



GAT Übersicht

Alexander Beck-Ratzka

DGI Workshop GAT, AEI Potsdam, 19.01.06



GAT Übersicht: Agenda



- Wieso GAT?
- Objekt- / Interfacemodell von GAT
- File Management
- Job Management
- Adaptoren



- Wieso GAT?
- Objekt- / Interfacemodell von GAT
- File Management
- Job Management
- Adaptoren



Wieso GAT?



GAT als einheitliche API für Zugriff auf heterogene Gridtechnologien / Gridmiddleware.

GAT ist nur ein Framework; die eigentlichen Operationen müssen durch Aaptoren erledigt werden. GAT bietet die Möglichkeit des Adaptoren-Einsatzes.

Neue Grid-Technologien müssen nur via Adaptor mit ans GAT gekoppelt werden -> Keine Änderungen mehr im Programm-Code nötig, auch nicht bei neuer Grid-Technologie.



GAT berücksichtigt...



Datei-Operation im Grid: sowohl copy / delete, als auch lesen und schreiben in Dateien.

Advertisable Management: überwachen, ob etwas bestimmtes passiert (Abfrage verfügbarer Files) -> Möglichkeit der Prozesskommunikation.

Resource Management: Setzen von Ressourcen. Welche Ressourcen sind für den Job wo verfügbar.

Job Management: submitten von Batchjob, Statusabfrage, beenden von Batchjobs.



- Wieso GAT?
- Objekt- / Interfacemodell von GAT
- File Management
- Resource Management
- Job Management
- Adaptoren



Objektmodell von GAT



GAT ist objektorientiert -> besondere Anforderungen an C-GAT^{*1)} .

In C-GAT gibt es für „Klassen“ (eigentlich sind es structs) gesonderte create und destroy-Funktionen. Übernehmen Aufgabe des Konstruktors und des Destruktors.

*1) Es gibt auch JAVA-GAT, einen C++-Wrapper, Python-GAT ist in Entwicklung



Objektfunktionen C-GAT



GATObject_Create(...) -> unterschiedliche Argumente möglich

GATObject_Destroy(GATObject *object)

void GATObject_GetType(GATObject_const object)

GATResult GATObject_Equals(GATObject_const 1st,
GATObject_const 2nd,
GATBool *iseq)

GATResult GATObject_Clone(GATObject_const object,
GATObject *result)

GATResult GATObject_GetInterface(GATObject_const object,
GATInterface iftype,
void const **ifp)



Beispiel GATTime



```
GATTime_Create(int timevalue)
```

```
GATTime_Destroy(GATTime *Time)
```

```
void GATTime_GetType(GATTime_const Time)
```

```
GATResult GATTime_Equals(GATTime_const 1st,  
                          GATTime_const 2nd,  
                          GATBool *iseq)
```

```
GATResult GATTime_Clone(GATTime_const Time, GATTime *result)
```

```
GATResult GATTime_GetInterface(GATTime_const Time,  
                               GATInterface iftype,  
                               void const **ifp)
```



