Ontology-based Integration of Cultural Heritage Metadata

Christos Papatheodorou[1,2]
Manolis Gergatsoulis[1], Lina Bountouri[1], Panorea Gaitanou[1],

1. Database & Information Systems Group
Laboratory of Digital Libraries and Electronic Publishing
Department of Archives and Library Sciences
Ionian University, Corfu, Greece.
{manolis, boudouri, rgaitanou, papatheodor}@ionio.gr

2. Digital Curation Unit
Institute for the Management of Information Systems (IMIS)
Athena R.C., Athens, Greece.

Acknowledgement

- Thank you for inviting me
- This work started during the last phase of DELOS: mapping metadata schemas to CIDOC CRM in cooperation with FORTH, Greece
- Thanks to the DBIS members
  - Except the authors, 3 MSc theses and 1 PhD dissertation has been completed

Metadata Interoperability

- Cultural Heritage (CH) institutions, such as archives, libraries and museums, host and develop various types of material often described by different metadata schemas
- Semantic heterogeneity between metadata elements
  - Example: the concept of “creator” in bibliografic metadata schemas
  - DC-Lib: element *Creator*
  - MODS: element *Name*: information about names
    - sub-element: *Role*, value creator
    - Attribute: *type*, values personal, corporate, conference andfamily.
  - UNIMARC Bibliographic Format: fields 700, 701, 710,711, 720 and721

Metadata Interoperability

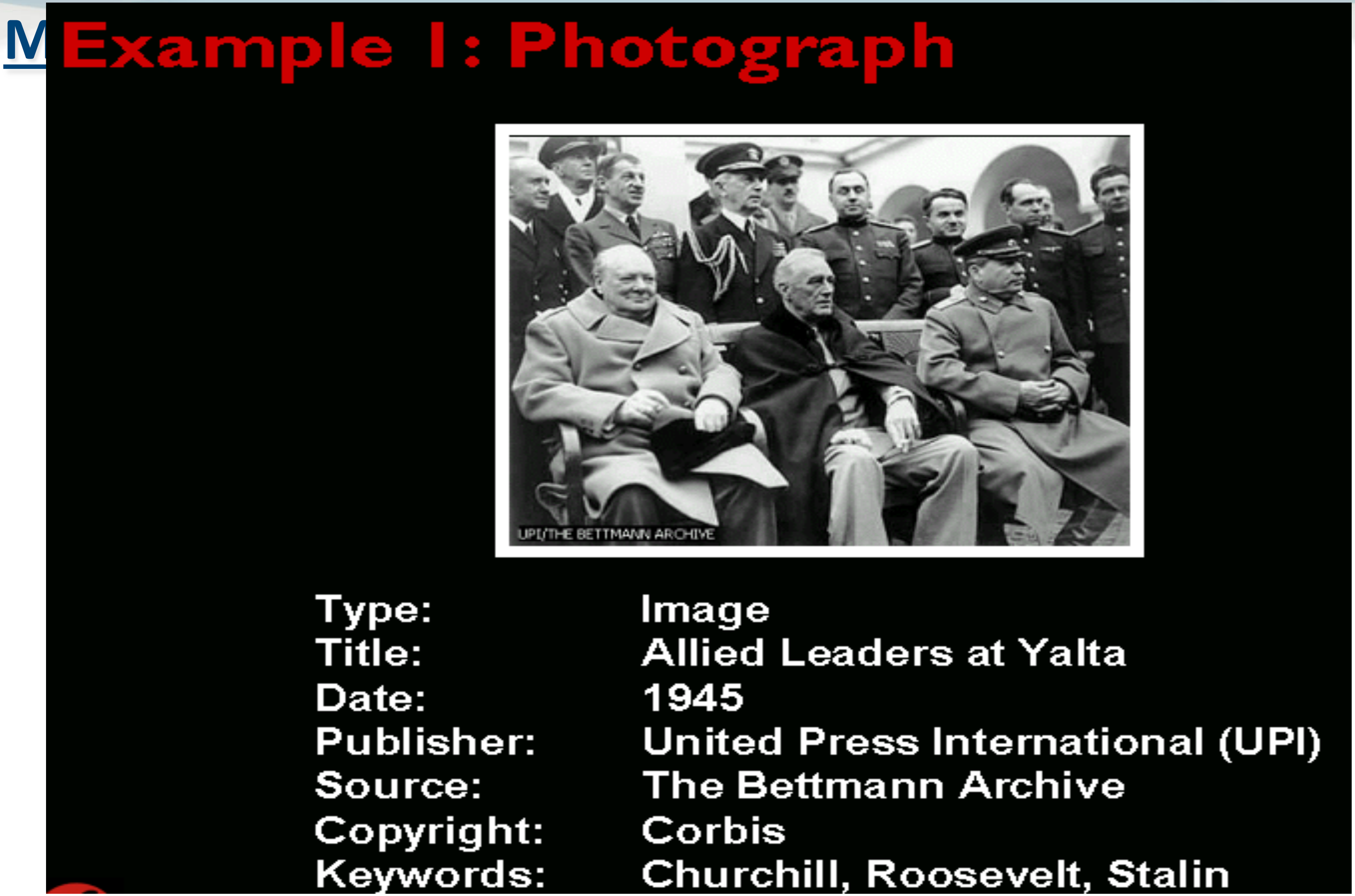
- Need for *Metadata Interoperability*
  - i.e. for metadata exchange, Information Retrieval
  - Mapping of the conceptual model among two or more metadata schemas
- Various methods have been implemented
  - Crosswalks,
  - Application Profiles (APs),
  - Switching Scema
  - *Ontology based Integration*, etc
- Ontologies can act as the *mediated schema* between heterogeneous sources
  - CIDOC CRM can act as the mediator for diverse CH metadata, since it provides a conceptual view of the CH domain

