Amonem Projekt v.1.0.0

Barbara Scheuner, Andreas Scherrer, Markus Vitalini

Distributed Systems Lab. WS 04/05. **März 2005**
Supervising Professor: G. Alonso
Supervising Assistant: A. Frei

Table of Contents

© ETHZ-IKS • ALL RIGHTS RESERVED

© ETHZ-IKS • ALL RIGHTS RESERVED

1.1 Motivation

Motivation

Ausgangslage

Jadabs, entwickelt von der IKS Gruppe an der ETH Zuerich, wurde speziell darauf ausgelegt, auch auf kleinen Geraeten (PDA und Handy) zu laufen. Um die im Aufbau begriffene Infrastruktur zu testen existierte ein Ansatz um Jadabs-Knoten auf einem (leistungsfähigen) Rechner zu simulieren.
Diese bestehenden Moeglichkeiten waren eher beschraenkt und erforderten viel Handarbeit. Es existierte nur ein einfaches GUI ueber welches Peers angezeigt und Bundles modifiziert werden koennen. Diese kleine Applikation entstand mit moeglichst wenig Aufwand v.a. aus dem Beduerfnis, die laufende Implementation zu testen.

Ziele

Das Ziel dieses Projektes war, ein Feature ("Buendel" von Plugins; welches einfach zu installieren ist) fuer Eclipse zu schreiben. Dieses Feature soll den Umgang mit der Jadabs Umgebung vereinfachen. Insbesondere soll es moeglich werden, auf eine einfache Art und Weise Peers (PDAs und/oder Handies) zu simulieren und die Konfiguration dieser Peers abzuspeichern und zu einem spaeteren Zeitpunkt wieder zu laden.

Weiter soll es moeglich sein, sich in Reichweite befindende (echte und simulierte) Knoten zu veraendern. Zu den Moeglichkeiten zaehlen das deinstallieren, starten und stoppen von Bundles, sowie in Zukunft auch das installieren von Bundles und hinzufuegen und entfernen von Pipes (Verbindungen zwischen Peers).
Fuer jeden Peer sollen die aktuellen Einstellungen angezeigt werden.

Es existiert theoretisch das Konzept von Gruppen, d.h. Peers lassen sich in Gruppen zusammenfassen. Diese Gruppen sollen ebenfalls dargestellt werden und Peers sollen zu Gruppen hinzugefuegt bzw. aus Gruppen entfernt werden koennen.
Da die Infrastuktur im Moment noch keine Gruppen unterstuetzt wurde dieser Punkt so weit als Moeglich vorbereitet, es existiert aber aktuell nur eine Gruppe (die `WorldGroup`) zu welcher alle Peers automatisch gehoeren.

© ETHZ-IKS • ALL RIGHTS RESERVED

1.2 Installation

Installation des Amonem-Features

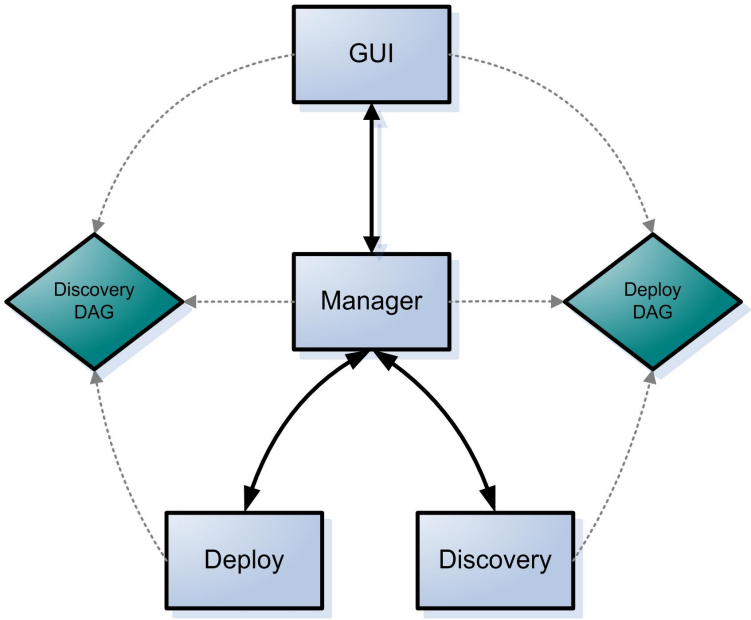
Da es sich bei Amonem um ein Eclipse-Feature handelt, ist die Installation einfach. Das Amonem Feature laesst sich analog zu allen anderen Eclipse Features installieren (siehe auch Eclipse Homepage). Das Feature muss in das Feature-Verzeichnis von Eclipse kopiert werden.

© ETHZ-IKS • ALL RIGHTS RESERVED

2.1 Architektur

Amonem Architektur

Dieses Bild stellt die Architektur des Amonem Features bezueglich seiner Funktionalitaet dar.



Beim Starten des Feature wird als erstes eine Instanz des GUI erzeugt. Dieses startet dann zuerst den Manager, welcher den Deployer und den Discoverer startet.

Wir haben dies so gewaehlt, um das GUI moeglichst von der darunterligenden Struktur zu trennen. Nur die DAGs und der Manager sind dem GUI bekannt.

Kurze Erklaerung der einzelnen Teile:

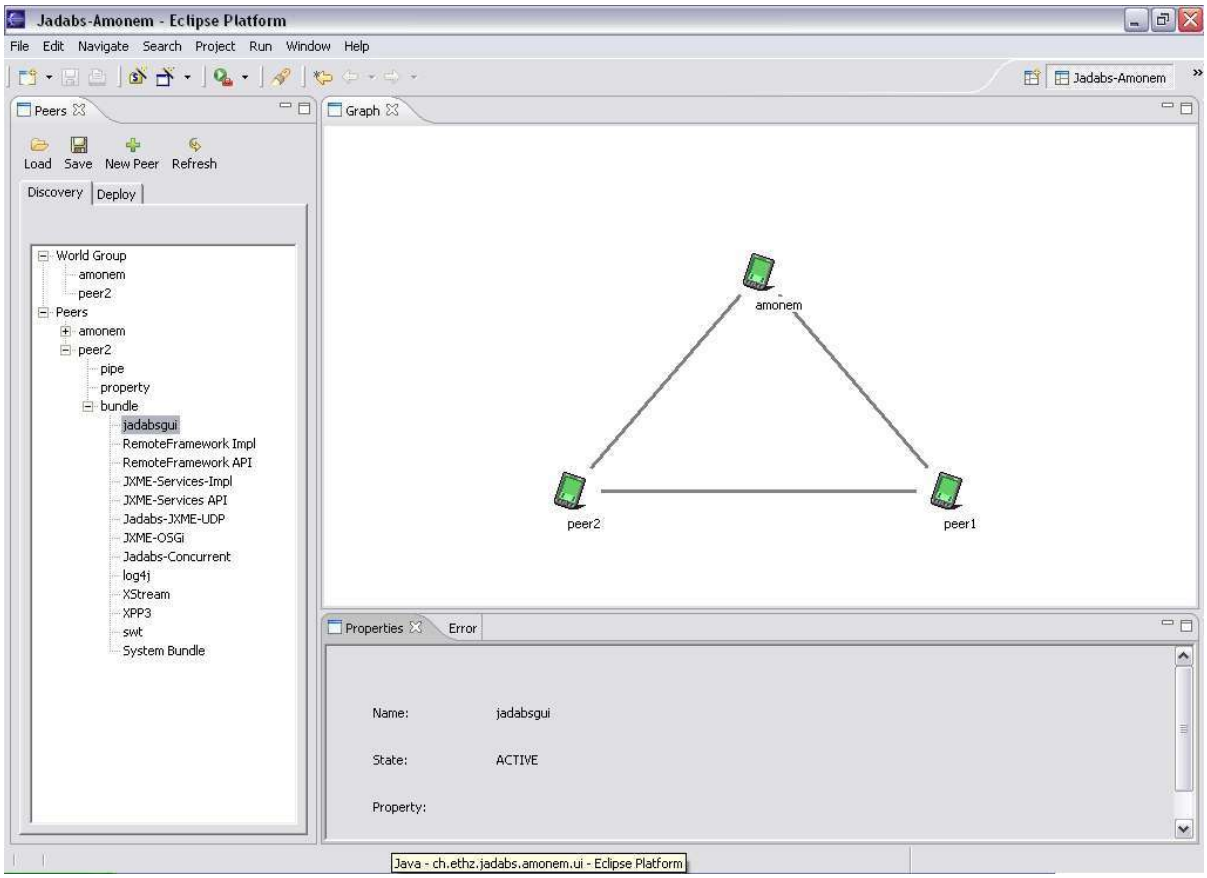
- **GUI :** Graphische Oberflaeche zur Bedienung von Amonem
- **Manager :** Managed die Kommunikation zwischen dem GUI und Deployer, Discoverer. Er ist verantwortlich fuer die DAGs und fuer Import/Export.
- **Deployer :** Verwaltet den Deploy-DAG, d.h. erstellt neue und modifiziert bestehende Peers.
- **Discoverer :** Verwaltet den Dicovery-DAG, d.h. beobachtet die Umgebung und meldet wenn neue Peers auftauchen oder bestehende verschwinden.
- **DAG :** Diese Komponente ist keine eigentliche Klasse, sondern eine Datenstruktur, in der alle Komponenten eines Peer-to-Peer Netzwerks gespeichert sind.

© ETHZ-IKS • ALL RIGHTS RESERVED

2.2 GUI

GUI

Das GUI fuer das Amonem Projekt wurde als Plugin in Eclipse implementiert. Dies war einerseits eine Vorgabe, stellte sich aber spaeter als lohnenswert heraus, weil Eclipse einige Funktionalitaeten wie CVS,... zur verfuegung stellt. Diese erleichtern das Arbeiten in der Gruppe. Eine weitere Bedingung war, das SWT (Standard Widget Toolkit) von IBM zu benutzen.



© ETHZ-IKS • ALL RIGHTS RESERVED

2.2.1 Eclipse

Eclipse

Eclipse bietet eine Vielzahl von Moeglichkeiten fuer die Integration von Plugins. Es koennen neue Menueeintraege, Knoepfe und sogar ganze Perspektiven in das Programm eingeführt werden (Perspektiven sind ganze Umgebungen, die sich aus verschiedenen Fenstern (Views) zusammensetzen).

Um die Entwicklung eines neuen Plugins moeglichst zu vereinfachen steht ein Plugin Wizard zur Verfügung. Dieser hilft bei der Zusammenstellung der verschiedenen Views und generiert automatisch ein `plugin.xml`, welches alle Informationen ueber das Plugin enthaelt. In dieser Datei müssen die Perspektiven und die Views definiert werden.

Perspective

Eine stark vereinfachte Fassung einer `plugin.xml` Datei, in welcher eine Perspektive deklariert wird.

```
...
point="org.eclipse.ui.perspectives"

<perspective>
        class="JadabsPerspective"
        name="Jadabs-Amonem"
        id="jadabs-amonem-perspective"
</perspective>
...
```

point: Damit Eclipse weiss, um was fuer eine Art von Plugin es sich handelt, muss man einen point deklarieren. Dieser gibt an, ob das Plugin eine neue Perspektive, ein Menueeintrag oder ein neuer Knopf sein soll.

class: Deklarationen, welche die Perspektive und die Views beschreiben, reichen noch nicht aus um das Layout des Plugins zu beschreiben. Es wird noch eine Klasse benoetigt, in welcher die verschiedenen Views an bestimmten Orten in der Perspektive platziert werden. Damit Eclipse auch erkennen kann, dass es sich bei dieser Klasse um eine "Perspective Factory" handelt, muss sie von `IPerspectiveFactory` erben.

name: Name des Plugins.

id: Das Plugin muss eine eindeutigen Identifikation haben.

Views

Eine stark vereinfachte Fassung einer `plugin.xml` Datei, in welcher zwei Views deklariert werden.

© ETHZ-IKS • ALL RIGHTS RESERVED

```
...
<view>
        class="GraphView"
        category="jadabs"
        name="Graph"
        id="GraphViewId">
</view>
<view>
        class="NewPeerView"
        category="jadabs"
        name="New Peer"
        id="NewPeerViewId"
</view>
...
```

class: Jede View wird von einer Klasse beschrieben. Der Name diese Klasse muss hier angegeben werden. Damit Eclipse sie auch als solche erkennt, muss sie von `ViewPart` erben. `ViewPart` ist eine abstrakte Klasse, die die Methode `createPartControl()` definiert. In dieser können alle SWT Elemente instanziert werden, die in den jeweiligen Views dargestellt werden sollen.

category: Eine View kann einer Kategorie zugeordnet werden. Dies dient der Uebersicht und kann nützlich sein, wenn eine View geschlossen wird und man sie spaeter wieder oeffnen will. Gibt es viele Views, die keiner Kategorie zugewiesen wurden, so wird die Suche unuebersichtlich.

name: Name der View.

id: Die View muss eine eindeutigen Identifikation haben.

Views registrieren

Wie bereits in der Perspektive erwaehnt, gibt man eine Klasse an, die von `IPerspectiveFactory` erbt. Hier wird konkret angegeben, wo in der Perspektive die verschiedenen Views plaziert werden sollen. Das folgende Beispiel soll zeigen, wie die "PeersView" von Amonem registriert wird.

```
...
IFolderLayout lists = layout.createFolder("List", IPageLayout.LEFT, (float) 0.26,
editorArea);
lists.addView("PeersViewId");
...
```

Der Perspektive von Eclipse (`editorArea`) wurde nun ein neues Fenster zugeordnet. Es befindet sich am linken Rand der Perspektive (`IPageLayout.LEFT`). Diesem Bereich kann man nun die benoetigten Views zuordnen, indem man sie mit der ViewId registriert (`lists.addView("PeersViewId")`.

