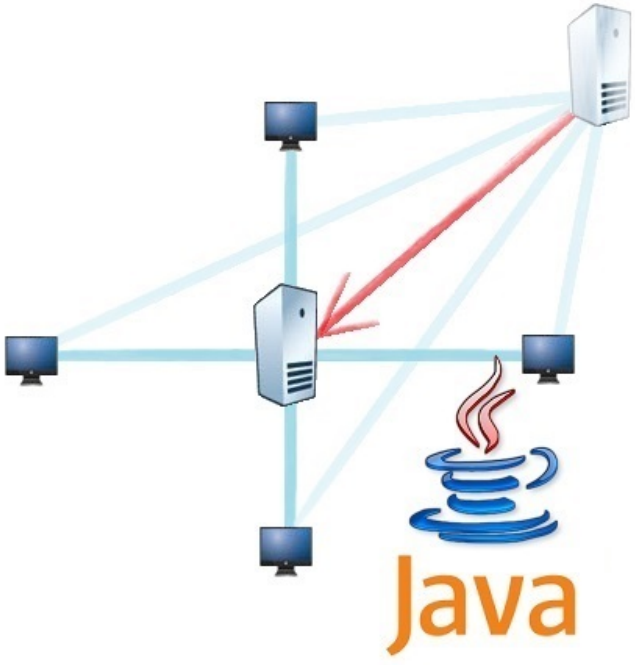
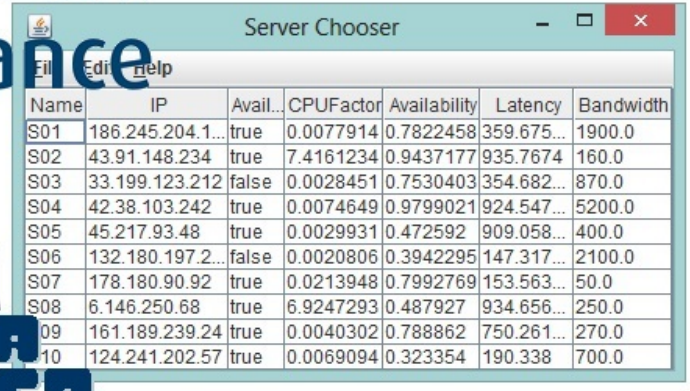
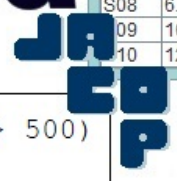
ulm university universität ulm





Name	IP	Avail...	CPUFactor	Availability	Latency	Bandwidth
S01	186.245.204.1...	true	0.0077914	0.7822458	359.675...	1900.0
S02	43.91.148.234	true	7.4161234	0.9437177	935.7674	160.0
S03	33.199.123.212	false	0.0028451	0.7530403	354.682...	870.0
S04	42.38.103.242	true	0.0074649	0.9799021	924.547...	5200.0
S05	45.217.93.48	true	0.0029931	0.472592	909.058...	400.0
S06	132.180.197.2...	false	0.0020806	0.3942295	147.317...	2100.0
S07	178.180.90.92	true	0.0213948	0.7992769	153.563...	50.0
S08	6.146.250.68	true	6.9247293	0.487927	934.656...	250.0
S09	161.189.239.24	true	0.0040302	0.788862	750.261...	270.0
S10	124.241.202.57	true	0.0069094	0.323354	190.338	700.0




```
1 discard (Latency #> 500)
2 minimize (Latency)
```

Michael Grönert

13. Mai 2013

Optimierung des Deployments vert. Anw. unter Verwendung einer domänenspez. Beschreibungssprache

Motivation (1)

- Verteilte Systeme haben verschiedene Probleme
  - → **Fehler-Auflösung** - Finden der Fehlerquelle und verhindern der Ausbreitung.
  - → **Ressourcen-Management** - Verwaltung & Beständigkeit von Daten, CPU-Leistungen und Bandbreiten.
  - → **Administration** - Konfiguration und Überwachung aller Komponenten.
  - → **Kommunikationsinfrastruktur** - Gewährleistung und Sicherung der Kommunikation.
- Fokus auf wichtige Eigenschaften:

  Bandbreite, *CPU-Leistung*, *Erreichbarkeit*, *Jitter*, *Latenz*, *Migrationskosten*

Motivation (2)

- **Problembekämpfung durch (Re)deployment** statt Kompensation
  - → Kompensation reicht oft nicht aus.

