

GEORG-AUGUST-UNIVERSITÄT  
GÖTTINGEN



Steffen Klemer moh@gmx.org  
Seminar: GRID Technologie in der Wissenschaft

Handout zum Vortrag

**gLite**

Eine Grid-Distribution der EGEE



# Inhaltsverzeichnis

<b>1 Übersicht</b>	<b>3</b>
<b>2 Geschichte</b>	<b>3</b>
2.1 Warum eine Distribution? . . . . .	3
2.2 Was sind eigentlich... Web Services? . . . . .	4
<b>3 Ein Weg durch gLite</b>	<b>6</b>
3.1 Authentifizierung . . . . .	6
3.2 Versenden von Jobs . . . . .	7
3.3 Monitoring und Abrechnung . . . . .	8
3.4 Das Datengrid . . . . .	8
<b>4 Probleme und ein Blick in die Glaskugel</b>	<b>9</b>
<b>5 Wichtige Abkürzungen</b>	<b>10</b>
<b>Literatur</b>	<b>10</b>

# 1 Übersicht

gLite ([1]) ist eine Distribution von Paketen um ein Computing Grid aufzubauen. Es wird vor allem von der europäischen Initiative EGEE (*Enabling Grid for E-science* [8]) und dem Hochenergiephysik-Zentrum CERN entwickelt. Die Ausrichtung ist sehr speziell auf ein Szenario gekoppelter Rechencluster und die Batch-Verarbeitung von Aufgaben abgestimmt.

## 2 Geschichte

Um gLite und seine Eigenheiten besser, zum Teil überhaupt, zu verstehen ist ein Blick auf die Entstehungsgeschichte notwendig.

Die Wurzeln reichen bis in das Jahr 1994, als an der Universität in Illinois erstmalig mehrere Cluster der USA zusammengeschaltet wurden, damals noch aus reiner, informationstechnischer Neugierde. Daraus entwickelte sich die staatlich geförderte *I-Way*-Initiative. In diesen ersten Jahren wurden einige wegweisende Entscheidungen getroffen. Eine davon ist die Einsicht, *Heterogenität nutzbar* machen zu wollen, anstatt eine Homogenität in der eingesetzten Hard- wie Software zu erstreben. Eine andere, das Grid<sup>1</sup> modular mit zahlreichen Zwischenschichten aufzubauen. Diese Zwischenschichten werden meist als *Middleware* bezeichnet.

Die führte zur Entwicklung des *Globus Toolkits 1.0* 1998. Es bestand bereits damals aus zahlreichen, zum Teil austauschbaren, Modulen. Nach der recht erfolgreichen Version 2.0 im Jahr 1998 erfolgte zur Jahrtausendwende ein starker Bruch durch den Schwenk auf Web Services, die mit Version 4.0 2006 zum größten Teil auf XML und HTTP basierte Schnittstellen portiert wurden.

Ebenfalls um das Jahr 2000 wurde am CERN im *LCG (LHC Computing Grid)* eine Lösung zur Verarbeitung der riesigen Datenmengen des LHC ab dem Jahr 2006 gesucht. Dies mündete im *AliEn* Framework, bestehend aus zahlreichen Perl-Modulen und frei verfügbaren Bibliotheken<sup>2</sup>. AliEn und ein Fork von Globus 2.0, *gLite 1.6*<sup>3</sup> mündeten schließlich innerhalb der europäischen *EGEE* - Initiative in *gLite 3*.

### 2.1 Warum eine Distribution?

Globus ist, wie sein Name ausdrückt, ein Toolkit. Innerhalb der streng definierten Grid-Schnittstellen stellt es eine sehr große Anzahl teilweiser redundanter Module bereit. Mit diesen können die verschiedensten Formen eines Grids erstellt werden - vom Cluster-Verbindenden bis zur *Rechenleistung aus der Steckdose* für jedermann. Dazu kommen noch viele weitere kleinere Projekte wie *Condor*<sup>4</sup>, die spezifische Module für einzelne Grid-Schichten anbieten.

