Flexible Streaming Infrastructure for UNICORE

Krzysztof Benedyczak[1], Aleksander Nowiński[2], and Piotr Bała[1,2]

[1] Faculty of Mathematics and Computer Science
Nicolaus Copernicus University
Chopina 12/18, 87-100 Toruń, Poland
[2] Interdisciplinary Center for Mathematical
and Computational Modelling
Warsaw University
Pawińskiego 5a, 02-106 Warsaw, Poland

Abstract. We present recent innovation in a field of advanced, multi-purpose streaming solutions for the grid. The described solution is based on the Unigrids Streaming Framework [7] which has been adopted to the UNICORE 6 middleware and extended. The main focus of this paper is the UGSF Data Flow Editor, which is an universal tool for powerful streaming composition. It has been developed to provide users with a graphical interface for streaming applications on the grid.

1 Introduction

Data streaming is one of the most advanced services available in the grid. The data streaming, as well as instrument and application steering, gets significant attention of the grid users and middleware developers. Unfortunately, there is still lack of good and stable solutions ready for wide deployment.

The early works on the data streaming in the UNICORE [1] were focused on solutions dedicated to the particular applications. Among others, specialized approaches based on the SSH and pre-OGSA version of UNICORE have been developed [2]. Currently SSH tunnels are used in COVS framework [3] to perfom streaming of visualisation data. However those developments can not be seen as universal streaming framework, as are limited to concrete applications.

The new version of OGSA [4] and therefore service oriented version of the UNICORE opened the possibility to develop much better solutions. As a result the UniGrids Streaming Framework (UGSF) has been made available. The UGSF is a middleware, which serves as an engine for the new generation of UNICORE streaming services. It contains also a library for the client development.

The UGSF established a new quality in comparison to the solutions developed for pre-web services versions of UNICORE. The large part of the UGSF - the *UGSF core* is not directly operated by the end-users. It contains a number of stream implementations which can be deployed and used without any additional effort. However, from the user's point of view, there were still few problems in the UGSF. The main one is lack of user-friendly, graphical tools to access the streaming services. The development of such tools was limited either to the programming from scratch[1] or to use GPE GridBeans technology [5]. The first solution is obviously hard, while the second one could be efficiently used only for concrete streaming scenarios.

The shortcomings of the existing solution motivated us to start the development of the UGSF Data Flow Client. It can be considered as an alternative to the UGSF clients, which have been developed in the GridBeans technology. The UGSF Data Flow Client can be seen as a streaming counterpart of the GPE Expert Client, which allows for composing arbitrary *live* stream connections instead of building jobs workflows.

In this paper we present the UGSF platform and describe the Data Flow Client. The detailed discussion of the plugin interface is performed. Example applications are presented in the second part. The final part of the paper describes ongoing and future developments.

2 The UGSF Platform

The aim of the UniGrids Streaming Framework (UGSF) is the provision of a direct data streaming for applications. The main part of UGSF is *UGSF core*, which is a middleware that allows developers to create dedicated streaming services. Every system based on the UGSF will use the *core* together with some application dependent code. The UGSF core provides basic functionality common for all streaming applications. This includes a creation or a shut down of a connection. UGSF contains also a large group of versatile software pieces which can be reused when creating actual implementations of streaming services. A good example is a component which allows for locating UNICORE job's working directory.

The detailed UGSF architecture is presented elsewhere [6] and here we will introduce its brief overview focused on the basic elements. The UGSF is built based on the fundamental concept of *stream*. A Stream is a logical entity that encapsulates some functionality on server-side. This functionality can be:

1. production of data (by any means, e.g. importing it, reading from a file or even creating it)
2. consumption of data (any kind of destination like file export to another streaming system)
3. data processing (filtering, changing data format, etc.)

The stream defines at least one of input ports, output ports or bidirectional ports. Those represent possibility to connect to different UGSF streams or ordinary clients. The stream can have many ports, with different characteristics (list of allowed formats is a good example of stream characteristics). It can also perform many logical data transfers - “streamings” in common sense. Every logical

[1] “Programming from scratch” refers here to the GUI part of such applications. The UGSF provides client-side library, which simplifies development of the logic of the application. Nevertheless, significant programming skills are necessary.

