Multimedia Database Support for Digital Libraries

Seminar at Schloss Dagstuhl, 29.08.-03.09.99

Improving the Performance of Media Servers Providing Physical Data Independence—Problems, Concepts, and Challenges

VirtualMedia VirtualMedia

Ulrich Marder

www.ulrich-marder.de

Outline

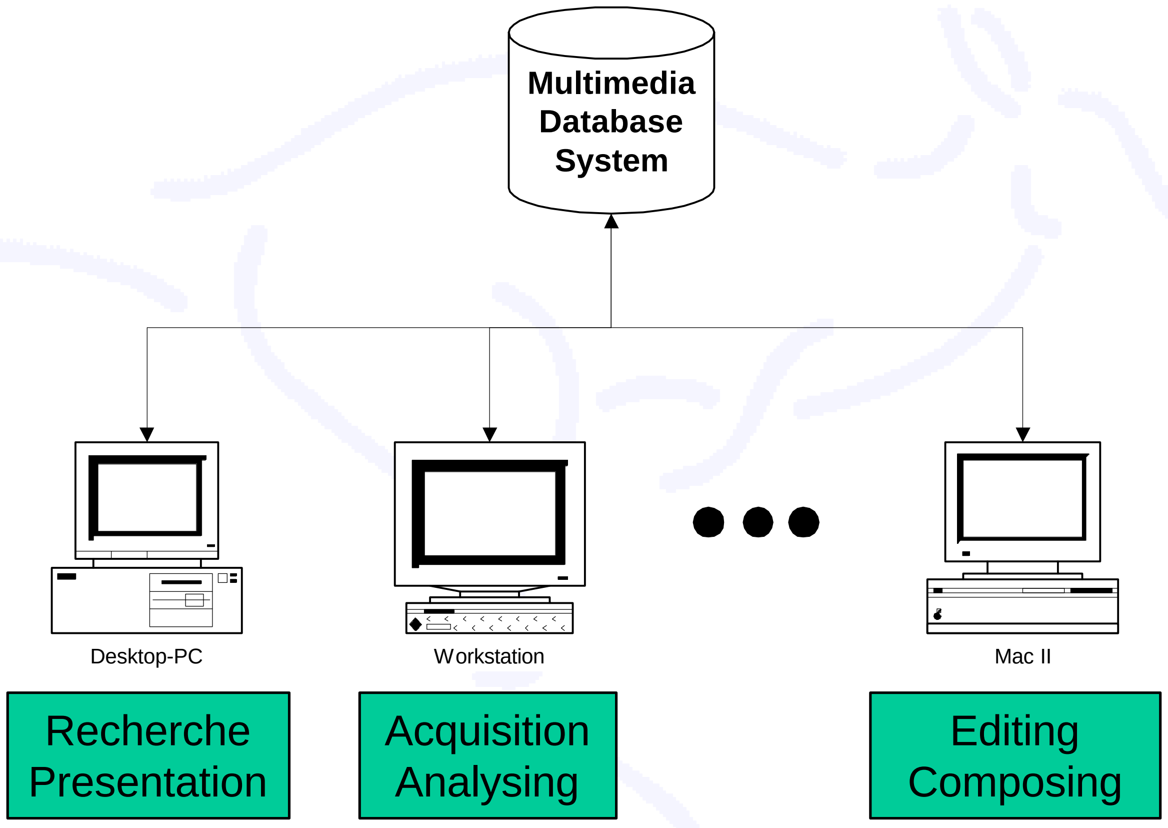
- Problems
  - Applications requiring physical data independence
  - Bad performance—the crux with unified data formats
- Concepts
  - The transformation independence abstraction
  - Interface considerations
  - Virtual media objects—Swanky quick-change artists on stage
  - Materialization—Saving the show on backstage
- Challenges
  - Solving the optimization problem(s)
  - Integration with common DBMS technology

Typical Applications

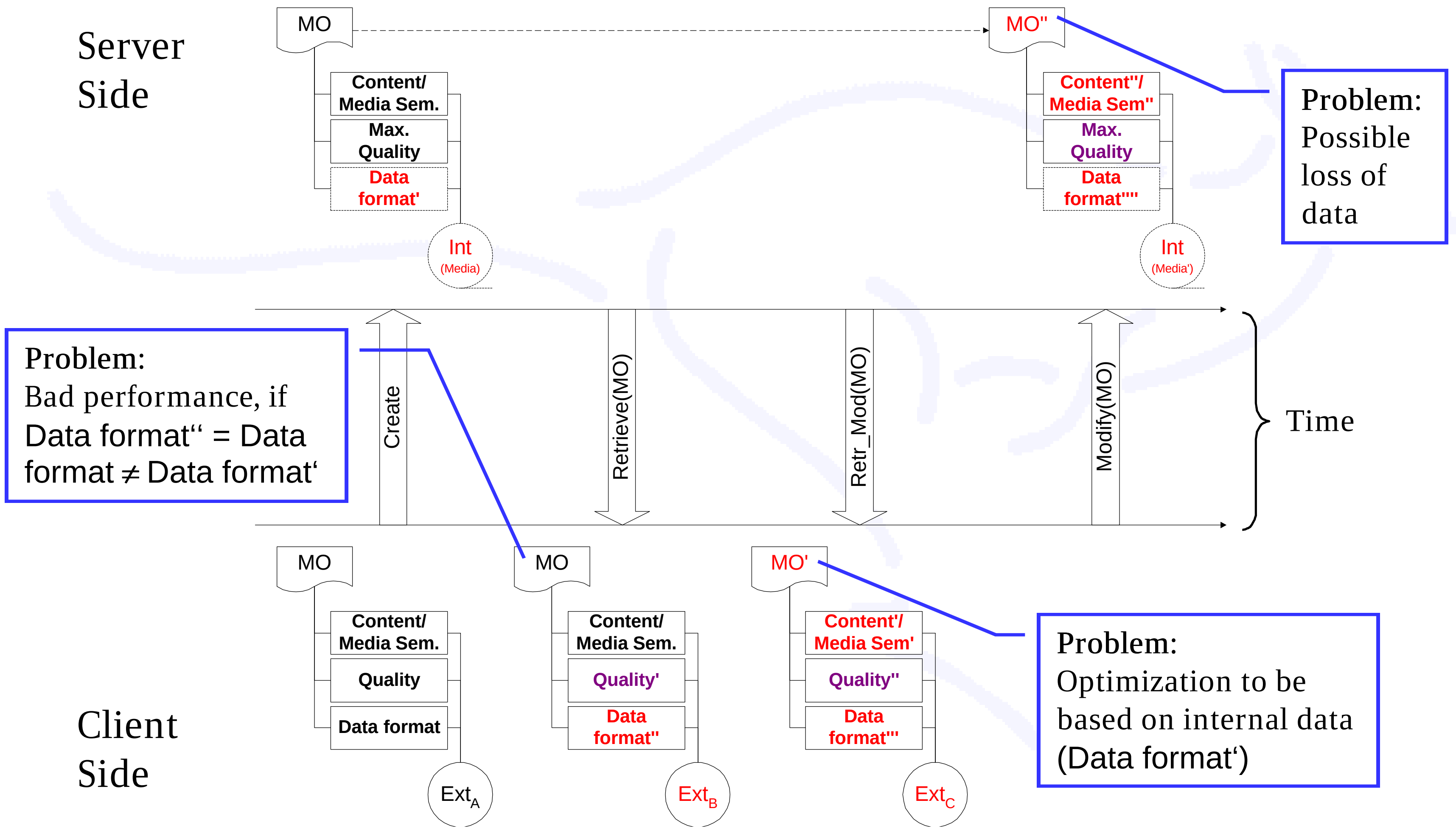
Global Media Data
(Media Assets)

Heterogeneous
Clients

Different
Applications



Problems Occuring during MO Life Cycle



Problem:
Possible loss of data

Problem:
Bad performance, if
Data format“ = Data format ≠ Data format‘

Problem:
Optimization to be based on internal data (Data format‘)

Consequences

- Allow modifications, but prevent information loss
  - Do some kind of versioning
    - Prioritize access path to older versions (originals)
- First, optimize the most common case
  - Use the most popular external data format(s) internally
    - May vary at run-time
    - May introduce redundance
- Second, make efforts to optimize the more special cases
  - Modify-operations must be expressable w/o any references to or assumptions on the internal data format
    - Dynamic optimization (at the server-side)
    - Mapping of abstract semantics to executable operations is hard and may produce imperfect results

Transformation Independence Abstraction

Freedom for Clients—Heavy Work for Servers

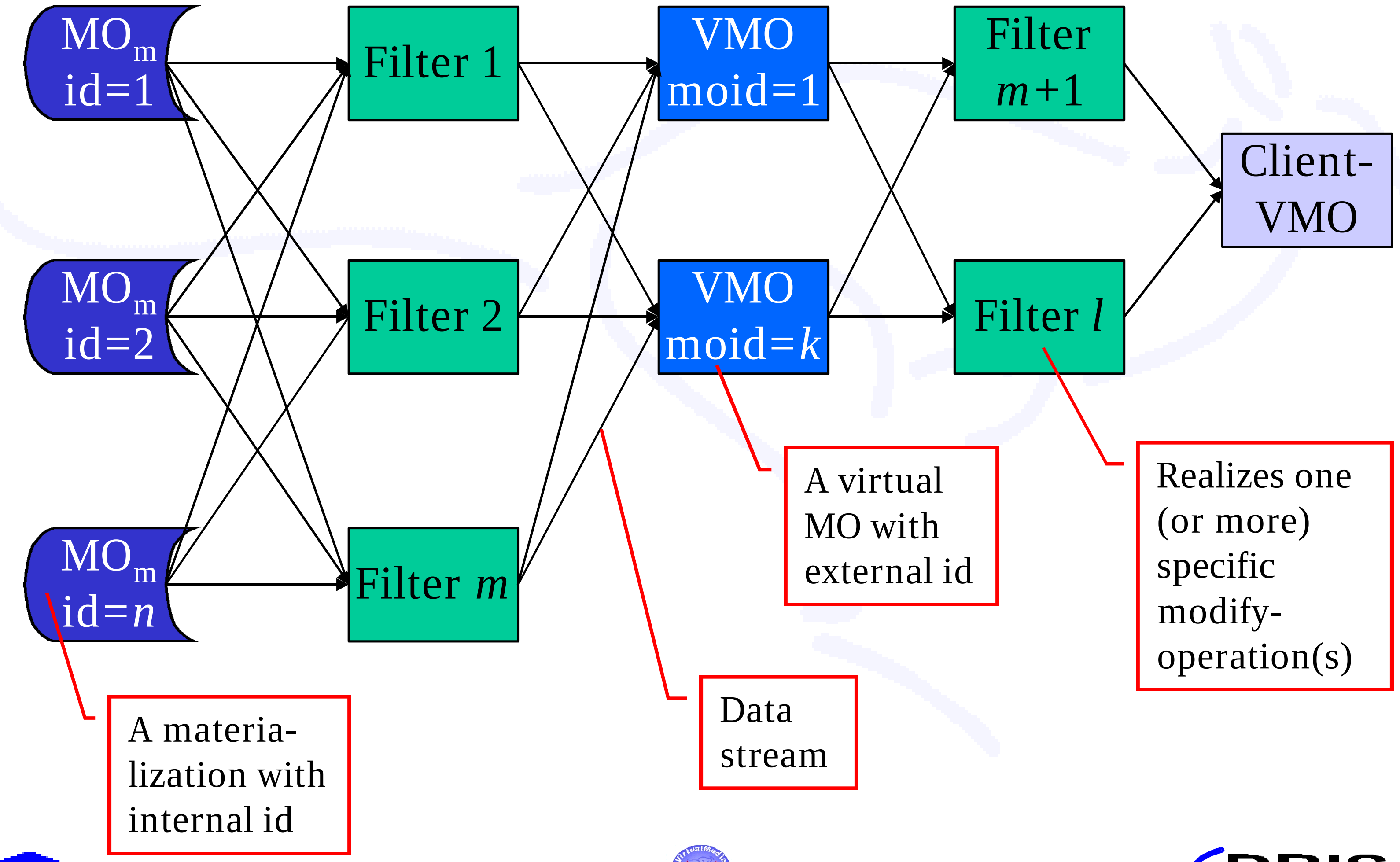
- Create, retrieve, and modify by *transformation request*
  - Describes what result the client expects
    - Data format & quality
    - Modify-operations
  - Does not prescribe an algorithm to achieve the result
    - No helper-operations
    - Only semantically significant operation sequences
  - Does not prescribe where to execute operations
  - Does not prescribe what is to be materialized
- Thus, three dimensions for *optimization*:
  - Algorithm: (Re-)ordering, substitution (of operations)
  - Place of execution: data-centric, client-centric, HW-centric,...
  - Materialization: redundant, non-redundant, client-centric,...