Innerhalb eines Projektes, wie dem LCG, bei dem die Ressourcenzahl zwar groß, aber überschaubar ist und es vor allem auf den reibungslosen Betrieb ankommt, ist eine standardisierte Auswahl von Modulen aus diesem Fundus erstrebenswert. Genau dies bietet gLite. Es besteht aus einigen selbsterstellten Komponenten, aber zusätzlich vielen Globusbestandteilen oder anderen externen Modulen.

gLite implementiert auf den meisten Ebenen die Standards des *OGF (Open Grid Forum)*<sup>5</sup> und basiert auf einer Service Orientierten Architektur (SOA). Wie die meisten Middlewares, besteht gLite grob aus vier Teilen:

- Benutzerauthentifizierung
- Jobsubmission/ Workload Management
- Informationsservices (Abrechnung, Logging, Fehlererkennung)
- Datenhaltung und -übermittlung

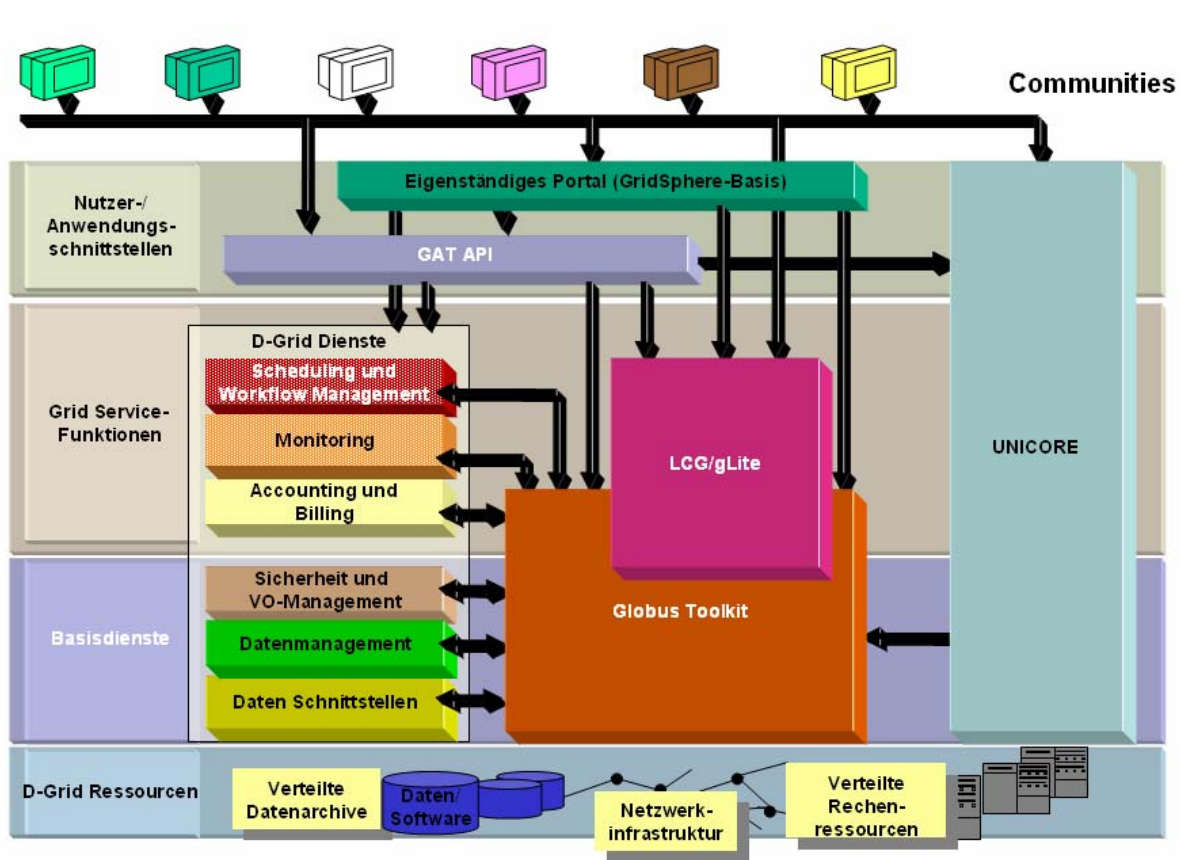


Abbildung 1: gLite im Kontext der D-Grid Initiative

Abb. 1 zeigt schematisch, an welcher *Stelle* des Grid<sup>6</sup> man sich gLite vorstellen kann. Auch der Überlapp mit Globus wird dargestellt.