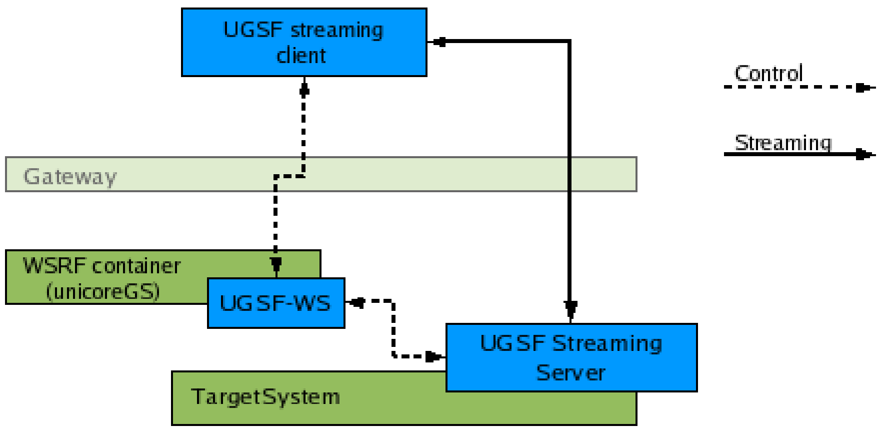


Fig. 1. The general architecture of UGSF

data transfer, started or ended at port of the stream is called a *flow*. To give an example: "Theora video decompressing stream" can have two flows; one input which accepts compressed Theora video, and one output flow, that transfers raw video frames. In this example the stream has defined data processing while neither production nor consumption of data (as defined above) are present. The implementation that provides defined functionality is called a *stream implementation*. Usage of the stream implementation requires creation of a corresponding resource which is called a *stream instance*.

The UGSF streams (or better *streams implementations*) have a metadata attached. For every flow there are defined capabilities such as reconnect capability, supported formats (or protocols) and others. It is possible to specify more than one format for a single flow as well as to express the only supported formats' combinations for the all flows of the stream together.

2.1 UGSF Architecture

The UGSF system is based on the WSRF compliant version of the UNICORE, (version 6) [1]. The first version of UGSF system was developed using UnicoreGS software [7] as the hosting environment. Recently, the hosting environment was changed to use mainstream UNICORE 6 server-side components: WSRFLite and UnicoreX.

The *UGSF core* consists of a *UGSF Web Service* part, *Streaming Server* part and a library to create clients. The usage of the last component is optional. The *UGSF Web Service* takes advantage of WSRF capabilities. It is used to control a set of available stream types, to create new streams and to manage already created ones. The *Streaming Server* part is managed by a *UGSF Web Service* and performs streaming. Client library is used to simplify the creation of the client-side software. Overall architecture is shown in Figure 1.

The *UGSF core* is complemented with stream implementations. Those consists of two server-side parts: streaming server and web service modules. Web service module implements control operations specific to the stream implementation. Streaming server module deals with a wire streaming protocol and data consumption or acquisition. The recent works provided a possibility to add client-side implementation for the stream implementation. The details are given in the section 3.

2.2 UGSF Web Service

The *UGSF Web Service* component consists of two kinds of web services. A base one (called *StreamingFrameworkService*) is responsible for connection authorization, creation of stream and its setup. During this process the new WS-Resource (called *StreamManagementService*) is created by a dedicated web service interface. This WS-Resource acts as a controller of an active streaming connection.

The *StreamingFrameworkService* is a WS-Resource which maintains list of *StreamManagementServices*. The *StreamingFrameworkService* allows users to get a list of available stream instances and to set up a connection to the specified one. The list of both owned and accessible streams is available. In addition, the *StreamingFrameworkService* has an administrative interface, which empowers system administrator to enable and disable particular stream types on the fly. The service reconfiguration such as addition or removal of stream types is also possible.

For each created stream an instance of the *StreamManagementService* allows user to perform universal operations for all streams. This includes shutting the stream down (by means of WS-Lifetime interface) or getting status and statistics of the connection. This functionality can be easily enriched by the developer. He can extend *StreamManagementService* with additional operations. The enriched implementations are free to consume any special XML configuration supplied to the *StreamingFrameworkService* and required for service setup and creation.

