GWD-R-P.134
OGSA-DMI WG

Mario Antonioletti, EPCC
Michel Drescher, Fujitsu
Allen Luniewski, IBM
Steven Newhouse, Microsoft
Ravi Madduri, Argonne National Laboratory
13 June 2008

OGSA-DMI Functional Specification 1.0

Status of This Document

This document specifies two port types to support the instantiation and management of data transfers within and across Grid deployments. Distribution is unlimited.

Abstract

The Open Grid Services Architecture Data Movement Interface (OGSA-DMI) specification defines a standardized mechanism for moving data from its source to its destination. By abstracting this movement, the client complexity for moving data within a grid is greatly reduced. OGSA-DMI defines two port types for initiating, scheduling and managing data transfers from a given source to a specified destination. The source and destination are described through Data End Point References (DEPRs), a specialized form of a WS-Address.

An OGSA-DMI service will use the properties of the data to be transferred and the properties of the infrastructure to decide what the best underlying transfer protocol to undertake the actual data transfer is for the given combination of data source and sink. By this means OGSA-DMI provides a common interface, though its abstraction layer, that can be used to span the preferred transfer mechanisms used across many different types of Grid infrastructures.

This version of the specification aims to ensure that the OGSA-DMI port types are composable with other Web Service (WS) specifications whilst being agnostic about any specific WS rendering. OGSA-DMI is only concerned with the moving of data, i.e. bytes. Any associated semantics are the responsibility of higher level services that leverage OGSA-DMI. The OGSA-DMI framework is general enough to cater for the movement of single elements of data or composites.

Mario Antonioletti, EPCC
Michel Drescher, Fujitsu
Allen Luniewski, IBM
Steven Newhouse, Microsoft
Ravi Madduri, Argonne National Laboratory
27 May 2008

Contents

Mario Antonioletti, EPCC
Michel Drescher, Fujitsu
Allen Luniewski, IBM
Steven Newhouse, Microsoft
Ravi Madduri, Argonne National Laboratory

1. Introduction

The Open Grid Services Architecture Data Movement Interface (OGSA-DMI) specification describes two port types that provide operations to co-ordinate the transfer of data from a source location to a sink (destination) location. What happens to the data at the source location once the data has been transferred is beyond the scope of this specification. The OGSA-DMI port types have been designed to encapsulate many different data transfer mechanisms – indeed this is the value provided by a service implementing these port types. It enables the user consuming the service to focus on what is important to them – the transfer of data from one location to another whilst observing a few optional constraints. The selection of the most suitable underlying data transfer protocol, from those that are supported by the service, is undertaken by the OGSA-DMI service – without any direct user involvement.

1.1 Goals

This version of the specification aims to:

- Ensure that the OGSA-DMI port types are composable with other Web Service (WS) specifications.
- Ensure that the OGSA-DMI specification defines the semantics of its port types but is agnostic about any specific WS rendering.
- Provide normative renderings of the OGSA-DMI specification for different 'flavors' of WS.
- Be agnostic about the semantics and type of data being transferred.
- Consider (from the DMI perspective) that the data transfer consists of a single logical data transfer moving from a data source to a data sink. This logical data transfer (e.g. a directory) may consist of multiple data items (i.e. files).

1.2 Non-Goals

For this version of the specification we consider the following topics to be out of scope. We will consider these important topics for consideration in future revisions:

- The notion of bandwidth negotiation or the establishing of service level agreements.
- The co-ordination, or scheduling, of a data transfer with any other activity.
- Encapsulation of data replication information, for the purposes of reliability or data transfer optimization, within the source or sink service and the propagation of this information within the OGSA-DMI architecture.
- Imposition of constraints as to how data is made available through a source or what happens with the bytes that are delivered to a sink.
- Knowledge of, or guarantees, being made regarding the semantics of the data being transferred.

Implementations MAY provide any of these capabilities on an *ad hoc* basis provided they are able to still interoperate with other implementations, i.e. without changing the interface or the semantics of any of the OGSA-DMI defined operations.

2. Notational Conventions

The key words 'MUST," "MUST NOT," "REQUIRED," "SHALL," "SHALL NOT," "SHOULD," "SHOULD NOT," "RECOMMENDED," "MAY," and "OPTIONAL" are to be interpreted as described in RFC 2119 [BRADNER1]

When describing abstract data models, this specification uses the notational convention used by the XML Infoset [XML Infoset]. Specifically, abstract property names always appear in square brackets (e.g., [some property]).

When describing concrete XML schemas [XML Schema Part 1, Part 2], this specification uses the notational convention of WS-Security [WS-Security]. Specifically, each member of an element's [children] or [attributes] property is described using an XPath-like notation (e.g., /x:MyHeader/x:SomeProperty/@value1). The use of {any} indicates the presence of an element wildcard (<xs:any/>). The use of @{any} indicates the presence of an attribute wildcard (<xs:anyAttribute/>).

2.1 XML Namespaces

This specification uses a number of namespace prefixes throughout; they are listed in Table 1. Note that the choice of any namespace prefix is arbitrary and not semantically significant (see [BRAY]).

Prefix	Namespace
s11	http://schemas.xmlsoap.org/soap/envelope
rns	http://schemas.ogf.org/rns/2006/09/rns
wsa	http://www.w3.org/2005/03/addressing
xs	http://www.w3.org/2001/XMLSchema
dmi	http://schemas.ogf.org/dmi/2007/05/dmi

Table 1: Namespaces and prefixes used in this document

3. Architecture

A common requirement within Grid environments is the ability to transfer data from one location to another. The services and protocols required to support such data transfers will vary between Grid deployments, the actor requesting a data transfer, and potentially the amount of data that is to be transferred. The sole purpose of this specification is to control and co-ordinate the transfer of bytes from a source (an emitter of an ordered byte sequence) and a sink (a receiver of an ordered byte sequence) that is independent of the protocol being used to undertake the data transfer.

To provide such a capability, OGSA-DMI defines two port types:

- A port type that defines how to set up a data transfer instance.
- A port type that defines operations for a data transfer instance in order to:
  - control its behavior, e.g. "resume" and "suspend"
  - return attribute values relating to its internal behavior, e.g. "number of bytes moved", "total size of the data moved".

Figure 1 gives an overview of the functional architecture and depicts the use of particular communication mechanisms:

- On the far left is an unspecified interaction between a user agent (human) and a client agent that will undertake the interaction with services that implement the OGSA-DMI port types.
- In the centre is an OGSA-DMI specific communication (as defined in this document) between a client and implementations of the Data Transfer Factory (DTF) and Data Transfer Instance (DTI) port types.
- On the far right of the diagram are the established native data transfer protocols for controlling and undertaking the data transfer, e.g. FTP[POSTEL], GridFTP[ALLCOCK01], HTTP[FIELDING], UDT[GU], etc.

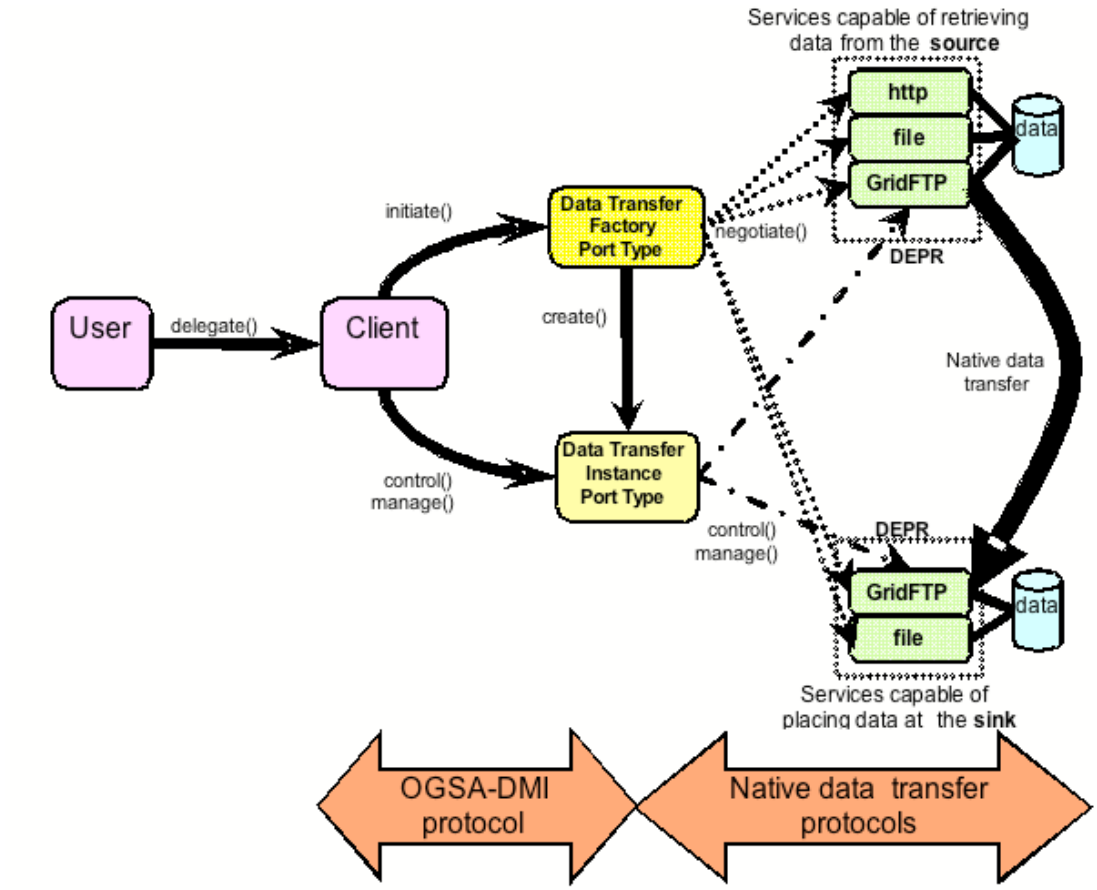

Figure 1: OGSA-DMI Architecture

It is important to note that the source and sink services need to ‘get’ and ‘put’ the data and, although they are shown separately in this diagram, they may be treated as the same functional entity. They are treated separated in this overview to make it clear that the source and sink can be addressed individually using a specific data transport protocol or with a specific identity token when they form part of different virtual organizations.