## 2.2 Was sind eigentlich... Web Services?

Im Zusammenhang mit Globus wird oft von Web Services oder SOA gesprochen. Web Services sind ein vom *W3C* (*World Wide Web Consortium* - eine der Hüterinnen der Webstandards) geprägter Begriff. Es bezeichnet das Anbieten von Informationsdienstleistungen über das Internet. Meist wird dies in lokalen Programmen als *Remote Procedure Call* implementiert. Es wird also einfach eine Funktion aufgerufen, die beim entfernten Anbieter ausgewertet und ein Ergebnis zurückgegeben wird.

Das w3c hat für das Übermitteln dieser Services einige Protokolle auf XML- Basis normiert. Die ausgetauschten XML-Nachrichten werden dabei in einen *SOAP* (*Simple Object Access Protocol*)-Rahmen eingehüllt und meist via *HTTP* übertragen. Es wurden aber auch *SMTP* und *XMPP* erfolgreich getestet.

Im Folgenden findet sich ein Beispiel aus *James Snell, Doug Tidwell & Pavel Kulchenko: Programming Web Services with SOAP; Oreilly 2001*. Es demonstriert den Aufruf einer Funktion `sayHello` mit dem Parameter `name=Coyote`.

<sup>1</sup>wobei es so erst seit dem Buch von Foster and Kesselman [3] von 1998 heißt

<sup>2</sup>Teile von Globus konnten verwendet werden, da sie meist unter OpenSource Lizenzen freigegeben werden

<sup>3</sup>daher der ursprüngliche Name *Globus Lite*

<sup>4</sup>ein Scheduling-Framework der Universität Wisconsin: [6]

<sup>5</sup>Es wurden bereits erfolgreich Jobs mit dem amerikanischen *Open Science Grid* ausgetauscht[5]

<sup>6</sup>Hier innerhalb der D-Grid Initiative, von deren Website auch das Bild stammt: <http://www.d-grid.de/>

```

1  <s:Envelope
2  xmlns:s="http://schemas.xmlsoap.org/soap/envelope/"
3  xmlns:xsi="http://www.w3.org/1999/XMLSchema-
4  instance"
5  xmlns:xsd="http://www.w3.org/1999/XMLSchema">
6  <s:Body>
7      <m:sayHello xmlns:m='urn:Example1'>
8          <name xsi:type='xsd:string'>Coyote</name>
9      </m:sayHello>
10 </s:Body>
11 </s:Envelope>
12

```

```

1  <s:Envelope
2  xmlns:s="http://www.w3.org/2001/06/soap-envelope"
3  xmlns:xsi="http://www.w3.org/1999/XMLSchema-
4  instance"
5  xmlns:xsd="http://www.w3.org/1999/XMLSchema">
6  <s:Body>
7      <n:sayHelloResponse xmlns:n="urn:Example1">
8          <return xsi:type="xsd:string">
9              Hello Coyote
10          </return>
11      </n:sayHelloResponse>
12 </s:Body>
13 </s:Envelope>
14

```

Als Rückgabewert wird ein String *Hello Coyote* versandt. Innerhalb des SOAP-Namespaces, der hier das Prefix *s* bekommen hat, wurde ein extra erstelltes XML-Schema *Example1*, markiert durch ein vorangestelltes *n*, verwendet.

Aktuelle Beispiele für *Grid-fremde* Web Services sind:

- Google Search API
- Flickr API
- Reuters Finanzdaten

### 3 Ein Weg durch gLite

Nun werden wir aus der Sicht eines Benutzers, der einen Arbeitsauftrag berechnen lassen möchte, durch das Grid, wie es gLite darstellt, wandern. So ein Arbeitsauftrag wird auch Job genannt und genau darum geht es (im Moment noch) bei gLite - das Ausführen von in sich abgeschlossenen Jobs. Die Abstrahierung besteht zur Zeit im Wesentlichen daraus, dass der Wissenschaftler am Rechner nicht mehr darüber nachdenken müsste *wo* sein Programm berechnet wird und *wo* seine Daten liegen. Im Hintergrund arbeiten weiterhin nur große Rechencluster, die dadurch eine viel höhere Auslastung erhalten sollen.

Eine stark vereinfachte Sicht auf das Grid ist in Abb. 2 zu finden.