2.3 UGSF Streaming Server

The UGSF *Streaming Server* is a stand-alone, modular application which performs streaming to and from the target system. The server is tightly connected with the *UGSF Web Service* which maintains stream definitions (however *UGSF Web Service* can control multiple *Streaming Servers* without a problem). The server is modular and configurable.

Streaming Server modules can be divided into two categories: entry point modules and stream modules. The first kind of modules is responsible for implementation of special handshake protocol used to start streaming connection. Thanks to the modular architecture there can be many of such protocols available, even concurrently. An addition of a new one is possible and easy. Currently HTTP and HTTPS entry modules are available (special connection parameters are passed in HTTP header).

The second category of modules is responsible for streaming implementation. These modules can operate simultaneously in both directions: pushing the data from a server or pulling to the server. Stream module implements required elements of functionality presented in section 2. One possible class of streams are "filtering" streams which provides neither source nor sink for data. The data is read from a client, then processed and finally written out to a (possibly another) client.

Integration of the streaming functionality with grid jobs is of great interest here. The UGSF, among its standard stream modules, supplies visualization stream implementation. It can stream any kind of file both from and to job's USpace determined based on the given UNICORE job's reference and file's name. There is also a set of other implementations available, including TCP tunnels, UDP over TCP tunnels or multiplexer which clones input into many copies to name a few.

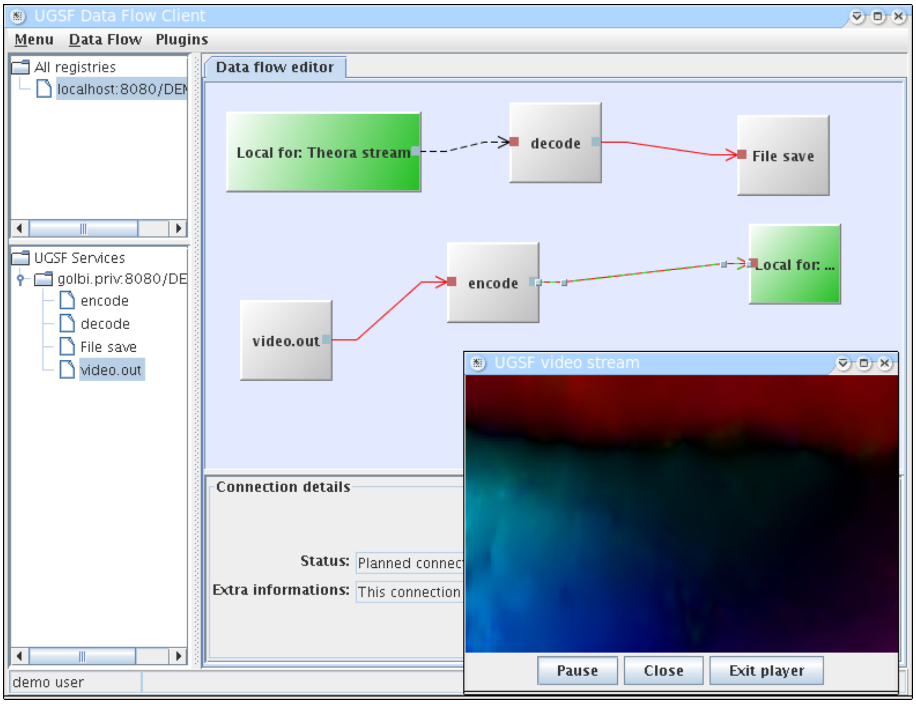
3 UGSF Data Flow Client

The UGSF Data Flow Client has been designed to provide solid base for creation of a specialized client applications which have to deal with *complex* streaming scenarios, including arbitrary data flow composition. The basic idea was to provide support for all, or nearly all generic features of UGSF and to support features available in sophisticated stream implementations by pluggable modules. Graphical approach was chosen for manipulating stream instances connections (i.e. data flow). To be fully functional, Data Flow Client must have possibility to act as a local endpoint for streaming in addition to control server to server connections. Local machine should be able to stream data in both directions to and from UGSF servers.

3.1 Generic Functionality

Data Flow Client manages the user's keystore which allows access to the grid. As the first entry point, some sort of resource discovery must be performed to locate streaming services. This is achieved in the usual manner for the UNICORE 6 — the user has to provide addresses of the registries. The same registry can be used for both UNICORE and UGSF services. The content of registries is automatically fetched and UGSF services are enumerated in side panel, called *services panel*. User can choose between having all services displayed or only those which are present in the actually chosen registry.