This architecture allows the DTF & DTI to be remote from the data source and data sink and to initiate the transfers from a remote location. Such ‘third party transfers’ are not supported by all protocols. When they are not supported the DTI (or another entity) may act as an intermediary downloading the bytes from the source and uploading the bytes to the sink. Clearly, protocols such as GridFTP which support third party transfers would be more effective in these scenarios.

This diagram does not show the interactions between the Client (or User) and the source and sink resources that are needed to select the data that is to be transferred. It also does not show the interactions that specify the destination to which the data is to be transferred to, and how the protocols that can be used to access the source and sink data locations are determined. These interactions are out of scope for OGSA-DMI even though their outcome is necessary in order to fully specify the data transfer activity through the information that is embedded within the Data Endpoint Reference (DEPR).

3.1 Topology

It is inevitable that different Grid deployments will support a number of different data transfer protocols through different service interfaces and that not all nodes in a Grid will support all of the available protocols.

In an OGSA-DMI enabled Grid, there will be at least one deployed service implementing the OGSA-DMI Data Transfer Factory port type available. Implementations of this port type will have been built using a number of transport protocols which will be advertised in the factory's 'supported protocols' attribute. Through this port type a client will be able to initiate a data transfer activity from a data source to a data sink using the most suitable protocol supported at both the source and sink locations that can be used to move the data. This data movement will be supported by data movement services located at the source and sink. A typical Grid deployment might use existing low-level data movement services, protocols, or mechanisms (e.g. GridFTP, file access, SRB, etc) that have been defined elsewhere.

3.2 Entities

This section describes the entities that participate in a data transfer. These entities can be natural (i.e. human beings), or agents. Non-human entities may be implemented through hardware or software only, or software governing/controlling hardware.

3.2.1 Data Endpoint Reference (DEPR)

A DEPR encapsulates all the information relating to how the data it describes can be accessed (a source DEPR) and where it is to be placed (a sink DEPR). Thus, a unique DEPR is needed for each source and sink. The transfer protocols by which these operations can be undertaken are encapsulated in these endpoint references and are used by the DTF to select the protocol for the underlying data transfer. Although this version of the specification does not make use of all of the WS-Addressing Endpoint Reference data structure, it is anticipated that future versions will do so. This approach keeps the interface stable, and re-tooling is kept at a minimum for both clients and implementations.

3.2.2 Source

The source of a data transfer contains all the bytes that need to be transferred to a sink. The internal organization of this data (e.g. a flat file, relational database, distributed across replicas, striped for performance, etc.) is not exposed to the DMI architecture.

From the viewpoint of the other data transfer entities the source's primary function is to emit an ordered byte sequence using one of several possible data transfer protocols.

3.2.3 Sink

The sink of a data transfer will eventually receive all the bytes that are at the source of a data transfer. Data semantic issues, such as byte ordering, are outside the scope of the DMI specification but may be supported natively by the data transfer protocol used. As with the source, the internal organization of the data once it is consumed by the sink is intentionally left unspecified.

3.2.4 Data Transfer Factory Port Type

The Data Transfer Factory (DTF) port type's sole role is to set up a data transfer between a data source and a data sink from a set of parameters supplied by the user.

To set up a data transfer, a DTF communicates with both the source and sink, using the

information encapsulated in the source and sink DEPRs. The DTF attempts to negotiate a data transfer protocol and, on success, it sets the protocol specific parameter settings in a manner consistent with the current parameter set and the configuration of the DTF. The DTF returns an Endpoint Reference (EPR) to a Data Transfer Instance (DTI) to the client. This EPR can be used to monitor and control the data transfer through an instance of a Data Transfer Instance (DTI) port type.

Within a typical Grid deployment, a DTF is a persistent well-known service which is expected to have a long lifetime, possibly measured in months or years.

3.2.5 Data Transfer Instance Port Type

The operations in a Data Transfer Instance (DTI) port type enable a DTI's configuration and current state to be discovered, as well as providing a means to control a data transfer through an appropriate EPR. To the Data Transfer Client, a DTI acts as a proxy for the underlying data transfer using the interface defined in this specification. Additional protocol-specific capabilities and information relating to a data transfer MAY be provided if this does not alter the semantics and operation defined in this specification.

Clients of a DTI may poll for information about its status, or apply management and control operations as described below by using the given EPR.

The EPR is used for one and only one data transfer and is therefore fairly short-lived compared to an EPR to a DTF. When the underlying data transfer has finished, the state corresponding to the EPR that is accessed through the DTI will be destroyed.

However, this destruction of state is not performed automatically after the last byte of the data has been transferred and successfully stored at the sink or if the data transfer fails for some unexpected reason. The DTI configuration allows this state to exist past the last transferred byte until the user has confirmed that it no longer needs access to this state as the data transfer is complete. In addition, the actor requesting the transfer may also specify a duration for the state information to be retained (Section 4.2.1.3 - the [stay alive time]).These leads to three scenarios:

- No [stay alive time] specified: State data is kept for an implementation specific duration.
- Sensible [stay alive time] specified: State data is kept for the requested duration.
- Unreasonable [stay alive time] specified: State data cannot be expected to be persisted for 'all' time and after an implementation specific duration it will be destroyed.

Therefore, implementations MAY enforce a policy of a maximum extended lifetime for this state (so that DTIs do not need to store this state indefinitely).

3.2.6 Data Transfer Client

A Data Transfer Client contacts a DTF to set up and configure a data transfer. It will receive a handle to a DTI, which it can use to query for the transfer's status and other relevant information.

The Client may invoke control and management operations, such as "Stop", "Start", "Suspend", "Resume", "State", etc. as available and applicable on the service, see Section 5.2.

A Data Transfer Client is not always the same entity as a Data Transfer User. Rather, the Data Transfer Client might be a software agent located at a remote execution management system that executes parts of a larger workflow that require data transfers on behalf of the Data Transfer User. It could also be a web based portal that lets the user manually set up and invoke data transfers.

3.2.7 Data Transfer User

A Data Transfer User is the only human actor in the Data Transfer Architecture. The identity of

the Data Transfer User is immutable. The identity of the Data Transfer User may have important security implications (see the Security section later on in this document).

3.3 Using the DEPRs

The DMI model is able to encompass three distinct use cases through the defined port-types and DEPR structure. In the following scenarios 'data' encompass structured data and data with no particular structure contained within a file. These use cases are driven by a practical realization that initially data services that provide information on data location and access will not be able to provide DEPRs. Until data services are able to map logical data transfer requests to actual protocols we provide a migration strategy as to how DEPRs can be used with DMI services.

3.3.1 Explicit Data Transfer

In this scenario the client knows the source and destination URLs for the data that is to be moved and the protocol that is required to undertake the movement. Through a client tool (or by explicitly constructing the required XML) a DEPR is provided for the source and the sink using the single URL specifying the real location of the data source and sink. The wsa:Address element in the DEPR is explicitly marked as **http://www.ogf.org/ogsa/2007/08/addressing/none** as defined in Section 4.2.1.2 to indicate that this DEPR is structure complete as it is.

As the source and the sink DEPRs each contain a single protocol element the DTF port type is able to use the provided protocols and URL to initiate the DTI. As the information has been explicitly provided the DTF is guaranteed to find a match if the requested protocol is supported by the DTI implementation.

3.3.2 Defined Data Transfers

In this scenario the client has already been in contact with a data registry or replica catalogue and has obtained the necessary information required to map their logical source and sink data names to the protocols by which the data can be accessed. The mechanism used to undertake this mapping process is not defined. Through a client tool (or by explicitly constructing the required XML) a DEPR is provided for the source and the sink. The source and sink DEPRs contain the set of protocols by which the data at the source and sink locations can be accessed. The wsa:Address element in the DEPR is explicitly marked as **http://www.ogf.org/ogsa/2007/08/addressing/none** as defined in Section 4.2.1.2 to indicate that this DEPR is structure complete as it is.

As the source and the sink DEPRs may each contain multiple protocol elements the DTF port type is able to use the provided protocols and URLs to select a protocol that is both supported at the source and sink, and is supported by the DTI implementation to initiate the DTI. It is possible that no matching protocol will be found within the source and sink DEPRs and a fault will be generated as discussed in Section 4.

3.3.3 Logical Data Transfers

In this scenario the client is able to provide the logical data names for the source and sink locations in the DEPR and a resolving service (using the wsa:Address element) that the DTF can use to retrieve the protocols supported by the data source or sink. Through a client tool (or by explicitly constructing the required XML) a DEPR is provided for both the source and the sink containing this information and these are passed to the DTF port type.

The DTF port type is not able to select and initiate a data transfer at this point as it has no information on the available protocols. By using the wsa:Address element contained in the source and the sink DEPR the DTF is able to contact a separate service that is able to provide the mapping between the logical data name and the protocols and URLs by which the data can be

accessed. This information, effectively the 'DataLocations' element in the DEPR, may be returned by annotation of the DEPR passed to a 'mapping' port type or by explicitly returning the 'DataLocations' element in the return message body. No statement is made by this document as to the normative definition of the 'mapping' port type.

