland, 1999.
<http://www.ai.mit.edu/people/mhcoen/metagluue.pdf>.
- [2] Computer Science and Artificial Intelligence Laboratory. MIT Project Oxygen, 2005.
<http://oxygen.csail.mit.edu/index.html>.