© ETHZ-IKS • ALL RIGHTS RESERVED

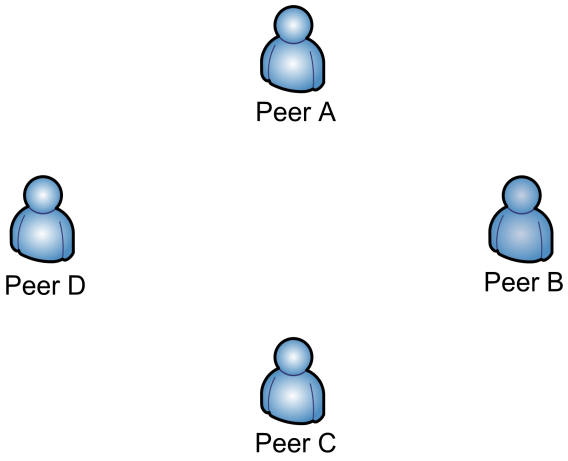
2.2.2 Graph

Graph

Die Generierung des Graphen bringt einige Probleme mit sich. Die groesste Schwierigkeit liegt wahrscheinlich in der Tatsache, dass sich die Peers in Gruppen befinden. Es ist auch moeglich, dass sich ein Peer in mehreren Gruppen befindet. Dies macht es noetig, einen DAG als Datenstruktur zur Speicherung der Peers zu benutzen. Wie dieser DAG genau aussieht, wird in einem eigenen Kapitel behandelt (Kapitel 2.3.1 DAG).

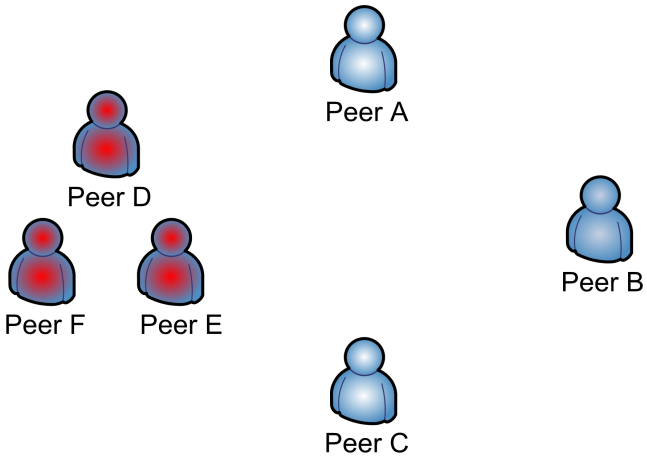
Graph generieren

Die Grundidee des Algorithmus ist sehr einfach. Die `WorldGroup` (Wurzelelement des DAGs) hat direkte Nachfolger. Ein solches Element kann ein Peer oder eine weitere Untergruppe sein. Diese werden nun, unabhaengig vom Typ, auf der zugrundeliegenden Zeichenflaeche auf einem Kreis angeordnet. Mit einer `WorldGroup`, bestehend aus vier Peers, wuerde sich folgendes Bild ergeben.



Beruecksichtigt man nun, dass es weitere Untergruppen in der `WorldGroup` geben kann, so wendet man den Algorithmus einfach nochmals auf diese Untergruppe an (als Zeichenflaeche verwendet man jetzt einen kleineren Bereich um das Gruppenelement). Mit dieser Strategie kann man solange weiterfahren, bis alle Gruppen behandelt wurden. Mit einer Untergruppe, bestehend aus drei Peers, wuerde sich dann folgendes Bild ergeben:

© ETHZ-IKS • ALL RIGHTS RESERVED



Befindet sich ein Peer in mehreren Gruppen, soll dieser natuerlich nur einmal im Graphen vorkommen. Aus diesem Grund wird jeder Peer (zur Darstellung) nur einer Gruppe zugeordnet.

Methoden

Folgende drei Methoden wurden implementiert:

Die Methode **`generateGraph()`** kann aufgerufen werden um den ganzen Graphen zu zeichnen. Sie verwendet `computeGraph()` und `computeLocation()`. Als Argumente sind lediglich die `WorldGroup` und die Zeichenflaeche notwendig.

`computeLocation()` nimmt eine Zeichenflaeche und einen Vector mit Gruppenelementen (Peers oder Untergruppen) entgegen. Die Positionen dieser Elemente werden so berechnet, dass sie auf einem Kreis innerhalb der Zeichenflaeche zu liegen kommen werden.

`computeGraph()` unterscheidet zwischen Peers und Gruppen. Hier wird sichergestellt, dass Untergruppen korrekt behandelt werden.

© ETHZ-IKS • ALL RIGHTS RESERVED

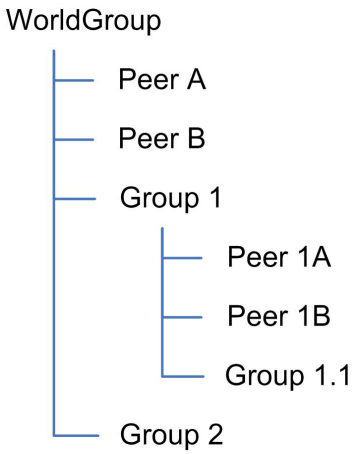
2.2.3 Peer Liste

Peer Liste

Die "Peer Liste" enthaellt alle Informationen ueber die Peers, Gruppen und Untergruppen. Hier ist ersichtlich, welche Peers in welcher Gruppe sind und welche Bundles und Eigenschaften ein Peer hat. Um diese Informationen uebersichtlich darzustellen, kann man die Infofrmationen, ausgehend von zwei Wurzelelementen, abrufen.

World Group

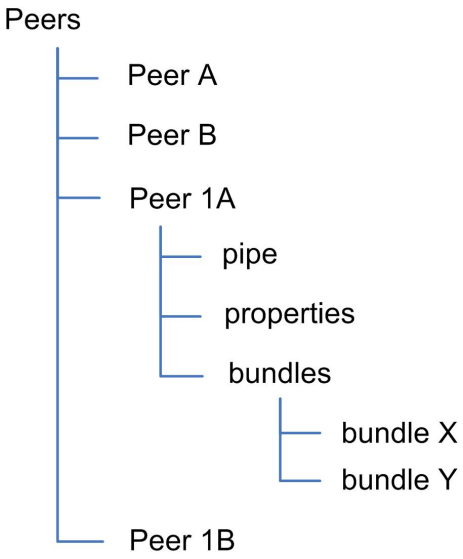
Die `WorldGroup` stellt die Struktur des DAGs dar. Hier ist ersichtlich, welche Elemente zu welcher Gruppe gehoeren. Dabei kann es vorkommen, dass ein Peer zweimal aufgelistet wird (da er in mehreren Gruppen sein kann).



Peers

In `Peers` werden alle Peers der `WorldGroup` aufgelistet. Von diesen aus koennen weitere Informationen ueber Bundles, Verbindungen und weitere Eigenschaften abgefragt werden.

© ETHZ-IKS • ALL RIGHTS RESERVED



© ETHZ-IKS • ALL RIGHTS RESERVED

2.3 Manager

Manager

Der Manager bildet die Schnittstelle zwischen Deployer , Discoverer und GUI . Er bietet dem GUI ueber sein Interface alle wichtigen Methoden von Deployer und Discoverer an. Darueber hinaus koordiniert er die Erzeugung von Peers, den Import und den Export von Konfigurationen und Einzelteilen der DAGs .