→ Entwickeln eines Dienstes, der ein solches Deployment anhand verschiedener Kriterien erleichtert.

→ Verwenden einer **Domänensprache** (**DSL**).

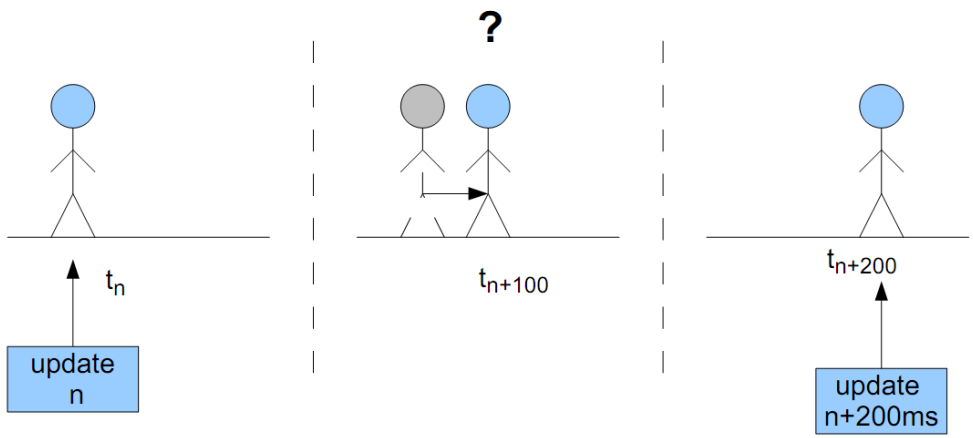
- → Einfach, leicht zu erlernen, intuitiv und erweiterbar.

Überblick

- Anwendungsfälle
  - → Beispiele für DSL-Formulierung
- Verwendete Techniken
  - → Domänensprache
- Dienst
  - → Aufbau
  - → Sichten
  - → Implementierung
- Demonstration
- Zusammenfassung & Ausblick

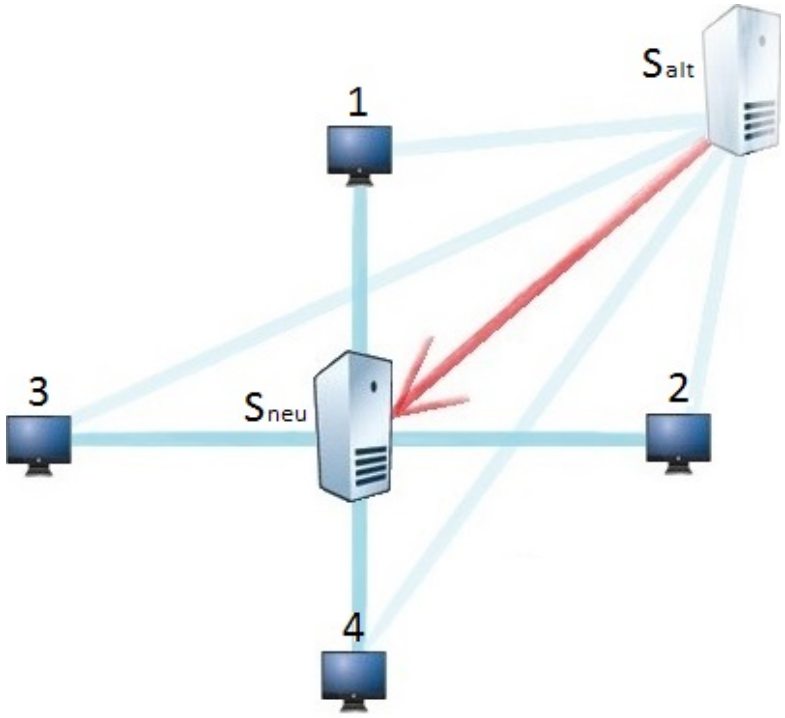
Anwendungsfall I - Multiplayerspiel

- Anwendungsbeispiel Echtzeit-Strategiespiel
  - → **„Harte Echtzeitanforderungen“**
  - → **Synchrones Gameplay unerlässlich** (gleiche Sicht auf Spielablauf)
- Hauptproblem: **(Unterschiedliche) Latenzen** und Bandbreiten
- In der Praxis oft ausgeklügelte Interpolationsalgorithmen, nachträgliche Korrekturen oder erzwungene Verzögerungen
  - → Nicht immer realisierbar oder unpassend/schlecht