4. Data Transfer Factory Port Type

This section defines the operations of the Data Transfer Factory (DTF) port type and its attributes. The purpose of the DTF is to set up a data transfer between a source and a sink.

The DTF provides a set of properties describing the current configuration of the Factory. The set of properties MUST minimally include the compulsory attributes defined in this specification. It MAY support the optional attributes defined in this specification, and using the defined extensibility mechanisms any implementation specific attributes. These attributes are defined in the following properties section.

The factory has a single operation that requests a data movement activity, "GetDataTransferInstance". Parameters given in invoking this operation define the requirements specified by the client for the data transfer activity. If creating the data transfer activity succeeds, then the DTF MUST create and return an EPR to a DTI back to the client. If creating the data transfer fails then the DTF MUST throw a fault and no EPR to a DTI is created. Details to the fault condition MAY be given in the fault data structure. The abstract interface definition is as follows:

Data Transfer Factory
[supported protocol]+
[service instance] GetDataTransferInstance([source DEPR], [sink DEPR], [transfer requirements]); [factory attributes] GetFactoryAttributesDocument()

Figure 2: Functional Data Transfer Factory Interface

The details of the DTF are described in below.

4.1 Properties

A Data Transfer Factory exports the following read-only properties that describe its configuration and capabilities.

4.1.1 [supported protocol]

Each [supported protocol] information element identifies a particular data transfer protocol that is recognized by the DTF (and any generated DTI) and can potentially be used to initiate a data transfer from a source or a sink. If a protocol specified within a DEPR is not listed as a supported protocol by the DTF then it will not be considered as an option when trying to find a matching protocol between those supported at the data source and sink, and the DMI implementation. Multiple [supported protocol] elements may be specified. A [supported protocol] element indicates which protocols this factory knows about. Each protocol will be characterized by one of several specific undo strategies as described below. There cannot be a default [undo strategy] for a DTF and any created DTI. If a DTF supports more than one data transfer protocol then it must provide an instance of [supported protocol] for each transfer protocol that it supports.

For the purpose of this specification, the following URIs identify the described data transfer protocol and are intended for use as values within the [supported protocol] information elements.

If an implementation supports one of the following data transfer protocols it MUST use one of the corresponding identifiers to identify it.

http://www.ogf.org/ogsa-dmi/2006/03/im/protocol/gridftp-v20
This identifier describes the GridFTP as specified in [ALLCOCK01].

http://www.ogf.org/ogsa-dmi/2006/03/im/protocol/http/v11
This describes the HTTP/1.1 as specified in [FIELDING].

http://www.ogf.org/ogsa-dmi/2006/03/im/protocol/ftp
This describes the FTP as specified in [POSTEL].

http://www.ogf.org/ogsa-dmi/2006/03/im/protocol/ftp-passive
This describes the “Passive Mode” for FTP when a sink only supports passive FTP transfers, e.g. because of firewall restrictions, as described in [POSTEL].

Implementations may choose to recognize other protocols identifiers.

4.1.1.1 [undo strategy]

If a data transfer being undertaken by a DTI should fail, then a DTI may have recovery options described in the [undo strategy] information element depending on the capabilities of the underlying data transfer protocol being used. The success or failure of the DTI in removing artifacts generated through the data transfer is described in the DTI’s Failed state.

The data type of the [undo strategy] information element is a URI. When an [undo strategy] is invoked, its resultant state {clean, unclean or unknown} MUST be reflected in the DTI’s ‘Failed’ state.

An implementation MUST describe the undo strategy that it supports for each supported protocol using one of the following identifiers. In declaring its support for such an undo strategy it MUST follow the declared semantics of the undo strategy, namely:

http://www.ogf.org/ogsa-dmi/2006/03/im/undo/full
The data transfer protocol guarantees a complete and verifiable cleanup, which ensures that:
- Any data transferred to the sink is removed from the sink;
- Any temporary data generated at the sink or the source are removed;
- Any remaining space reservations done at the sink are removed;
- Any remaining bandwidth reservations are undone;

By declaring such guarantee a failed data transfer, after the undo has been applied MUST advance to the state :Failed:Clean”. Conversely, it MUST NOT assume either “Fauled:Unknown” or “Failed:Unclean”.

http://www.ogf.org/ogsa-dmi/2006/03/im/retry/best-effort
A best effort indicates that a full undo is attempted, as described earlier, but the outcome of this effort cannot be guaranteed or verified. An OGSA-DMI implementation MAY provide additional information about the best-effort undo activity such as which of the undo efforts succeeded or failed.
Depending on the outcome of the undo operation the resultant state MAY be either of the “Failed:{Clean, Unclean, Unknown}” states. The underlying undo operations, if invoked, may be able to completely tidy up the system with no traces left behind. In such case the resultant state SHOULD be “Failed:Clean”. If the underlying undo operations are known to

not tidy up completely, then the resultant state SHOULD be “Failed:Unclean”. However, if it cannot be safely determined whether one or more undo operations were successful, the resultant state SHOULD be “Failed:Unknown”.

http://www.ogf.org/ogsa-dmi/2006/03/im/retry/none
The protocol does not guarantee to undo any of the changes caused at the entities participating in the data transfer (e.g. releasing reserved resources, removing the partial data transferred at the storage space, etc.).
In case the underlying data transfer fails the resultant state MUST be “Failed:Unclean”.

Clients MAY take the appropriate actions to perform any clean up on their own, which is out of scope for this specification.

4.1.1.2 XML Representation

The [supported protocol] information element is rendered in XML as:

```
<dmi:SupportedProtocol name=”xs:anyURI”>
    <dmi:UndoStrategy name=”xs:anyURI”/>
</dmi:SupportedProtocol>
```

Where:

/dmi:SupportedProtocol
Represents the [supported protocol] information element. Its XML type is defined as xs:anyURI.
/dmi:UndoStrategy
Represents the [undo strategy] information element. Its XML type is defined as xs:anyURI.

4.2 Operations

A DTF defines two operations: one that requests a data transfer and another to retrieve information relating to its configuration.

4.2.1 GetDataTransferInstance

This operation is used to create a DTI in response to a request for a specific data transfer to take place with specific constraints.

Invoking this operation will, on success, create an EPR to a DTI that is then returned by this operation. The DTI encapsulates access to the underlying data transfer.

To create an EPR to the DTI, a DTF MUST perform some matching between the protocols that are supported by both the source and sink (captured within the DEPRs [available protocols] element, and the protocols supported by the DTI which are captured in the [supported protocols] attribute of the DTF. A simple implementation of the OGSA-DMI specification may decide that if no common protocol exists between the source, the sink and the DTI then a data transfer is not possible. A more sophisticated implementation may explore the use of intermediaries to find a transfer mechanism. If no data transfer is possible then the operation MUST exit with a NoSourceSinkProtocolMatchFault (See 4.3.2). The particular fault returned MAY provide further information about the fault condition.

The following information elements further describe the parameters used in the “GetDataTransferInstance” operation.

4.2.1.1 [service instance]

A [service instance] information element describes a WS-Addressing Endpoint Reference (EPR) to a DTI. This EPR is minted by a DTF during the “GetDataTransferInstance” operation and is sent back to the client of the DTF. The instantiation of an EPR for that DTI indicates that the setting up of a native data transfer was successful and that it has been encapsulated in the DTI.

Once the Client receives the EPR, its communication with the DTF is complete for the current data transfer, and all future communication by the client for this data transfer MUST be conducted via the DTI using the EPR returned.

4.2.1.1.1 XML Representation

When rendered in XML, the [service instance] information element is described as follows:

```
<dmi:ServiceInstance> wsa:EndpointReferenceType </dmi:ServiceInstance>
```

The following describes the attributes and elements in the Pseudo Schema given above:

/dmi:ServiceInstance

This represents the [service instance] information element. Its XML type is defined as wsa:EndpointReferenceType.

4.2.1.2 [source DEPR] or [sink DEPR]

The [source DEPR] information element describes the source of a data transfer and has the type of a Data Endpoint Reference (DEPR). The [sink DEPR] information element describes the sink of a data transfer and has the type of a DEPR. Although both the source and sink DEPRs have the same type they are semantically very different.

The DEPR that is used as either the sink or source is obtained in a manner that is out of bands for this specification. The DTF client passes these DEPRs to the DTF A DEPR may be minted by a Web Service that has knowledge of the data store and the protocols by which it can be accessed. This Web Service may be identical to the one that handles and manages the actual data represented by the DEPR. The DEPR is of type wsa:EndpointReferenceType and MUST NOT be null.

The Web Service that mints and issues the DEPR element MUST provide at least one protocol contained within a [data] element describing how the data represented by the DEPR must be accessed. If more than one [data] element is specified each element MUST contain a different protocol. The [data] elements are wrapped in the container element [data locations]. The [data locations] element MUST be present in the [metadata] section of the DEPR.

In effect, the [data EPR] element (introduced conceptually in Section 3.2.1) defines a profile in the [metadata] section of a generic WS-Addressing EndpointReferenceType. However, this does not prevent the Web Service that mints the [data EPR] element from adding other metadata using the described extensibility mechanism, which is out of scope for this specification.

In addition, a [data EPR] may be used to represent data aggregates such as a collection of files. How this is represented within the [data EPR] is out of scope for this document. Nevertheless, this abstraction allows a DTI to represents the bulk transfer of data aggregates between a source and sink. In practice, if one were using GridFTP, the transfer of a directory with its associated content could be achieved by adding a "/" at the end of the URL representing the data within the [data EPR]. Aggregation thus becomes an implementation detail but it is nevertheless possible within the context of the present DTI specification.