Aufgaben

Die Hauptaufgabe des Managers ist die Verwaltung der DAGs. Davon gibt es zwei Instanzen, den DiscoveryDAG und den DeployDAG.
Der *DeployDAG* enthaelt alle Peers, die vom Benutzer erzeugt wurden. Erzeugt werden kann direkt ueber ein View oder (wie spaeter erklaert) durch das laden eines XML-Files. Der *DiscoveryDAG* speichert jene Peers, welche vom Discoverer entdeckt wurden. Diese Peers sind lokal oder remote aktiv.
Diese beiden Datenstrukturen muessen unterschieden werden, denn ein vom Benutzer erzeugter Peer kann viel mehr Information enthalten, weil zum Starten eines Peers mehr Information noetig sind, als aus einem "nur" aktiven Peer vom Discoverer herausgelesen werden kann. Eine solche Information ist z.B. der Prozess, in dem der Peer laeuft. Dieser wird fuer das Stoppen der Entitaet benoetigt. Auch Informationen ueber die installierten Bundles mit der URL, wo sie sich befinden, werden in einem vom Benutzer erzeugten Peer gespeichert.

Der Manager ist auch dafuer verantwortlich, den Discoverer und den Deployer zu starten, und ihnen Zugang zum jeweiligen DAG zu gewaeren. Der Informationsfluss zwischen diesen beiden Objekten und dem GUI laeuft vollstaendig ueber den Manager. Das GUI propagiert die Inputs des Users nach unten zu den beiden Objekten. Umgekehrt wird das GUI informiert, wenn eines von ihnen an seinem DAG etwas geaendert hat (einfuegen/loeschen eines Peers, Aenderung von Bundle Informationen). Dies geschieht mittels `PeerListener`s und `GroupListener`s.

Methoden fuer das GUI

- `start()`: Diese Methode dient dazu den Manager zu starten. Der Manager erzeugt dann die ROOT-Gruppen fuer die beiden DAGs, und einen Deployer und einen Discoverer.
- Methoden fuer die Manipulation von Peers: Dieses Aufgabenpaket umfasst das Starten/Stoppen von Peers und das Kontrollieren dessen Bundles (installieren, starten, stoppen). Diese Methoden werden jeweils an die zustaendige Klasse (Deployer , Discoverer) weitergereicht.
- Methoden fuer das Importieren und Exportieren von Peers oder ganzen Konfigurationen. Hierfuer wurde SAX verwendet.
- Eine Methode, welche die Bundle-Informationen eines XML-Files herausliest, und dem GUI als Vektor zurueckgibt.

© ETHZ-IKS • ALL RIGHTS RESERVED

Export

Es koennen nur Peers exportiert werden, die sich im DeployDAG befinden. Dies bedeutet, dass eine speicherbare Konfiguration dem ganzen DeployDAG entspricht, und nicht dem DiscoveryDAG, also den aktiven Peers. Gruppen koennen nicht einzeln abgespeichert werden, da sie auch nicht erzeugt werden koennen. Im XML-File der ganzen Konfiguration treten Gruppen zwar auf, werden beim Einlesen aber im Moment noch ignoriert. Dies muesste zu dem Zeitpunkt angepasst werden, an dem die Gruppenerzeugung moeglich ist.
Von einem Peer werden folgende Attribute abgespeichert: `name, deploypath, javapath, platform, bundles, parents`. Im Tag `<bundles>` werden die einzelnen Bundles mit ihren Attributen gespeichert. Dies sind: `bundle-name, update-location, uuid`. Der Tag `<parents>` enthaelt momentan bei allen Peers nur die ROOT-Gruppe.

Das exportieren eines ganzen DAGs oder aber auch nur einzelner Peers funktioniert gleich. Dabei werden die folgenden Schritte ausgefuehrt:

- Zuerst wird ein DOM-Dokument (`org.w3c.dom.Document`) erzeugt. Diese Struktur besteht aus Elementen, die den XML-Tags entsprechen. Der ganze DAG (oder aber auch nur ein einzelner Peer) wird dann gemaess der geforderten XML-Struktur in dieses Dokument uebertragen.
- Dieses Dokument wird dann in ein XML-File geschrieben. Wenn eine ganze Konfiguration gespeichert wird, bekommt diese vom Benutzer einen Namen. Daraufhin wird ein Ordner mit diesem Namen erzeugt, und darin die einzelnen XML-Files abgelegt. Es gibt ein File fuer die ganze Konfiguration, und je eines fuer die einzelnen Peers. Gruppen werden nicht einzeln abgespeichert.

Import

Wie oben erwaehnt, werden beim Importieren nur die Peers wieder eingelesen. Bei diesem Vorgang wird auch geprueft, ob alle noetigen Tags vorhanden sind. Ist dies nicht der Fall, wird der Peer nicht geladen.

Es gibt zwei Arten von Files die geladen werden koennen. Die eine enthaelt einen einzelnen Peer, die andere eine ganze Konfiguration. Diese beiden Faelle werden unterschiedlich behandelt.

- Beim Laden einer Konfiguration werden zuerst alle Vorhandenen Peers im DeployDAG geloescht. Dann wird der DAG mit den eingelesenen Peers gefuellt.
- Wird ein einzelner Peer geladen, dann wird der DeployDAG beibehalten. Es wird ueberprueft, ob sich in diesem DAG schon ein Peer mit demselben Namen befindet. Ist dies der Fall, wird der alte Peer durch den importierten ersetzt.

In beiden Faellen kann der Benutzer waehlen, ob er die Peers nur laden, oder auch noch starten will.

Beim Importieren eines Files (egal ob ein Peer, oder eine ganze Konfiguration) wird zuerst mit einem Parser ein DOM-Dokument `org.w3c.dom.Document` generiert.
Aus dessen Baumstruktur von Elementen werden dann die noetigen Informationen ausgelesen, und gleichzeitig auf ihr Vorhandensein geprueft.

```
Document D= DocumentBuilderFactory.newInstance().newDocumentBuilder().parse(new
File(name))
```

© ETHZ-IKS • ALL RIGHTS RESERVED

2.3.1 DAG

DAG

Diese Datenstruktur verwaltet die Gruppen und deren Peers. Sie ist ein "Directed Acyclic Graph" , da es moeglich ist, dass ein Peer oder eine Gruppe in mehrere Gruppen gehoert. Die ueber allen stehende Gruppe, der alle angehoeren, heisst `WorldGroup` und wird meist als ROOT bezeichnet.

DAGMember

Hier sind alle Informationen gespeichert, die auf Peers und Gruppen zutreffen.

```
protected NamedResource Resource;
protected Framework FW;
protected int X;                // Koordinaten (Graph)
protected int Y;
protected int Type;                // 1=Goup; 2=Peer
protected String Name;
protected Vector Parents;
protected Hashtable Properties;
```

Beide Klassen, `DAGGroup` und `DAGPeer`, erweitern diese Klasse mit speziefischen Attributen und Methoden.

DAGGroup

Die Gruppe besitzt neben dem Vector mit den Uebergruppen (`Parents`) auch noch einen Vector, der die Kinder enthaelt (`Children`).
Einer Gruppe kann man Kinder anfuegen und wieder entfernen. Ueber den Namen kann man in einer Gruppe nach einem Kind-Element suchen.
Anmerkung: Gruppen koennen von diesem Fueature nicht erzeugt werden. Im Moment gibt es nur eine `WorldGroup`, welche alle Peers enthaelt. Die Datenstruktur erlaubt jedoch einen komplexeren Graphen.