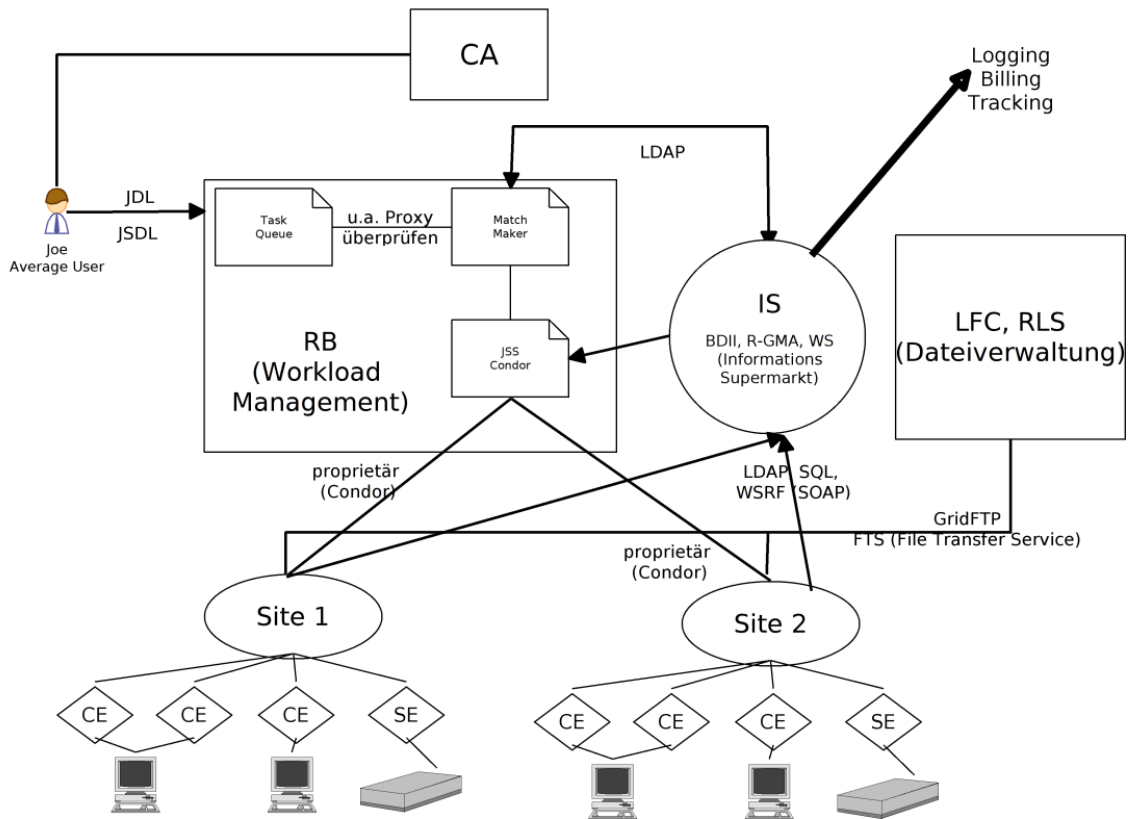


Abbildung 2: Schema des Grid in der gLite-Interpretation

#### 3.1 Authentifizierung

Entlang des Weges muss sich der Job an zahlreichen Stellen *ausweisen*. Das erste Mal geschieht dies beim Absenden an einen der zentralen *Ressource Broker* (RB). Später beispielsweise kurz vor der Ausführung oder beim Zugriff auf Daten im Grid. Hierfür wurde die erprobte *Public-Key Infrastruktur* (PKI) nach dem Verzeichnisstandard *X.509* übernommen. Die eigentliche Implementierung ist die *OpenSSL*. Hierbei hat jeder Nutzer einen signierten Schlüssel, mit dem er sich ein rund 12h gültiges *Ticket*<sup>7</sup> erstellen kann. Für länger laufende Jobs, gibt es Lösungen in Form eine Daemons wie *myproxy*[9].

Heruntergebrochen auf die notwendige Nutzerinteraktion sieht es so aus, dass dieser 2 Dateien in seinem Homeverzeichnis hat, `.globus/usercert.pem` und `.globus/userkey.pem` und zum Erstellen des Tickets folgendes aufruft:

<sup>7</sup>im Grid Umfeld meist *Proxy* genannt

```
~> voms-proxy-init -voms acme
```

Hierbei ist *acme* die Virtuelle Organisation, für die er arbeitet.