CIDOC Conceptual Reference Model (CRM)

- A formal ontology for the CH domain, consisting of a hierarchy of 86 *entities* (named also *classes*) and 137 *properties* and expressing semantics as a sequence of class/property path(s)
  - Entities group items, called *class instances*, sharing common characteristics
  - A *class* may be the *domain* or the *range* of properties, which are binary relations between classes
  - An *instance of a property* is a relation between an instance of its domain and an instance of its range

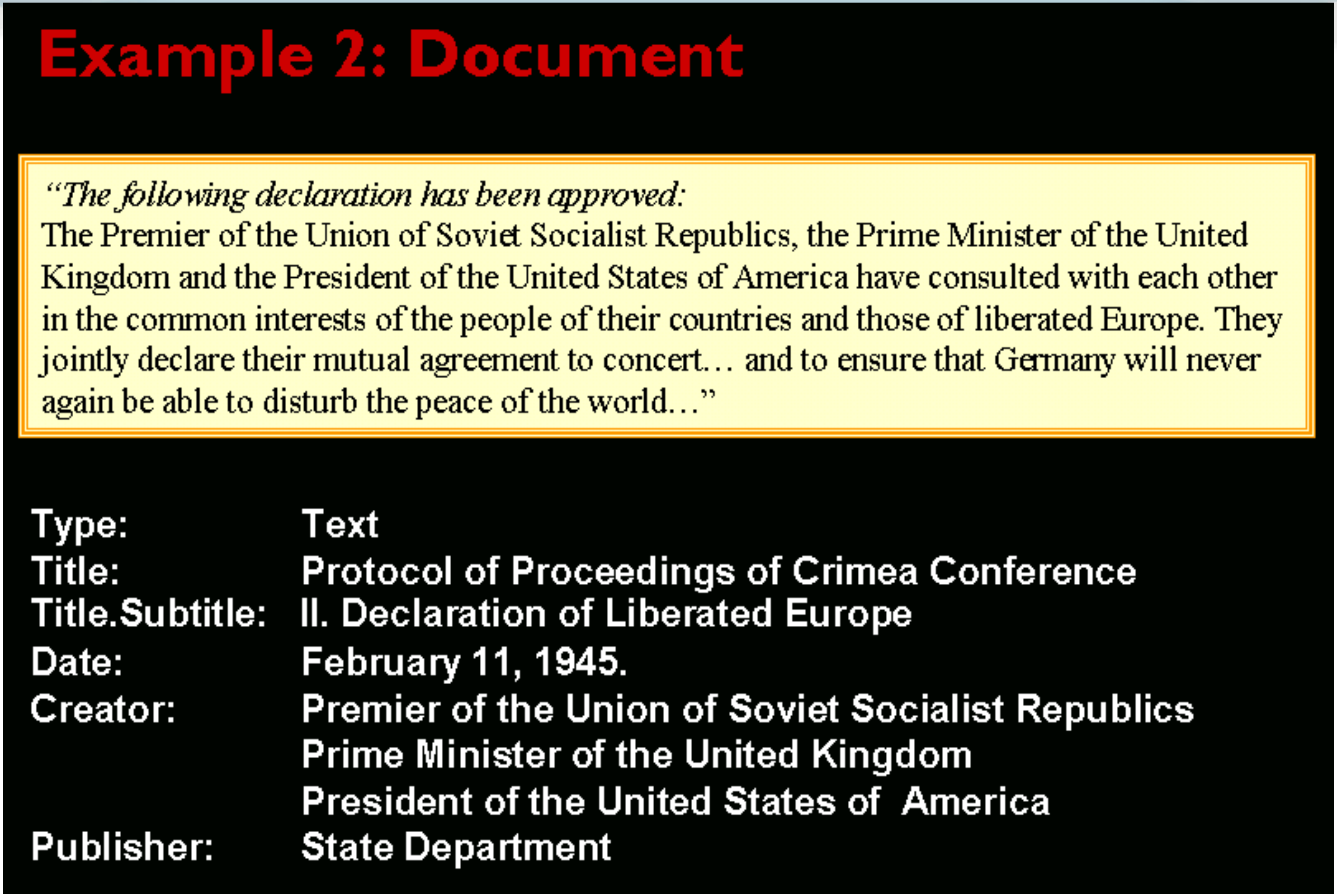
Property	Domain Class	Range Class
P72 has language (is language of)	E33 Linguistic Object	E56 Language
P102 has title (is title of)	E71 Man-Made Thing	E35 Title
P108 has produced (was produced by)	E12 Production	E24 Physical Man-Made Thing

