# Multiagentensysteme

*Technologien, Architekturen, Plattformen*

Prof. Dr. Stefan Bosse

Universität Koblenz - FB Informatik - Praktische Informatik

# Verteiltes Rechnen mit JAM Agenten (ABC): Mobilität und Netzwerke

Bisher wurden MAS auf einer Plattform ausgeführt.

- Die Agenten konnten über Signale und Tupelräume interagieren
- Wichtige MAS Eigenschaften wie Replikation und Terminierung waren schon möglich

# Verteiltes Rechnen mit JAM Agenten (ABC): Mobilität und Netzwerke

Bisher wurden MAS auf einer Plattform ausgeführt.

- Die Agenten konnten über Signale und Tupelräume interagieren
- Wichtige MAS Eigenschaften wie Replikation und Terminierung waren schon möglich

Jetzt soll auch die Mobilität der Agenten über Plattformnetzwerke ermöglicht und diskutiert werden.

# Mobile Agenten

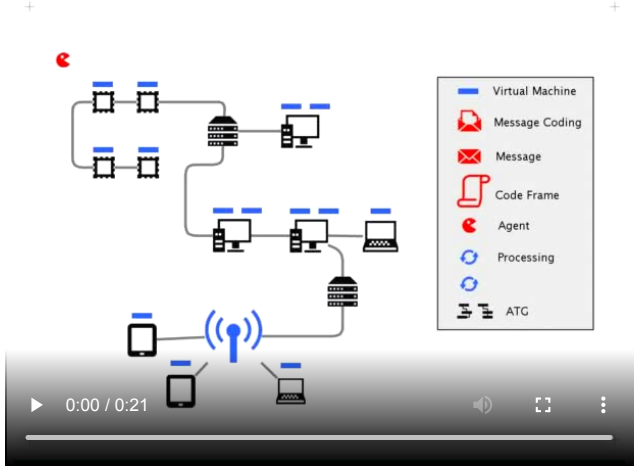
- Ein Agent ist charakterisiert durch:
  - Einem Programm dass das Verhalten des Agenten beschreibt (hier ATG)
  - Daten und Datenzustand (hier Körpervariablen)
  - Einem Standort und eine Umgebung
- Programm und Daten bilden ein funktionales Objekt
- Ein Agent in Ausführung wird durch einen Prozess repräsentiert.
- Der Prozesszustand setzt sich zusammen aus:
  - Körpervariablen (Datenzustand)
  - Nächste Aktivität und Mikroschedulingblöcke (Kontrollzustand)
  - Signale, Nachrichten
  - Pfad im Netzwerk (Plattformen auf denene der Agent bereits ausgeführt wurde)
  - Verhalten (D.h. ATG)!

# Mobile Agenten

- **Mobile Agenten bedeuten mobile Prozesse**!
- Bei der Migration muss ein Agentenprozess in einen
  1. **Prozessschnappschuss** (Container) eigenfroren und serialisiert werden,
  2. Der Schnappschuss in textueller Form (JSON+) von einer Plattform zu einer anderen übertragen werden, und
  3. Der Prozessschnappschuss wieder deserialisiert werden, und der Prozess wird weiter ausgeführt.

# Mobiler Code

- Anstelle von mobilen Daten (Kommunikation) wird durch Agenten mobiler Code mit eingebetteten Daten in Netzwerken transportiert.
- Dabei werden Code und Daten als Kontainerobjekte und Prozesschnappschüsse serialisiert und auf der Zielplattform wieder deserialisiert.



## Serialisierung und Prozessschnappschuss

- Damit ein Agent seinen Standort wechseln kann muss das gesamte funktionale Objekt und sein Kontrollzustand serialisiert werden:
  - JSON Format für Daten
  - JSON+ Format für Daten und Funktionen
- Der Kontrollzustand ist vor allem durch die *next* Variable gegeben und bereits Bestandteil des Agentenobjekts.

## JSON+ Format

- Das JavaScript Object Notation Format ist ein weit verbreitetes Format um Daten portabel und unabhängig von Rechner und Betriebssystem zwischen Programmen austauschen zu können.
  - Skalare Daten (Strings, Numerische Werte, Boolean Werte)
  - Vektorielle Daten (Aber nur generische Arrays, typisierte Arrays bedürfen eine "Sonderbehandlung"
  - Objekte, aber ohne Methoden (Prototypen) Bindung! bemerkung: Der Agent ist ein Objekt aber ohne gebundene Prototypenfunktionen!

Wie sollen nun aber die Funktionen aus der Aktivitäts-, Übergangs- und Behandlungssektion serialisiert werden?