The stream instances managed by UGSF services are displayed in the same *services panel* in the form of a tree. From the context menus the user can create new instances, and destroy existing ones. While destroying is simple, the creation of a new stream instance is a more advanced operation and involves configuration of the instance. The client shows a pop-up dialog similar to shown in fig. 3. The dialog allows user to choose among all stream types defined in the selected streaming service. Further on, the user can choose the name for the job, set the



Status: Planned connec
Extra informations: This connection

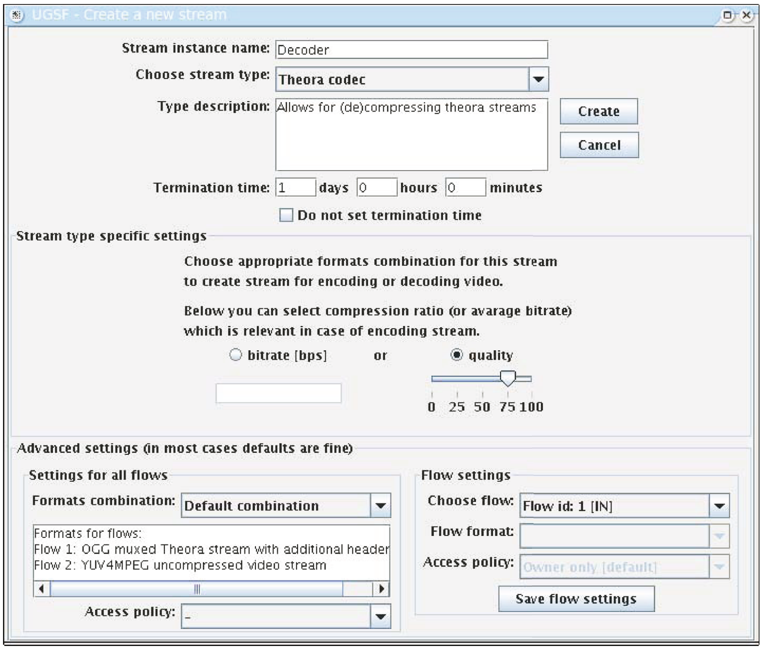
Fig. 2. The example of Data Flow Editor usage: Two simple data flows are prepared; one streams a local video file to a remote grid node, where UGSF filtering stream decompress the video and pushes it to the input file of the grid job. The job processes the input and its output is (after compression) sent back to the client, where live results are presented (optical flow of the input video sequence in this example).

termination time and select initial set of formats to be used. The dialog makes use of plugins to render GUI for preparation of any stream specific configuration that is needed.

When stream instance is created, its entry appears in the *services panel* tree. From there the stream instance can be added to the main workbench of the program: the graphical data flow editor. The editor visualizes data flow as a directed graph. Vertexes symbolize stream instances. Every vertex can have multiple *ports*, i.e. places of edges' attachment. Edges are used intuitively to represent stream connections (flows). Each fundamental property of a flow specifies direction of data transport.

Every stream can have many flows and adequate number of ports. There are also different kinds of ports: input, output and bidirectional rendered in a slightly different color at a different side of a vertex.

Usage of a stream instance in the data flow editor is accomplished by context menus. There are two menus: one for the stream instance to perform stream manipulation, and another one for every port to invoke operations related to the particular flows. For example, they allow for creating a new connection. The


UGSF - Create a new stream

Stream instance name: Decoder
Choose stream type: Theora codec
Type description: Allows for (de)compressing theora streams
Termination time: 1 days 0 hours 0 minutes
☐ Do not set termination time

Stream type specific settings
Choose appropriate formats combination for this stream
to create stream for encoding or decoding video.
Below you can select compression ratio (or avarage bitrate)
which is relevant in case of encoding stream.
bitrate [bps] or [✓ quality]

Advanced settings (in most cases defaults are fine)
Settings for all flows
Formats combination: Default combination
Formats for flows:
Flow 1: OGG muxed Theora stream with additional header
Flow 2: YUV4MPEG uncompressed video stream
Access policy: _

Flow settings
Choose flow: Flow id: 1 [IN]
Flow format:
Access policy: Owner only [default]

Fig. 3. Dialog used to create a stream instance in Data Flow Client. The parameters of video compression are set in subpanel which is provided by a plugin. The rest of the dialog window is generic.

client does not allow to connect two ports (an therefore flows) with incompatible formats or data transport directions.