DAGPeer

Ein Peer bietet Attribute mit den zugehoerigen Gettern und Settern an, um alle Informationen abzuspeichern, welche fuer den Benutzer interessant sein koennten, und die zum starten eines Peers benoetigt werden.

```
private Vector Connections;
private Process myProcess;
private Vector Jars;
private String JavaPath;
private Vector Bundles;
private String Platform;
```

© ETHZ-IKS • ALL RIGHTS RESERVED

```
private String DEPLOY_PATH;
private BundleInfoListener listener;
private boolean Deployed;
```

Nicht alle diese Informationen koennen aus einem beliebigen (nicht mit diesem Feature gestarteten) Peer herausgelesen werden. Deshalb werden alle Felder nur im DeployDAG verwendet.
Neben den ueblichen Gettern und Settern gibt es noch eine Methode mit der man nach einem Bundle suchen kann. Wahlweise kann man mit der `BundleID` oder dem Namen des Bundles suchen.
Anmerkung: Verbindungen (`Connections`) zwischen zwei Peers sind im Moment noch nicht abrufbar. Deshalb wird die im Code vorbereitete Klasse `DAGPipe` nicht verwendet.

DAGBundle

Ein `DAGBundle` speichert alle vorhandenen Informationen zu einem Bundle. `Name, UpdateLocation` und `UUID` sind noetig, um einen neuen Peer zu starten und werden deshalb auch beim Exportieren im XML-File gespeichert und daraus wieder ausgelesen.
`BundleID` und `State` sind nur relevant, wenn der Peer, und somit auch das Bundle, aktiv ist.

```
private long BundleID;
private String Name;
private int State;
private String Property;
private String UpdateLocation;
private String UUID;
```

DAGIterator

Diese Klasse stellt verschiedene Methoden zur Verfuegung, welche das Arbeiten mit dem DAG erleichtern. Im Speziellen kann man eine Liste aller Peers abrufen, bei der jeder Peer nur einmal aufgefuehrt wird. Dasselbe gibt es auch fuer die Gruppen. Der bei der Konstruktion uebergebene ROOT-Knoten dient als Grundlage fuer folgende Methoden:

- `newPeerEnumeration(), getNextPeer(), hasMorePeers()` werden gebraucht, um die Liste aller Peers durchlaufen zu koennen. Dabei wird beim Aufruf von `newPeerEnumeration` eine Liste der momentanen im DAG befindlichen Peers erzeugt. Auf dieser Liste werden dann die weiteren Aufrufe getaetigt.
- `newGroupEnumeration(), getNextGroup(), hasMoreGroups()` ermoeglichen eine Auflistung aller Gruppen auf die gleiche Wiese wie bei den Peers.
- `PeerIsMember(DAGPeer peer), GroupIsMember(DAGGroup group)` testen, ob sich eine Gruppe, oder ein Peer schon im DAG befindet. Dabei handelt es sich um einen Test der Objektgleichheit. Es wird nicht getestet, ob ein Element mit demselben Namen schon im DAG ist.

© ETHZ-IKS • ALL RIGHTS RESERVED

2.4 Discovery

Discovery

Der Discovery "Service" ist dafuer zustaendig, andere Jadabs-Knoten (Peers) zu entdecken und in den Discovery-DAG einzufuegen, damit sie im GUI angezeigt werden koennen. Dabei wird nicht zwischen simulierten und echten Peers unterschieden.
Das heisst, auch von Amonem simulierte Peers werden erst im GUI sichtbar, nachdem sie vom Discovery Service erkannt wurden.
Umgekehrt ist der Discovery Service auch fuer das Entfernen von Peers aus dem Discovery-DAG zustaendig. Dies bedeutet insbesondere, dass ueber das GUI beendete Peers erst nach einer gewissen Zeit aus der Anzeige verschwinden.

Der Discovery Service reagiert v.a. auf Events des Jadabs-Service Bundles (`EnterFramework-` und `LeaveFrameworkEvent` des `RemoteFrameworkListeners`). Aus diesem Grund ist er sehr abhaengig vom Funktionieren dieses Bundles.

Weiter soll der Discovery Service verwendet werden koennen, um Gruppen von Peers zu erkennen. Da diese im Moment noch nicht implementiert sind, ist diese Funktionalitaet nicht gewaehrleistet.
Es werden einfach alle gefundenen Peers in die `WorldGroup` eingefuegt.

Die Grundidee

Der Amonem Discovery Service ist selber ein Jadabs Knoten auf welchem die `ch.ehtz.jadabs.jxme.services` laufen. So kann der Discovery ueber dieselben Mechanismen wie die "richtigen" Knoten herausfinden, welche Peers in seiner Umgebung vorhanden sind. Aus diesem Grund erscheint die Amonem Applikation auch im Amonem-Graphen.
Da (noch) keine Sicherheitsfunktionalitaet existiert, kann der Discovery auf diese Art und Weise unkompliziert alles herausfinden, was das Amonem Tool ueber einen Peer wissen muss.

Der Discovery wird sobald das Plugin gestartet wird vom Amonem Manager gestartet. Von diesem bekommt er dann auch die zur Erfuellung seiner Aufgaben noetigen Referenzen auf den `ch.ehtz.jadabs.jxme.services.GroupService` der `WorldGroup` und den `ch.ethz.jadabs.remotefw.FrameworkManager`.

Peers einfuegen und entfernen

Neuer Peer kommt hinzu

Kommt ein neuer Peer zum Netzwerk hinzu, loest dies in Jadabs einen `EnterFrameworkEvent` aus. Auf diesen Event reagiert der Amonem Discovery Service.
Die folgenden Schritte werden in der Reihenfolge in der sie aufgelistet sind ausgefuehrt:

- Ueberpruefen, ob im Discovery-DAG schon ein Peer mit diesem Namen existiert. Ist dies der Fall

© ETHZ-IKS • ALL RIGHTS RESERVED

werden die fogenden Schritte nicht mehr ausgefuehrt

- Neuen Peer (DAGPeer Objekt) mit dem Namen des neuen Peers erstellen
- Jadabs Framework "merken", d.h. die Referenz die mit dem Event mitgeliefert wurde im DAGPeer Objekt speichern
- Einen `ch.ethz.jadabs.remotefw.BundleInfoListener` registrieren (damit Aenderungen an auf dem Peer installierten Bundle angezeigt werden koennen) und die Referenz speichern, damit der Listener später wieder entfernt werden kann
- Den Peer analysieren (welche Bundles sind in welchem Zustand auf dem Peer)

Dabei ist anzumerken, dass der letzte Punkt einige Probleme aufweist. Das Problem ist, dass die Bundle-Informationen von `ch.ethz.jadabs.remotefw.Framework` geliefert werden muessten, dies aber nur manchmal funktioniert. Oft kommt der Event nicht oder zu einem unerwarteten Zeitpunkt (z.B. immer "einen Event zu spaet").