Konvertierungsfunktionen



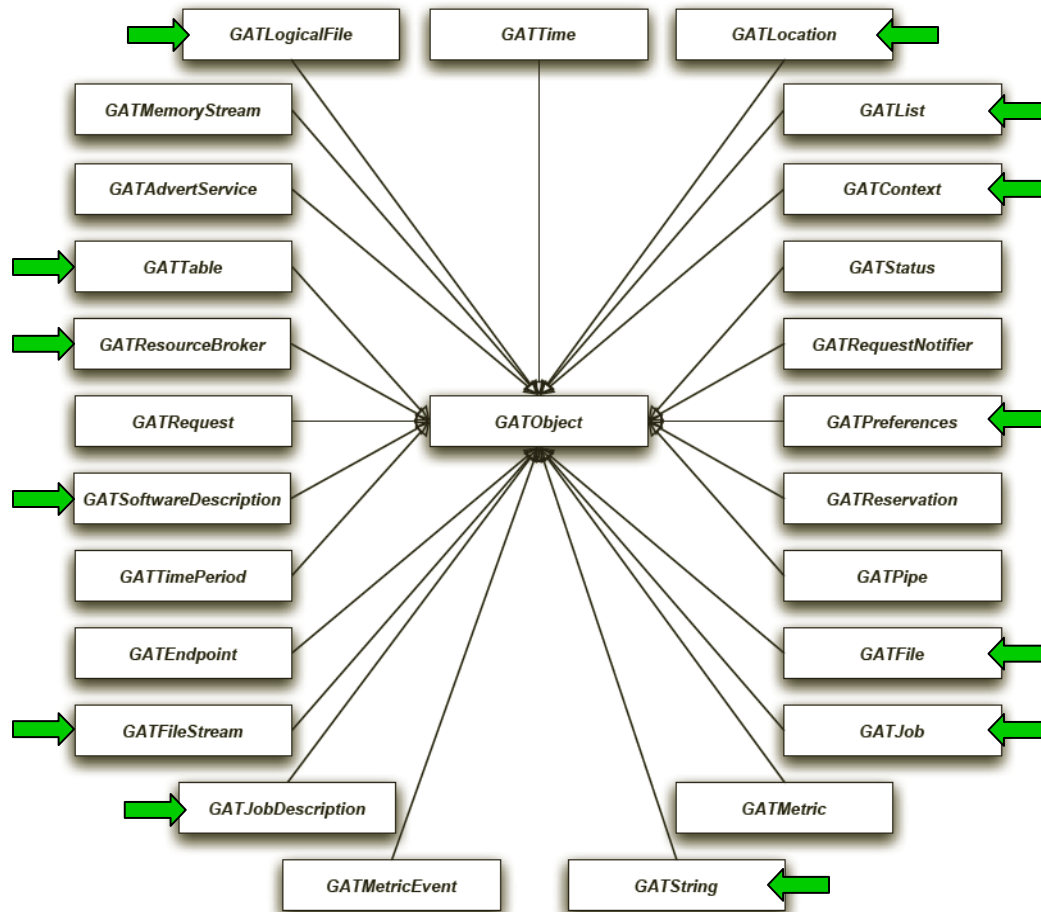
GATTime GATObject_ToGATTime(GATObject object)

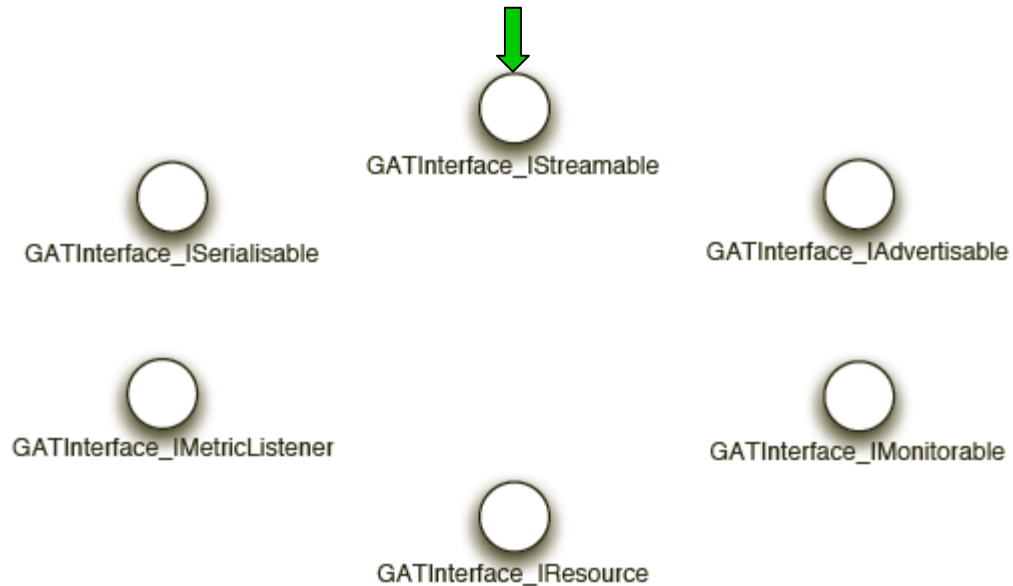
GATObject GATTime_ToGATObject(GATTime time)