Interface Considerations

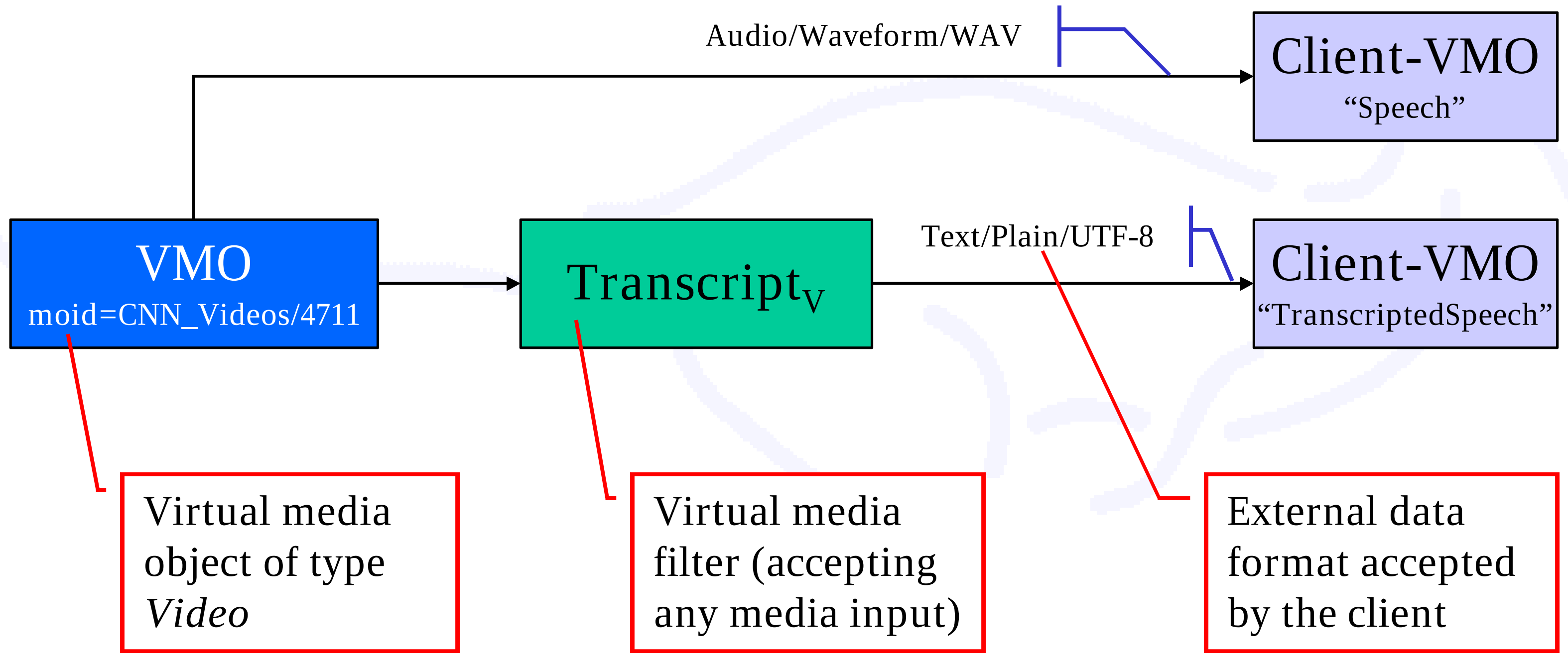
A sample transformation request (VirtualMedia descriptor)

```
<?XML version="1.0" standalone="yes"?>
<VMDESC>

 <SOURCE>
  <MOID ALIAS="BC_Video"
   REF="CNN_Videos/4711">
  </MOID>
 </SOURCE>

 <VIRTUAL NAME="TranscriptedSpeech"
  MAINTYPE="TEXT"
  SUBTYPE="PLAIN"
  ENCODING="UTF-8">

  <TRANSFORMATION NAME="Transcription">
   <OPERATION NAME="Transcript">
    <INPUT NAME="BC_Video"/>
    <PARAM NAME="Language" VALUE="EN"/>
   </OPERATION>
  </TRANSFORMATION>

 </VIRTUAL>

 <VIRTUAL NAME="Speech"
  MAINTYPE="AUDIO"
  SUBTYPE="WAVEFORM"
  ENCODING="WAV">

  <QUALITY>
   <SAMPLING_FREQUENCY>44100</SAMPLING_FREQUENCY>
   <SAMPLE_DEPTH>16</SAMPLE_DEPTH>
   <!-- Alternatively specify
   <PROFILE>CD-AUDIO</PROFILE>
   -->
  </QUALITY>

  <TRANSFORMATION>
   <OPERATION NAME="NOP">
    <INPUT NAME="BC_Video"/>
   </OPERATION>
  </TRANSFORMATION>

 </VIRTUAL>

</VMDESC>
```

Virtual Media Objects—A schematical view



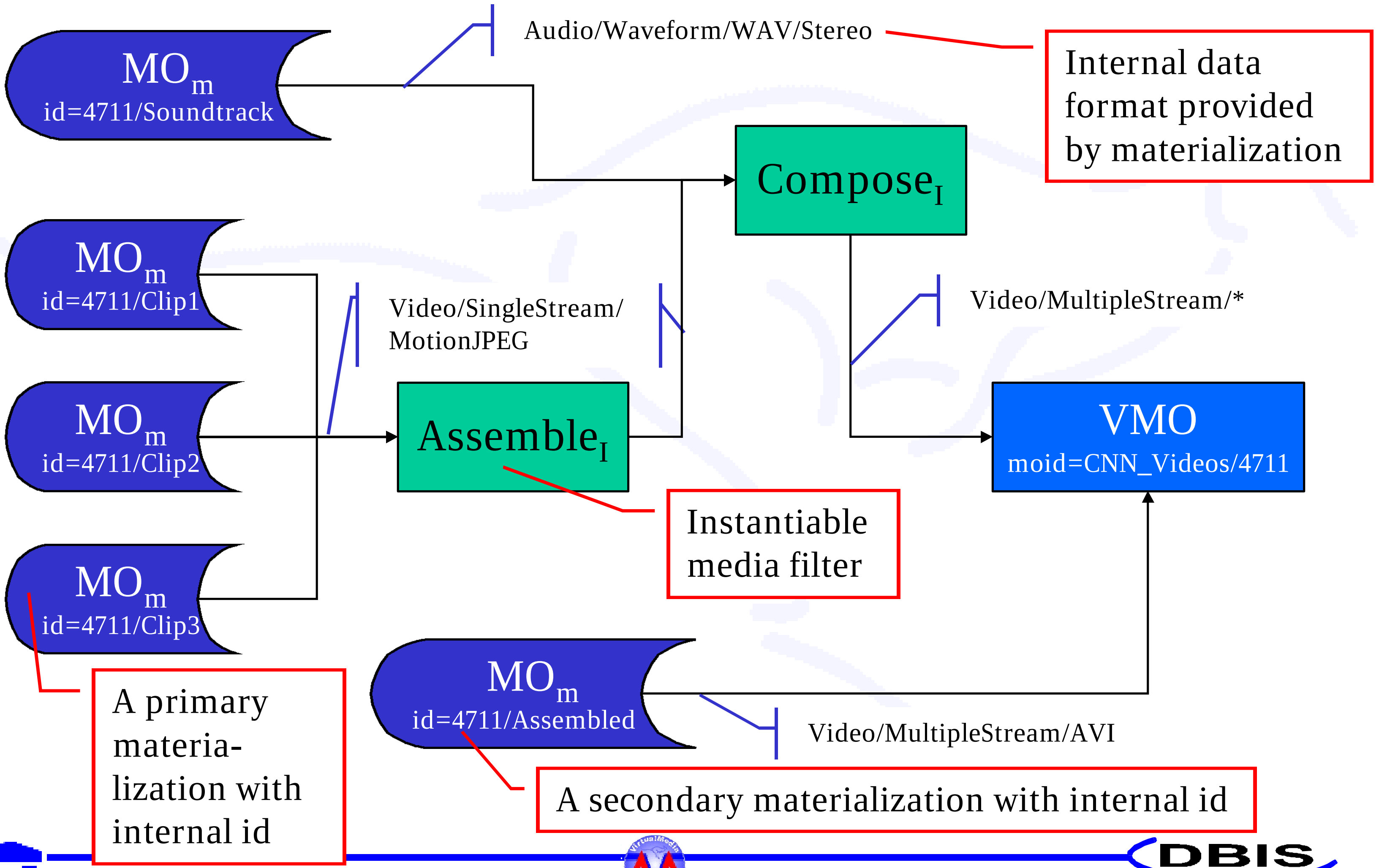
VirtualMedia Transformation Request Graph


Audio/Waveform/WAV
Client-VMO
"Speech"
VMO
moid=CNN_Videos/4711
Transcript$_V$
Text/Plain/UTF-8
Client-VMO
"TranscriptedSpeech"
Virtual media object of type *Video*
Virtual media filter (accepting any media input)
External data format accepted by the client

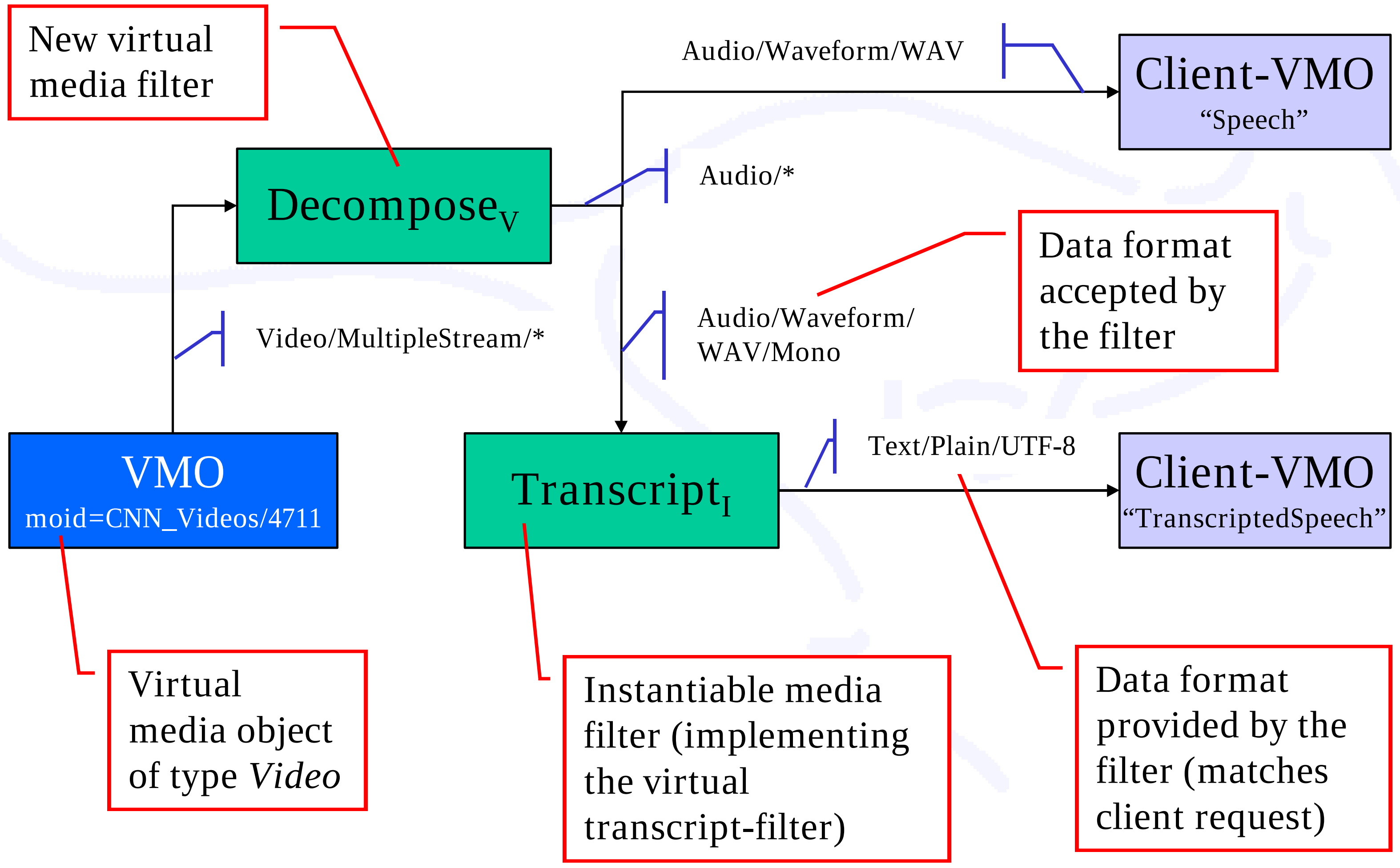
Tasks:

- Find appropriate materializations of VMOs
- Replace virtual filters with optimal, semantically correct implementation

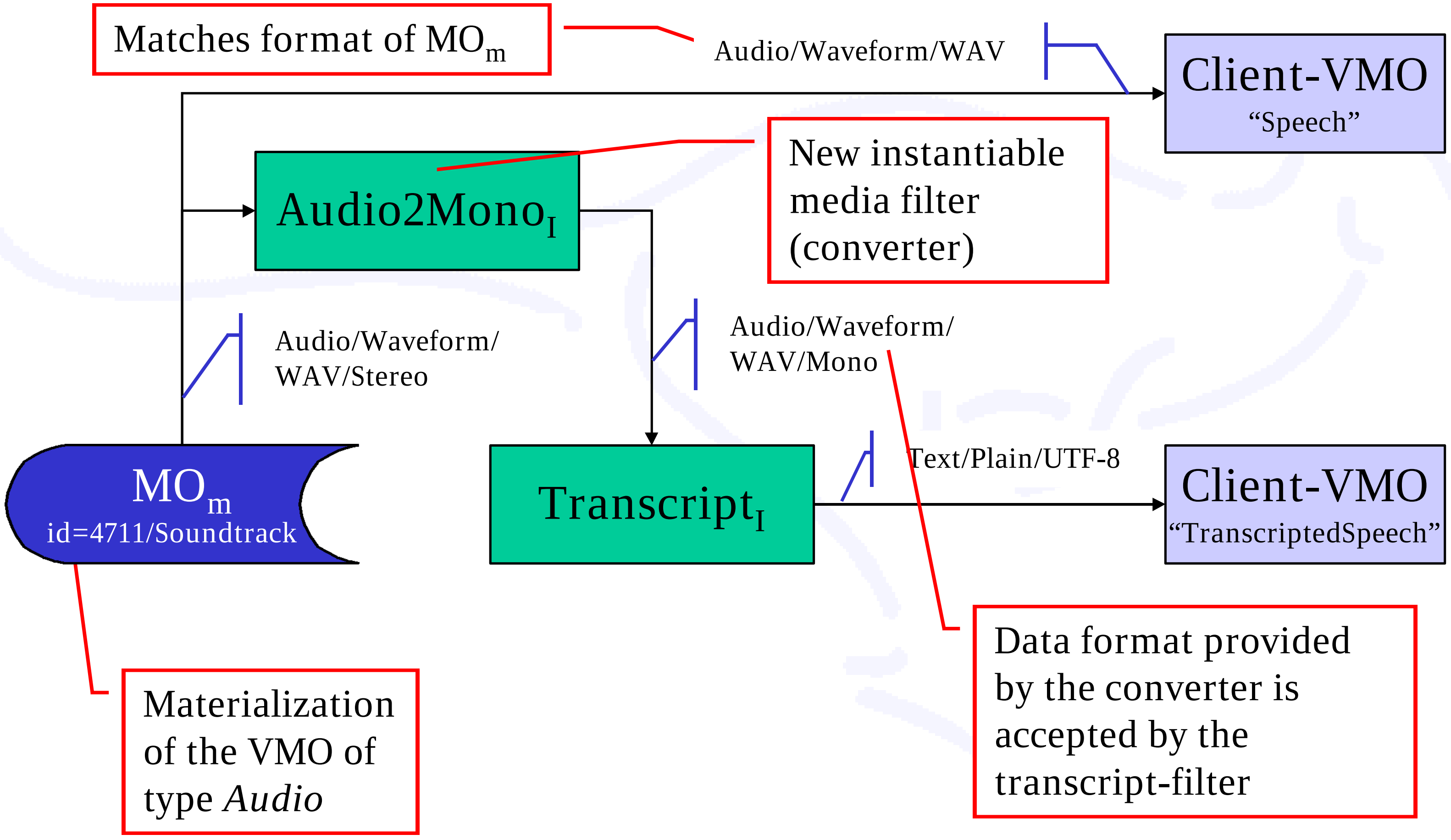
VirtualMedia Materialization Graph


MO$_m$
id=4711/Soundtrack
Audio/Waveform/WAV/Stereo
Internal data format provided by materialization
Compose$_I$
MO$_m$
id=4711/Clip1
Video/SingleStream/ MotionJPEG
Video/MultipleStream/*
MO$_m$
id=4711/Clip2
Assemble$_I$
VMO
moid=CNN_Videos/4711
Instantiable media filter
MO$_m$
id=4711/Clip3
MO$_m$
id=4711/Assembled
Video/MultipleStream/AVI
A primary materia-lization with internal id
A secondary materialization with internal id

