For the strawman, my feeling is that we should do the following;

a) First propose a set of 'standalone' XSD constructs/elements (e.g. DMI-Common) and document those as a re-usable element set. These elements should all be defined in a single schema under the same namespace. DMI-Common should include all the common/reusable elements that would provide generic value, such as the BulkDataCopy, DataCopy, DataEPR, Credentials, InstanceAttributes (which is currently defined in the WS rendering schema? not sure why), Standard fault types (again, these are currently defined in the WS rendering schema), TransferRequirements, BulkDataCopyStatus? etc.
b) Secondly, document how the DMI-Common element set should be composed in order to be compatible with the DMI v1/v2 (DMIB) 'usage' profile(s). These usage profiles could either be presented as a set of separate documents, or as sub-sections in the strawman document.

In doing part 1) above, other activities/projects are then free to re-use the DMI-Common element set as required without having to support the DMI-specific 'usage' profiles. I think this would facilitate maximum re-use and the most widespread adoption (e.g. a new BES-DMI profile could then choose which elements to re-use, this would not impact the DMI profiles).

Comments on your questions:

- "xsd:any" elements in the DMI Functional specification: We agreed to put xsd:any elements in the existing schema wherever it is required, because the current specification lacks structure for extensibility and more complex requirements.
Agreed (some examples below). There are potentially some more when I start putting together the new strawman. New xsd:any as a child of 'dmi:DataType' complex type and new xsd:any as child of 'dmi:DataLocationsType' complex type.

```
<!--
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" />
  <!-- new xsd:any definition -->
  <any namespace="##other" minOccurs="0" maxOccurs="unbounded" />
</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" />
  <!-- new xsd:any definition -->
  <any namespace="##other" minOccurs="0" maxOccurs="unbounded" />
</sequence>
</complexType>

<element name="DataLocation" type="dmi:DataLocationType" />
```

- Instead of defining new DTF service, we can extend the DataTransferFactory interface by changing parameters to support DataBulkCopy elements.
Agreed: I think the DTF interface could be extended to support a single BulkDataCopy document as suggested in Figure 6 in the paper.

- We need to have DTF service properties such as SupportedProtocols
Ok.

- We don't need to record BulkDataCopy's Status property, may be TotalTranferredBytes are worthy enough.
Not sure? If we are treating the bulk copy activity as a single 'atomic' activity, then maybe we do need to report the status of the BulkDataCopy when considered as whole? At present, I think we need to support elements similar to the <BulkDataCopyStatus/> with nested <DataCopyStatus/> elements as suggested in Figure 7.

- We can keep the current DMI state model without introducing new or reusing BES state model
Not sure? I don't think DTS/dataMINX will support the 'dmi:start' and 'dmi:activate' steps since the DTS job will be scheduled and started following submission of the request. Could these states therefore be made optional?. Also, I think other profiles should be able to re-use the XSD constructs and element definitions that we are proposing for use within other profiles (such as BES). Therefore, I guess my feeling is that we should a) first propose a set of 'standalone' XSD constructs/elements and document those as a re-usable element set, and b) secondly profile how these elements can be used within the DMI v1 (and DMIB v2?) 'usage' profile(s). In doing this, other activities (e.g. BES) are free to re-use the standalone elements as required without having to support the DMI-specific 'usage' profiles. I think the definition of the standalone elements/constructs would facilitate maximum re-use (inc. BulkDataCopy doc, InstanceAttributes, Standard fault types, TransferRequirements, BulkDataCopyStatus? etc) .

- DataEPRs: can be the same as they are in the current specification
Agreed (provided we add the additional extension points). Also, we may need to introduce an optional identifier for the DataEPR if we are going to support element referencing (Figure 5 in the

paper). In summary, I believe we should support 'in-line' XML document construction that use direct element nesting as a minimum (as per the current spec). We should also then consider whether to additionally support element references in conjunction with 'in-line' definitions (if you have used the Spring framework; much like the bean element references). I'll canvas opinion on this design choice at OGF.

- Transfer Requirements
Single out transfer requirements from individual elements. In the current proposal it is referenced by all data copies elements. One could specify general transfer requirements in the Resources section which will be valid for all the individual datacopy elements, but if any of the child DataCopy elements needs to have separate transfer requirements then it must reference them explicitly from the Resources section. This model can be improved a bit to have more user convenience.
Agreed, but I think there will be some finer points to address. E.g. Transfer Requirements element defines a 'batch window' using 'dmi:StartNotBefore' and 'dmi:EndNoLaterThan' elements. Therefore, should each DataCopy element be able to override the global TransferRequirements with its own TransferRequirements definition with different StartNotBefore and EndNoLaterThan values? Maybe we need two transfer requirement elements, e.g. a mandatory <BulkCopyTransferRequirements/> and an optional <DataCopyTransferRequirements/> so that we can have finer grained control. Also, I think we could propose some more child elements for the TransferRequirements, e.g. including 'FileSelector' elements used to define a RegEx for the purposes of selecting files that match a particular name or that have a particular file extension(s).

- Credentials
They can be considered in the same manner as transfer requirements.
Agreed.

- Bulk data copy
  - no need to name have datatransferinstance (may be an optional service)
  - properties of individual transfer can be represented as instance properties
Agreed. Sounds good.

Additional / Misc:

Should the state and attempts be made optional as below.

```
<!-- Supplementary elements -->
<complexType name="InstanceAttributesType">
<sequence>
  <element ref="dmi:StartTime" minOccurs="0"/>
  <element ref="dmi:State" minOccurs="0"/>
  <element ref="dmi:CompletionTime" minOccurs="0" />
  <element ref="dmi:TotalDataSize" minOccurs="0" />
  <element ref="dmi:BytesTransferred" minOccurs="0" />
  <element ref="dmi:Attempts" minOccurs="0"/>
  <any namespace="##other" minOccurs="0" maxOccurs="unbounded" />
</sequence>
</complexType>
```