For example, a DataEPR that describes that the directory “/name/of/the/dir/” including its contents may be transferred using GridFTP 2.0 or SRM may look like this:

```
<dmi:DataEPR>
  <wsa:Address>
    http://www.ogf.org/ogsa/2007/08/addressing/none
  </wsa:Address>
  <wsa:Metadata>
    <dmi:DataLocations>
      <dmi:Data
           ProtocolUri="http://www.ogf.org/ogsa-
dmi/2006/03/im/protocol/gridftp-v20"
           DataUrl="gsiftp://example.org/name/of/the/dir/">
      </dmi:Data>
      <dmi:Data
           ProtocolUri="urn:my-project:srm"
           DataUrl="srm://example.org/name/of/the/dir/">
      </dmi:Data>
    </dmi:DataLocations>
  </wsa:Metadata>
</dmi:DataEPR>
```

Note the difference in the declaration of the protocol: This version of the specification does not define a normative URI for the SRM protocol hence a project specific identifier for the SRM protocol.

4.2.1.2.1 XML Representation

When rendered in XML, the [source DEPR], or [sink DEPR], element will be described as follows:

```
<dmi:DataEPR>
    <wsa:Address/>
    <wsa:ReferenceParameters/>?
    <wsa:Metadata>
        <xs:any/>*
        <dmi:DataLocations>
            <dmi:Data ProtocolUri=”xs:anyURI” DataUrl=”xs:anyURI”>+
                <dmi:Credentials>?
                     <xs:any namespace=”##other”/>+
                </dmi:Credentials>
            </dmi:Data>
        </dmi:DataLocations>
    </wsa:Metadata>
</dmi:DataEPR>
```

The following describes the attributes and elements in the Pseudo Schema given above:

/dmi:DataEPR

This represents the [source], or [sink], information element. Its XML type is defined as wsa:EndpointReferenceType.

/dmi:DataEPR/wsa:Address

This represents the [address] element.
If this DataEPR is intended to be used as a container element for a dmi:AvailableProtocols element only, and not as a genuine WS-Addressing EndpointReferene that points to a live

Web Service, then the wsa:Address element MUST have the value:
“http://www.ogf.org/ogsa/2007/08/addressing/none”

/dmi:DataEPR/tns:ReferenceParameters
This represents the WS-Addressing [reference parameters] element.

/dmi:DataEPR/wsa:Metadata
This represents the WS-Addressing [metadata] element.

/dmi:DataEPR/wsa:Metadata/xs:any
This represents the extensibility point for further information about the data as defined in WS-Addressing.

/dmi:DataEPR/wsa:Metadata/dmi:DataLocations
This represents the [DataLocations] element. This element serves as a container element for all data transfer protocols that can be used to access the data addressed by the parent dmi:DataEPR element.

/dmi:DataEPR/wsa:Metadata/dmi:DataLocations/dmi:Data
The [data] element describes the protocol Uri and protocol specific information by which the data can be retrieved or stored.

/dmi:DataEPR/wsa:Metadata/dmi:DataLocations/dmi:Data/@dmi:ProtocolUri
The ProtocolUri attribute contains the normative identifier for the described protocol. Section 4.1.1 defines normative identifiers for particular data transfer protocols. These MUST be used to identify the supported data transfer protocol.
As an extensibility option, the name attribute MAY contain identifiers for data transfer protocols that are not normatively referenced in this specification. An implementation MAY ignore any [data] elements with such an attribute.

/dmi:DataEPR/wsa:Metadata/dmi:DataLocations/dmi:Data/@dmi:DataUrl
The DataUrl attribute contains data transfer protocol specific access information. If the DTF selects the data transfer protocol that is identified by the value of the sibling dmi:ProtocolUri attribute, then the DTI MUST use the information provided in the dmi:DataUrl attribute to access the data.

/dmi:DataEPR/wsa:Metadata/dmi:DataLocations/dmi:Data/dmi:Credentials
This OPTIONAL element contains one or more credentials that can be used to request access to the specified data URL, e.g. a Serialized X509 GSI proxy, a username & password, SecureID one time password challenges, etc.
The credentials are initially inserted by the service generating the DEPR (if it is generated) using the credential that the user used to identify themselves to that service, in a process that is out of scope for DMI. Those credentials MAY require that the DMI User or DMI Client add elements to the dmi:Credentials section, e.g. the response to a SecureID challenge. Entities that create or augment the dmi:Credentials element SHOULD use digital cryptography technology to protect and ensure confidentiality, authenticity and validity of the contained credentials. For scenarios that allow more than one entity to augment the dmi:Credentials section it is RECOMMENDED that the individual child elements are digitally signed, instead of the dmi:Credentials element itself.
The credential do not need to be the user’s credential but could instead be a specific credential that could only be used once to gain access anonymously to the specified data URL.

/dmi:DataEPR/wsa:Metadata/dmi:DataLocations/dmi:Data /dmi:Credentials/xs:any
This extensibility point describes any security information that is required to enable the Source or Sink to properly authenticate and authorize the requested Data Transfer. The

format of such data must conform to the security framework employed in the current Grid infrastructure.

4.2.1.3 [transfer requirements]

A [transfer requirements] information element describes various constraints on the data transfer request.

The data type of the [transfer requirements] element is complex, i.e. it does not define or allow content other than child information elements.

This element is optional when requesting a data transfer. When present, the DTF MUST use these values to drive the selection of the underlying protocols and services. If not specified then the DTF will use the default values.

4.2.1.3.1 XML Representation

When rendered in XML, the [transfer options] information element is described as follows:

```
<dmi:TransferRequirements>
    <dmi:StartNotBefore/>?
    <dmi:EndNoLaterThan xsi:nillable=true />
    <dmi:StayAliveTime/>?
    <dmi:MaxAttempts />?
    <xs:any namespace="##other"/>*
</dmi:TransferRequirements>
```

The following describes the attributes and elements in the Pseudo Schema given above:

/dmi:TransferRequirements
This represents the [transfer requirements] information element. Its XML type is defined as xs:complexType.

/dmi:TransferRequirements/dmi:StartNotBefore
This represents the [startNotBefore] information element. The data transfer encapsulated by the DTI MUST NOT start before the date, time and time zone specified by this element. If not specified its default value will be the date and time that the DTI was created. The DTI MUST NOT enter the transferring state before this time unless the “Start” operation on the DTI is invoked.

/dmi:TransferRequirements/dmi:EndNoLaterThan
This represents the [endNoLaterThan] information element. The value MUST specify a date and time greater than or equal to the value of the [startNotBefore] element. The default value that will be used by the DTI will be null indicating that there is no date and time by which the data transfer must complete by.

If a value is specified for this element and the DTI has not reached the “Done” state by the time the specified date and time is reached, then the DTI will be deemed to have failed and MUST enter the “Failed” state. The DTI MUST attempt to halt the underlying data transfer using the selected protocol. Other mechanisms MAY be used by the DTI to halt the data transfer.

If the [undo] element exists then the results of the partial transfer at the sink MUST be removed so the DTI can enter the appropriate cleanup related state, i.e. clean, unclean, unknown.

/dmi:TransferRequirements/dmi:StayAliveTime

This represents the [stay alive time] information element. It is used to specify the minimum time, once the DTI has entered the “Done” state that the DTI will be maintained before it is automatically cleaned up. If this element is not specified then there will be no automatic cleanup. A value of zero indicates that the DTI will be automatically destroyed, once it has entered the “Done” state, at the discretion of the implementation: The client MUST NOT expect its existence after the “Done” state has been reached.
A DMI implementation MAY enforce a policy of a maximum stay alive time. The details of such a situation arising are out of scope of this specification, but they MAY change the semantics of this value. If this is the case the DMI implementation MUST communicate the enactment of such policy using means that are out of scope of this specification.

/dmi:TransferRequirements/dmi:MaxAttempts

This represents the [MaxAttempts] information element. It is used to specify the number of times a data movement operation (transfer) is made to move the data before a DTI enters the “Failed” state. A DMI implementation MUST have a default value of 1 for the MaxAttempts in case of a failure. If the [endNoLaterThan] is reached before the number of Attempts has reached [MaxAttempts] then the service MUST NOT continue with the retries and MUST enter the “Failed” state. When a transfer is failed, the Attempts counter in the Instance Attributes of a DTI is incremented. When the number of Attempts is equal to maxAttempts attribute of the /dmi:TransferRequirements then the state is changed to one of the “Failed” states.

/dmi:TransferRequirements/xs:any

This represents an extensibility element for the [transfer requirements]. DMI Clients MAY add elements of other namespaces to the [transfer requirements] when requesting a data transfer. A DTF implementation MUST return an "UnsatisfiableRequestOptionsFault" as specified in section 4.3.1 if it does not recognize or support the extension elements in the [transfer requirements] element.

4.2.2 GetFactoryAttributesDocument

This operation is used to retrieve the current configuration of a DTF. Invoking this operation will, on success, return an XML document containing the current configuration.

The following information elements further describe the return value of the “GetFactoryAttributesDocument”. The operation has no input parameters.

4.2.2.1 [factory attributes]

The [factory attributes] information element describes the document returned by the “GetFactoryAttributesDocument” operation.

4.2.2.1.1 XML Representation

When rendered in XML, the [factory attributes] information element is described as follows:

```
<dmi:FactoryAttributes>
    <dmi:SupportedProtocol name=”xs:anyURI”>*
        <dmi:UndoStrategy name=”xs:anyURI”/>
    </dmi:SupportedProtocol>
    <xs:any namespace=”##other”/>*
</dmi:FactoryAttributes>
```

The following describes the attributes and elements in the Pseudo Schema given above that have not already been defined elsewhere in this specification:

/dmi:FactoryAttributes

This represents the [factory attributes] information element. It is a complex type listing the protocols supported by the DTF.