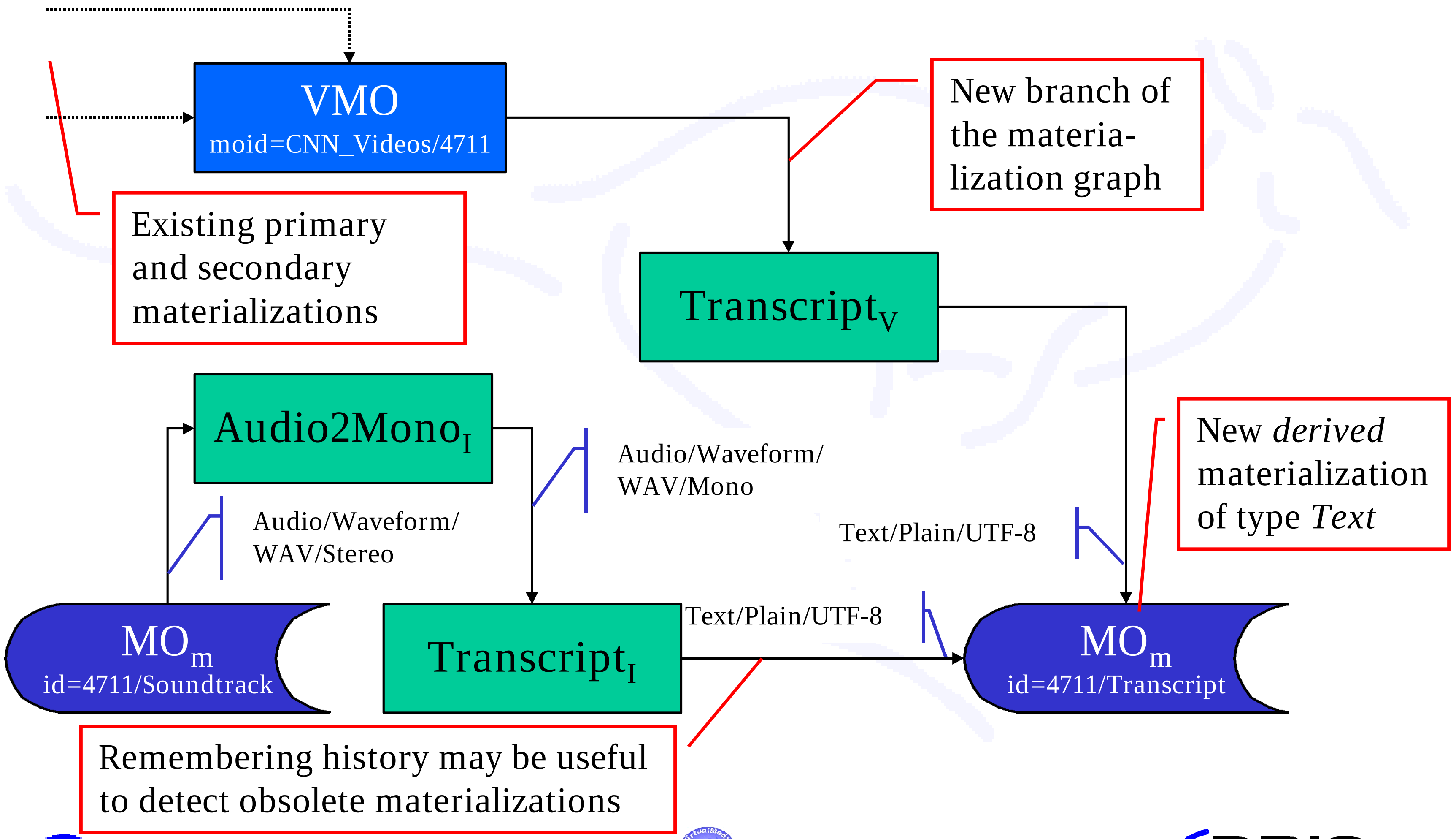
Transformation Request Resolution (1)



Transformation Request Resolution (2)


Matches format of MO_m
Audio/Waveform/WAV
Client-VMO
"Speech"
$Audio2Mono_I$
New instantiable media filter (converter)
Audio/Waveform/WAV/Stereo
Audio/Waveform/WAV/Mono
MO_m
id=4711/Soundtrack
$Transcript_I$
Text/Plain/UTF-8
Client-VMO
"TranscriptedSpeech"
Data format provided by the converter is accepted by the transcript-filter
Materialization of the VMO of type *Audio*

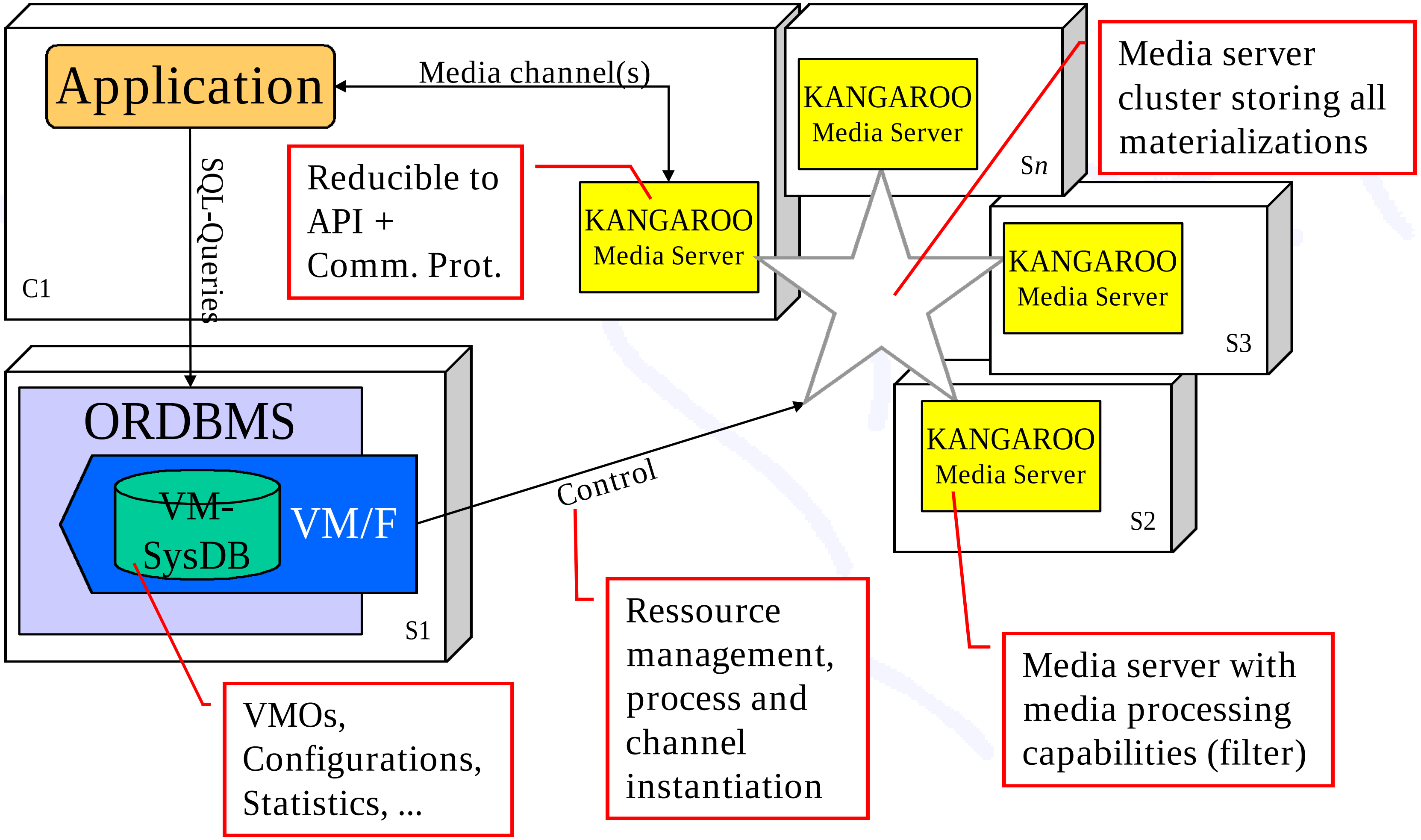
Adding a Materialization


VMO
moid=CNN_Videos/4711
Existing primary and secondary materializations
New branch of the materia-lization graph
Transcript$_V$
Audio2Mono$_I$
Audio/Waveform/ WAV/Mono
Audio/Waveform/ WAV/Stereo
Text/Plain/UTF-8
New *derived* materialization of type *Text*
MO$_m$
id=4711/Soundtrack
Transcript$_I$
Text/Plain/UTF-8
MO$_m$
id=4711/Transcript
Remembering history may be useful to detect obsolete materializations

Challenges

- Refine VM-graph-transformation algorithm
  - Approach: Rule-based algebraic optimization (well understood)
  - Floorers: Secondary and derived materializations—how to control the risk of trading quality for performance?
  - ⇘ Quality assessment of materializations (use fuzzy system?)
- Refine VirtualMedia model
  - Parametrized VMOs (templates)
  - Hierarchically structured VM-graphs (encapsulation)
- Integration with common DBMS technology
  - Exploit extensibility
  - Object-relational features facilitate tight integration
  - But: media server especially adapted to VirtualMedia required

Software Architecture Proposal


Application
Media channel(s)
SQL-Queries
Reducible to API + Comm. Prot.
KANGAROO Media Server
C1
KANGAROO Media Server
Sn
Media server cluster storing all materializations
KANGAROO Media Server
S3
KANGAROO Media Server
S2
ORDBMS
VM-SysDB
VM/F
S1
Control
VMOs, Configurations, Statistics, ...
Ressource management, process and channel instantiation
Media server with media processing capabilities (filter)

Conclusions

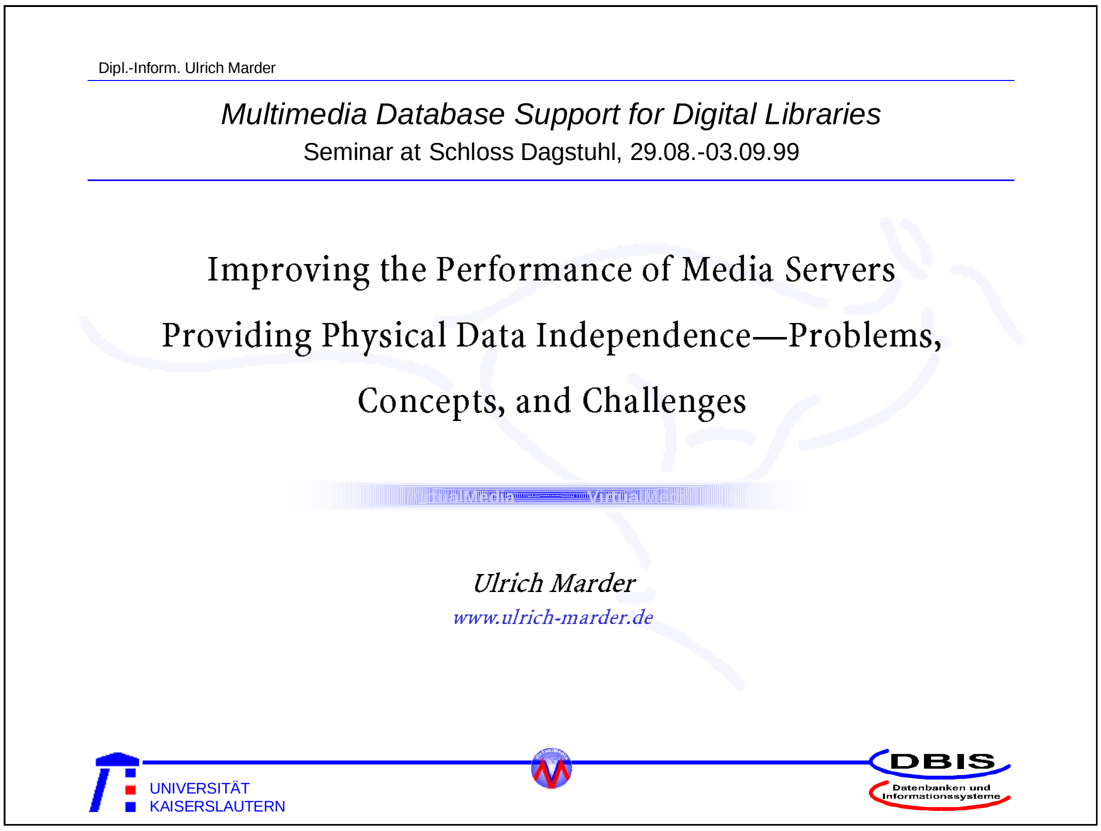
- Problems with physical data independence
  - Bad performance due to frequent format conversions
  - Loss of data due to irreversible modify-operations
  - All optimization must be based on internal data representation
- Concepts
  - Transformation independence
    - Abstract semantics at API-level
    - Multiple optimization dimensions (algorithm, materialization, ...)
  - VirtualMedia
    - Optimal matching of transformation request graphs and materialization graphs
- Challenges
  - Pretty to much to enumerate...