An automatic policy setting is used for created connections and on default only creator can connect to the stream. This leads to the problem whenever one stream instance created in Data Flow Client has to connect to another instance. In this case the Data Flow Client sets up permissions behind the scene. The client fetches the identity of the stream A which initiates connection. In the next step it changes the authorization policy of the target stream, to accept the connections with the A.

There is also a possibility to manually control the most of generic features of the stream from the vertex, port and edge context menus. For example, if the stream supports multiple formats, which is the generic UGSF feature, it is possible to change them. The streams in the UGSF allow for choosing a format of the individual flow or of all flows together. Whenever there are format dependencies, a change of a format of one flow will affect the format of another.

The UGSF Data Flow Client also provides features to monitor data flow state. This is the function of a dedicated panel located below the data flow editor panel. The information about selected element is displayed there. If a port is selected, then related flow properties are shown including (among others) flow status CONNECTED, DISCONNECTED, statistics of transferred bytes in any direction, average transfer speed and the time of last activity. The status is updated either automatically or manually by the user.

The Data Flow Client also provides auto-discovery of data flows. This is necessary to avoid problems, when there are stream instances created by other clients or in other sessions. The available streaming services allow for doing this except for the connections between UGSF streams and external clients. The auto-discovery process can be divided into two parts. The more simple one is used whenever stream instance is added to the data flow editor. The instance is checked if it has some active connections. If so, the another side of connection is determined. If it is already known by the editor, the connection is automatically added. If it is not present, an "orphaned" connection is drawn to mark that the stream is used.

There is also a more advanced feature which finds all vertices with orphaned edges, tries to locate them on the grid (not only among streams in the editor) the peers and add them to the editor. The process is repeated until no more orphaned connections exist or there is no known element to be added.

3.2 Plugins

The generic functionality of the UGSF cannot be used without dedicated stream implementations which offer specialized control and configuration possibilities. The Streaming Framework Client uses dynamically loaded plugins to manage stream implementations.

There are cases where plugin for the stream implementation is not needed. The example is multiplexer stream which does not need any special configuration because it uses only standard operations of UGSF platform.

The client offers extensibility points for the plugins. Usage of most of them is optional and then some default values/components are used instead. The UGSF Data Flow Client can provide:

– GUI to ask for these stream creation parameters which are implementation dependent. The GUI has to return those parameters. It is used in stream creation dialog, for example to specify compression parameters for Theora stream encoding (see fig. 3).
– Menu with operations related to the stream. Such a menu is added to the context menu of vertex in data flow editor panel.
– Menu with operations applicable to the particular flow of the stream. Such a menu is attached to an appropriate port menu of the stream.
– Ability to create local endpoints.

Local endpoint feature is designed to allow user's machine to act as a peer in a data flow. In the UGSF the description how to connect to the remote stream is implementation dependant. Implementation provides data to stream and describes data format or application protocol. Local source or sink of data needs to be defined as well. This functionality is reserved exclusively for the plugins.

When local endpoint is created, it appears in a data flow editor as a vertex (but of different color than ordinary stream instances). The functionality resembles the simplified stream vertex — there are ports and two kinds of context menus. The only difference is that the contents of the menus are coming nearly exclusively from plugin implementation.

To give an example, the IVis stream used to stream files to/from UNICORE job work directory allows for creating local endpoint. This endpoint offers two features activated in its context menu: to create output flow and to create input flow. In the first case local file needs to be specified for streaming it out. In the latter case, the name of a local file to store streamed data is required.

4 Related Work

In general there are few general frameworks which integrate computing grid infrastructure with advanced streaming capabilities and flexible data flow creation.