Bewegungsinterpolation bei Dead-Reckoning [2]

Anwendungsfall I - „Gerechtes“ Deployment (zur Laufzeit)



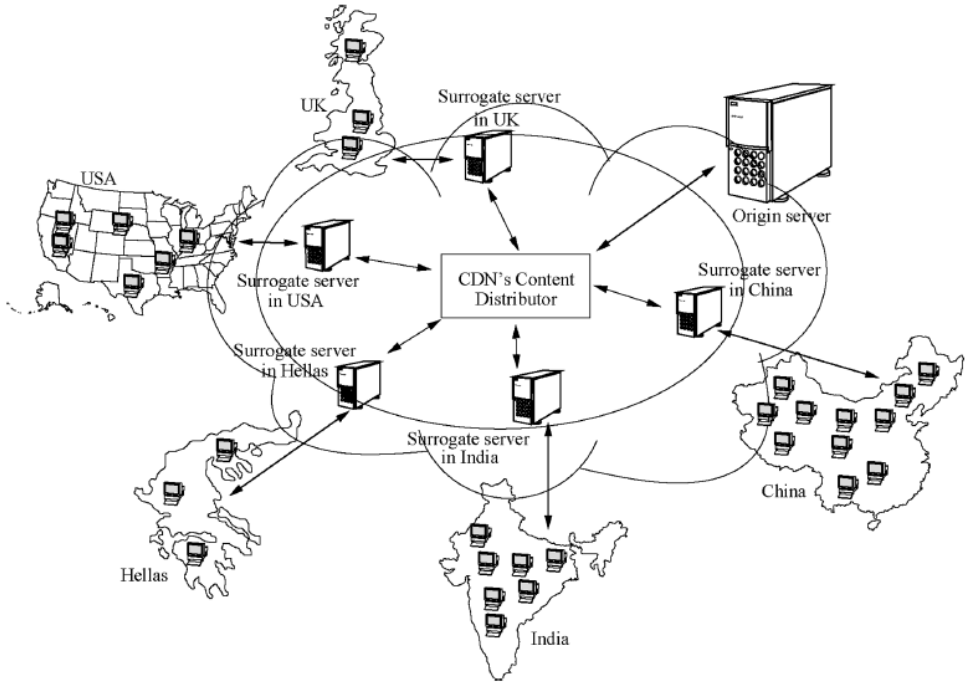
→ z.B. Server „verschieben“, damit alle ähnliche Latenzen haben [1,3] Gleicher „Abstand“ der Clients (1,2) & (3,4) zu S_{alt} untereinander. Große Unterschiede, wenn alle miteinander spielen. S_{neu} ist für alle gleich fair.

„Verwerfe Server mit Latenzen größer als 500ms und sortiere die restlichen nach aufsteigenden Latenzen.“

Anwendungsfall II - Content Delivery Network (CDN)

- Großflächig verteiltes Netzwerk zum Zugriff auf große Datenmengen

→ Beispiel: **YouTube** - Videoplattform, verteilt in 53 Ländern [A]



Modell eines Content Delivery Network [4]

Anwendungsfall II - Last vs. Latenz

- Last durch Benutzerzugriffe gleichmäßig verteilen.
- Nutzen naher Quellen, um Latenzen und Komplexität des Routings gering zu halten.