Additional Material

- VirtualMedia home page at
  http://pfau.informatik.uni-kl.de/virtualmedia
- SMIL-presentation based on the present slides
  - Available after the seminar (scheduled for sept.)
  - URL (a link will be provided at the home page)
    http://pfau.informatik.uni-kl.de:8090/ramgen/presentations/Dagstuhl99/real/slideshow.smi
  - RealPlayer® G2 required for playback

Appendix 1: Sample Create-request

```
<?XML version="1.0" standalone="yes"?>
<VMDESC>

 <SOURCE>
  <MOM ALIAS="Sound"
   REF="File://Client1/Media/sound.wav"
   MAINTYPE="AUDIO" SUBTYPE="WAVEFORM"
   ENCODING="WAV">
   <QUALITY>
    <SAMPLING_FREQUENCY>44100
    </SAMPLING_FREQUENCY>
    <SAMPLE_DEPTH>16</SAMPLE_DEPTH>
    <CHANNELS>Stereo</CHANNELS>
   </QUALITY>
  </MOM>
  <MOM ALIAS="Clip1"
   REF="File://Client1/Media/clip1.avi"
   MAINTYPE="VIDEO" SUBTYPE="SINGLESTREAM"
   ENCODING="AVI/MJPEG">
   <QUALITY>
    <IMAGE_WIDTH>704</IMAGE_WIDTH>
    <IMAGE_HEIGHT>480</IMAGE_HEIGHT>
    <FRAME_RATE>25</FRAME_RATE>
   </QUALITY>
  </MOM>
  <MOM ALIAS="Clip2" ...>...</MOM>
  <MOM ALIAS="Clip3" ...>...</MOM>
 </SOURCE>

 <VIRTUAL NAME="BC_Video" MAINTYPE="VIDEO"
  SUBTYPE="MULTIPLESTREAM">

  <TRANSFORMATION>
   <OPERATION NAME="Assemble">
    <INPUT NAME="Clip1"/>
    <INPUT NAME="Clip2"/>
    <INPUT NAME="Clip3"/>
    <PARAM SEQUENCE="Clip1, Clip2, Clip3"/>
    <OUTPUT NAME="Clip"/>
   </OPERATION>
   <OPERATION NAME="Compose">
    <INPUT NAME="Sound"/>
    <INPUT NAME="Clip"/>
   </OPERATION>
  </TRANSFORMATION>

 </VIRTUAL>

</VMDESC>
```



Dipl.-Inform. Ulrich Marder

Multimedia Database Support for Digital Libraries

Seminar at Schloss Dagstuhl, 29.08.-03.09.99

Improving the Performance of Media Servers Providing Physical Data Independence—Problems, Concepts, and Challenges

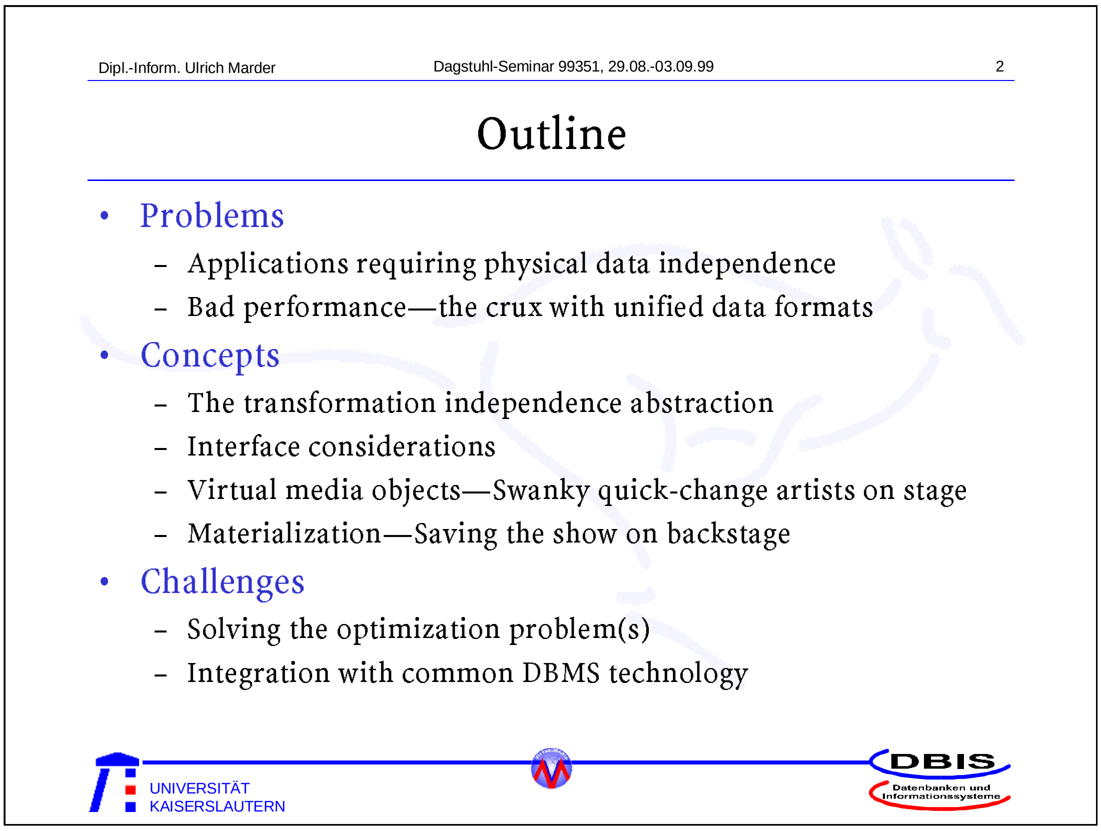
Ulrich Marder

www.ulrich-marder.de

UNIVERSITÄT KAISERSLAUTERN

DBIS Datenbanken und Informationssysteme

This presentation deals with media servers providing physical data independence. Today's media servers - especially continuous media servers - usually do not provide physical data independence at all. One - if not the main - reason for this is performance. Physical data independence *without optimization* costs a lot of performance. Therefore, we are looking for a solution of this optimization problem.



Outline

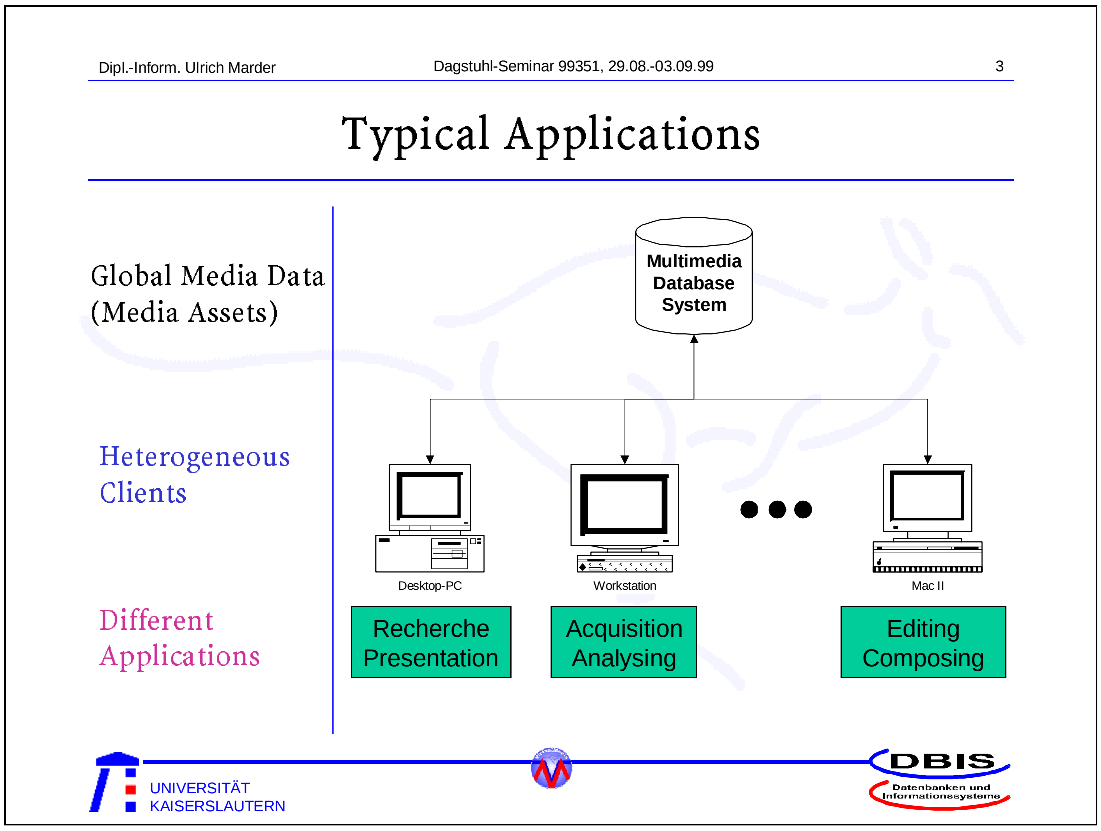
- Problems
  - Applications requiring physical data independence
  - Bad performance—the crux with unified data formats
- Concepts
  - The transformation independence abstraction
  - Interface considerations
  - Virtual media objects—Swanky quick-change artists on stage
  - Materialization—Saving the show on backstage
- Challenges
  - Solving the optimization problem(s)
  - Integration with common DBMS technology

This is a short overview of the talk.

We begin with looking at the problems that actually cause the performance loss. Most of this has to do with the ‘standard solution’ for providing physical data independence, that is using unified data formats. There is a quotation from Herbert Mencken which is certainly true in this case: “For every problem there is one solution which is simple, neat, and wrong.”

Next, concepts for a better solution are introduced. The most abstract version of these concepts is the so-called transformation independence abstraction. Then, we will show how this abstraction can be realized by virtual media objects.

Finally, some remarks on the challenges still waiting will be made.



Typical Applications

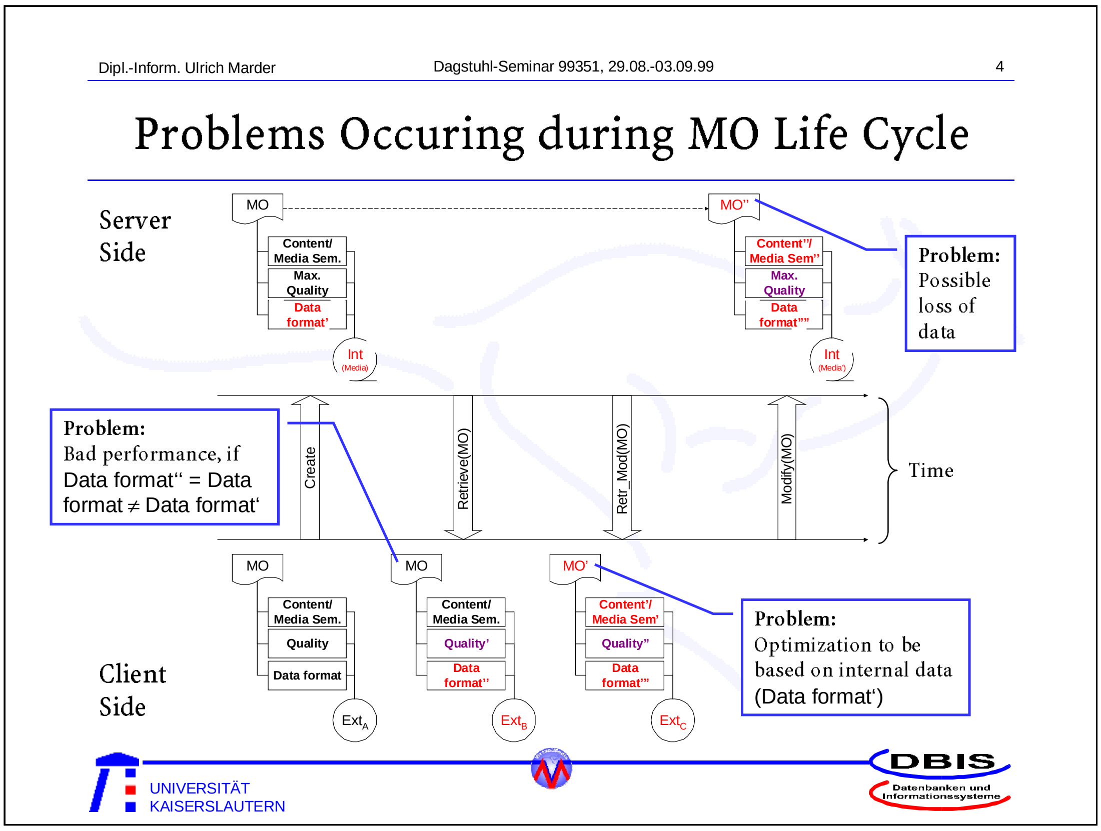
Global Media Data (Media Assets)

Heterogeneous Clients

Different Applications

Multimedia Database System
Desktop-PC
Workstation
Mac II
Recherche Presentation
Acquisition Analysing
Editing Composing
UNIVERSITÄT KAISERSLAUTERN
DBIS Datenbanken und Informationssysteme

This is an application scenario showing a typical situation where physical data independence would be highly beneficial. There is global media data - sometimes called media assets - stored in a MMDBMS. There are heterogeneous clients with different capabilities of storing, processing, and presenting media data. And there are different applications. Some only want to retrieve media objects for presentation or possibly printing. Others create or modify media objects. And again others create media objects by editing and combining existing ones. But note that there is sort of unbalance: Usually there are many applications of the presentation type but few of the other types. The latter, however, often have stronger quality demands.



Dipl.-Inform. Ulrich Marder Dagstuhl-Seminar 99351, 29.08.-03.09.99 4

Problems Occuring during MO Life Cycle

Problem: Possible loss of data

Problem: Bad performance, if Data format" = Data format ≠ Data format'

Problem: Optimization to be based on internal data (Data format')

On this slide part of the life cycle of a media object (MO) in such a system is shown.