### 3.2 Versenden von Jobs

Innerhalb des Grid gibt es eine zentrale Instanz, die Jobs entgegennimmt und anschließend auf die Ressourcen verteilt. Passenderweise nennt sich diese *Ressource Broker* (RB). Er bekommt die Jobs und zugehörige Informationen in neuen Versionen von gLite via Web Services (mit Apache als Backend), also SOAP+WSDL (Web Services Description Language), übergeben. Bis dahin wurde ein proprietäres Protokoll verwendet.

Damit der Job im Sinne des Anwenders ausgeführt wird und der Scheduler eine gut passende Ressource findet, muss eine JDL-Datei (*Job Description Language*) erstellt werden. Diese sieht zum Beispiel folgendermaßen aus:

```

1  VirtualOrganisation = "acme";
2  Executable = "mandelbrot.sh";
3  Comment = "My first submitted job";
4
5  StdOutput = "mbrot.out";
6  StdError = "mbrot.err";
7
8  InputSandbox = {"mandelbrot.sh","inputdata.txt","worker.py"};
9  OutputSandbox = {"mbrot.out", "mbrot.err"};
10
11 RetryCount = 2;
12 ShallowRetryCount = 10;
13
14 Requirements = Member( "VO-acme-production-0.0.1",
15                        other.GlueHostApplicationSoftwareRunTimeEnvironment)
16                        && other.GlueCEPolicyMaxCPUTime > 1000
17                        && other.GlueCEStateWaitingJobs < 20 ;

```

Mit **Sandbox** ist die abgeschottete Umgebung, in der der Prozess auf dem Ziel dann laufen wird, gemeint. In diese können kleine Dateien mitgenommen und direkt nach der Ausführung zurück transportiert werden.

Das entscheidende Feld ist **Requirements**. Hier können zum einen Fakten wie die minimal nötige CPU-Zeit festgelegt, aber auch Dinge wie benötigte Rechnerarchitektur, spezielle Bibliotheken u.ä. gefordert werden. Es wird dann sichergestellt, dass der Job auf einer Ressource landet, die dies erfüllt. Hiermit sind weiterhin Optimierungen auf bestimmte Prozessorarchitekturen, aber auch die Parallelisierbarkeit mittels der Standardbibliotheken *libPVM* oder *libMPI* möglich. Diese Anforderungen können aber auch als gerichtete Graphen über andere Jobs formuliert werden, um komplexe, parallele Abhängigkeiten zu modellieren.

Abgeschickt wird der Job dann einfach mit

```
~> glite-wms-job-submit -a mandelbrot.jdl
```

oder einem grafischen Tool wie **Ganga**[4]. Der Nutzer erhält daraufhin eine Job ID, mit der er den Status abfragen, den Job killen, resubmiten u.ä. kann.

Nun gelangt der Job innerhalb des RB in eine Warteschlange und schließlich in den Scheduler **Condor**. Dieser fragt im *Information Supermarket* (IS), einer weiteren zentralen Instanz, nach Informationen über die vorhandenen Ressourcen. Daraufhin wird der Job dann an die passendste Ressource, also eines der angeschlossenen Rechenzentren geleitet. Innerhalb eines Rechenzentrums hat man verschiedene *Computing Elements* (CE). Diese beziehen sich auf verschiedene Hardware, aber auch auf identische mit unterschiedlichen Policies, wie der Rechenzeit, bis ein Job gekillt wird. Außerdem können *Storage Elements* (SE) dazugehören.

### 3.3 Monitoring und Abrechnung

Wie bereits erwähnt, gibt es einen zentralen Informationsservice. Dieser bekommt seine Daten von allen angeschlossenen Ressourcen geschickt. Die Übermittlung und Abfrage erfolgt über eines von drei Protokollen, *BDII* (LDAP-ähnlich), *R-GMA* (SQL-ähnlich, aus dem Globus-Projekt), *SOAP* (recht neu). Die Beschreibung ist im *GLUE* Standard[7] festgeschrieben und erfolgt objektorientiert. Der IS ist auch für das Logging, in Zukunft das Abrechnen und die Fehlererkennung zuständig.

### 3.4 Das Datengrid

Der Job im Beispiel sieht so aus:

```

_____ mandelbrot.sh _____
1  export LFC_HOST='lcg-infosites --vo acme lfc'
2  python worker.py inputdata.txt
3  lcg-cr --vo acme -v -n 16 file://$PWD/result.txt \
4      -l lfn:/grid/acme/users/rrunner/'date +%Y-%m-%d' /result.txt

```

Es wird also mittels `python` ein Programm ausgeführt, wobei ihm zusätzlich die vorher mit übermittelten Daten übergeben werden. Das Programm soll eine Datei mit dem Namen `result.txt` erstellen.