  „Verwerfe Server, zu denen momentan schon mehr als 300 Benutzer verbunden sind und sortiere die restlichen nach absteigenden Bandbreiten.“

Anwendungsfall III - Cloud-Computing

- Verteilte Infrastruktur, bestehend aus Rechen- und Netzwerkkapazität, Datenspeicher, Hard- und Software
  - → Beispiel: **Amazon S3-Cloud** - Webservice-Schnittstelle zum Speichern und Abrufen beliebiger Datenmengen, jederzeit, überall [B]

„Nutze nur Server, die $Ressource_x$ und $Ressource_y$ anbieten und sortiere die restlichen nach absteigenden Bandbreiten.“

Verwendete Techniken

- ***OSGi* (Open Services Gateway initiative)**
- ***Java* & *Scala*** - Für Dienst und Domänensprache
- ***JaCoP* (Java Constraint Programming solver)**
- ***DSL* (domain-specific language)**

DSL (domain-specific language)

- Formale Sprache, für ein bestimmtes Problemfeld modelliert
  - → **Lesbarkeit:** Natürliche Grammatik
  - → **Intuitiv:** Leicht zu verstehen und zu erlernen
  - → **Geringer Umfang:** Weniger Fehlerquellen und Lernaufwand
- **Interne/Eingebettete** *DSLs*
  - Nutzen Komponenten ihrer Wirtssprache
  - Geringerer Implementierungsaufwand
- **Externe** *DSLs*
  - Von Grund auf neu definierte Sprachen

DSL - Domäne

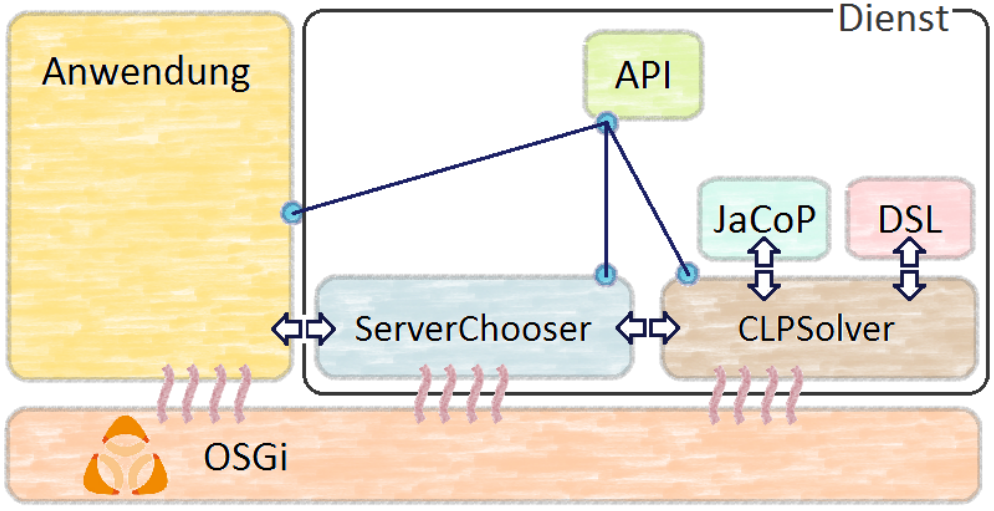
- **Problemdomäne** - Prozesse, Bestandteile und Beschränkungen
  - → Bandbreite, CPU-Leistung, Erreichbarkeit, Jitter, Latenz, Migrationskosten, Nutzung
- **Lösungsdomäne** - Werkzeuge, Techniken und Funktionalitäten
  - → `use`, `maximize`, `minimize`, `discard`, `discardWithout`
- **Grammatik** - Abbilden von Problemdomäne auf Lösungsdomäne

```
<DSL>       ::= <Funktion> | <Scala...> ;
<Funktion>  ::= ( <FunkTyp> '(' <Gleichung> ')' ) | ...;
<FunkTyp>   ::= <Maximum> | <Minimum> | <Verwurf> | ...;
<Maximum>   ::= 'maximizing' | 'max' | 'maximiere' ;
```

Verwendung der DSL

- von Anwendungsfall I - Multiplayerspiel
  - `discard (Latency #> 500)`
  - `minimize (Latency)`
- von Anwendungsfall II - Content Delivery Network
  - `discard (UsedBy #> 300)`
  - `maximize (Bandwidth)`
- von Anwendungsfall III - Cloud-Computing
  - `discardWithout (UsedBy #& 1)`
  - `discardWithout (UsedBy #& 2)`
  - `maximize (Bandwidth)`