# Serialisierung von Funktionen

1. In JavaScript kann jede Funktion durch die `f.toString()` Methode (wieder) in Text umgewandelt werden.

2. IN JSON könnten diese Funktionen dann direkt als Text gespeichert werden. Aber wie soll man diese Zeichenketten von Daten unterscheiden? Und Programmcode enthält z.B. Zeilenumbrüche und Zeichen die in JSON kodiert (Escaped) werden müssen.

3. In JSON+ werden Funktionen als Zeichenketten mit einem Prefix im base64 Format gespeichert (erspart Kodeüberarbeitung).

## Serialisierung von Funktionen

- Es gibt aber noch mehr zu tun. Der Programmkode eines Agent wird je nach Plattform vor der Ausführung modifiziert.
  - JavaScript ist nicht präemptiv und es gibt keine Koroutinen. Ein Agent darf aber nur eine bestimmte Zeitscheibe laufen und muss dann unterbrochen werden (Scheduling). Bei generischen JS VM Plattformen müssen zusätzliche Anweisungen in den Programmkode "injiziert" werden um ein Scheduling zu ermöglichen.
  - Dieser zusätzliche Programmkode muss vor der Serialisierung entfernt werden!

```
while (ε) { ... }
-------------------------
while (ε && schedule()) { ... }

for (;ε;) { ... }
-------------------------
for (;ε && schedule();) { ... }

function foo() { ... }
-------------------------
function foo () { schedule(); ... }
```

# Serialisierung von Agentenobjekten

```
{ x,
  act:{a1:() => {i1;i2;..}},
  trans:{a1:() => {i1;i2;..}},
  next}
```

---

```
{ "x":v,
  "act":{"a1":"__PxEnUf_aTE7aTI7Li4=",..},
  "trans":{"a1":...},
  next:"a2"}
```

Def. 1. Serialisierung eines Agentenobjekts in das JSON+ Format

# AgentJS Mobilität

Für mobile Agenten wird benötigt:

1. Mindestens zwei JAM Plattformen (physisch oder logisch)
2. Verbindung der Agentenplattformen über serielle Kommunikationskanäle
3. Ein Ziel: Z.B. die IP Adresse einer anderen Plattform oder der Plattformname
4. Eine Funktion: `moveto(destination)`

Die Verbindung von JAM Plattformen findet über AMP (Agent Management Port) und Kommunikationskanälen statt.

- HTTP, HTTPS (Verteilt: Browser, Native, Server)
- TCP,UDP (Verteilt: Native, Server)
- Pipes (Lokal: Native, Server)
- Virtual Channel (Lokal, nur virtuelle logische Knoten)

# Vernetzung von Agentenplattformen

- Eine Agentenplattform repräsentiert einen physikalischen Knoten in einem Virtuellen Netzwerkgraphen
  - Es gibt virtuelle und physikalische Kommunikationsverbindungen zwischen den Netzwerkknoten
  - Beispiel: Internet Protokoll (IP/UDP/HTTP) wäre die physische Verbindung
- *Ports* von Plattformen verbinden und stellen *Links* her
- Ein Agent kann auf einen anderen Knoten (Plattform) migrieren indem er die *moveto* Operation ausführt (innerhalb einer Aktivität):
  - Dabei muss der Agent als Argument das Ziel angeben
  - Ziel kann ein Knotenname (`DIR.NODE(<nodename>)`, eine IP Adresse (`DIR.IP(<url>)`), oder bei gerichteten Verbindungen die Richtung sein (z.B. `DIR.NORTH`)

# Vernetzung von Agentenplattformen

- Alle aktuell mit einer Plattform verbundenen anderen Plattformen (Namen) können mit der *link* Funktion (vom Agenten) abgefragt werden;
  - `link(DIR.IP('%')` liefert JAM Plattformnamen
  - `link(DIR.IP('*')` liefert IP Adressen der Plattformen

```
{
  var connected=link(DIR.IP('%'));
  if (connected.length) {
    var next = random(connected);
    moveto(DIR.NODE(next));
  }
}
```

## Kommunikation und Netzwerke

- JAM Plattformen können in beliebigen Netzwerken miteinander verbunden werden.
- Eine Vielzahl von Kommunikationsprotokolle sind verwendbar:
  - RS232/USB
  - UDP
  - TCP
  - HTTP
- Beliebige Netzwerktopologien können gebildet werden (physisch wie logisch):
  - Stern
  - Gitter (1D/2D/3D)
  - Bus
  - Intranet und Internet (allg. Graphen)
- AMP: **Agent Management Port** als gemeinsames Protokoll und Interface in heterogenen Systemen

# Agent Management Port

- AMP definiert eine Menge von Nachrichten die dem Transport von
  - Agenten (Prozessschnappschüsse),
  - Signalen und Tupeln,
  - und Handshakes dienen.
- Weiterhin kann via AMP ein Debugging und Monitoring betrieben werden.