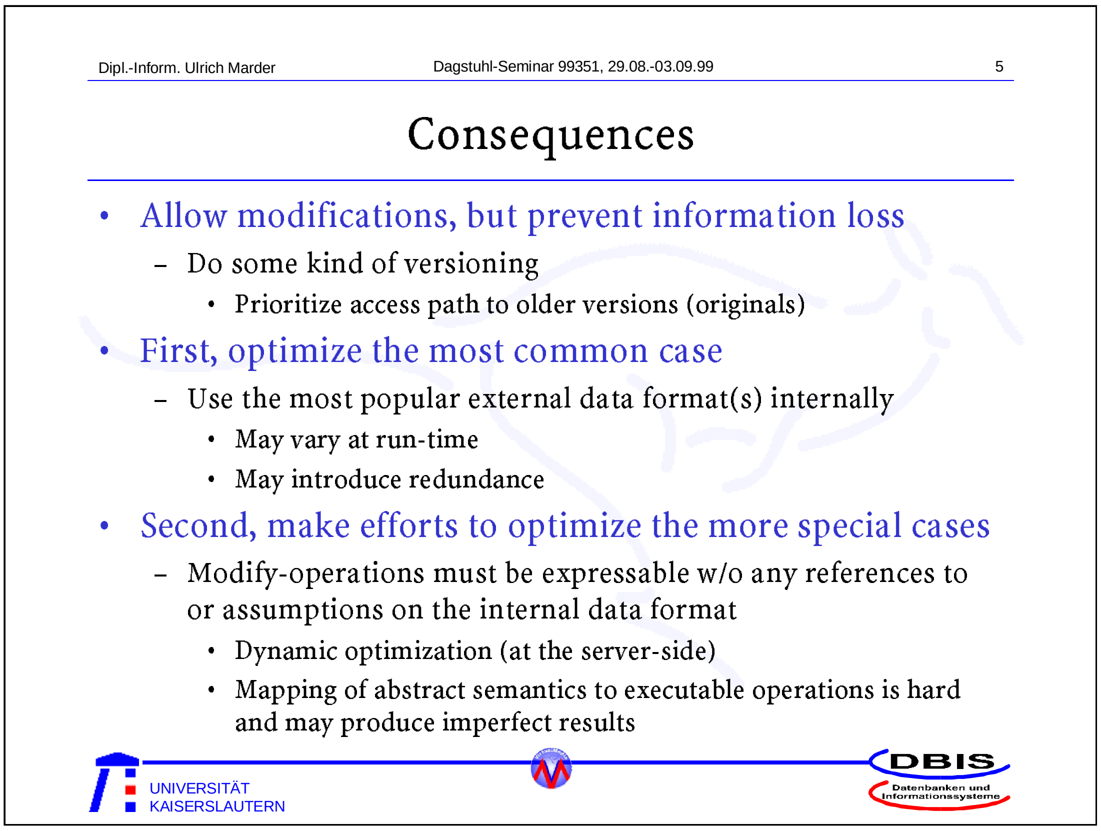
First, the MO is created by some application. The external and internal data formats may be different. Second, another application will eventually retrieve this object. Again, the external and internal data format may be different. Also, the quality of the retrieved object may change. In the third case, some modifications of the MO are invoked prior to to retrieving it. The modifications are only to be applied to the MO delivered, but not to be made persistent. In the last case, the MO is updated in the database.

One can observe three problems here:

First, we get bad performance if the external formats are equal but different from the internal format. We can assume that this will happen quite frequently (except the creating application happens to choose an exotic external format). Hence, any solution enforcing data conversion to a unified internal format is far from being optimal.

Second, all optimization regarding modify-operations has to be based on the internal data format. Thus, there is no chance of leaving optimization to the client application.

And third, the update-operation generally causes a loss of information because many modifications are irreversible. This problem is not particularly dependent on physical data independence. Its solution, however, is dependent on the materialization strategy and, thus, subject to optimization.



Dipl.-Inform. Ulrich Marder Dagstuhl-Seminar 99351, 29.08.-03.09.99 5

Consequences

- Allow modifications, but prevent information loss
  - Do some kind of versioning
    - Prioritize access path to older versions (originals)
- First, optimize the most common case
  - Use the most popular external data format(s) internally
    - May vary at run-time
    - May introduce redundance
- Second, make efforts to optimize the more special cases
  - Modify-operations must be expressable w/o any references to or assumptions on the internal data format
    - Dynamic optimization (at the server-side)
    - Mapping of abstract semantics to executable operations is hard and may produce imperfect results

UNIVERSITÄT KAISERSLAUTERN

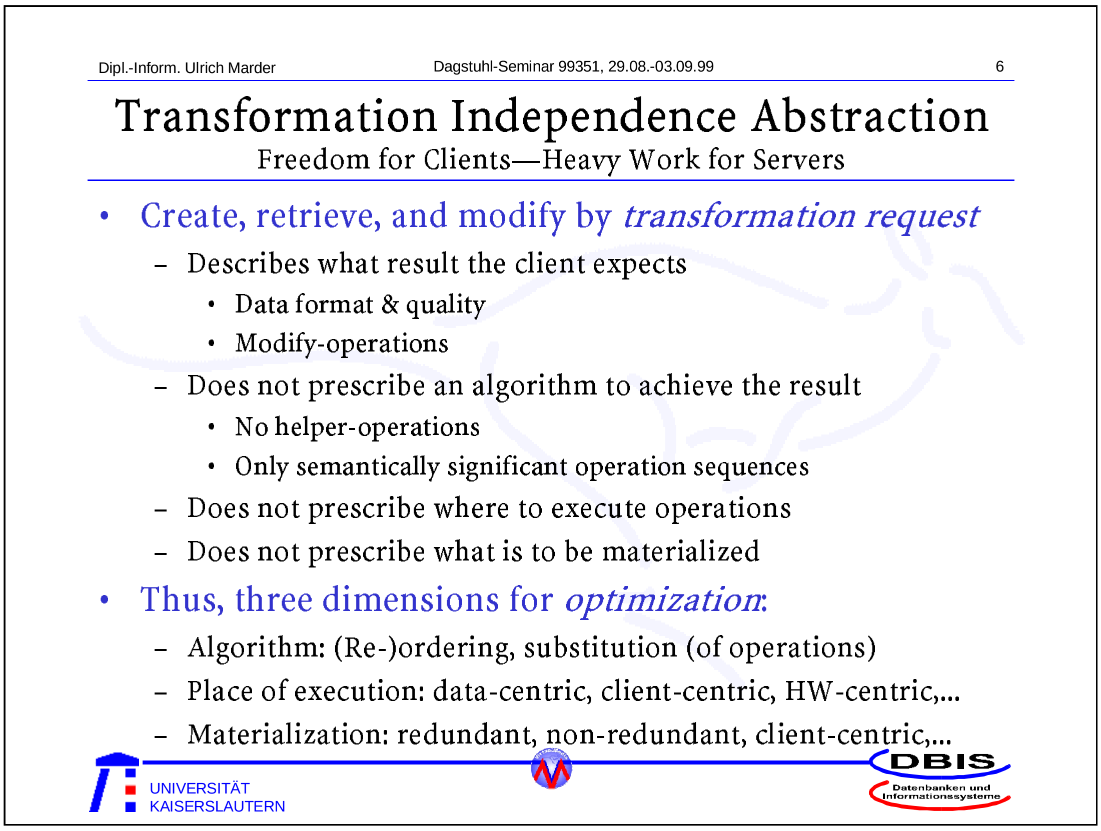
DBIS Datenbanken und Informationssysteme

The following consequences can be drawn from the previous observations.

First, we certainly want to allow modifications, but we also want to prevent information loss. Therefore, some kind of version control will be required. In contrast to more traditional versioning models, the access path to the older version is always to be prioritized. That means, an application maintaining an external reference to an MO will always get it unmodified by other applications.

Considering optimization, we should optimize the most common case in the first place. And that means, quite obviously, that we have to use the most popular external formats also as internal formats in order to prevent format conversions as often as possible. Of course, what *is* popular may vary during the life cycle of an MO. This variation may also introduce redundance - that is materialization in different formats at the same time.

But we should also attempt to optimize the more special cases. This means optimizing media modifications at the server. As a precondition such operations must be expressablewithout any reference to or even assumptions on the internal format. Then, it would be possible to do optimization by dynamic programming. However, mapping of such abstract operations to really executable operations is subject to some semantic fuzziness and, hence, may occasionally produce imperfect results.



Dipl.-Inform. Ulrich Marder Dagstuhl-Seminar 99351, 29.08.-03.09.99 6

Transformation Independence Abstraction

Freedom for Clients—Heavy Work for Servers

- Create, retrieve, and modify by *transformation request*
  - Describes what result the client expects
    - Data format & quality
    - Modify-operations
  - Does not prescribe an algorithm to achieve the result
    - No helper-operations
    - Only semantically significant operation sequences
  - Does not prescribe where to execute operations
  - Does not prescribe what is to be materialized
- Thus, three dimensions for *optimization*:
  - Algorithm: (Re-)ordering, substitution (of operations)
  - Place of execution: data-centric, client-centric, HW-centric,...
  - Materialization: redundant, non-redundant, client-centric,...

UNIVERSITÄT KAISERSLAUTERN

DBIS Datenbanken und Informationssysteme

Transformation independence is the most abstract version of our media server concept. It is kind of extension of the traditional data independence paradigm because it considers data format, data quality, and data modifications together. All these aspects are described by the client in a so-called transformation request, which can be used to create, retrieve, and modify MOs.

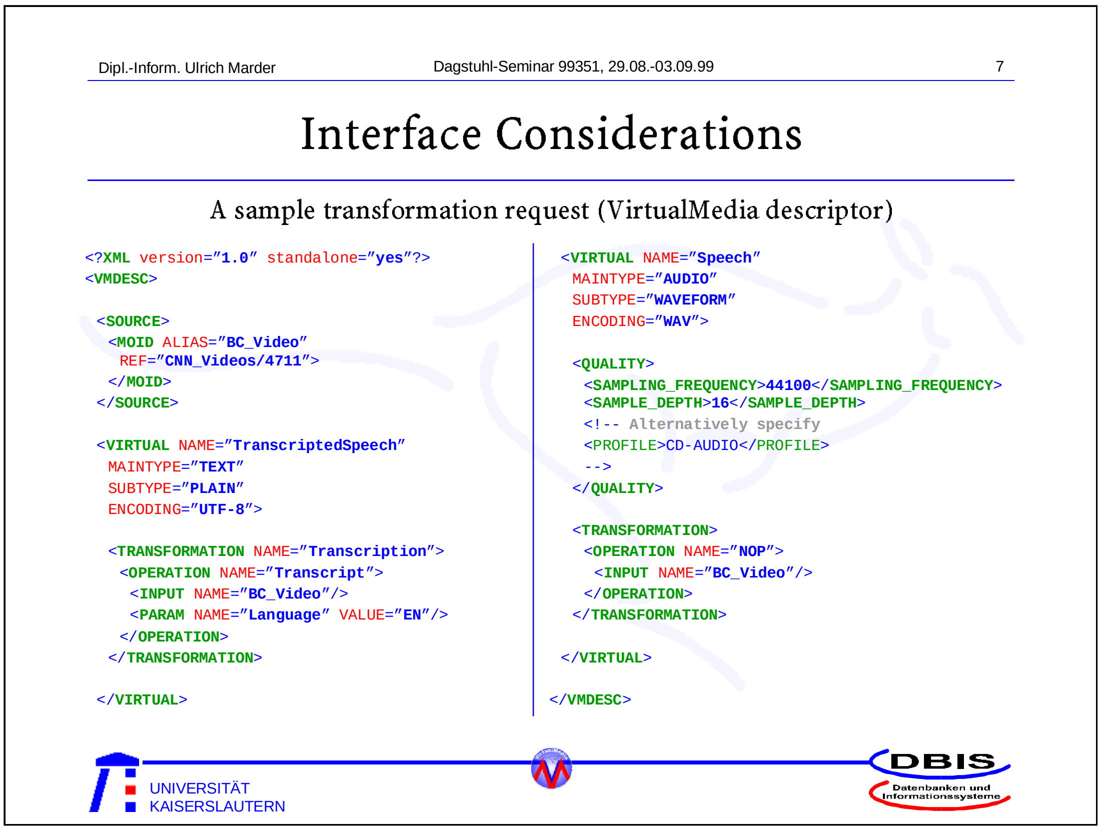
Also note that a transformation request

- does not prescribe an algorithm to achieve the result,

- does not prescribe where to execute operations,

- and does not prescribe what is to be materialized.

Thus, we get three dimensions for optimizing at the server. The server can reorder or substitute operations according to appropriate rules. The server can decide for each operation wether it should be executed where the data is, where the client is, or where special hardware supporting it is. And the server may choose to materialize MOs redundantly (or not) - or even at the client’s machine.