4.3 Faults

If the Data Transfer Factory is unable to create a Data Transfer Instance, then it MUST throw a fault indicating the error condition.

4.3.1 UnsatisfiableRequestOptionsFault

This fault describes the situation when the options, as resolved from [transfer options], do not allow successful creation of a Data Transfer. A DTF MAY give additional information in the extensibility element of this fault.

4.3.2 NoSourceSinkProtocolMatchFault

This fault describes the situation when the attempt to find a common data transfer protocol between the source and sink DEPRs, and a protocol supported by the DMI has failed. A possible cause may be mismatching [source protocol] entries at the source and [sink protocol] entries at the sink; or different implementations of the same data transfer protocol that renders an interoperable use of that transfer protocol impossible.

4.3.3 NoDataLocationsSpecifiedInEprFault

The [sourceDEPR] or [sinkDEPR] while correctly formatted XML do not contain the required elements in the metadata section to be processed by the DTF.

4.3.4 CustomFault

This fault element MAY be used to describe any other situations that are unrelated to, or specific to, the implementation of the DTF.

5. Data Transfer Instance Port Type

This section defines the Data Transfer Instance (DTI) port type. The purpose of the DTI port type is to monitor and control a data transfer between the source and the sink. Users can use an EPR to a DTI instance to query the status and progress of the underlying data transfer.

The DTI exports a set of properties describing its current state and configuration. The set of properties is not limited to the attributes defined in this specification. Implementers may choose to add attributes as necessary. However, the obligatory attributes defined in this document MUST be supported by any implementation. The abstract interface definition is as follows:

Data Transfer Instance
[start time] [state] [completion time]? [total data size]? [bytes transferred]? [attempts]

Start()
Activate()
Stop()
Resume()
Suspend()
[state] GetState()
[instance attributes] GetInstanceAttributesDocument()

Figure 3: Functional Data Transfer Instance Interface

The details of the Data Transfer Instance are described in the following sections.

5.1 Properties

The Data Transfer Instance exports the following properties. They are all informational properties, i.e. they can be accessed, but not modified. All properties are dynamic, in the sense that their respective value may change during the lifetime of the Data Transfer Instance.

5.1.1 [start time]

When the DTI is in the Created state the [start time] information element describes the point in time and the time zone that the transfer is due to start. When the DTI is not in the Created state the [start time] information element describes the point in time and time zone that marks the start of the current attempt for the data transfer, i.e. when the DTI initiates the transfer through the selected protocol. Implementations MUST support the [start time] attribute.

An implementation of this specification MUST guarantee that the [start time] will be observed. If the data transfer does not start at the scheduled start time then it MUST enter a FAILED state.

If scheduling information is not present, or available, for the current DTI, then the value of the [start time] element is null until the transfer attempt starts – by moving into the Scheduled state. In such circumstances the data transfer MUST be started manually through the 'Start()' operation.

The data type of the [start time] element is a composite of a Gregorian calendar date and a time and time zone on that day.

5.1.1.1 XML Representation

When rendered in XML, the [scheduled start time] information element is described as follows:

```
<dmi:StartTime xsi:nillable=”true”>
    xs:dateTime
</dmi:StartTime>
```

The following describes the attributes and elements in the Pseudo Schema given above:

/dmi:StartTime

This represents the [start time] information element. Its XML type is defined as xs:dateTime. The cardinality of the XML element is 1, i.e. it MUST be present at all times. If no meaningful value is available, i.e. no time has been specified for the Data Transfer to start then this MUST be reflected using the attribute and value `xsi:nil=”true”` as defined in [THOMPSON].

5.1.2 [state]

The [state] information element describes information about the state of a DTI at the time when it is queried for its state. It MUST be supported by all implementations of this port type.

The data type of the [state] information element is complex, i.e. a composition of attributes and child elements.

5.1.2.1 XML Representation

When rendered in XML, the [state] information element is described as follows:

```
<dmi:State value=”xs:string”>
    <dmi:Detail>
        xs:any*
    </dmi:Detail>?
</dmi:State>
```

The following describes the attributes and elements in the Pseudo Schema given above:

/dmi:State
This represents the [state] information element. Its XML type is defined as xs:complexType.

/dmi:State/@value
This represents the [state value] information element as described further below.

/dmi:State/dmi:Detail
This represents a container for any additional information a DMI implementation MAY want to provide to its clients. This may include non-normative, non-interoperable pieces of information, but also normative information elements as defined below, which MUST be understood by DMI clients.

/dmi:State/dmi:Detail/xs:any
This represents the extensibility point for additional information a DMI implementation may want to provide to clients.

5.1.3 [state value]

The [state value] information element describes the current state of a DTI, encoded in a character string.

The data type of the [state value] information element is xs:string. Value comparison MUST be done character by character.

The following values are defined for the [state value] information element, representing the possible states as described in section 5.4.

Created
This identifier describes the state “Created” as described in section 5.4.1.1.

Scheduled
This identifier describes the state “Scheduled” as described in section 5.4.1.2.

Transferring
This identifier describes the state “Transferring” as described in section 5.4.1.3.

Done
This identifier describes the state “Done” as described in section 5.4.1.4.

Suspended
This identifier describes the state “Suspended” as described in section 5.4.1.5.

Failed
This identifier describes the state “Failed” as described in section 5.4.1.6

Failed:Clean
This identifier describes the state “Failed:Clean” as described in section 5.4.1.7.

Failed:Unclean
This identifier describes the state “Failed:Unclean” as described in section 5.4.1.8.

Failed:Unknown
This identifier describes the state “Failed:Unknown” as described in section 5.4.1.9.

5.1.4 [completion time]

The [completion time] information element describes a point in time, while the DTI is not in the Done or one of the Failed states, that marks the projected time at which the Data Transfer is expected to finish, i.e. when the underlying transfer protocol has reported completion of the data transfer. Once the DTI has entered the Done or one of the Failed states the [completion time] records the time at which it entered that state.

Support for the [completion time] attribute is OPTIONAL for all implementations of this interface.

The [completion time] is only of informative nature. That is, if an implementation supports this attribute, then the implementation is not obliged whatsoever to guarantee the completion of the underlying Data Transfer at the given date and time. An implementation MAY change the value of this attribute at any given time until the Data Transfer is Done or has Failed.

The data type of the [completion time] element is a composite of a Gregorian calendar date and a time on that day.

5.1.4.1 XML Representation

When rendered in XML, the [estimated completion time] information element is described as follows:

```
<dmi:CompletionTime>
    xs:dateTime
</dmi:CompletionTime>
```

The following describes the attributes and elements in the Pseudo Schema given above:

/dmi:CompletionTime
This represents the [estimated completion time] information element. Its XML type is defined as xs:dateTime.

5.1.5 [total data size]

The [total data size] information element is an OPTIONAL attribute that records in bytes the total size of the data transfer that is to be transferred (excluding any overhead from the data transfer

itself, e.g. control traffic or data retransmission)

5.1.5.1 XML Representation

When rendered in XML, the [total data size] information element is described as follows:

```
<dmi:TotalDataSize> xs:unsignedLong </dmi:TotalDataSize>
```

The following describes the attributes and elements in the Pseudo Schema given above:

/dmi:TotalDataSize
This represents the [total data size] information element. Its XML type is defined as xs:unsignedLong.

5.1.6 [bytes transferred]

The [bytes transferred] information element records how many bytes have been transferred to the sink as part of the data transfer activity represented by the DTI. Support for the [bytes transferred] attribute is OPTIONAL for all implementations of this interface. No requirements are imposed on the underlying data transfer and the implementation as to how frequently this figure should be updated. It is therefore acceptable for implementations to provide values that are updated at intervals during the transfer and are therefore implicitly 'inaccurate' at times between these updates.

5.1.6.1 XML Representation

When rendered in XML, the [bytes transferred] information element is described as follows:

```
<dmi:BytesTransferred> xs:unsignedLong </dmi:BytesTransferred>
```

The following describes the attributes and elements in the Pseudo Schema given above:

/dmi:BytesTransferred
This represents the [bytes transferred] information element. Its XML type is defined as xs:unsignedLong.

5.1.7 [attempts]

The [attempts] information element records the number of times a transfer has been attempted. Therefore, once the DTI enters the Transferring state for the first time this value is set to 1. If the transfer attempt fails, a new attempt will be made to transfer the data unless the number of [attempts] equals the value of [max attempts] (default 1) that is set in the TransferRequirements element when initializing the DTI through the factory operation. Support for the [attempts] is MUST and should be present all the time.

5.1.7.1 XML Representation

When rendered in XML the [attempts] information element is described as follows:

```
<dmi:Attempts> xs:unsignedInt </dmi:Attempts>
```

5.2 Operations

5.2.1 Start

This operation is used to manually start the data transfer represented by the DTI. The DTI will move to the “Scheduled” state from the “Created” state on successful completion of this operation. If the DTI is not in the “Created” state then this operation will generate a fault.

5.2.2 Activate

This operation is used to manually initiate a transfer represented by the DTI that is currently in the “Scheduled” state to the “Transferring” state on successful completion of this operation. If the DTI is not in the “Scheduled” state then this operation will generate a fault.

5.2.3 Stop

On invoking this operation the DTI will, if in the “Transferring” or “Suspended” state, move into the “Failed” state by attempting to stop the underlying data transfer. Depending on the success of terminating the data transfer the DTI will move into one of the qualifiers – “Clean”, “Unclean” or “Unknown”. If the DTI is not in the “Transferring” or “Suspended” state then the operation will generate a fault.

5.2.4 Resume

On invoking this operation the DTI will, if in the “Suspended” state, attempt to move to the “Transferring” state. If the DTI is not in the “Suspended” state then this operation will generate a fault.