# Agent Management Port

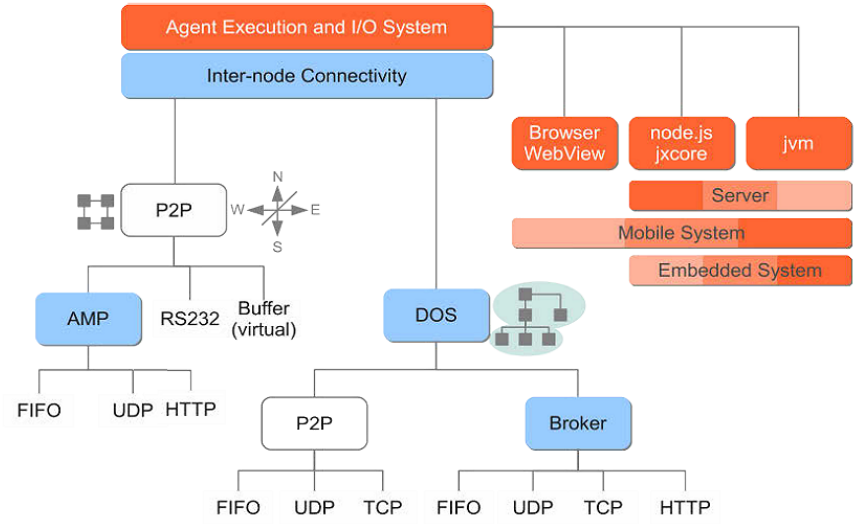



Abb. 1. JAM Konnektivität und ein breites Spektrum an unterstützten JavaScript Host Plattformen. DOS ist ein optionale Distributed Organization System Schicht und implementiert verteilte Systeme via Remote Procedure Calls (RPC)

# Agent Management Port

- JAM Plattformen können über eine oder mehrere virtuelle Kanäle (AMP) miteinander verbunden werden
  - Verwendung von Relaisstationen führt zu sternartigen Kommunikationssnetzwerken
  - Browser und mobile JAM Knoten können nur an im Internet sichtbare JAM Knoten über HTTP sich verbinden → erfordert Relais Knoten (ohne Nutzerinteraktion, kopflos)

```
// Service Provider
port(DIR.IP(9999),{proto:'http'})

// Service User
port(DIR.IP('http://localhost:*'));
locate(print);  // start localization
connect(DIR.IP('http://localhost:9999'));
```

Code 1. In der JAM Shell können AMP Service Endpunkte (außer im Browser) und AMP Verbindungen erstellt werden

# Agenten Interaktion

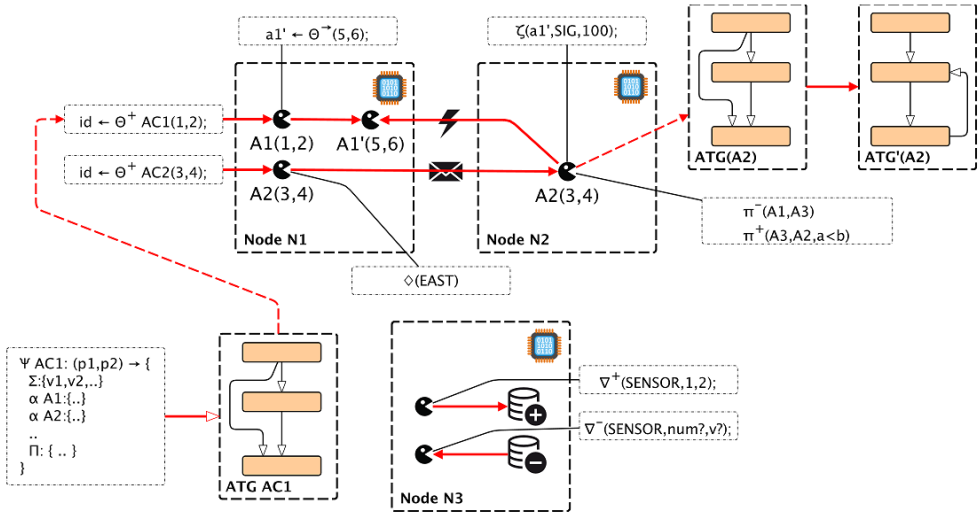



Abb. 2. Effekt von verschiedenen AAPL/AgentJS Anweisungen zur Laufzeit auf Agenten und Plattformen

# JAM Shell

- Die JAM shell *jamsh* integriert eine JAM Plattform.
- Beim Start wird ein JAM Knoten (d.h. einer virtuelle/logische JAM APP) erzeugt.
- Nach dem Start können einzelne Anweisungen ausgeführt werden, wie das Laden von Agentenklassen oder das Herstellen von Verbindungen zu anderen JAM Knoten (physisch oder logisch)