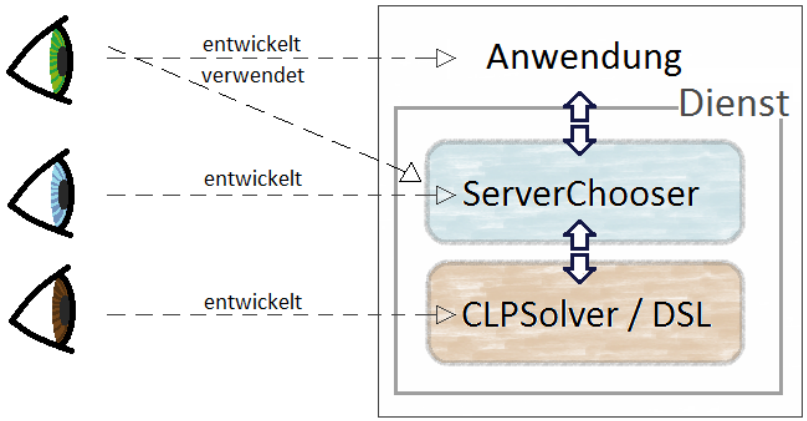
Architektur und Komponenten des Dienstes



Architektur des Gesamtsystems [3]

- **ServerChooser** - Schnittstelle zur Anwendung
  - Bietet verschiedene Auswahlverfahren.
- **CLPSolver** - Setzt die Nutzung einer Domänensprache um.
  - **JaCoP** - Java-Constraint Solver
  - **DSL** - Enthält alle Bestandteile der Domänensprache.