GATTime_const GATObject_ToGATTime_const
(GATObject_const object)

GATObject_const GATTime_ToGATObject_const
(GATTime_const time)

Von GATObject abgeleitete Klassen





Jedes GAT-Interface besteht aus einer **struct** mit Funktionspointern. Diese Funktionen führen die Interface-Aktionen aus.



- Wieso GAT?
- Objekt- / Interfacemodell von GAT
- File Management
- Job Management
- Adaptoren



File-Management



GATFile

GATFileStream

GATLogicalFile



Die GATFile-Klasse (1)



Benötigt zur Erzeugung GATLocation und GATContext

```
GATLocation GATLocation_Create(const char *URI*1))
```

```
void GATLocation_Destroy(GATLocation *location)
```

```
GATContext GATContext_Create(void)
```

```
void GATContext_Destroy(GATContext *context)
```

```
GATFile GATFile_Create( GATContext context,  
                        GATLocation_const location,  
                        GATPreferences_const preferences*2))
```

```
void GATFile_Destroy(GATFile *file)
```

*1) z.B. <file:///home/gatuser/gatfile> *2) Wichtig für Adaptern



Die GATFile-Klasse (2)



Funktionen für **kopieren**, **löschen**, **verschieben**...

```
GATResult GATFile_Copy(GATFile_const file,  
                        GATLocation_const targetLocation, GATFileMode mode)
```

GATFileMode: FailIfExists -> Kein Überschreiben existenter Dateien.
GATFileMode: Overwrite -> In jedem Fall überschreiben.



Die GATFile-Klasse (3)



Funktionen für kopieren, löschen, verschieben...

```
GATResult GATFile_Move(GATFile_const file,  
                        GATLocation_const targetLocation,  
                        GATFileMode mode)
```

```
GATResult GATFile_Delete(GATFile_const file)
```



Die GATFile-Klasse (4)



Funktionen zur Überprüfung von File-Instanzen

GATResult GATFile_IsReadable(GATFile_const file)

GATResult GATFile_IsWritable(GATFile_const file)

GATResult GATFile_GetLength(GATFile_const file, unsigned long *length)

GATResult GATFile_LastWriteTime(GATFile_const file, GATTime *lw_time)

GATLocation_const GATFile_GetLocation(GATFile_const file)



Filestream-Management (1)



Operationen auf geöffneten Dateien -> lesen und schreiben

GATFileStream GATFileStream_Create (GATContext context,
GATPreferences_const preferences,
GATLocation location,
GATFileStreamMode mode)

mode:

GATFileStreamMode_Read -> Lesen von Anfang
GATFileStreamMode_Write -> Schreiben von Anfang
GATFileStreamMode_Append -> Schreiben an Ende
GATFileStreamMode_ReadWrite -> Lesen und Schreiben von Anfang

GATFileStream GATFileStream_Destroy (GATFileStream file)



Filestream-Management (2)



Lesen und schreiben über Interface **GATInterface_Istreamable**

Lesen:

```
GATResult GATStreamable_Read(GATObject object,  
                               void *buffer,  
                               GATuint32 size,  
                               GATuint32 *read_bytes)
```

Achtung zuvor: object = GATFileStream_ToGATObject(fileStream);

Schreiben:

```
GATResult GATStreamable_Write(GATObject object,  
                               void *buffer,  
                               GATuint32 size,  
                               GATuint32 *written_bytes)
```



Filestream-Management (3)



Positionierung

```
GATResult GATStreamable_Seek(GATObject object,  
                                GATOrigin origin,  
                                GATuint32 offset,  
                                GATuint32 *new_position)
```

origin:

GATOrigin_Set -> Positionierung relativ zu Beginn.
GATOrigin_Current -> Positionierung relativ zur aktuellen Position.
GATOrigin_End -> Positionierung relativ zum Ende.