M



Example 1: Photograph

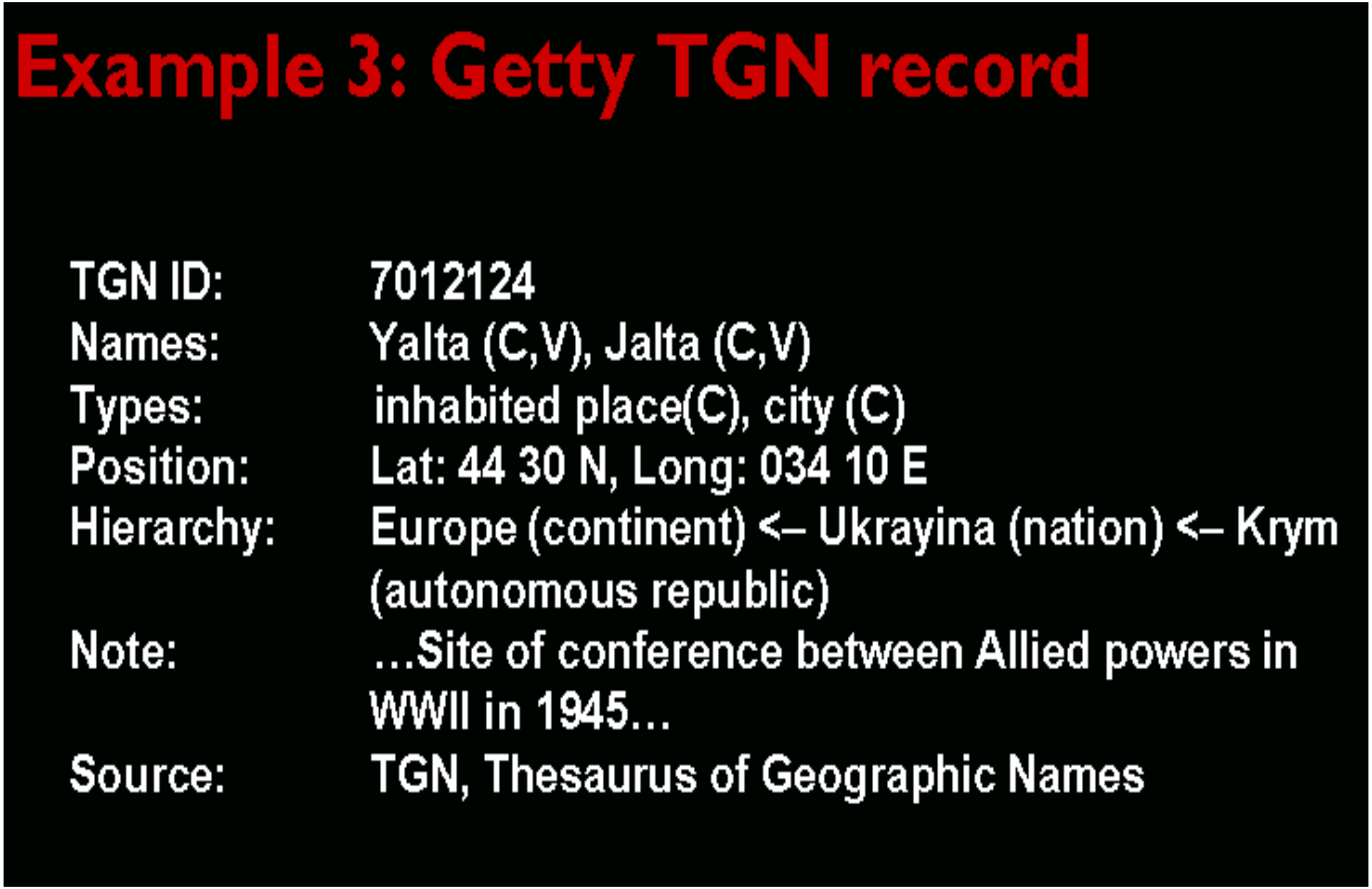
Type:	Image
Title:	Allied Leaders at Yalta
Date:	1945
Publisher:	United Press International (UPI)
Source:	The Bettmann Archive
Copyright:	Corbis
Keywords:	Churchill, Roosevelt, Stalin



Example 2: Document

> *"The following declaration has been approved:*
> The Premier of the Union of Soviet Socialist Republics, the Prime Minister of the United Kingdom and the President of the United States of America have consulted with each other in the common interests of the people of their countries and those of liberated Europe. They jointly declare their mutual agreement to concert… and to ensure that Germany will never again be able to disturb the peace of the world…"