Dipl.-Inform. Ulrich Marder Dagstuhl-Seminar 99351, 29.08.-03.09.99 7

Interface Considerations

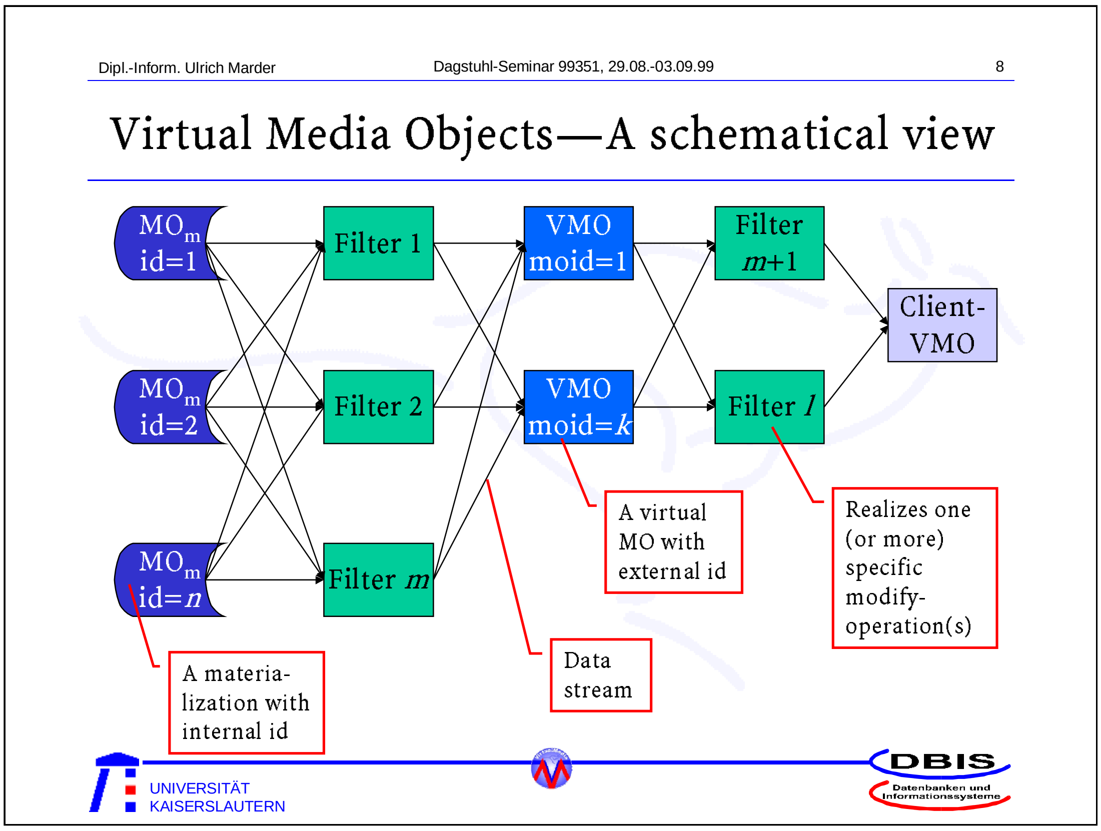
A sample transformation request (VirtualMedia descriptor)

```
<?XML version=”1.0” standalone=”yes”?>
<VMDESC>

 <SOURCE>
  <MOID ALIAS=”BC_Video”
   REF=”CNN_Videos/4711”>
  </MOID>
 </SOURCE>

 <VIRTUAL NAME=”TranscriptedSpeech”
  MAINTYPE=”TEXT”
  SUBTYPE=”PLAIN”
  ENCODING=”UTF-8”>

  <TRANSFORMATION NAME=”Transcription”>
   <OPERATION NAME=”Transcript”>
    <INPUT NAME=”BC_Video”/>
    <PARAM NAME=”Language” VALUE=”EN”/>
   </OPERATION>
  </TRANSFORMATION>

 </VIRTUAL>

 <VIRTUAL NAME=”Speech”
  MAINTYPE=”AUDIO”
  SUBTYPE=”WAVEFORM”
  ENCODING=”WAV”>

  <QUALITY>
   <SAMPLING_FREQUENCY>44100</SAMPLING_FREQUENCY>
   <SAMPLE_DEPTH>16</SAMPLE_DEPTH>
   <!-- Alternatively specify
   <PROFILE>CD-AUDIO</PROFILE>
   -->
  </QUALITY>

  <TRANSFORMATION>
   <OPERATION NAME=”NOP”>
    <INPUT NAME=”BC_Video”/>
   </OPERATION>
  </TRANSFORMATION>

 </VIRTUAL>

</VMDESC>
```

UNIVERSITÄT KAISERSLAUTERN

DBIS Datenbanken und Informationssysteme

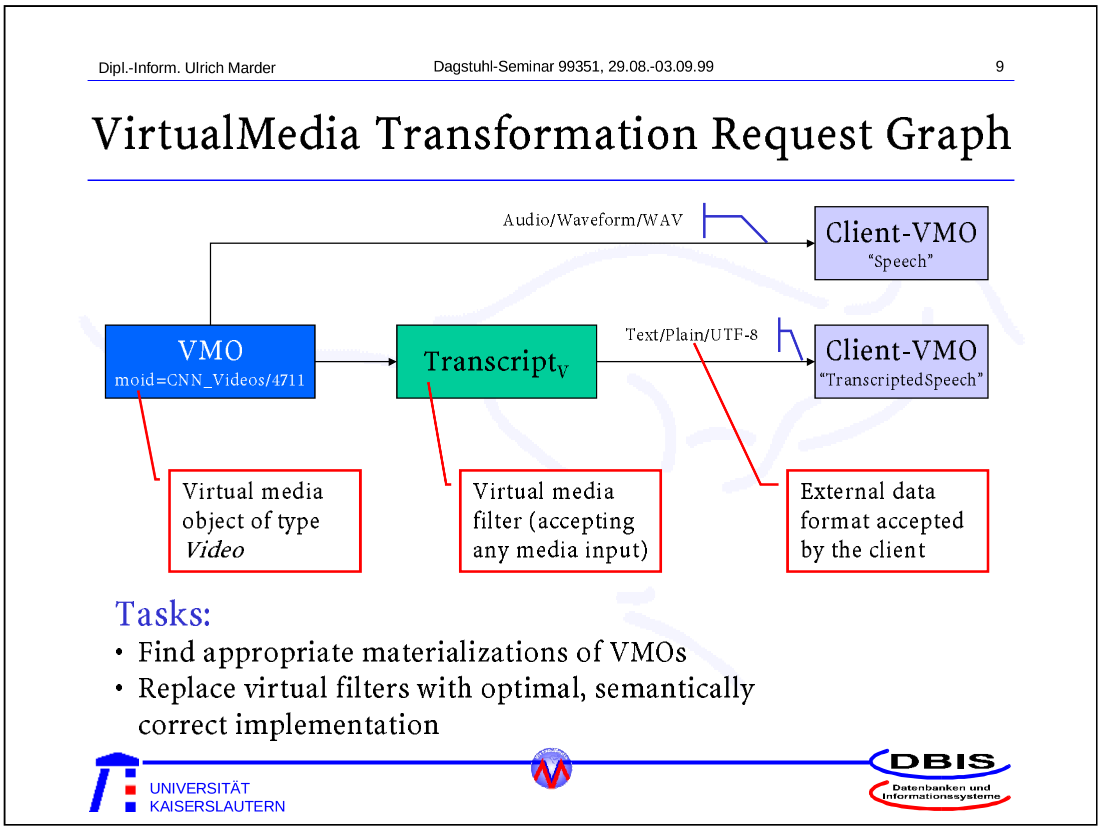
This a sample transformation request that the client may send to the server. It is also sometimes called a *VirtualMedia descriptor*. This request first references a media object which is stored in the database. This is a *Video* object. Next, the request describes two so-called *virtual objects* that are to be delivered to the client. Each virtual object has a type and format description, transformation description, and optionally a quality description. The transformation description of the first virtual object means that it should be a transcription of the video referenced in the source-part. The second virtual object is the audio-part of the same video-object. Its quality is specified very detailed - so, another possibility would be using certain *quality profiles*.



Virtual Media Objects—A schematical view

MO$_m$ id=1
MO$_m$ id=2
MO$_m$ id=n
Filter 1
Filter 2
Filter m
VMO moid=1
VMO moid=k
Filter m+1
Filter l
Client-VMO
A materia-lization with internal id
Data stream
A virtual MO with external id
Realizes one (or more) specific modify-operation(s)
UNIVERSITÄT KAISERSLAUTERN
DBIS Datenbanken und Informationssysteme

This graph should clarify what we mean by *virtual media objects* (VMO). The virtual objects described in the transformation request are actually client-VMOs appearing as end-nodes of this graph. Database objects that are visible to the client are also VMOs having an external media object id (moid). The graph also shows the relation between materializations and VMOs. The materializations, however, are not visible to clients and, therefore, have only internal ids. The edges of the graph determine how the data has to flow in order to materialize the VMOs. At arbitrary points within that data flow *filters* can be placed that realize one (or maybe more) specific modify-operation.



Dipl.-Inform. Ulrich Marder Dagstuhl-Seminar 99351, 29.08.-03.09.99 9

VirtualMedia Transformation Request Graph

Audio/Waveform/WAV

Client-VMO "Speech"

VMO moid=CNN_Videos/4711

Transcript_V

Text/Plain/UTF-8

Client-VMO "TranscriptedSpeech"

Virtual media object of type *Video*

Virtual media filter (accepting any media input)

External data format accepted by the client

Tasks:

- Find appropriate materializations of VMOs
- Replace virtual filters with optimal, semantically correct implementation

UNIVERSITÄT KAISERSLAUTERN

DBIS Datenbanken und Informationssysteme

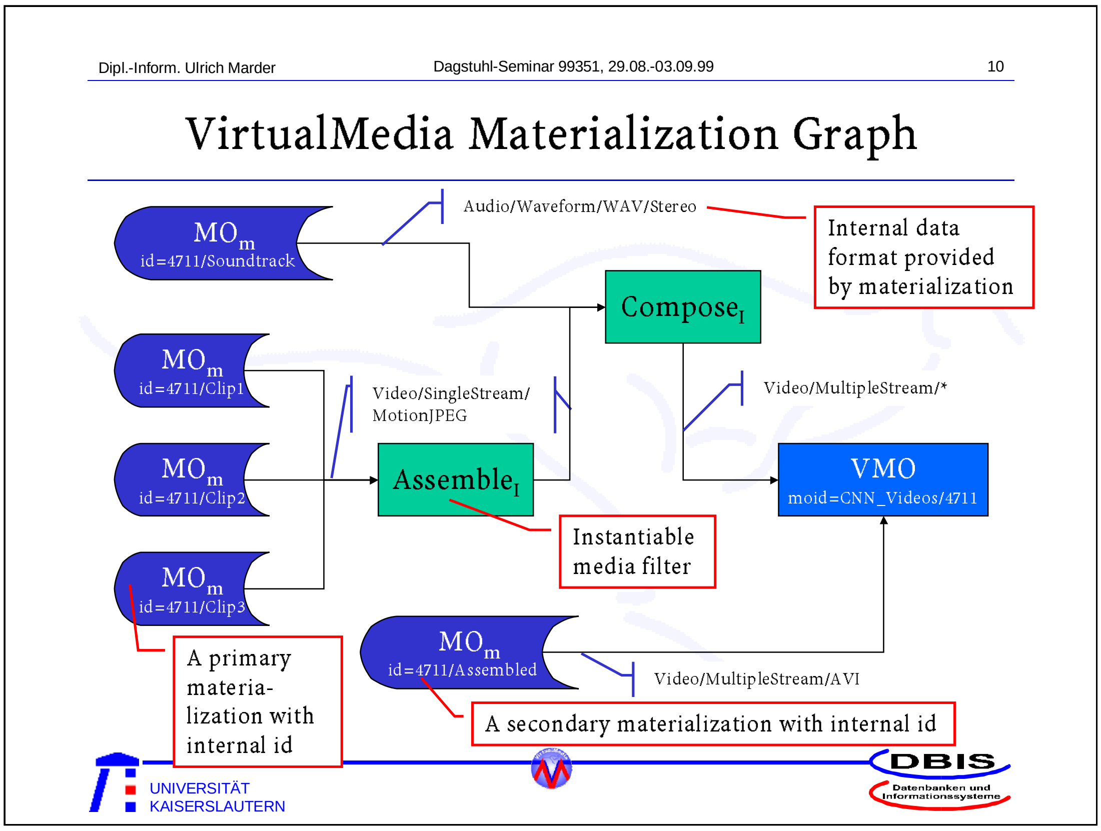
This is the previously shown transformation request translated into its graph representation.

The start-node of this graph is the video-object which is the source object of the transformation request. The two virtual objects of the transformation request become end-nodes of the graph. The edges pointing to these end-nodes are labeled with the requested type and format specification. The transcript operation specified in the request is turned into an according *virtual media filter*, which is placed within the the data flow from the source object to the text-object. The filter must be virtual because its input is virtual.

There are now two tasks for the server:

Find an appropriate materialization of the video-object. And replace the virtual filter with its optimal, sematically correct implementation.

Before explaining that algorithm in some more detail, we should take a look at the *materialization graph* of the virtual video-object.



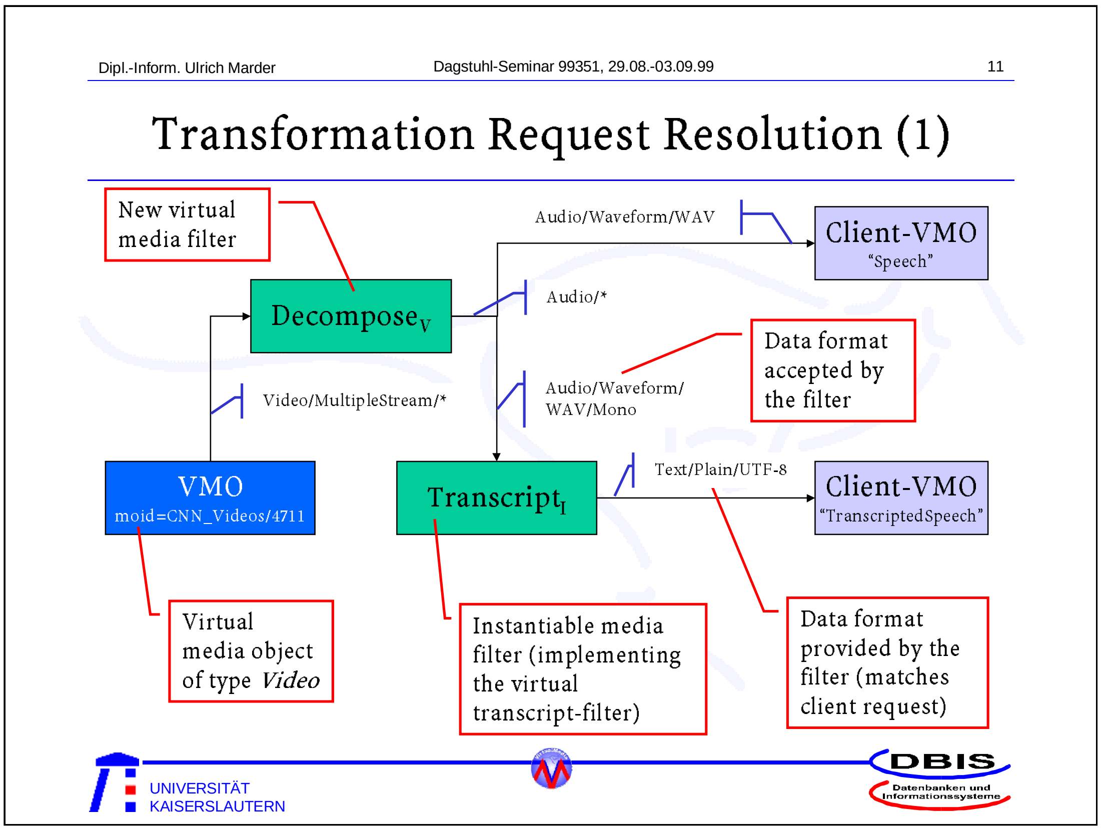
VirtualMedia Materialization Graph

MO$_m$ id=4711/Soundtrack
Audio/Waveform/WAV/Stereo
Internal data format provided by materialization
Compose$_I$
MO$_m$ id=4711/Clip1
Video/SingleStream/ MotionJPEG
Video/MultipleStream/*
MO$_m$ id=4711/Clip2
Assemble$_I$
VMO moid=CNN_Videos/4711
Instantiable media filter
MO$_m$ id=4711/Clip3
MO$_m$ id=4711/Assembled
Video/MultipleStream/AVI
A primary materia-lization with internal id
A secondary materialization with internal id

UNIVERSITÄT KAISERSLAUTERN

DBIS Datenbanken und Informationssysteme

The materialization graph of a VMO contains all its materializations and how they are related to the VMO. There are different types of materializations: primary, secondary, and derived materializations. The latter will be considered on one of the following slides. Primary materializations are provided at create-time of the VMO and are assumed to provide the maximum available quality of the VMO. Secondary materializations are created by the server purely for optimization purposes - usually without informing the applications. Hence, the server may create or destroy secondary materializations whenever this seems likely to improve the performance of the MMDBMS. As shown in the example, materialization graphs can also contain media filters. But in contrast to the transformation request graph, these media filters are already instantiable. This is possible because the input data formats are always known.