5.2.5 Suspend

On invoking this operation the DTI will if in the “Transferring” state attempt to move to the “Suspended” state. If the DTI is not in the “Transferring” state then this operation will generate a fault.

5.2.6 GetState

This operation returns the current state of the DTI. Invoking this operation will, on success, return an XML document containing the current state.

The following information element describes the return value of the “GetState” operation. The operation accepts no input parameters.

5.2.6.1 [state]

The [state] information element describes the document returned by the “GetState” operation. The XML representation and semantics of this element have been previously defined in Section 5.1.2.

5.2.7 GetInstanceAttributesDocument

This operation returns the current configuration of the DTI. Invoking this operation will, on success, return an XML document containing the current configuration.

The following information elements further describe the return value of the “GetInstanceAttributesDocument”. The operation accepts no input parameters.

5.2.7.1 [instance attributes]

The [instance attributes] information element describes the document returned by the "GetInstanceAttributesDocument" operation.

5.2.7.1.1 XML Representation

When rendered in XML, the [instance attributes] information element is described as follows:

```
<dmi:InstanceAttributes>
  <xs:any namespace="##other"/>+
  <dmi:ScheduledStartTime/>
  <dmi:State/>
  <dmi:EstimatedCompletionTime/>?
  <dmi:TotalDataSize/>?
  <dmi:BytesTransferred/>?
  <dmi:Attempts/>
</dmi:InstanceAttributes>
```

The following describes the attributes and elements in the Pseudo Schema given above that have not already been defined elsewhere in this specification:

/dmi:InstanceAttributes

This represents the [instance attributes] information element. It is a complex type encapsulating XML elements defined elsewhere for the DTI.

5.3 Faults

The state change operations (5.2.1 to 5.2.5) MUST, if an error occurs, throw a fault indicating the error condition.

5.3.1 IncorrectStateFault

This fault indicates that the DTI is not in the correct state for the requested operation. For instance, this fault would be generated if the "Start" operation is called and the DTI is not in the "Created" state.

5.3.2 FailedStateTransitionFault

This fault indicates that, although the DTI was in the correct state to initiate the state transition operation, the DTI failed to reach the destination state. For instance, this fault would be generated if the "Suspend" operation was called while in the "Transferring" state but the DTI did not reach the "Suspended" state.

5.3.3 TransferProtocolNotInstantiatableFault

This fault indicates that the DTI failed to instantiate the underlying data transfer protocol. This fault type, if occurring, MUST be conveyed in the dmi:Detail element of the dmi:State attribute as defined in section 5.1.2, and the DTI MUST assume the state "Failed", and initiate any defined undo strategy, before entering one of its defined qualified sub-states as defined in sections 5.4.1.7, 5.4.1.8 and 5.4.1.9.

The data type of the TransferProtocolNotInstantiatableFault element is complex, i.e. it does not define or allow content other than child information elements.

5.3.3.1 XML Representation

When rendered in XML, the TransferProtocolNotInstantiatableFault is described as follows:

```
<dmi:TransferProtocolNotInstantiatableFault>
    <Protocol name=”xs:anyURI”/>
    <Timestamp>xs:dateTime</Timestamp>
    <dmi:SourceDataEPR>
        wsa:EndpointReferenceType
    </dmi:SourceDataEPR/>?
    <dmi:SinkDataEPR>
        wsa:EndpointReferenceType
    </dmi:SinkDataEPR/>?
    <xs:any namespace=”##other/>*
</dmi:TransferProtocolNotInstantiatableFault>
```

The following describes the attributes and elements in the Pseudo Schema given above:

/dmi:TransferProtocolNotInstantiatableFault
This represents the TransferProtocolNotInstantiatableFault. Its XML type is defined as xs:complexType.

/dmi:TransferProtocolNotInstantiatableFault/Protocol
This MANDATORY element represents information about the underlying data transfer protocol that could not be instantiated. This XML element is strictly local to the dmi:TransferProtocolNotInstantiatableFault element and need no further namespace qualification. Its XML type is defined as xs:complexType.

/dmi:TransferProtocolNotInstantiatableFault/Protocol/@name
This MANDATORY element represents the normative identifier for the underlying data transfer protocol. Its value MUST be one of the values that the DTF advertises as one of the supported transfer protocols as defined in section 4.1.1. The value of this field MAY be one of the normative values as defined in 4.1.1. Its XML type is defined as xs:anyURI.

/dmi:TransferProtocolNotInstantiatableFault/Timestamp
This MANDATORY element represents the date and time when the DTI attempted to instantiate the underlying data transfer protocol. Its XML type is defined as xs:dateTime. If the value does not specify a time zone, then the date and time specified MUST be assumed to be given for the UTC time zone. This element is strictly local to the dmi:TransferProtocolNotInstantiatableFault element and need no further namespace qualification.

/dmi:TransferProtocolNotInstantiatableFault/dmi:SourceDataEPR
This OPTIONAL element represents a DataEPR as defined in section 3.2.1. If present, it constitutes a DataEPR for the Source of the data transfer, amended by the DMI implementation to reflect the current fault condition, e.g. by having the faulty data transfer protocol removed from the DataEPR’s dmi:Protocols section. A DMI implementation MAY provide a dmi:SourceDataEPR for the convenience of its clients. DMI implementations MUST NOT expect clients to use this information, if provided.

/dmi:TransferProtocolNotInstantiatableFault/dmi:SinkDataEPR
This OPTIONAL element represents a DataEPR as defined in section 3.2.1. If present, it constitutes a DataEPR for the Sink of the data transfer, amended by the DMI implementation to reflect the current fault condition, e.g. by having the faulty data transfer protocol removed from the DataEPR’s dmi:Data section. A DMI implementation MAY provide a dmi:SinkDataEPR for the convenience of its clients. DMI implementations MUST

NOT expect clients to use this information, if provided.

/dmi:TransferProtocolNotInstantiatableFault/xs:any
This OPTIONAL element represents the extensibility point for additional information a DMI implementation may want to provide to clients.

5.3.4 RequestedStateNotSupportedFault

This fault indicates that a state transition has been requested of the DTI that cannot be supported by the underlying data transfer protocol. The requested state transition does not take place. For instance, if a “Suspend” operation is called while in the “Transferring” state and the underlying data transfer protocol could not be suspended, a “RequestedStateNotSupportedFault” would be generated.

5.4 Lifecycle

The lifecycle of the DTI reflects the overall progress of the underlying data transfer activity, starting from its creation to its completion. The transition between states can be initiated manually through the operations described in section 5.2 or may be initiated by the DTI as the transfer progresses. When the state changes, either manually or automatically, then events are emitted. The event mechanisms will not be defined in this version of the specification.

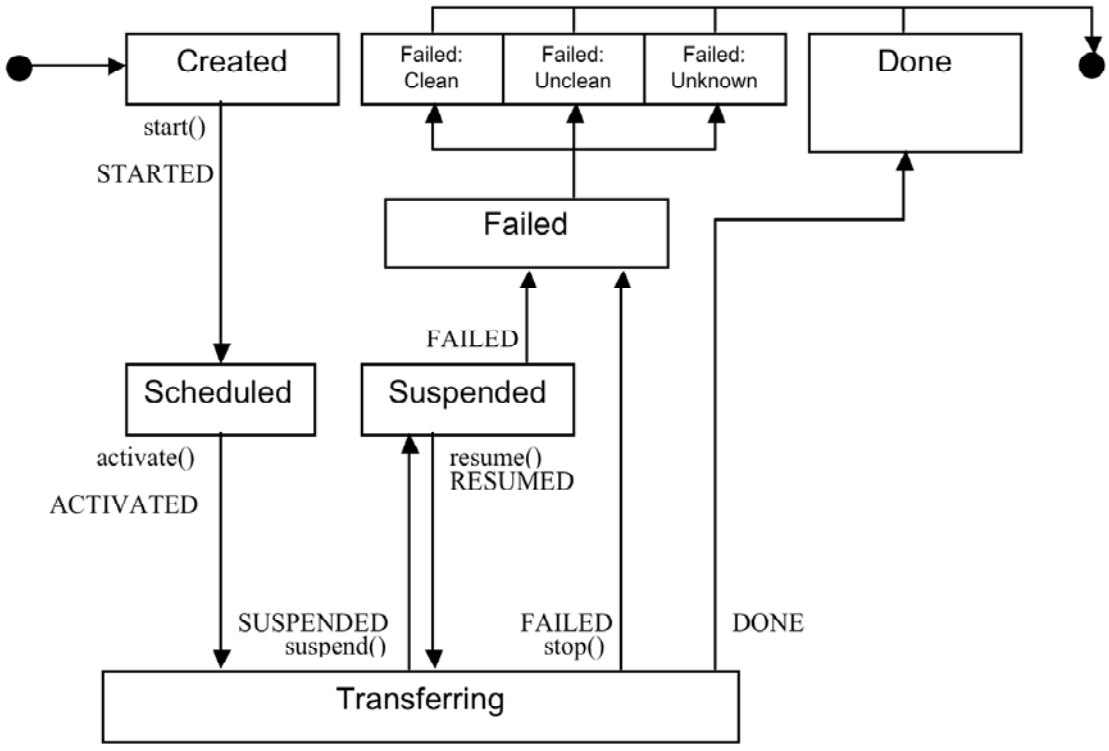

Figure 4: DTI state diagram showing "operations()" and "EVENTS"

Figure 4 outlines the DTI states with the allowed transitions, the operations that can manually initiate state changes, and the events that are generated when these state changes take place.

5.4.1 Lifecycle States

This section defines the semantics of the states as outlined in Figure 4.