In the case of UNICORE platform there is no such solution known to the authors. However frameworks that offers (some) streaming capabilities exist. One example is COVS framework [3]. It's aim is to support online visualisation and steering of scientific applications run on the grid, with collaboration support. The COVS implementation uses VISIT library [8] as underlying technology. Therefore COVS application is available only for VISIT enabled software. The COVS framework uses SSH to tunnel VISIT protocol and extedns it with web service management capabilities. The data flow is fixed: one application run on the grid node can be steered and visualised by one or more end-users. The UGSF approach to the streaming is far more powerful. It does not restrict streaming to one (eg. SSH) low level protocol. Usage of other protocols can bring large performance gain, especially when encryption is not required. UGSF allows for client $\leftrightarrow$ server communication as COVS does, but also for server $\leftrightarrow$ server. Last but not least, UGSF can be used with any streaming application run on the grid, not only those VISIT enabled. In conclusion we can state that UGSF and VISIT overlap only in small part of functionality. UGSF provides low level mechanism and COVS could be built on top of it.

The length of this paper doesn't allow for performing through comparison with streaming frameworks for other than UNICORE grid platforms as NaradaBrokering [9], GridKit [10] or GATES [11]. However we can state here that, whilst most of such platforms offers very extensive features in case of streaming itself, their integration with computational grid is very limmited. Also the visual data flow editor described in this paper is a significant advantage of UGSF compared to other solutions.

5 Conclusions and Future Work

The presented solution is a big step forward in providing streaming capabilities for UNICORE. It is a convenient and easy base to be used for universal stream composition.

Some of the presented streaming features can be performed using standard UNICORE technology such as GridBeans and UNICORE Clients. This is not possible for special cases, where user wants to execute workflow and stream data between tasks executed on the different target systems. With the UGSF Data Flow client, such task can be built with a few mouse clicks.

The solution presented here solves the most important problems related to the data streaming in the grid but needs some further development. One of the important features is possibility to save and restore data flow composition, which is different from saving graphical representation of workflow as it can involve many complicated situations. One example is recreation of already destroyed stream instances and connections. This might require development of the dedicated service acting as data stream broker. The another required feature is UGSF administrative interface, which will allow to define stream types and deploy and manage streaming services. Such work is now in progress.

This work was partially supported by European Commission under IST grant UniGrids (No. 004279).

References

1. UNICORE project (May 2007), `http://sourceforge.net/projects/unicore`
2. Benedyczak, K., Nowiński, A., Nowiński, K., Bała, P.: Real-Time Visualisation in the Grid Using UNICORE Middleware. In: Wyrzykowski, R., Dongarra, J., Meyer, N., Waśniewski, J. (eds.) PPAM 2005. LNCS, vol. 3911, pp. 608–615. Springer, Heidelberg (2006)
3. Riedel, M. et al: Requirements and Design of a Collaborative Online Visualization and Steering Framework for Grid and e-Science infrastructures. German e-Science Conference, (May 2007)
4. Foster, I., Kesselman, C., Nick, J., Tuecke, S.: The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration. Globus Project, 2002 (May 2007),
   `http://www.globus.org/alliance/publications/papers/ogsa.pdf`
5. GPE4GTK project (May 2007),
   `http://gpe4gtk.sourceforge.net`
6. Benedyczak, K., Nowinski, A., Nowinski, K., Bala, P.: UniGrids Streaming Framework. In: Kågström, B., Elmroth, E., Dongarra, J., Waśniewski, J. (eds.) PARA 2006. LNCS, vol. 4699, Springer, Heidelberg (2007)
7. UniGrids project (May 2007), `http://www.unigrids.org`
8. Visualization Interface Toolkit (May 2007),
   `http://www.fz-juelich.de/zam/visit`
9. Pallickara, S., Fox, G.: NaradaBrokering: A Distributed Middleware Framework and Architecture for Enabling Durable Peer-to-Peer Grids. In: Endler, M., Schmidt, D.C. (eds.) Middleware 2003. LNCS, vol. 2672, pp. 41–61. Springer, Heidelberg (2003)
10. Grace, P., et al.: GRIDKIT: Pluggable Overlay Networks for Grid Computing. In: Meersman, R., Tari, Z. (eds.) CoopIS/DOA/ODBASE (2). LNCS, vol. 3291, pp. 1463–1481. Springer, Heidelberg (2004)
11. Chen, L., Reddy, K., Agrawal, G.: GATES: A Grid-Based Middleware for Processing Distributed Data Streams. In: HPDC, IEEE Computer Society, Los Alamitos (2004)