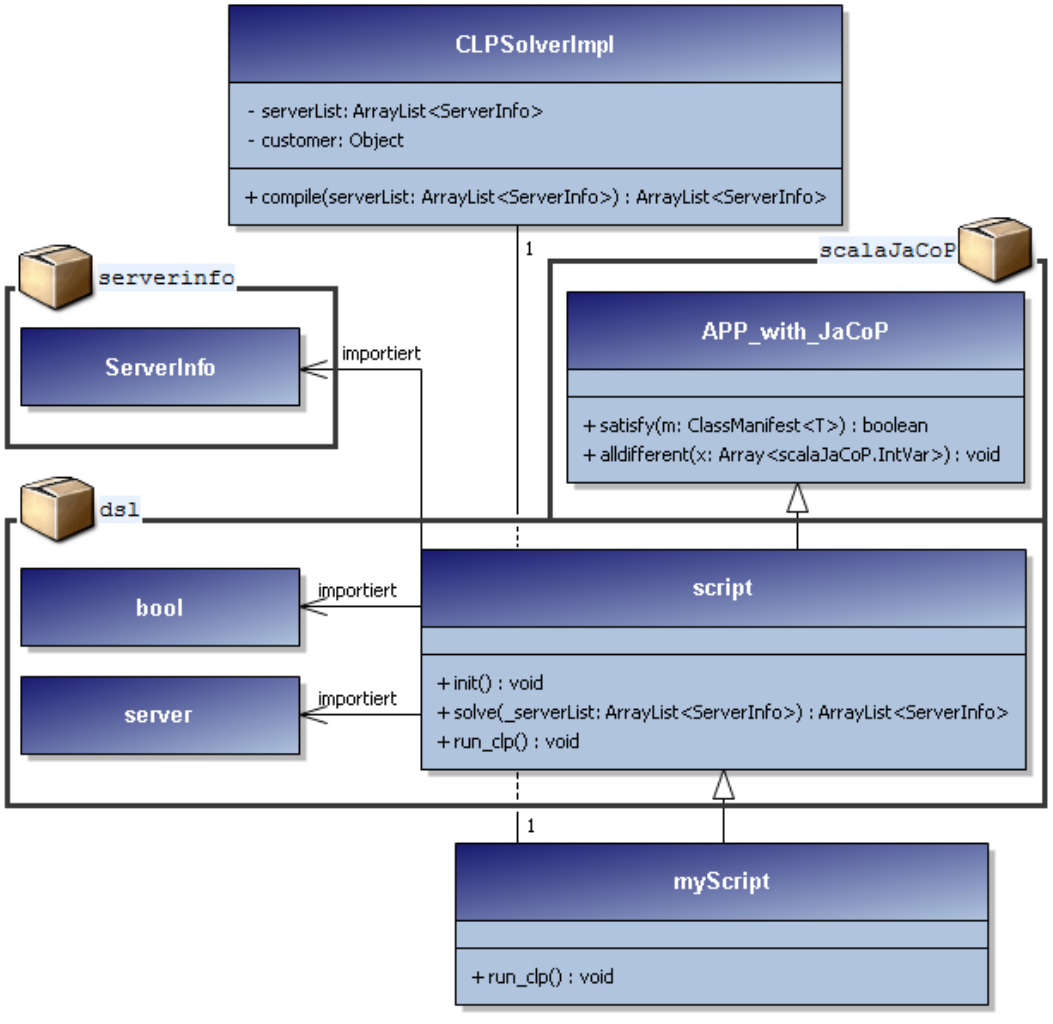
Sichten auf den Dienst



Drei verschiedene Sichten auf den Dienst [3]

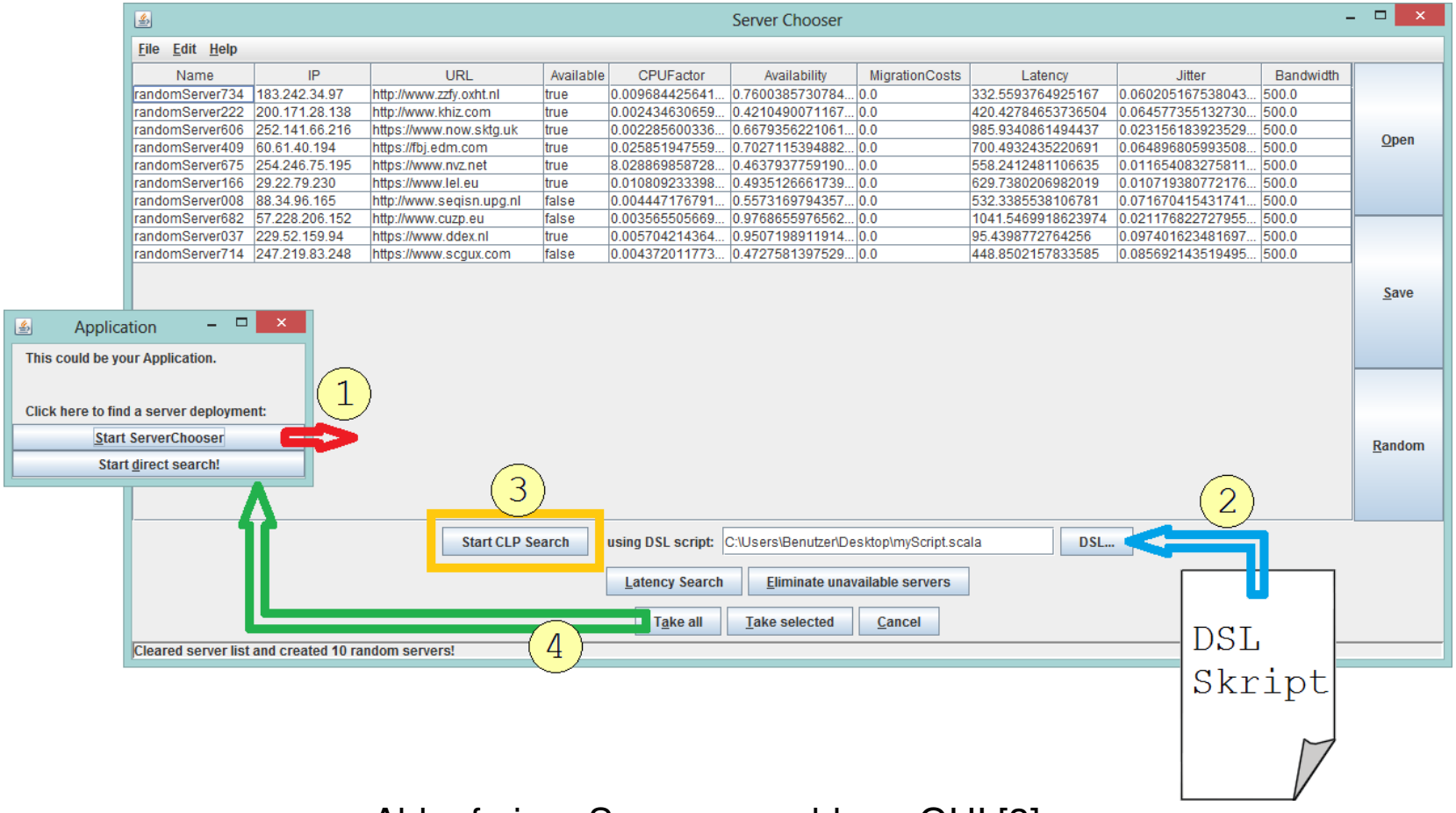
- **1.** - Endanwender, bzw. Entwickler eines Zielsystems.
- **2.** - Entwickler für neue Serverauswahlmethoden.
- **3.** - Entwickler für die Domänensprache.