Peer verlaesst das Netzwerk

Auf das Verlassen des Netzwerks durch einen Peer reagiert Jadabs ebenfalls mit einem Event (`LeaveFrameworkEvent`). In dieser Situation werden folgende Schritte in der Reihenfolge in der sie aufgelistet sind ausgefuehrt:

- Peer im Discovery-DAG suchen. Zuerst wird nach einem Peer mit demselben `ch.ethz.jadabs.remotefw.Framework` gesucht, gibt es keinen solchen, wird nach einem mit demselben Namen gesucht. Gibt es auch den nicht, werden die folgenden Schritte nicht ausgefuehrt.
- Den `ch.ethz.jadabs.remotefw.BundleInfoListener` vom DAGPeer entfernen
- Den Peer aus dem Discovery-DAG herausnehmen (Achtung: falls es einmal mehr Gruppen geben kann, muss hier Code angepasst werden).
- Den Amonem Manager informieren, dass sich etwas geaendert hat (Listener aufrufen).

Gruppen erkennen

Diese Funktionalitaet ist nicht implementiert. Alle Peers befinden sich (ausschliesslich) in der `WorldGroup`, einer immer existierenden Obergruppe. Es bleibt auch noch zu definieren, was mit einer Gruppe passiert, wenn ihr letztes Mitglied das Netzwerk verlaesst.

© ETHZ-IKS • ALL RIGHTS RESERVED

2.5 Deploy

Deploy

Der Deploy Teil der Amonem Anwendung ist dafuer verantwortlich, Peers die simuliert werden sollen zu "deployen", also die benoetigten Dateien zu kopieren und den Peer zu starten (d.h. eine neue Java Virtual Machine (JVM) zu starten). Weiter muss er an bereits laufenden Peers Veraenderungen machen koennen (neue Bundles installieren, ein Bundle starten/stoppen oder ein Bundle deinstallieren).
Der Deploy verwaltet den Deploy-DAG. Dieser enthaelt nur Eintraege fuer Peers, die von Amonem simuliert werden, keine fuer Peers die vom Discovery entdeckt wurden.
Der Deploy wird vom Manager gestartet und erhaelt von diesem dabei die Root des Deploy-DAGs.

Der Deploy kann die fuer die Lauffaehigkeit eines Peers noetigen JAR-Dateien anhand einer URL (also entweder aus einem lokal vorhandenen Verzeichnis oder vom Internet) kopieren und den Peer dann mit diesen starten. Die Liste der zur Verfuegung stehenden JARs wird aus einem XML-Dokument nach ???berlios??? erstellt.
Da der Deploy selbstverstaendlich nicht wissen kann, welche Bundles installiert werden sollen, muss der Benutzer dies angeben.

Grundidee

Der Deploy soll nach Anweisung des Benutzers JAR-Dateien kopieren und neue Peers starten. Um den Speicherbedarf in Grenzen zu halten soll er alle noetigen JAR-Dateien nur einmal kopieren. Um die Hexibilitaet zu steigern, soll es *moeglich* sein, als Quelle fuer ein JAR eine URL ins Internet anzugeben. Die URL kann aber auch auf ein File auf der lokalen Platte zeigen.
Dasselbe gilt fuer die Datei, welche angibt, welche JAR-Dateien zur Auswahl stehen und wo diese zu finden sind (wir nennen diese Datei im Weiteren Repository-XML).

Urspruenglich sollte der Deploy auch Dateien auf ein entferntes System kopieren koennen. Diese Idee mussten wir aus Zeitgruenden leider verwerfen, so dass im Moment nur in ein lokales Verzeichnis deployed werden kann.

Neuen Peer deployen

Der Deploy bekommt ein vorgefertigtes `AmonemDeploySkeleton` mit allen fuer das Deployment wichtigen Informationen. Diese umfassen

- Name des Peers
- JAR der Platform (UUID und URL)
- UUIDs der zu installierenden JARs (Bundles)
- URLs der zu installierenden JARs (Bundles)
- temporaeres Verzeichnis

© ETHZ-IKS • ALL RIGHTS RESERVED

- Deploy Verzeichnis
- Pfad zum xargs-Template (muss lokal sein)

Das temporaere Verzeichnis wird benoetigt um Dateien (JARs) die heruntergeladen werden muessen zu speichern. Im Deploy Verzeichnis wird ein Ordner mit dem Namen des Peers erstellt und darin alle fuer den Peer relevanten Daten abgelegt (xargs Datei und die Runtime-Dateien). Die JARs werden direkt im Deploy Verzeichnis abgelegt (einmal fuer alle Peers mit diesem Deploy-Verzeichnis). Das xargs-Template ist eine Vorlage fuer das zu erstellende xargs-File. Es enthaelt an zwei Stellen Platzhalter, die dann ersetzt werden. Die Platzhalter heissen `<CH.EHTZ.JADABS.AMONEM.JARSPACE>` und `<CH.EHTZ.JADABS.AMONEM.STARTSPACE>`.
Achtung: Die Platzhalter heissen wirklich so (...EHTZ...), das ist ein Tippfehler den wir nicht mehr korrigieren konnten.

Erstellen der `xargs` Datei

Die xargs Datei wird aus dem oben erwaehnten Template erstellt, indem die Platzhalter durch peerspezifische Anweisungen ersetzt werden. `<CH.EHTZ.JADABS.AMONEM.JARSPACE>` wird dabei durch Eintaege der Form `-install <Pfad zum JAR File>` ersetzt.
`<CH.EHTZ.JADABS.AMONEM.STARTSPACE>` durch `-start <Nummer des JAR Files>`.
Hiermit werden zur Laufzeit alle Bundles mit einem `-install` Eintrag auf dem Peer installiert und alle mit einem `-start` Eintrag zusaetzlich gestartet. In der momentanen Implementation wird für jeden `-install` Eintrag auch ein `-start` Eintrag geschrieben.
Dann wird das neue `xargs` File ins Deploy Verzeichnis in einen Ordner mit dem Namen des Peers kopiert.

Kopieren der benoetigten Dateien

Die noetigen JARs werden entweder von der angegebenen URL heruntergeladen (in das temporaere Verzeichnis). Dann werden sie vom temporaeren Verzeichnis (oder wenn eine lokale URL angegeben wurde von dort) ins Deploy Verzeichnis kopiert. Existiert im temporaeren Verzeichnis schon eine Datei mit demselben Namen wie eine, die heruntergeladen werden soll, wir diese *nicht* heruntergeladen. Dasselbe gilt für das kopieren ins Deploy Verzeichnis: existiert dort schon eine Datei mit demselben Namen, wird *nicht* kopiert.