Diese wird daraufhin via GridFTP aus dem aktuellen Verzeichnis in das Datengrid kopiert und in der hierarchischen Dateistruktur abgelegt. Auffällig ist das `lfn:`, welches für *Logical Filename* steht. Jede Datei kann beliebig viele solcher *logischer* Dateinamen haben. Ein zentraler Server, der *Logical File Catalog* (LFC), mappt diese auf eine *Global Unique ID* (GUID). Mit dieser GUID wiederum sind eine oder mehrere *Site URLs*, also physikalische Adressen verbunden.

Damit die Datenintegrität sichergestellt ist, können Dateien nur erstellt und gelöscht, nicht aber editiert werden. Die Datenübertragung zwischen den Storage Elementen beziehungsweise zwischen SE und dem Benutzer erfolgt über GridFTP, eine für Breitbandverbindungen optimierte Version des FTP.

## 4 Probleme und ein Blick in die Glaskugel

Aus der Entstehungsgeschichte heraus ist klar, dass gLite auf ein System optimiert wurde. Schließlich konnte die überschaubare Rechnerzahl damit installiert werden. Dies erwies sich beim Übergang vom alten (Scientific Linux 3.0) auf ein neues System (Scientific Linux 4.0) als sehr problematisch. Es dauerte über ein Jahr, bis alle Module lauffähig waren. Aufgrund von bekannten Sicherheitslücken in SL 3.0 wurden im LHC-Grid zum Teil Dienste gesperrt. Der Teil, der auf den Computing Elements läuft, ist aber durchaus portierbar, wie die Website <http://cagraidsvr06.cs.tcd.ie/porting/> belegt.

Auffällig ist weiterhin, dass selbst mit größeren Anstrengungen gLite wohl nicht auf nicht-Unix Systeme portierbar ist.

Für die Zukunft ist, wie vom Open Grid Forum vorgegeben, auch bei gLite ein Übergang zu Web Services (über SOAP) auf allen wichtigen Zwischenschichten geplant. Die Module hierfür tragen alle das Wort *Cream* im Namen (Siehe auch Abb. 3).

### → LCG-CE (GT2 GRAM)

- Not ported to GT4. To be dismissed

### → gLite-CE (Condor-C+GSI)

- Deployed (GT2 version) but still needs tuning

### → CREAM (WS-I)

- Prototype. OGF-BES (see demo at SC'06)

- Possible developments:
  - GT4 → BLAH submissions?

Choose your preferred path to the Batch System!