# JAM Shell

```
# node jamsh
JAM Shell. Version 1.1.14 (C) Dr. Stefan Bosse
[JAM] Created world MUJIVOTO.
[JAM] Created root node mujivoto (0,0).
> help
> port(DIR.IP(10001))
iprouter: add link localhost:10001
[AMP C2:EF:38:1F:F5:EC IP(10001)] Starting 127.0.0.1:10001 [MUL] (proto udp)
[AMP C2:EF:38:1F:F5:EC IP(10001)] IP port 141.26.71.28:10001 (proto udp)
> open('code/hello.js')
> start()
> create('hello',{text:"hello world"})
dicexiro
> stats('node')
```

# JAM Shell

- Skripte (mit JAM Shell Kommandos und eingebetteten Agentenkonstruktorfunktionen) können mittels des `script(file)` Kommandos geladen werden.

## Beispiel

```
function hello(msg) {
  this.msg=msg;
  this.act = {
    init: function () {
      log('hello '+this.msg);
    },
    end: function () {}
  }
  this.trans = {
    init:end
  }
  this.next=init;
}
compile(hello);
```

# JAM Shell

## Initialisierung

- Die eigentliche JAM APP muss (einmalig) mit dem Kommando `start()` gestartet werden um Agenten auszuführen:

```
> start()
[JAM] Starting JAM loop ..
```

- Schon vor dem Start können Agenten erzeugt werden!
- Mit dem `stop()` Kommando wird die *Ausführung von Agenten* durch JAM gestoppt, jedoch bleiben die Agenten erhalten. Eine nochmalige Verwendung des `start()` Kommandos führt die Ausführung der Agenten weiter.

## Netzwerke

- JAM Knoten können z.B. über IP Netzwerke miteinander verbunden werden. Dazu muss
  - ein Kommunikationsport auf jeder JAM APP erzeugt werden, und
  - die Ports miteinander verbunden werden. Achtung: Es existiert kein IP Verbindungsaufbau, die Kopplung ist locker!

# JAM Shell

- Mit dem `port(dir)` Kommando kann ein neuer Kommunikationsport für eine JAM APP erzeugt werden, i.A, `port(DIR.IP("IP:IPPORT"))`
- Mit dem `link(dir)` Kommando werden zwei Ports miteinander verbunden, i.A. `port(DIR.IP("IP2:IPPORT2"))`.

```
APP-A > port(DIR.IP('IP1:IPPORT1'))
iprouter: add link localhost:10001
[AMP 63:35:E4:66:FA:43 IP(localhost:10001)] Starting 127.0.0.1:10001 [MUL] (proto udp)
[AMP 63:35:E4:66:FA:43 IP(localhost:10001)] IP port 141.26.71.164:10001 (proto udp)
APP-B > port(DIR.IP('IP2:IPPORT2'))
iprouter: add link localhost:10002
[AMP CE:EE:47:39:61:1A IP(localhost:10002)] Starting 127.0.0.1:10002 [MUL] (proto udp)
[AMP CE:EE:47:39:61:1A IP(localhost:10002)] IP port 141.26.71.164:10002 (proto udp)
APP-A > link(DIR.IP('IP2:IPPORT2'))
[AMP 63:35:E4:66:FA:43 IP(localhost:10001)] Trying link to 127.0.0.1:10002
iprouter: add route 141.26.71.164:10001 -> 127.0.0.1:10002#negodoqe
[AMP 63:35:E4:66:FA:43 IP(localhost:10001)]
  Linked with ad-hoc udp 127.0.0.1:10002, AMP CE:EE:47:39:61:1A, Node negodoqe
APP-B >
iprouter: add route 141.26.71.164:10002 -> 127.0.0.1:10001#quxibumi
[AMP CE:EE:47:39:61:1A IP(localhost:10002)]
  Linked with ad-hoc udp 127.0.0.1:10001, AMP 63:35:E4:66:FA:43, Node quxibumi
```

# JAM Netzwerke

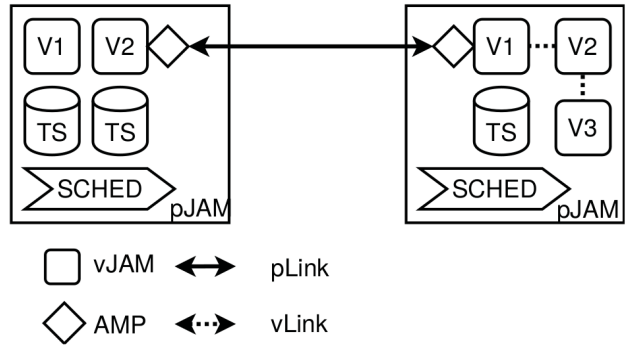



Abb. 3. Grundlegende Vernetzung von virtuellen (innerhalb einer physischen) und physischen JAM Plattformen