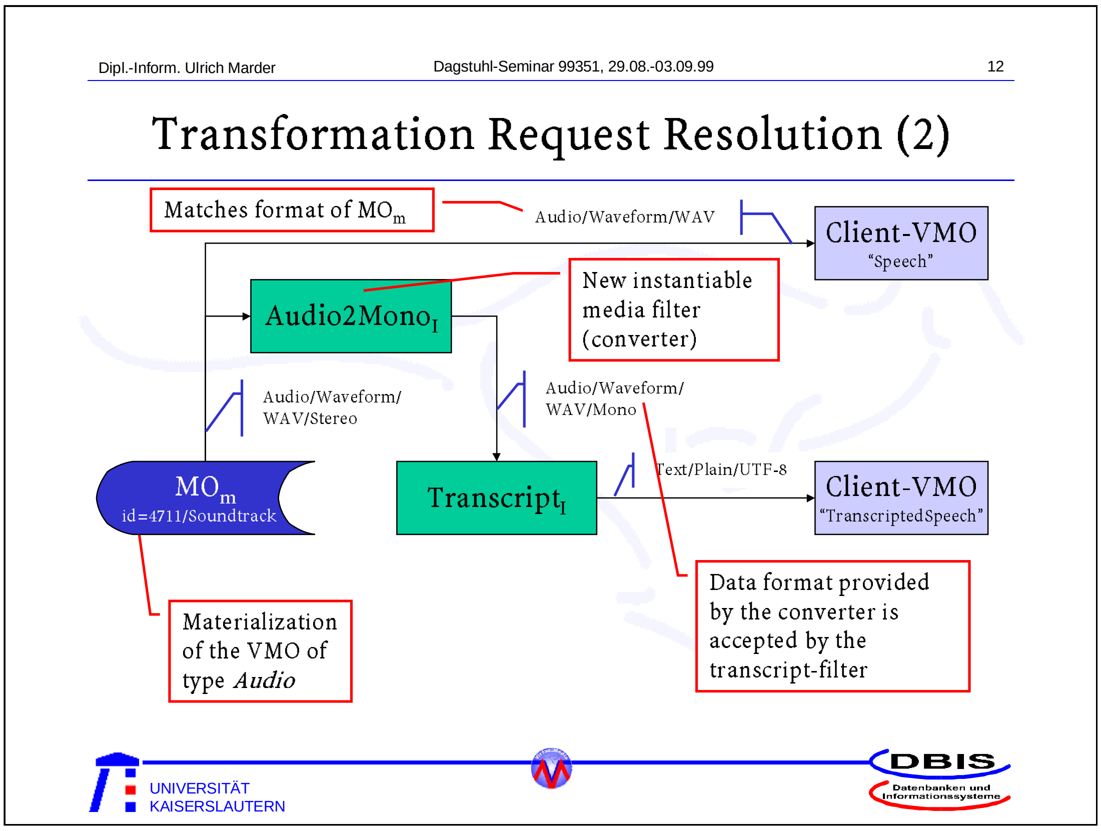


Transformation Request Resolution (1)

We are now considering the transformation request resolution algorithm by looking at how it transforms the original request graph of the example.

In the snapshot shown on the slide the virtual transcript-filter is replaced by some instantiable transcript filter. This filter introduces new requirements on the data formats. While the output format of the filter nicely matches the client's request, the input specification does not fit at all: it has to be an audio-object whereas the input supplied by the VMO is a video-object. Since the transformation request does not say anything about how to turn the video into audio, the resolution algorithm chooses a default rule for resolving this mismatch, which is inserting a (virtual) decompose-filter. Obviously, the same considerations apply to the second client-VMO.

To continue the resolution process this intermediate graph must now be connected with the materialization graph.



Dipl.-Inform. Ulrich Marder Dagstuhl-Seminar 99351, 29.08.-03.09.99 12

Transformation Request Resolution (2)

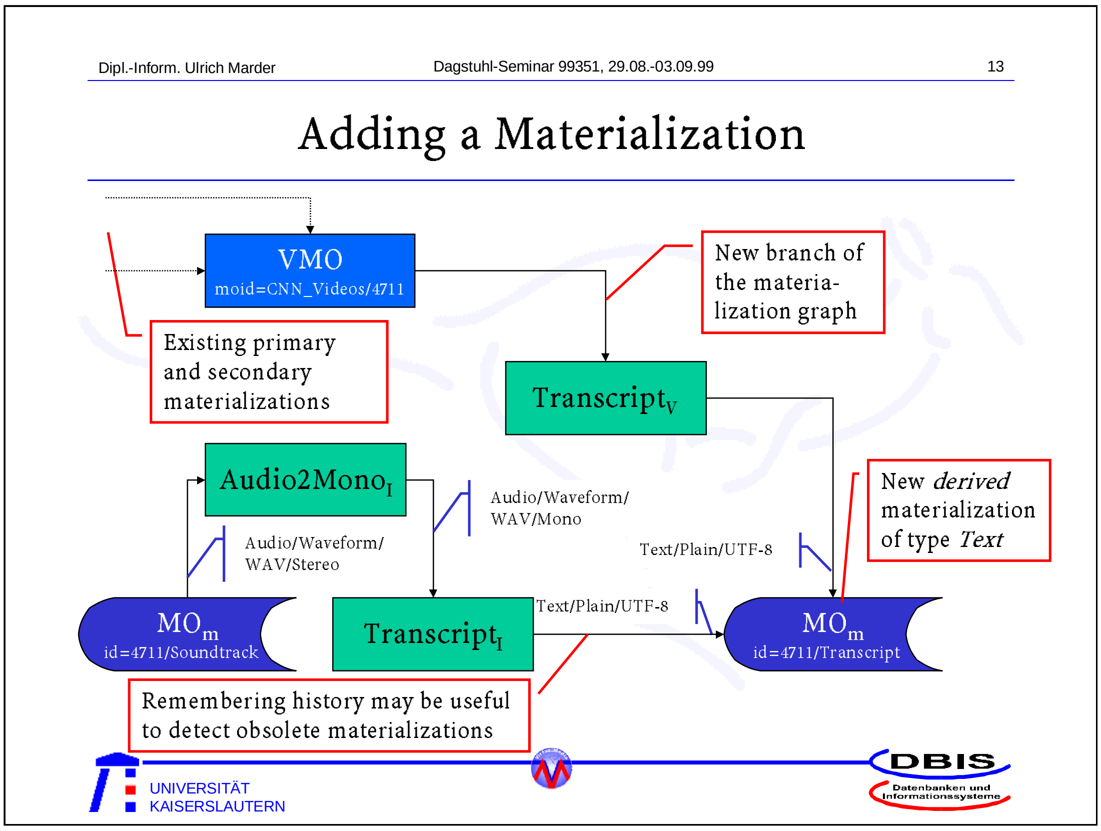
Matches format of MO_m
Audio/Waveform/WAV
Client-VMO
"Speech"
$Audio2Mono_I$
New instantiable media filter (converter)
Audio/Waveform/WAV/Stereo
Audio/Waveform/WAV/Mono
MO_m
id=4711/Soundtrack
$Transcript_I$
Text/Plain/UTF-8
Client-VMO
"TranscriptedSpeech"
Data format provided by the converter is accepted by the transcript-filter
Materialization of the VMO of type *Audio*

UNIVERSITÄT KAISERSLAUTERN

DBIS Datenbanken und Informationssysteme

This slide shows the result of the transformation request resolution.

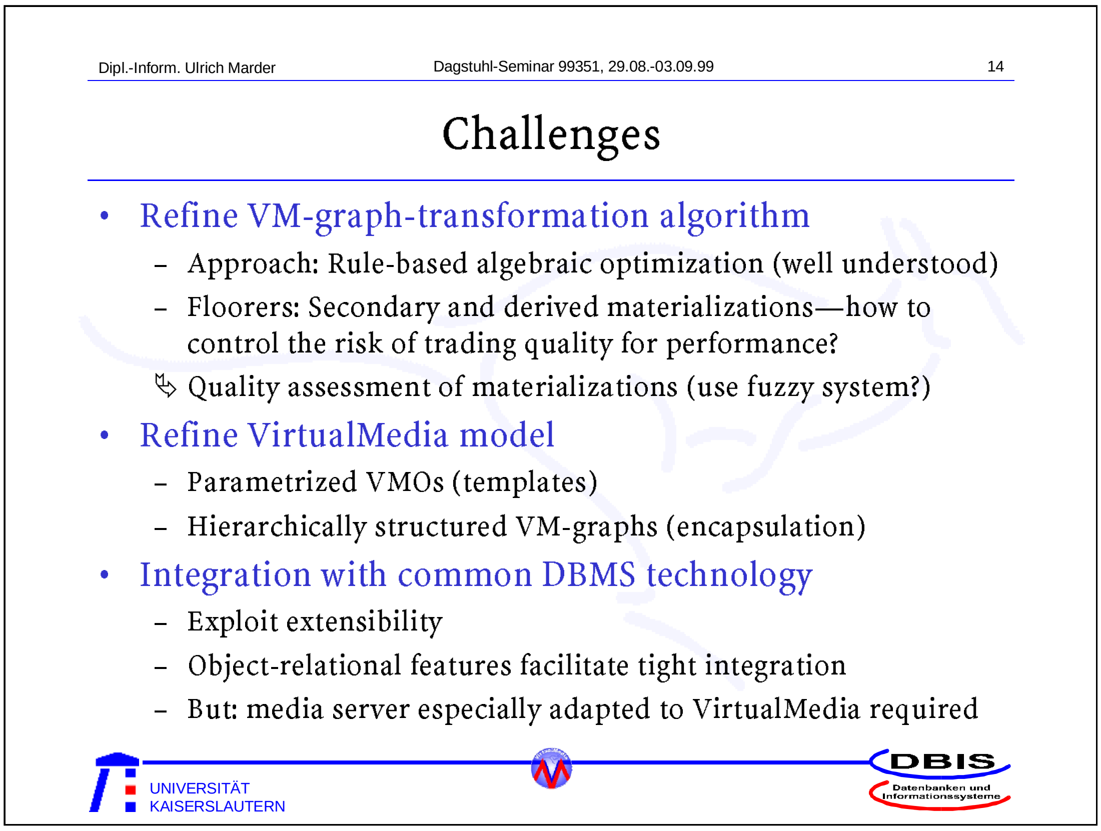
The virtual decompose-filter which had been introduced intermediately could be eliminated again, because it is the inverse of the compose-filter in the materialization graph. Thus, the resolution algorithm finds the primary materialization of the video-soundtrack being the optimal source object for the client's request. There is a slight format mismatch between the materialization and the input-specification of the transcript-filter, which is resolved by inserting a suitable converter-filter. Note that, to make this work, the algorithm must be told that such converters are *semantically neutral*. This might, however, be untrue in some rare situations, which is the reason why we cannot expect the algorithm to always work perfectly with respect to preserving the semantics of the client's request.



Adding a Materialization

VMO
moid=CNN_Videos/4711
New branch of the materia-lization graph
Existing primary and secondary materializations
Transcript$_V$
Audio2Mono$_I$
Audio/Waveform/ WAV/Mono
Audio/Waveform/ WAV/Stereo
Text/Plain/UTF-8
New *derived* materialization of type *Text*
MO$_m$
id=4711/Soundtrack
Transcript$_I$
Text/Plain/UTF-8
MO$_m$
id=4711/Transcript
Remembering history may be useful to detect obsolete materializations
UNIVERSITÄT KAISERSLAUTERN
DBIS
Datenbanken und Informationssysteme

After resolving the transformation request the server may detect that the costs of creating the transcription are very high compared to the costs of storing it for later reuse. To enable reuse a new branch of the materialization graph must be created, leading from the VMO to the *derived materialization*. Obviously, this branch must include all virtual filters, thus describing the semantics of the new materialization. Additionally, it is advisable to include the real origin of derived materializations in the graph. This helps assessing the actuality and quality of the materialization, which is necessary to decide wether the client’s quality demands are met if the server uses the materialization to fulfill a certain request.