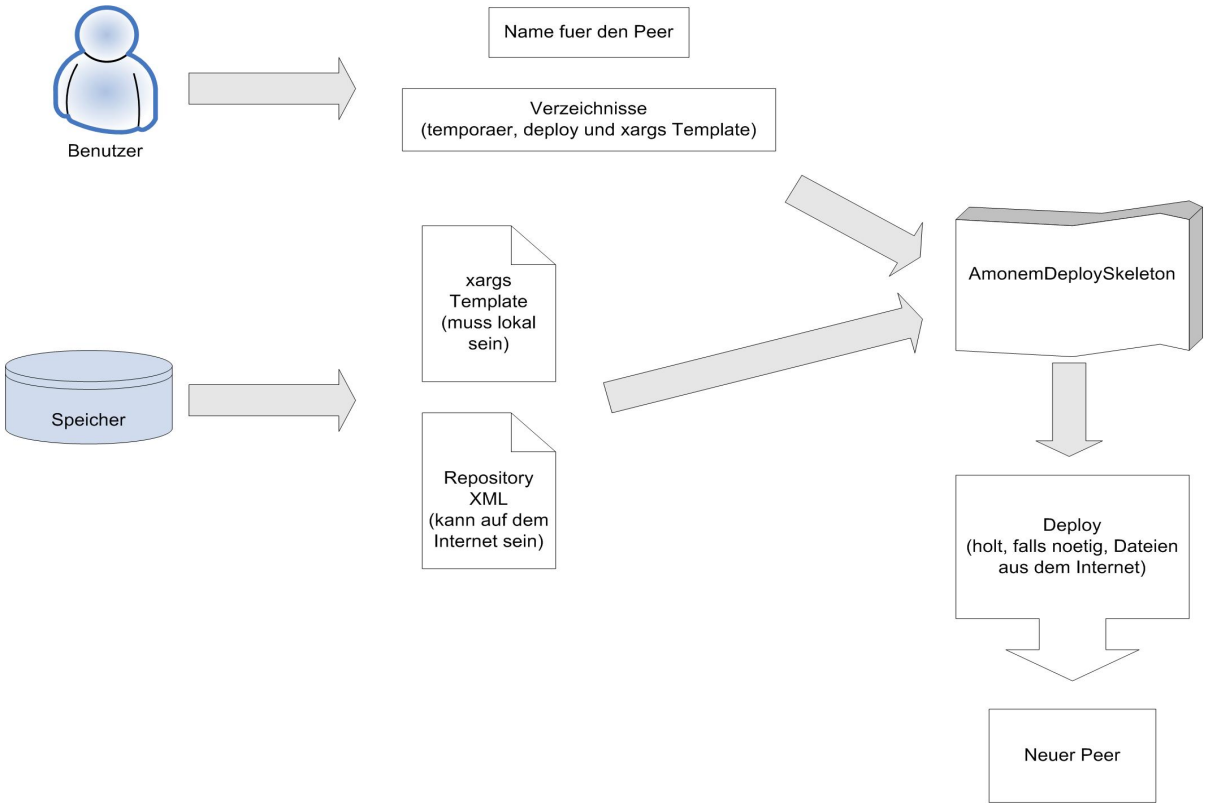
Starten des Peers

Dann wird ueber `Runitme.exec` eine neue JVM gestartet. Hierbei muss darauf geachtet werden, dass (mindestens auf WindowsXP) die Output-Streams (Ouput und Error) des Prozesses gelesen werden, da die Applikation sonst blockiert. Ein Aufruf von `Runtime.exec` sieht ungefaehr folgendermassen aus:

```
C:\Programme\Java\j2re1.4.2_05/bin/java -Dch.ethz.jadabs.jxme.peeralias=gintonic
-Dorg.knopflerfish.gosg.jars=file:C:\tmp\ -jar
C:\tmp\osgi/jars/framework-1.3.0-aop.jar -xargs C:\tmp\gintonic/gintonic.xargs
```

Schematische Darstellung

© ETHZ-IKS • ALL RIGHTS RESERVED



Bestehenden Peer veraendern

Bundles auf einem bestehenden (simulierten oder echten) Peer koennen veraendert werden. Es stehen folgende vier Moeglichkeiten zur Verfuegung:

- Bundle starten
- Bundle stoppen
- Bundle (de-)installieren

Diese Funktionalitaet wird von `ch.ethz.jadabs.remotefw.Framework` zur Verfuegung gestellt, der Deploy gibt nur die Aufrufe weiter. Starten, stoppen und entfernen geschieht ueber die Bundle ID (BID), installieren braucht den (lokalen) Pfad zum JAR File, das installiert werden soll.
Da nur Datenpakete die maximal 8 KByte gross sind verschickt werden koennen, darf das zu installierende Bundle (bzw. das JAR) hoechstens 8 KByte gross sein.
Aktuell scheint das Installieren von Bundles auf der Jadabs-Seite noch fehlzuschlagen. Auf jeden Fall gibt `installBundle` konsequent -1 zurueck (hartcodiert).

Dateien

- `xargs-Skeleton` Beispiel
- `xargs` Beispiel
- `Repository-XML` Beispiel

© ETHZ-IKS • ALL RIGHTS RESERVED

2.5.1 xargs skeleton

Beispiel fuer ein xargs Skelett

```
#
# Generated from template.xargs
# Knopflerfish release 1.3.0a4
#

-Dorg.knopflerfish.verbosity=0

-Dorg.knopflerfish.framework.debug.packages=false
-Dorg.knopflerfish.framework.debug.errors=true
-Dorg.knopflerfish.framework.debug.classloader=false
-Dorg.knopflerfish.framework.debug.startlevel=false
-Dorg.knopflerfish.framework.debug.ldap=false

-Dorg.osgi.framework.system.packages=ch.ethz.jadabs.osgiaop,org.codehaus.nanning,org.codehaus.nanning.cor

-Dorg.knopflerfish.startlevel.use=true

#-Dch.ethz.jadabs.jxme.peeralias=peer1
-Dch.ethz.jadabs.jxme.tcp.port=9001
-Dch.ethz.jadabs.jxme.seedURIs=tcp://ikdesk3.inf.ethz.ch:9002

-init

## Basic KF bundles

## jxme-eventsystem
-initlevel 1
<CH.EHTZ.JADABS.AMONEM.JARSPACE>

-launch
<CH.EHTZ.JADABS.AMONEM.STARTSPACE>
```

© ETHZ-IKS • ALL RIGHTS RESERVED

2.5.2 xargs Beispiel

Beispiel fuer ein xargs File

```
#
# Generated from template.xargs
# Knopflerfish release 1.3.0a4
#

-Dorg.knopflerfish.verbosity=0

-Dorg.knopflerfish.framework.debug.packages=false
-Dorg.knopflerfish.framework.debug.errors=true
-Dorg.knopflerfish.framework.debug.classloader=false
-Dorg.knopflerfish.framework.debug.startlevel=false
-Dorg.knopflerfish.framework.debug.ldap=false

-Dorg.osgi.framework.system.packages=ch.ethz.jadabs.osgiaop,org.codehaus.nanning,org.codehaus.nanning.cor

-Dorg.knopflerfish.startlevel.use=true

#-Dch.ethz.jadabs.jxme.peeralias=peer1
-Dch.ethz.jadabs.jxme.tcp.port=9001
-Dch.ethz.jadabs.jxme.seedURIs=tcp://ikdesk3.inf.ethz.ch:9002

-init

## Basic KF bundles

## jxme-eventsystem
-initlevel 1
-install swt/jars/swt-3.0RC1-linux-osgi.jar
-install xpp/jars/xpp3-1.1.3.3_min-osgi.jar
-install xstream/jars/xstream-1.0.1-osgi.jar
-install log4j/jars/log4j-1.2.8-osgi.jar
-install jadabs/jars/concurrent-0.7.1.jar
...

-launch

-start 1
-start 2
-start 3
-start 4
-start 5
...
```

© ETHZ-IKS • ALL RIGHTS RESERVED

2.5.3 repository

Beispiel fuer ein Repository-XML

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/xsl" href="repository.xsl"?>


<bundles>
        <repository-bundle-name>OSGi-Repository</repository-bundle-name>
        <date>
                Thu Dec 23 09:53:18 UTC 2004
        </date>
        <dtd-version>1.0</dtd-version>

        <bundle>
                <bundle-name>swt</bundle-name>
                <bundle-version>3.0RC1-linux-osgi</bundle-version>
                <bundle-uuid>swt:swt:3.0RC1-linux-osgi:</bundle-uuid>
<bundle-updatelocation>file:///home/bam/.maven/repository/swt/jars/swt-3.0RC1-linux-osgi.jar</bundle-upda
        </bundle>

...