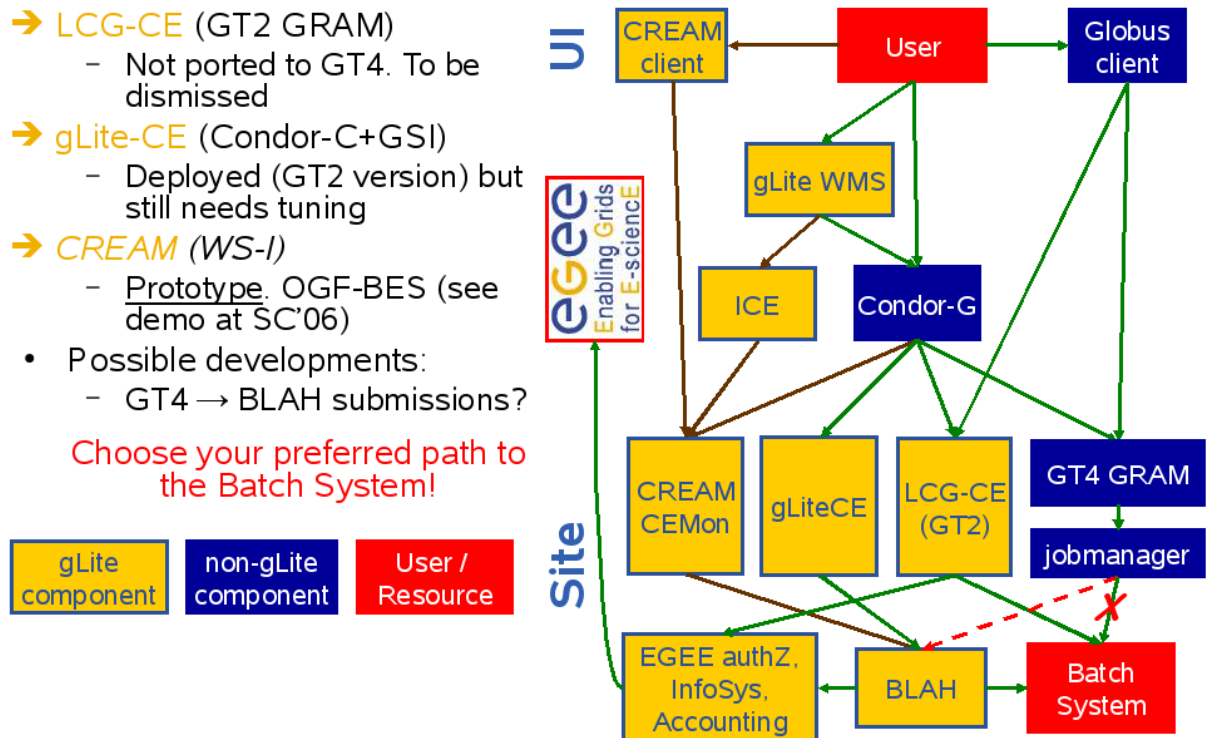


Abbildung 3: Aufbau von gLite; Abb. vom gLite Projekt.

## 5 Wichtige Abkürzungen

- *CERN* - Conseil Européen pour la Recherche Nucléaire; Europäische Organisation für Kernforschung
- *EGEE* - Enabling Grid for E-science
- *GLUE-Schema* - Grid Laboratory Uniform Environment Schema - Objektorientierte Beschreibung von Grid Ressourcen
- *GridFTP* - FTP basiertes Protokoll zur Übertragung von großen Datenmengen
- *GUID* - Global Unique ID - automatisch generierter Identifikationsstring, der aufgrund der Generierung einzigartig sein sollte - vgl. [10]
- *JDL* - Job Description Language - Sprache um Jobs und ihre Abhängigkeiten zu beschreiben
- *LCG* - LHC Computing Grid - Grid-Arbeitsgruppe des CERN
- *OGF* - Open Grid Forum - Standardisierungsgremium
- *SOA* - Service Orientierte Architektur - In diesem Zusammenhang das Anbieten von Dienstleistungen über das Internet mittels genormter Schnittstellen
- *SOAP* - Simple Object Access Protocol - XML basiertes Netzwerkprotokoll für die Modellierung von Remote Procedure Calls und den Datenaustausch
- *W3C* - World Wide Web Consortium - Standardisierungsgremium (z.B. HTML)
- *WSDL* - Web Services Description Language - Beschreibungssprache für WS auf XML Basis
- *X.509* - ein ITU-T-Standard für eine Public-Key-Infrastruktur; im Web meist analog zum RFC 3280 umgesetzt
- *XML* - Extensible Markup Language - Auszeichnungssprache des W3C für hierarchisch strukturierte Daten

## Literatur

- [1] Website des gLite-Projektes: <http://glite.web.cern.ch/glite/>; 2007-11-25
- [2] Website des Globus-Projektes: <http://www.globus.org/>; 2007-11-25
- [3] Ian Foster and Carl Kesselman: "The Grid: Blueprint for a new computing infrastructure", Morgan Kaufmann Publishers 1998
- [4] GUI für die Jobverwaltung: <http://ganga.web.cern.ch/ganga/>; 2007-11-25
- [5] Interopabilität zwischen gLite und dem OSG: <http://osg-docdb.opensciencegrid.org/cgi-bin/ListBy?topicid=20>; 2007-11-25
- [6] Beliebter Batch-Scheduler: <http://www.cs.wisc.edu/condor/>; 2007-11-25
- [7] Der GLUE-Standard: <http://forge.ogf.org/sf/projects/glue-wg>; 2007-11-25
- [8] Die EGEE im Internet: <http://public.eu-egee.org/>; 2007-11-25
- [9] MyProxy: <http://grid.ncsa.uiuc.edu/myproxy/>; 2007-11-25
- [10] Beispiele für GUID Generatoren: [http://en.wikipedia.org/wiki/Globally\\_Unique\\_Identifier](http://en.wikipedia.org/wiki/Globally_Unique_Identifier); 2007-12-14