Dipl.-Inform. Ulrich Marder   Dagstuhl-Seminar 99351, 29.08.-03.09.99   14

Challenges

- Refine VM-graph-transformation algorithm
  - Approach: Rule-based algebraic optimization (well understood)
  - Floorers: Secondary and derived materializations—how to control the risk of trading quality for performance?
  - ↳ Quality assessment of materializations (use fuzzy system?)
- Refine VirtualMedia model
  - Parametrized VMOs (templates)
  - Hierarchically structured VM-graphs (encapsulation)
- Integration with common DBMS technology
  - Exploit extensibility
  - Object-relational features facilitate tight integration
  - But: media server especially adapted to VirtualMedia required

UNIVERSITÄT KAISERSLAUTERN   DBIS Datenbanken und Informationssysteme

The development of the VirtualMedia concept has just started. Hence, it should be no surprise that we are still facing a considerable number of challenges.

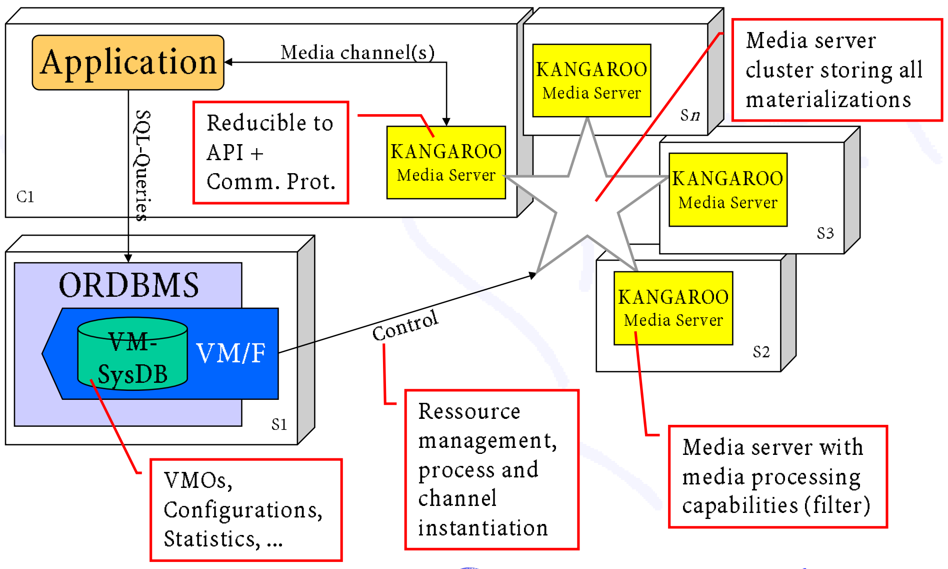
The next one to be tackled is refinement and formalization of the graph transformation algorithm for resolving and optimizing the client’s requests. We are currently considering rule-based algebraic optimization as a promizing approach. One of the most difficult parts of the algorithm probably is controlling the employment of secondary and derived materializations, because, ultimately, this requires sort of quality assessment of materializations.

Especially from the client’s point of view, there certainly is a need for refining the VirtualMedia model itself, e. g. by introducing parametrized VMOs as easy-to-use templates or by enabling hierarchical structures in VM-graphs.

For realizing the VirtualMedia concept we currently favorite integration with common DBMS-technology. Thereby, the extensibility features of modern DBMSs could be exploited. Such DBMSs, however, are not capable of processing media streams in real-time. Therefore, we propose an architecture where the work is shared between an appropriately extended OR-DBMS and several specialized media servers.

Dipl.-Inform. Ulrich Marder Dagstuhl-Seminar 99351, 29.08.-03.09.99 15

Software Architecture Proposal

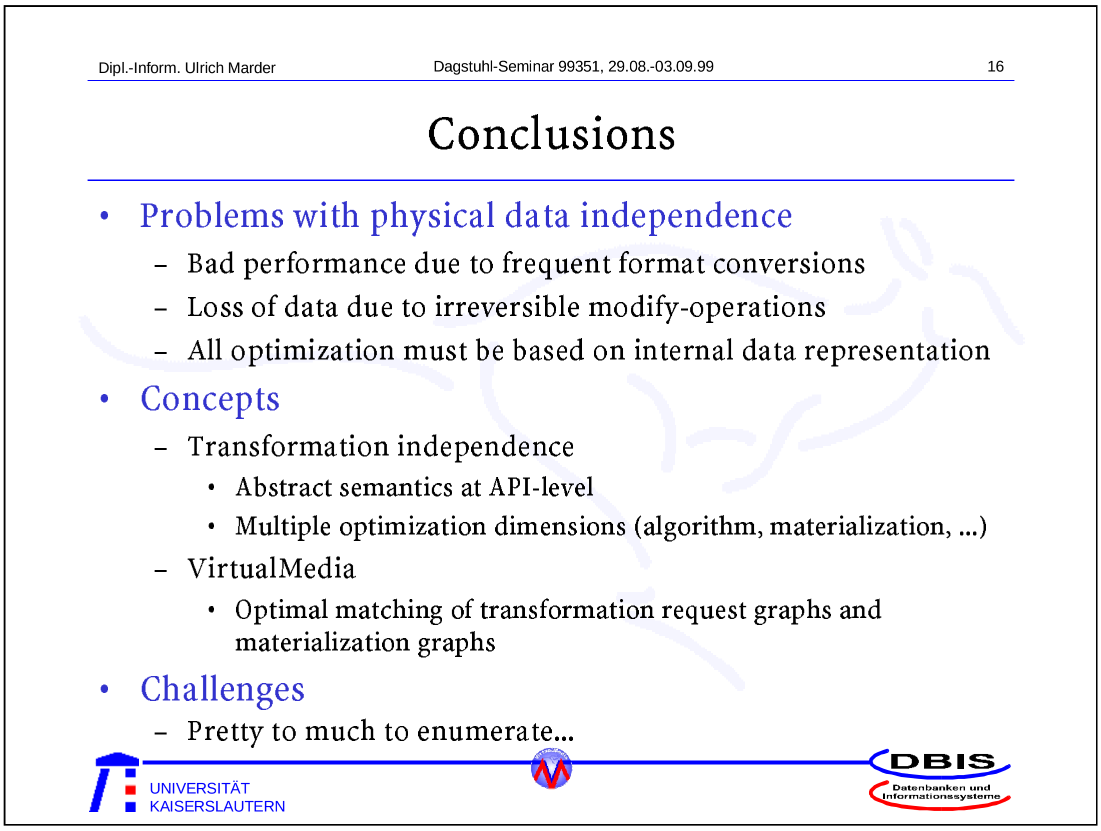


Application
Media channel(s)
SQL-Queries
Reducible to API + Comm. Prot.
KANGAROO Media Server
C1
KANGAROO Media Server
Sn
Media server cluster storing all materializations
KANGAROO Media Server
S3
KANGAROO Media Server
S2
ORDBMS
VM-SysDB
VM/F
S1
Control
VMOs, Configurations, Statistics, ...
Ressource management, process and channel instantiation
Media server with media processing capabilities (filter)

UNIVERSITÄT KAISERSLAUTERN

DBIS Datenbanken und Informationssysteme

The OR-DBMS-extension stores all the VMOs. Additionally, it manages the whole system configuration, statistics, and so on. And, of course, it must implement the request resolution algorithm. Besides that, there is a cluster of media servers managing the materializations and providing the media processing capabilities, where each server may specialize on certain media types or filter operations. The whole cluster is fully controlled by the OR-DBMS-extension. That means, the client application can only open a channel to the media servers by sending an appropriate transformation request to the OR-DBMS. Usually, there is also a media server on the client machine. This could be a fully functional server allowing filtering or even materialization directly on the client, or it could be merely a stub providing only the interface between the application and the media server cluster. Anyway, it is only the server - not the application - that knows about this and has to consider it for optimization.



Dipl.-Inform. Ulrich Marder Dagstuhl-Seminar 99351, 29.08.-03.09.99 16

Conclusions

- Problems with physical data independence
  - Bad performance due to frequent format conversions
  - Loss of data due to irreversible modify-operations
  - All optimization must be based on internal data representation
- Concepts
  - Transformation independence
    - Abstract semantics at API-level
    - Multiple optimization dimensions (algorithm, materialization, ...)
  - VirtualMedia
    - Optimal matching of transformation request graphs and materialization graphs
- Challenges
  - Pretty to much to enumerate...

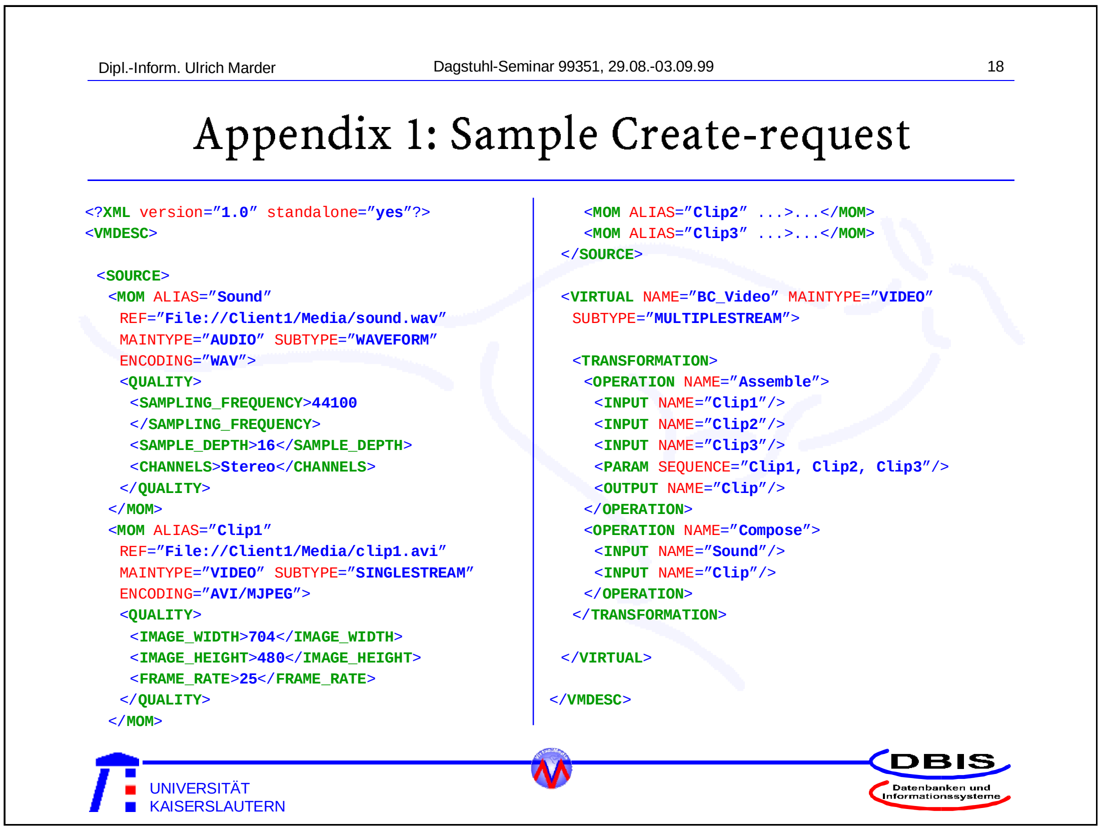
UNIVERSITÄT KAISERSLAUTERN

DBIS Datenbanken und Informationssysteme

We have shown that there are considerable problems when attempting to provide physical data independence with a media server (or MM-DBMS). Such systems tend to require frequent format conversions resulting in bad performance. They may inadvertently lose data due to irreversible updates. And hiding the internal data representation from the client obviously means that all the strongly necessary optimization is to be accomplished by the server.

Our proposed media server concept is based on a generalization of data independence called transformation independence. This abstraction reduces the creation, retrieval, and modification of media objects to what can be called the “pure application semantics”. The consequence are multiple optimization dimensions being left for exploitation at the server. The VirtualMedia concept realizes transformation independence based on virtual media objects being described by filter graphs. With this concept, optimization can be basically characterized as the process of matching transformation request graphs and materialization graphs.

Since we are just at the beginning of developing these concepts, there are still a lot of challenges to be mastered, like formalizing and evaluating the algorithms and realizing the concept using available DBMS-technology and media server components.



Dipl.-Inform. Ulrich Marder Dagstuhl-Seminar 99351, 29.08.-03.09.99 18

Appendix 1: Sample Create-request

```
<?XML version="1.0" standalone="yes"?>
<VMDESC>

 <SOURCE>
  <MOM ALIAS="Sound"
   REF="File://Client1/Media/sound.wav"
   MAINTYPE="AUDIO" SUBTYPE="WAVEFORM"
   ENCODING="WAV">
   <QUALITY>
    <SAMPLING_FREQUENCY>44100
    </SAMPLING_FREQUENCY>
    <SAMPLE_DEPTH>16</SAMPLE_DEPTH>
    <CHANNELS>Stereo</CHANNELS>
   </QUALITY>
  </MOM>
  <MOM ALIAS="Clip1"
   REF="File://Client1/Media/clip1.avi"
   MAINTYPE="VIDEO" SUBTYPE="SINGLESTREAM"
   ENCODING="AVI/MJPEG">
   <QUALITY>
    <IMAGE_WIDTH>704</IMAGE_WIDTH>
    <IMAGE_HEIGHT>480</IMAGE_HEIGHT>
    <FRAME_RATE>25</FRAME_RATE>
   </QUALITY>
  </MOM>
  <MOM ALIAS="Clip2" ...>...</MOM>
  <MOM ALIAS="Clip3" ...>...</MOM>
 </SOURCE>

 <VIRTUAL NAME="BC_Video" MAINTYPE="VIDEO"
  SUBTYPE="MULTIPLESTREAM">

  <TRANSFORMATION>
   <OPERATION NAME="Assemble">
    <INPUT NAME="Clip1"/>
    <INPUT NAME="Clip2"/>
    <INPUT NAME="Clip3"/>
    <PARAM SEQUENCE="Clip1, Clip2, Clip3"/>
    <OUTPUT NAME="Clip"/>
   </OPERATION>
   <OPERATION NAME="Compose">
    <INPUT NAME="Sound"/>
    <INPUT NAME="Clip"/>
   </OPERATION>
  </TRANSFORMATION>

 </VIRTUAL>

</VMDESC>
```

UNIVERSITÄT
KAISERSLAUTERN

DBIS
Datenbanken und Informationssysteme

This is the transformation request (with some omissions) that would result in the creation of the materialization graph shown on slide no. 10 (without the secondary materialization).