        <bundle>
                <bundle-name>remotefw-impl</bundle-name>
                <bundle-version>0.7.1</bundle-version>
                <bundle-uuid>jadabs:remotefw-impl:0.7.1:</bundle-uuid>
<bundle-updatelocation>file:///home/bam/.maven/repository/jadabs/jars/remotefw-impl-0.7.1.jar</bundle-upd
        </bundle>
        <bundle>
                <bundle-name>jadabs-maingui</bundle-name>
                <bundle-version>0.7.1</bundle-version>
                <bundle-uuid>jadabs:jadabs-maingui:0.7.1:</bundle-uuid>
<bundle-updatelocation>http://n.ethz.ch/student/scherand/download/jadabs-maingui-0.7.1.jar</bundle-update
        </bundle>

</bundles>
```

© ETHZ-IKS • ALL RIGHTS RESERVED

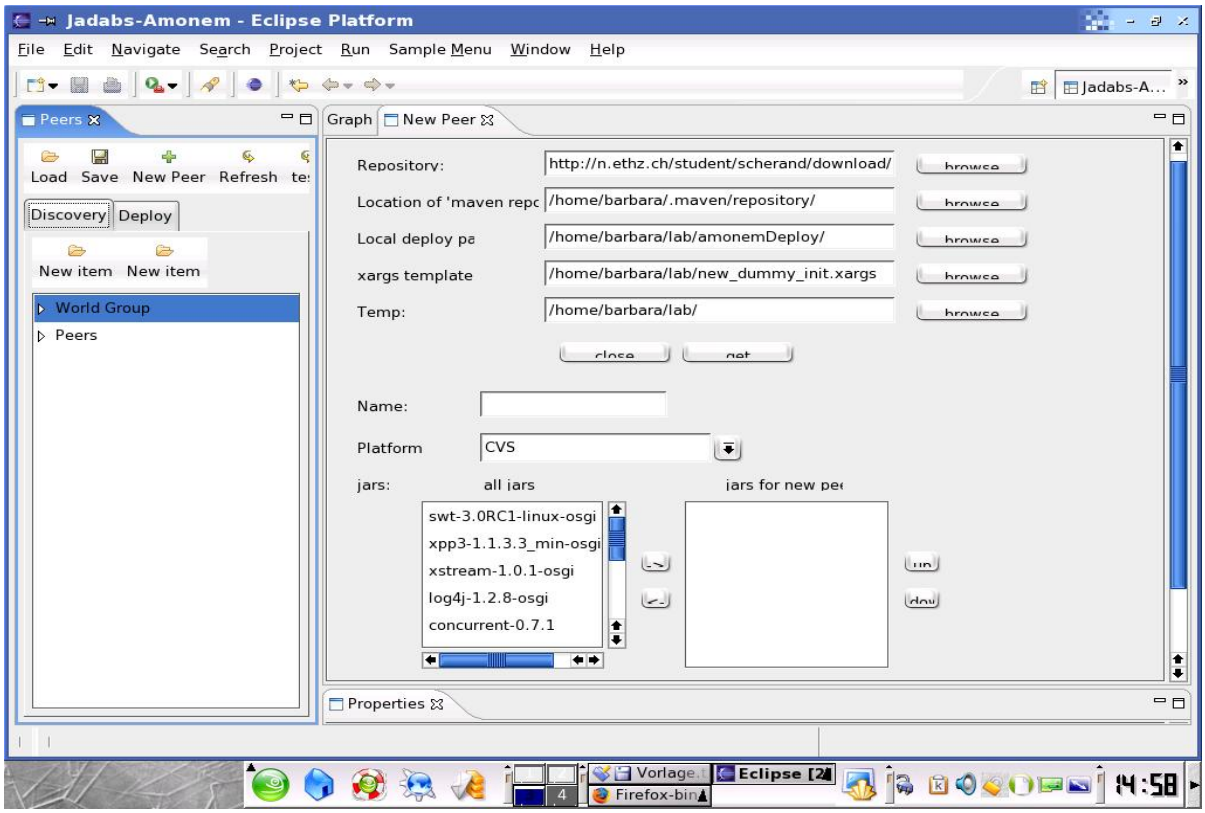
2.6 Beispiel

Beispiel

Zur Illustration eines Benutzerinputs wird hier das Erstellen eines Peers exemplarisch gezeigt. Die verwendeten Umgebungsvariablen sollten aber immer angepasst werden.

Erzeugen eines Peers

Nach dem Starten des Features werden eventuell Peers angezeigt, welche schon aktiv im System vorhanden sind. Man sollte beim neuen Peer darauf achten, dass dieser nicht einen Namen bekommt, der schon an einen anderen Peer vergeben ist.
Das Anwaehlen von "new Peer" laesst den oberen Teil des folgenden Dialogs erscheinen:



Nun kann man die Umgebungsvariablen eingeben. Diese Variablen werden dann solange beibehalten, bis sie vom Benutzer wieder geaendert werden.

- `Repository:` Hier werden die URL's oder lokalen Pfade von XML-Dateien angegeben, welche die Informationen zu den benoetigten Bundles enthalten.
- `Location of maven repo:` Lokaler Pfad, der auf das Maven-Repository verweist.

© ETHZ-IKS • ALL RIGHTS RESERVED

- `Local deploy path:` Lokaler Pfad, der auf jenen Ordner verweist, in welchem die Informationen zu einem Peer gespeichert werden sollen.
- `xargs template:` Pfadname jener Datei, in der sich das "Skelett" des xargs-Files befindet.
- `Temp:` Pfadname des temporaeren Verzeichnisses, in welches die Repository-Files nach dem Downloaden gespeichert werden. Auch die Jas der Bundles werden dort gespeichert, sofern sie nicht schon vorhanden sind.

Wenn alle diese Variablen gesetzt sind, waehlt man den `get`-Knopf. Nun werden die verschiedenen jars der durch `Repository` verfuegbaren Bundles aufgelistet.

Nun kann man den Namen des Peers bestimmen und auswaehlen mit welcher Platform er laufen soll. Aus der Liste der JARs koennen die Gewuenschten ausgesucht werden. Zu beachten ist dabei, dass die Reihenfolge in der sie dann auf der rechten Seite erscheinen auch die Reihenfolge, ist in der die Bundles gestartet werden.
Mit `ok` bestaetigt man die gemachten Definitionen, und der Peer wird daraufhin erzeugt und gestartet. Er escheint nach einigen Sekunden im Graph und in beiden Baeumen der linken Fensterseite.

© ETHZ-IKS • ALL RIGHTS RESERVED