5.4.1.1 Created

The “Created” state describes the configuration of the DTI after it has been created. The DTI will remain in this state until it reaches the defined ‘ScheduledStartTime’. If the ‘ScheduledStartTime’ element is null then the DTI will immediately move into the ‘Scheduled’ state once it enters the ‘Created’ state.

5.4.1.2 Scheduled

In the “Scheduled” state the DTI is ready to use the underlying data transfer services compliant with the specified requirements and will, if required, use any previously negotiated resources to make the data transfer.

5.4.1.3 Transferring

A DTI is in the “Transferring” state if the data transfer, through the underlying protocols specified when the DTI was created, has been initiated and bytes are starting to be transferred.

5.4.1.4 Done

A DTI reaches the “Done” state (from the “Transferring” state) when the protocol selected to undertake the data transfer has signaled completion.

5.4.1.5 Suspended

The “Suspended” state indicates that the underlying data transfer has been halted. This may not be supported by all data transfer protocols in which case the state will not be reached when triggered by the suspend event generating a FailedStateTransition fault.

5.4.1.6 Failed

The “Failed” state indicates that non-recoverable problems occurred in the underlying data transfer so that it cannot be completed successfully. This state indicates that the problem has been detected; the planned [undo strategy] has either not been invoked yet, or has not completed yet. A DTI may transition to the “Failed” state any time while it is in the “Transferring” or “Suspended” state.

5.4.1.7 Failed:Clean

The “Failed:Clean” state indicates that non-recoverable problems occurred in the underlying data transfer and that the planned [undo strategy] has completed successfully by cleaning up any traces of the failed data transfer attempt. A DTI may transition to the “Failed:Clean” state anytime while it is in the “Failed” state.

5.4.1.8 Failed:Unclean

The “Failed:Unclean” state indicates that non-recoverable problems occurred in the underlying data transfer and that the planned [undo strategy] completed but did not clean up all traces of the failed data transfer attempt. A DTI may transition to the “Failed:Unclean” state anytime while it is in the “Failed” state.

5.4.1.9 Failed:Unknown

The “Failed:Unknown” state indicates that non-recoverable problems occurred in the underlying data transfer and that the planned [undo strategy] completed without being able to assess whether all, none or some of the traces of the failed data transfer attempt have been cleaned up. A DTI may transition to the “Failed:Unknown” state anytime while it is in the “Failed” state.

5.4.2 Lifecycle Events

This section describes the lifecycle events that may occur during the lifecycle of a DTI completing the design of the DTI. In this version of the specification no mechanism is defined how these events may be delivered to interested clients as this is out of scope for this specification. Renderings for this specification, e.g. the WS-I rendering or the WSRF rendering, or alternative project-specific renderings MAY define or compose in event notification mechanisms, such as WS-Notification [GRAHAM] or WS-Eventing [BOX].

5.4.2.1 Started

The ‘Started’ event is generated when the DTI moves from the “Created” to the “Scheduled” state either by the “start” operation being invoked or if the time specified in the “ScheduledStartTime” is reached.

5.4.2.2 Activated

The “Activated” event is generated when the DTI starts the data transfer using the protocol and service instances (and any resource reservations) selected earlier.

5.4.2.3 Done

The “Done” event is generated when the DTI has completed the data transfer.

5.4.2.4 Suspended

The “Suspended” event is generated when the DTI attempts to move from a “Transferring” state to a “Suspend” state. This requires that the underlying data transfer protocols are able to halt and resume the data transfer on demand. If this capability is not available in the underlying protocol, the DTI will remain in the “Transferring” state. If the capability is available, and is successfully applied, then the DTI enters the “Suspended” state. If the capability is available and the protocols fail to halt the transfer then the DTI enters the “Failed” state.

5.4.2.5 Resumed

The “Resumed” event is generated when the DTI attempts to move from the “Suspended” to the “Transferring” state. If the data transfer fails to be resumed the DTI enters the “Failed” state.

5.4.2.6 Failed

The “Failed” event is generated when the DTI moves to the “Failed” state through some abnormal termination of the data transfer.

6. Security Considerations

It is of paramount importance that the transfers created and mediated by a DMI protect the data being transferred from being viewed or modified (i.e. data integrity, privacy and confidentiality) in unauthorized ways. As of the writing of this document, security considerations in a grid environment are still under investigation and being defined. Thus this document cannot provide

security related specifics. Instead, we discuss security related requirements here.

The following are the security related requirements that DMI must meet:

- The credentials required to access a specific data source at the source and a particular data set at the sink may be different.
- Credentials may be specific to the transfer protocol being used.
- The credentials presented at the source through the selected protocol must authorize the reading of the data being transferred from the source.
- The credentials presented at the sink, through the selected protocol, must authorize the storing of the data being transferred to the sink.
- A DMI may be configured to use the capabilities of a data transfer protocol to ensure that the data cannot be modified while in transit and that it cannot be read by any agent other than those directly involved in implementing the transfer protocol.
- The sink credentials used to transfer data to the sink must allow the sink to enforce appropriate access controls on the data once it has been successfully transferred to the sink.
- The source and sink MUST evaluate the credentials supplied to them through the selected protocol. The source and sink MUST say “no, not allowed” if the provided credentials are not understood or if they do not permit the requested transfer.
- DMI must be agnostic to the security infrastructure being used in the environment hosting DMI.

Some security infrastructures are based upon notions such as user identity and role. DMI must support these notions if the security infrastructure requires it. We conjecture that the protocol specific credentials that can be provided at either the source or the sink are sufficient to meet these needs but this will need to be confirmed until a security infrastructure has been defined. The DataEPR presented in this specification contains security credentials to access data resources. As such it contains personal data that should be protected and used only by the individual whose credentials were used to obtain the DataEPR. The DataEPR should therefore be stored securely and should not be passed in plain text over unencrypted communication channels.

The <Credential> element in the DataEPR provides an extensibility point that allows one or more authentication tokens to be inserted in order to provide access to the data resource. The supported authentication tokens will be specified through profiles, but would be expected to include:

- Username/password
- GSI proxy credential
- Activity Credential

7. Author Information

Michel Drescher
Fujitsu Laboratories of Europe, Ltd.
Hayes Park Central, Hayes End Road
Hayes, Middlesex UB4 8FE
United Kingdom

Allen Luniewski
IBM Corp.
555 Bailey Avenue
San Jose, California 95141
United States

Mario Antonioletti
EPCC
JCMB
The King's Buildings
Mayfield Road
Edinburgh EH9 3JZ

Steven Newhouse
Microsoft Corporation
One Microsoft Way
Redmond
USA
WA 98053

Ravi Madduri
Globus Project
Argonne National Laboratory
Chicago
USA
IL

8. Acknowledgements

We would like to acknowledge the contributions from James Casey, Bill Allcock, Andrew Grimshaw, Dave Snelling, Gavin Mccance and Alex Sim during the development of this specification.

9. Intellectual Property Statement

10. Disclaimer

11. Full Copyright Notice

12. References

[ALLCOCK01] Allcock, W., Bester, J., Bresnahan, J., Chervenak, A., Liming, L., and Tuecke, S. GridFTP: Protocol Extensions to FTP for the Grid. August 2001. http://www-fp.mcs.anl.gov/dsl/GridFTP-Protocol-RFC-Draft.pdf.

[BIRON] Biron, P. et al. XML Schema Part 2 (Second Edition). October 2004. http://www.w3.org/TR/xmlschema-2/

[BRADNER1] Bradner, S. Key Words for Use in RFCs to Indicate Requirement Levels, RFC 2119. March 1997.

[BRADNER2] Bradner, S. The Internet Standards Process – Revision 3, RFC 2026. October 1996. http://tools.ietf.org/html/rfc2026,

[BRAY] Bray, T., Hollander, D., Laymann, A., Tobin, R. Namespaces in XML 1.0, http://www.w3.org/TR/REC-xml-names/. W3C, 16 August 2006.

[BOX] Box, D., Cabrera, L., Critchley, C., et al. Web Services Eventing (WS-Eventing), W3C Member Submission, March 2006

[CATLETT] Catlett, C. GFD-1: Grid Forum Documents and Recommendations: Process and Requirements. Lemont, Illinois: Global Grid Forum. April 2002.

[COWAN] Cowan, J. et al. XML Information Set (Second Edition) February 2004. http://www.w3.org/TR/xml-infoset.

[FIELDING] Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P. and Berners-Lee, T. Hypertext Transfer Protocol -- HTTP/1.1. RFC 2616. June 1999. http://tools.ietf.org/html/rfc2616.

[GRAHAM] Graham, S., Hull, D., Murray, B., Web Services BaseNotification 1.3. OASIS Standard. October 2006.

[GU] Gu, Y. and Grossman, R.L.. UDT: UDP-based Data Transfer Protocol. IETF Draft. October 2007. http://www.ietf.org/internet-drafts/draft-gg-udt-02.txt.

[POSTEL] Postel, J. and Reynolds,J. File Transfer Protocol. RFC 959. October 1985. http://tools.ietf.org/html/rfc959.

[RESCORLA] Rescorla, E. Guidelines for Writing RFC Text on Security Considerations. RFC 3552. July 2003.