Implementierung des CLPSolvers



Verbindung der Klassen im CLPSolver-Bundle [3,5]

Demonstration des Dienstes



Ablauf einer Serverauswahl per GUI [3]

Zusammenfassung

- Dienst zur Unterstützung bei der Wahl eines Serverdeployments für verteilte Systeme
  - Zur Optimierung verschiedener systemspezifischer Kriterien

    - Minimieren von Laufzeiten

    - Vermeidung einer Auslastung von Servern

    - ...
  - Leicht zu integrieren, Bedienung ist benutzerfreundlich
  - Domänenspezifische Sprache (einfach, intuitiv, leicht zu erlernen, natürlich wirkende Grammatik)
  - Dienst liefert umsortierte und eingegrenzte Zusammenstellung von Servermenge

Ausblick

- Erweiterungsmöglichkeiten auf verschiedenen Ebenen
  - **DSL** - Grammatik, Alphabet bis zur Wirtssprache, nicht-textuelle DSL
  - **CLPSolver** - Interpretieren, Kompilieren, Makros
  - **ServerChooser** - Alternative Auswahlmethoden, Sortierverfahren, Netzwerkkoordinatendienst [C]
- Miteinbeziehen von Beziehungen zwischen Servern untereinander
- Versuche in der Praxis
  - Wie oft ein Redeployment?
  - Benutzerfreundlichkeit?

Quellen (1)

- Bilderquellen:

[1] Titelbild aus verschiedenen Quellen zusammengestellt:

[1a] http://www.barracudanetworks.ca/images/thumbs/0000623.jpg 28.01.2013

[1b] http://www.scala-lang.org/sites/default/files/newsflash_logo.png 28.01.2013

[1c] http://www.blog.qarea.com/wp-content/uploads/2011/05/java256px.png 28.01.2013

[1d] http://upload.wikimedia.org/wikipedia/de/thumb/e/e1/Java-Logo.svg/170px-Java-Logo.svg.png 28.01.2013

[1e] http://www.osgi.org/wiki/uploads/Main/logo1.jpg 28.01.2013

[1f] http://fileadmin.cs.lth.se/jacop/jacop.png 30.01.2013

[2] Vorlesungsfolien (E.11.1) Virtuelle Präsenz, Winter 2010, P. Schulthess, VS Informatik, Ulm

[3] Michael Grönert

[4] Konstantinos Stamos, George Pallis, Athena Vakali, Dimitrios Katsaros, Antonis Sidiropoulos, and Yannis Manolopoulos. Cdnsim: A simulation tool for content distribution networks. ACM Trans. Model. Comput. Simul., 20(2):10:1-10:40, May 2010.

[5] Erstellt mit http://nclass.sourceforge.net/ 03.02.2013

[6] http://www.osgi.org/wiki/uploads/About/layering-osgi.png 28.01.2013

[7] http://fileadmin.cs.lth.se/jacop/jacop.png 30.01.2013

Quellen (2)

- Wichtige genutzte Literatur:

[A] LLC 2013 YouTube. Youtube - statistics. http://www.youtube.com/yt/press/statistics.html 27.04.2013.