LogicalFile-Paket (1)



Liste identischer Dateien, die an unterschiedlichen Stellen im Grid liegen können

Prinzipielle Möglichkeit zur Ermittlung der Datei, auf die man am schnellsten zugreifen kann

GATLogicalFile GATLogicalFile_Create(GATContext context,
GATLocation location,
GATLogicalFileMode mode,
GATPreferences_const preferences)

mode:

GATLogicalFileMode_Open -> Öffne nur vorhandenen GATLogicalFile

GATLogicalFileMode_Create -> Erzeuge nicht vorh. GATLogicalFile

GATLogicalFileMode_Truncate -> Öffne in jedem Fall

void GATLogicalFile_Destroy(GATLogicalFile *logicalFile)



LogicalFile-Paket (2)



Operationen des LogicalFile-Pakets

GATResult GATLogicalFile_AddFile(GATLogicalFile logicalFile,
GATFile_const file)

GATResult GATLogicalFile_RemoveFile(GATLogicalFile logfile,
GATFile_const file)

GATResult GATLogicalFile_Replicate(GATLogicalFile_const logfile,
GATLocation_const target)

GATResult GATLogicalFile_GetFiles(GATLogicalFile_const logfile,
GATList_GATFile *files)



- Wieso GAT?
- Objekt- / Interfacemodell von GAT
- File Management
- **Job Management**
- Adaptoren



Job Management (1)



GATSoftwareDescription

GATSoftwareDescription_Create (GATTable_const attributes)

attributes

Name	Type	Description
location	GATLocation	Software location
arguments	List<String>	Aufruf-Argumente
environment	GATTable	Software Env.
stdin ^{**})	GATFile	Stdin für Job
stdout ^{**})	GATFile	Stdout für Job
stderr ^{**})	GATFile	Stderr für Job
pre-staged files	List<GATFile>	Muss vor Ausführung da sein.
post-staged files	List<GATFile>	Sollte nach Ausführung zurück geliefert werden.

void GATSoftwareDescription_Destroy (GATSoftwareDescription *resource)

^{**}) Derzeit **nicht** zur Laufzeit



Job Management (2)



JobDescription Instanz: GATJobDescription

srd=GATResourceBroker GATResourceBroker_Create (GATContext context,
GATPreferences_const preferences,
GATString vo_name)

**hrd=GATHardwareResourceDescription
GATHardwareResourceDescription_Create** (GATTable requirements)

**GATJobDescription
GATJobDescription_Create_Description** (GATContext context, **srd**,
GATHardwareResourceDescription_ToGATResourceDescription(**hrd**))

void GATJobDescription_Destroy (GATJobDescription *resource)



Job Management (3)



Job Instanz: GATJob

GATResult

GATResourceBroker_SubmitJob (GATResourceBroker broker,
GATJobDescription_const description,
GATJob *job)

void GATJob_Destroy (GATJob *resource)

GATResult GATJob_GetState (GATJob_const object, GATJobState *state)

GATJobState Wert	Bedeutung
GATJobState Unknown	Status nicht ermittelbar
GATJobState Initial	Im Initialisierungsstatus
GATJobState Scheduled	z.B. Queued
GATJobState Running	Job im executing
GATJobState Stopped	Job fertig oder gestoppt



Job Management (4)



Weitere Funktionen...