[THOMPSON] Thompson, H. et al. XML Schema Part 1: Structures (Second Edition). October 2004. http://www.w3.org/TR/xmlschema-1/

13. Reference XML Schema

This section defines the reference XML Schema for OGSA-DMI core components. All XML

messages that are involved in OGSA-DMI communications MUST validate against this XML Schema.

```
<?xml version="1.0" encoding="UTF-8"?>
<!--
  The OGF takes no position regarding the validity or scope of any
  intellectual property or other rights that might be claimed to
  pertain to the implementation or use of the technology described in
  this document or the extent to which any license under such rights
  might or might not be available; neither does it represent that it
  has made any effort to identify any such rights.  Copies of claims
  of rights made available for publication and any assurances of
  licenses to be made available, or the result of an attempt made to
  obtain a general license or permission for the use of such
  proprietary rights by implementers or users of this specification
  can be obtained from the OGF Secretariat.

  The OGF invites any interested party to bring to its attention any
  copyrights, patents or patent applications, or other proprietary
  rights which may cover technology that may be required to practice
  this recommendation.  Please address the information to the OGF
  Executive Director.

  This document and the information contained herein is provided on an
  "As Is" basis and the OGF disclaims all warranties, express or
  implied, including but not limited to any warranty that the use of
  the information herein will not infringe any rights or any implied
  warranties of merchantability or fitness for a particular purpose.

  Copyright (C) Open Grid Forum (2007-2008). All Rights Reserved.

  This document and translations of it may be copied and furnished to
  others, and derivative works that comment on or otherwise explain it
  or assist in its implementation may be prepared, copied, published
  and distributed, in whole or in part, without restriction of any
  kind, provided that the above copyright notice and this paragraph are
  included on all such copies and derivative works. However, this
  document itself may not be modified in any way, such as by removing
  the copyright notice or references to the OGF or other organizations,
  except as needed for the purpose of developing Grid Recommendations
  in which case the procedures for copyrights defined in the OGF
  Document process must be followed, or as required to translate it
  into languages other than English.

  The limited permissions granted above are perpetual and will not be
  revoked by the OGF or its successors or assignees.
-->
<schema targetNamespace="http://schemas.ogf.org/dmi/2008/05/dmi"
  xmlns="http://www.w3.org/2001/XMLSchema"
  xmlns:dmi="http://schemas.ogf.org/dmi/2008/05/dmi"
  xmlns:wsa="http://www.w3.org/2005/08/addressing"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  elementFormDefault="qualified">

  <import namespace="http://www.w3.org/2005/08/addressing"
    schemaLocation="http://www.w3.org/2006/03/addressing/ws-addr.xsd"/>
```

```
<!-- ======================================================= -->
<!-- ==                                                   == -->
<!-- == Data Model for OGSA DMI                           == -->
<!-- ==                                                   == -->
<!-- == This XML Schema document defines the common       == -->
<!-- == OGSA-DMI data model used to describe the          == -->
<!-- == normative interfaces for the Data Transfer        == -->
<!-- == Factory (DTF) and the Data Transfer Instance (DTI).== -->
<!-- ======================================================= -->


<!-- ============================================ -->
<!-- == Data Transfer Factory related data model == -->
<!-- ============================================ -->

<!--
  dmi:UndoStrategy
-->
<complexType name="UndoStrategyType">
  <annotation>
    <documentation>
      An undo strategy is executed to clean up traces of a failed
      data transfer attempt. Normative values for the
      dmi:UndoStrategy are defined in the OGSA-DMI Functional
      Specification.
    </documentation>
  </annotation>
  <attribute name="name" type="anyURI" use="required" />
</complexType>
<element name="UndoStrategy" type="dmi:UndoStrategyType" />

<!--
  dmi:SupportedProtocol
-->
<complexType name="SupportedProtocolType">
  <annotation>
    <documentation>
      A DTF must advertise which data transfer protocols it supports.
      Normative values for supported protocols are defined in the
      OGSA-DMI Functional Specification. For each supported protocol,
      the DTF must announce which undo strategy will be executed when
      the requested data transfer has failed.
    </documentation>
  </annotation>
  <sequence>
    <element name="UndoStrategy" type="dmi:UndoStrategyType" />
  </sequence>
  <attribute name="name" type="anyURI" use="required" />
</complexType>
<element name="SupportedProtocol" type="dmi:SupportedProtocolType" />

<!--
  dmi:Credentials
-->
<complexType name="CredentialsType">
  <annotation>
    <documentation>
```

```
        Credentials in OGSA-DMI are used to describe the security
        related information elements that are necessary to invoke a
        data transfer using a specific transport protocol. This element
        does not define any normative structure so that OGSA-DMI can be
        orthogonally composed with security infrastructures.
      </documentation>
    </annotation>
    <sequence>
      <any namespace="##other" minOccurs="1" maxOccurs="unbounded" />
    </sequence>
  </complexType>
  <element name="Credentials" type="dmi:CredentialsType" />

  <!--
    dmi:Data
  -->
  <complexType name="DataType">
    <annotation>
      <documentation>
        The dmi:Data element describes for each data transfer protocol
        (using the normalised values defined for dmi:SupportedProtocol)
        the specific information that must be used to access the data.
      </documentation>
    </annotation>
    <sequence>
      <element name="Credentials" type="dmi:CredentialsType"
        minOccurs="0" />
    </sequence>
    <attribute name="ProtocolUri" type="anyURI" use="required" />
    <attribute name="DataUrl" type="anyURI" use="required" />
  </complexType>
  <element name="Data" type="dmi:DataType" />

  <!--
    dmi:DataLocations
  -->
  <complexType name="DataLocationType">
    <annotation>
      <documentation>
        This element serves as a container aggregating one or more
        dmi:Data elements. This container item MUST appear in the
        wsa:Metadata section of the SourceDEPR and SinkDEPR as defined
        in the OGSA-DMI Functional Specification 1.0.
      </documentation>
    </annotation>
    <sequence>
      <element name="Data" type="dmi:DataType" maxOccurs="unbounded" />
    </sequence>
  </complexType>
  <element name="DataLocation" type="dmi:DataLocationType" />

  <!--
    dmi:TransferRequirements
  -->
  <complexType name="TransferRequirementsType">
    <annotation>
      <documentation>
```

```
        A client may specify a number of constraints to the data
        transfer such as start and end time (e.g. when resource
        reservations have been made).
      </documentation>
    </annotation>
    <sequence>
      <element name="StartNotBefore" type="dateTime" minOccurs="0" />
      <element name="EndNoLaterThan" type="dateTime" minOccurs="0"
        xsi:nillable="true" />
      <element name="StayAliveTime" type="unsignedInt" minOccurs="0" />
      <element name="MaxAttempts" type="unsignedInt" minOccurs="0" />
      <any namespace="##other" minOccurs="0" maxOccurs="unbounded" />
    </sequence>
  </complexType>
  <element name="TransferRequirements"
    type="dmi:TransferRequirementsType" />


  <!-- ================================================ -->
  <!-- == Data Transfer Instance related data model == -->
  <!-- ================================================ -->


  <!--
    dmi:StartTime
  -->
  <element name="StartTime" type="dateTime" xsi:nillable="true">
    <annotation>
      <documentation>
        The start time describes the point in time when an instantiated
        data transfer is due to start moving bytes over the wire.
      </documentation>
    </annotation>
  </element>

  <!--
    dmi:Detail
  -->
  <complexType name="DetailType">
    <annotation>
      <documentation>
        The StateDetail is an extension element that an implementation
        may use to provide proprietary information.
      </documentation>
    </annotation>
    <sequence>
      <any namespace="##other" minOccurs="0" maxOccurs="unbounded" />
    </sequence>
  </complexType>
  <element name="Detail" type="dmi:DetailType" />

  <!--
    dmi:StatusValue
  -->
  <simpleType name="StatusValue">
    <annotation>
      <documentation>
```

```
        This enumeration lists all possible values for the
        dmi:Status/@value attribute.
      </documentation>
    </annotation>
    <restriction base="string">
      <enumeration value="Created" />
      <enumeration value="Scheduled" />
      <enumeration value="Transferring" />
      <enumeration value="Done" />
      <enumeration value="Suspended" />
      <enumeration value="Failed" />
      <enumeration value="Failed:Clean" />
      <enumeration value="Failed:Unclean" />
      <enumeration value="Failed:Unknown" />
    </restriction>
  </simpleType>

  <!--
    dmi:State
  -->
  <complexType name="StateType">
    <annotation>
      <documentation>
        The status describes the current configuration of the Data
        Transfer Instance within its lifetime.
      </documentation>
    </annotation>
    <sequence>
      <element name="Detail" type="dmi:DetailType" minOccurs="0" />
    </sequence>
    <attribute name="value" type="dmi:StatusValue" />
  </complexType>
  <element name="State" type="dmi:StateType" />

  <!--
    dmi: CompletionTime
  -->
  <element name="CompletionTime" type="dateTime">
    <annotation>
      <documentation>
        The completion time gives the point in time when the underling
        data transfer is estimated to complete or has completed. The
        value of this element is expected to be highly volatile, and
        does not offer any level of guarantee of accuracy.
      </documentation>
    </annotation>
  </element>

  <!--
    dmi:TotalDataSize
  -->
  <element name="TotalDataSize" type="unsignedLong">
    <annotation>
      <documentation>
        This element defines the total number of bytes that the
        underlying data transfer will send over the wire in order to
        complete the data transfer.
```

```
      </documentation>
    </annotation>
  </element>

  <!--
    dmi:BytesTransferred
  -->
  <element name="BytesTransferred" type="unsignedLong">
    <annotation>
      <documentation>
        This element defines the number of bytes transferred at the
        time of request for this element. No guarantee of accuracy is
        given for the value; as it is - if supported - highly dependent
        on the underlying data transfer protocol.
      </documentation>
    </annotation>
  </element>

  <!--
    dmi:Attempts
  -->
  <element name="Attempts" type="unsignedInt">
    <annotation>
      <documentation>
        This element contains the number of attempts (including the
        current one, if th e state is "Transferring") for the requested
        data transfer. If the data transfer is scheduled to commence in
        the future, this element must have the value of "0".
      </documentation>
    </annotation>
  </element>

</schema>
```