[B] Inc. Amazon Web Services. Amazon simple storage service (amazon s3). http://aws.amazon.com/de/s3/ 14.04.2013

[C] Dominik Guggemos. Entwurf und Implementierung eines Netzwerkkoordinatendienstes für Instant-X. Diplomarbeit an der Universität Ulm

[-] Hall, Richard S. OSGi in Action: Creating Modular Applications in Java. Greenwich [Conn.: Manning, 2011]

[-] Yin, Q., A. Schupbach, J. Cappos, A. Baumann, and T. Roscoe. „Rhizoma: A Runtime for Self-deploying, Self-managing Overlays.“ Lecture Notes in Computer Science. 5896 (2009): 184-204. Print.

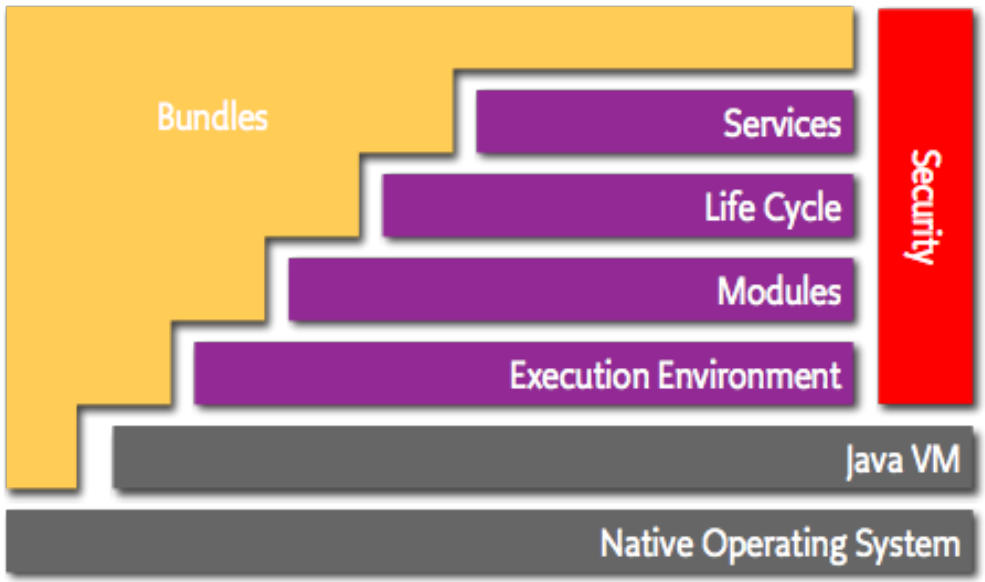
Anhang: *OSGi* (Open Services Gateway initiative)

- Hardwareunabhängige Softwareplattform
- Anwendungen/Dienste modularisieren

→ Durch Komponentenmodell (*Bundles* & *Services*)

→ Verwalten dieser durch eine *Service Registry*

- Baut auf *JVM* (Java Virtual Machine) auf



OSGi - Schichtenmodell [6]

Anhang: *Java* und *Scala*

- ***Java***: **1**. Unisprache, **2**. OSGi, **3**. Aufbau auf existende Projekte

→ Erster Gedanke: Einsatz in ***Space-X*** mit ***Instant-X***

- ***Scala***: **1**. Gut für *DSL*s geeignet, **2**. Perfekte Anbindung an *Java*

→ Syntax erlaubt Anweisungen in Form natürlicher Sprachen

→ Scala kompiliert in *Java*-Bytecode

→ *Java* & *Scala* können Klassen des jeweils anderen direkt verwenden

Anhang: *JaCoP* (Java Constraint Programming solver)

- Open-source *Java*-Bibliothek
  - → Geeignet für *Java* & *Scala*
  - → Experimentelle Entwicklungen: *DSL*s in *Scala*
- Wird seit 2001 aktiv entwickelt

  „The initial reason to write a constraint library in Java was the lack of constraint tools, which could be easily extended and manipulated.“

JaCoP - Open-source Java-Bibliothek [7]