~~GATResult GATJob_GetJobID (GATJob_const object, GATJobID_const *jobid)~~

GATResult GATJob_GetNativeID (GATJob_const object, GATJobID_const *jobid)

GATResult GATJob_UnSchedule (GATJob_const object)

GATResult GATJob_Stop (GATJob_const object)

GATResult GATJob_Checkpoint (GATJob_const object)

GATResult GATJob_Migrate (GATJob_const object,
GATHardwareResource_const hr,
GATJob *migratedJob)

GATResult GATJob_Clone (GATJob_const object,
GATHardwareResource_const hr,
GATJob *cloned_job)



- Wieso GAT?
- Objekt- / Interfacemodell von GAT
- File Management
- Job Management
- **Adaptoren**



Adaptoren (1)



Adaptoren werden eingebunden über CPI-Instanzen

Beispiel:

GATResourceBroker_SubmitJob -> GATResourceBrokerCPI_SubmitJob

Eine CPI-Instanz besteht aus einem Satz Funktionszeiger. Diese Funktionen führen die adaptorspezifischen Aktionen aus.



Adaptoren (2)



Prinzipiell mögliche Adaptoren

GATAdvertService
GATEndpoint
GATFile
GATFileStream
GATJob
GATLogicalFile
GATMonitorable
GATPipe
GATRequest
GATReservation
GATResourceBroker
GATResource
GATServiceResourceBroker



PBS-Adaptor (1)



Einzeladaptoren für PBS-Adaptor

GATResourceBroker
GATSelf
GATResource
GATJob

Mögliche Funktionen für PBS-GATResourceBroker-Adaptor

pbs_resourcebroker_adaptor_GATResourceBrokerCPI_CreateInstance -> create_instance
pbs_resourcebroker_adaptor_GATResourceBrokerCPI_DestroyInstance -> destroy_instance
pbs_resourcebroker_adaptor_GATResourceBrokerCPI_EqualsInstance -> equals_instance
pbs_resourcebroker_adaptor_GATResourceBrokerCPI_FindResources -> find_resources
pbs_resourcebroker_adaptor_GATResourceBrokerCPI_SubmitJob -> submit_job



PBS-Adaptor (2)



Funktionen für PBS-GATJob-Adaptor

pbs_resourcebroker_adaptor_GATJobCPI_CreateInstance	-> create_instance
pbs_resourcebroker_adaptor_GATJobCPI_DestroyInstance	-> destroy_instance
pbs_resourcebroker_adaptor_GATJobCPI_Stop	-> stopjob
pbs_resourcebroker_adaptor_GATJobCPI_GetJobDescription	-> get_jobdescription
pbs_resourcebroker_adaptor_GATJobCPI_GetInfo	-> get_jobinfo
pbs_resourcebroker_adaptor_GATJobCPI_GetJobID	-> get_jobid
pbs_resourcebroker_adaptor_GATJobCPI_GetState	-> get_state



PBS-Adaptor (3)



Ein PBS-Adaptor-Durchlauf (submit, Statusabfrage, stoppen)

GATResourceBroker_Create -> GATResourceBrokerCPI_CreateInstance

↳ liefert GATResourceBroker Broker

Erzeugung einer GATSoftwareDescription_const: PBS_SoftDescr

Erzeugung einer GATHardwareDescription_const: PBS_Hrd

Erzeugung GATJobdescription:

->

PBS_JobDescr=GATJobDescription_Create_Description(context,PBS_SoftDescr,PBS_Hrd)

↳ GATJobDescription PBS_JobDescr

Job Submit:

GATResourceBroker_SubmitJob (Broker,PBS_JobDescr,&PBSJob)

->**GATResourceBrokerCPI_SubmitJob**

↳ GAT_SUCCESS oder Fehler zurueck

↳ Pointer auf GATJob (&PBSJob)



PBS-Adaptor (4)



Ein PBS-Adaptor-Durchlauf (Statusabfrage, stoppen)

Abfrage Job-Status:

GATJob_GetState (PBSJob,GATJobState &PBSState) -> **GATJobCPI_GetState**

↳ GATJobState-Zeiger (&PBSState)

Loeschen eines Jobs:

GATJob_Stop (PBSJob) -> **GATJobCPI_Stop**

↳ GAT_SUCCESS oder Fehler zurueck



PBS-Adaptor (5)



GATTable für GATSoftwareDescription (erweiterbar)

"location"
"arguments"
"machine.queue"
"memory.size"
"cpu.walltime"
"cpu.nodes"
"file.size"
"file.yeo"
"file.Jobname"

Notwendige Aufrufe zur Belegung der Attribute-Gat-Tabelle fuer eine Softwaredescription:

- 1) **GATTable attributes** = **GATTable_Create ();**
- 2) **GATLocation location** = **GATLocation_Create (Executable);**
Executable kann auch URI sein!!
- 3) **GATTable_Add_GATObject (attributes, "location",**
GATLocation_ToGATObject_const (location));

-> GATLocation muss in ein GATObject konvertiert werden!

4) Wenn Argumente fuer das Executable beim Aufruf dabei, dann folgende Sequenz erforderlich:

```
if (args)
{
    arguments = GATList_String_Create ();

    for (ii = 0; ii < nargs; ii++)
    {
        GATList_String_Insert (arguments,
                               GATList_String_End (arguments),
                               args[ii]);
    }
    GATTable_Add_GATObject (attributes, "arguments",
                           GATList_String_ToGATObject_const
                               (arguments));
}
```



PBS-Adaptor (8)



5) Der Rest wird als String hinzugefügt:

```
GATTable_Add_String (attributes, "machine.queue", JobArgs->Queue);  
GATTable_Add_String (attributes, "memory.size", JobArgs->Memsize);  
GATTable_Add_String (attributes, "cpu.walltime", JobArgs->Walltime);  
GATTable_Add_String (attributes, "cpu.nodes", JobArgs->Nodes);  
GATTable_Add_String (attributes, "file.size", JobArgs->Filesize);  
GATTable_Add_String (attributes, "file.yeo", JobArgs->YankeE0);  
GATTable_Add_String (attributes, "file.Jobname", JobArgs->Jobname);
```



PBS-Adaptor (8)



Umsetzung in qsub-Skript

machine.queue ↪ -q Queue
cpu.walltime, file.size, memory.size, cpu.nodes, machine.queue
↪ -lwalltime=[..],file=[..],mem=[..],nodes=[..]:[..]

file.yeo ↪ -j yankeo
file.JobName ↪ -N jobname
location + args ↪ executable arg1 arg2 agr3 etc...



PBS-Adaptor (9)



Beispiel

```
machine.queue=dque  
memory.size=1024M  
cpu.walltime=00:00:10  
cpu.nodes=2  
file.size=1G  
file.yeo=eo  
file.Jobname=jname
```

```
executable=/home/alibeck/test-qsub/test.job  
args = 1, 2, 3, 4
```

wird:

```
#PBS -q dque  
#PBS -lwalltime=00:00:10,file=1G,mem=1024M,nodes=2:debug  
#PBS -V  
#PBS -N jname  
#PBS -j eo
```

```
/home/alibeck/test-qsub/test.job 1 2 3 4
```



PBS-Adaptor (10)



Aufruf PBS_JobControl

- e <executable^{*4)}> The executable to be transmitted with PBS.
- q <queue> The queue where the executable shall be started in.
- N <jobname> The jobname; default is the login user
- n <nodes> Nodes (Default is 1)?
- m <memsize> Necessary memory for the job.
- t <walltime> The time needed for the job. Default is 00:01:00
- f <filesize> The maximum filesize.
- j <YankeE0> How to yank stderr and stdout. Default is eo

*4) <executable> = <"exec arg1 arg2 arg3">



Links für GAT



GAT allgemein: <http://www.gridlab.org/WorkPackages/wp-1/>

GAT CVS: cvs.gridlab.org

cvsroot: pserver:readonly@cvs.gridlab.org:/cvs/gridlab

Passwort: anon

GAT-Quellen: wp-1/Codes

GAT-Dokumente: wp-1/Documents

Nur GATEngine: wp-1/Codes/GATEngine

Download tarball:

<http://www.gridlab.org/WorkPackages/wp-1/gat/releases.html>

<http://www.gridlab.org/WorkPackages/wp-1/adaptor/releases.html>

GAT Mailing-Liste: GAT@d-grid.de