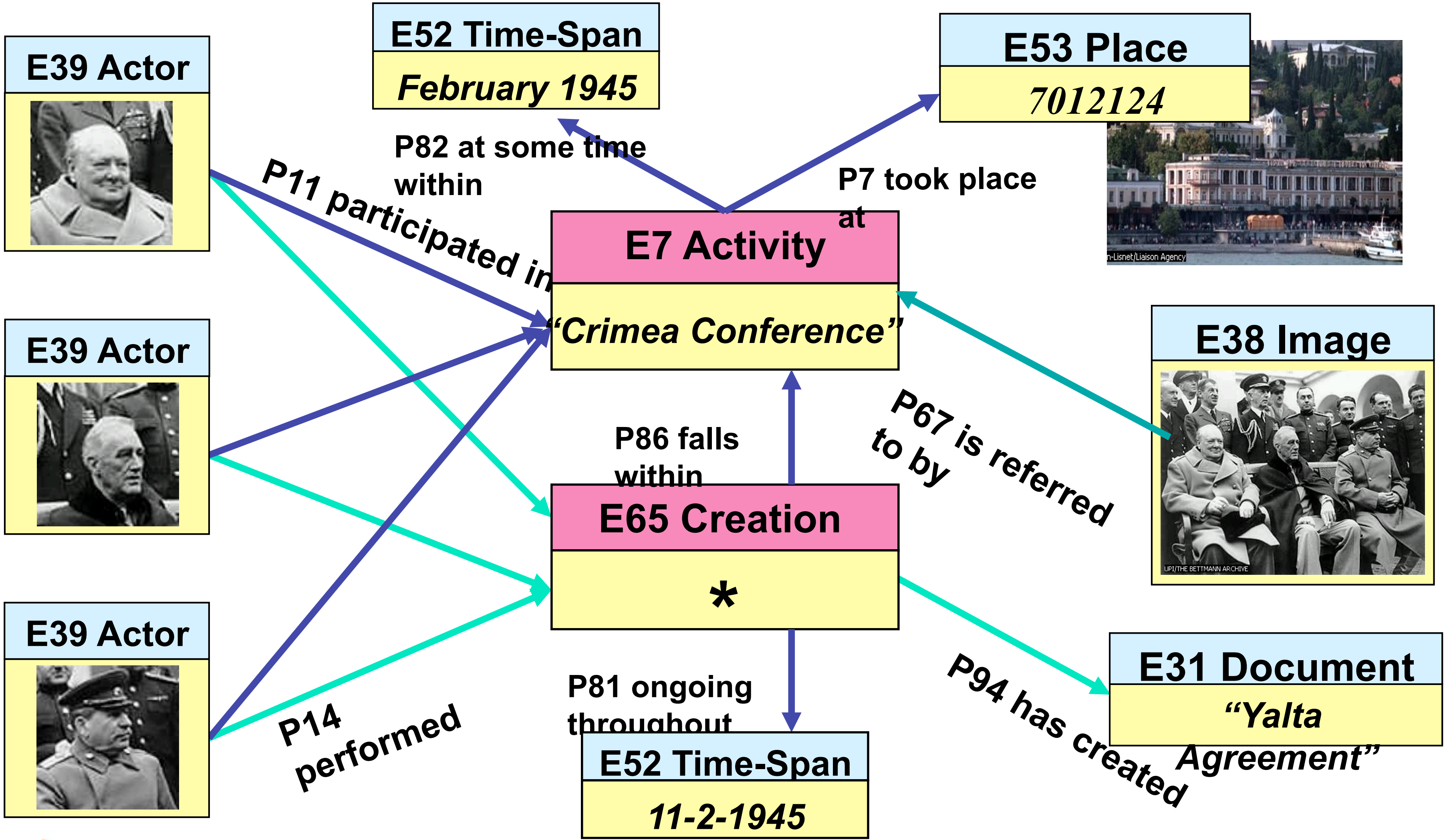
Type:	Text
Title:	Protocol of Proceedings of Crimea Conference
Title.Subtitle:	II. Declaration of Liberated Europe
Date:	February 11, 1945.
Creator:	Premier of the Union of Soviet Socialist Republics Prime Minister of the United Kingdom President of the United States of America
Publisher:	State Department



Example 3: Getty TGN record

TGN ID:	7012124
Names:	Yalta (C,V), Jalta (C,V)
Types:	inhabited place(C), city (C)
Position:	Lat: 44 30 N, Long: 034 10 E
Hierarchy:	Europe (continent) <– Ukrayina (nation) <– Krym (autonomous republic)
Note:	…Site of conference between Allied powers in WWII in 1945…
Source:	TGN, Thesaurus of Geographic Names

Event – based metadata integration



E39 Actor

E39 Actor

E39 Actor

E52 Time-Span

February 1945

P82 at some time within

E53 Place

7012124

P7 took place at

P11 participated in

E7 Activity

"Crimea Conference"

E38 Image

P67 is referred to by

P86 falls within

E65 Creation

*

P14 performed

P81 ongoing throughout

E52 Time-Span

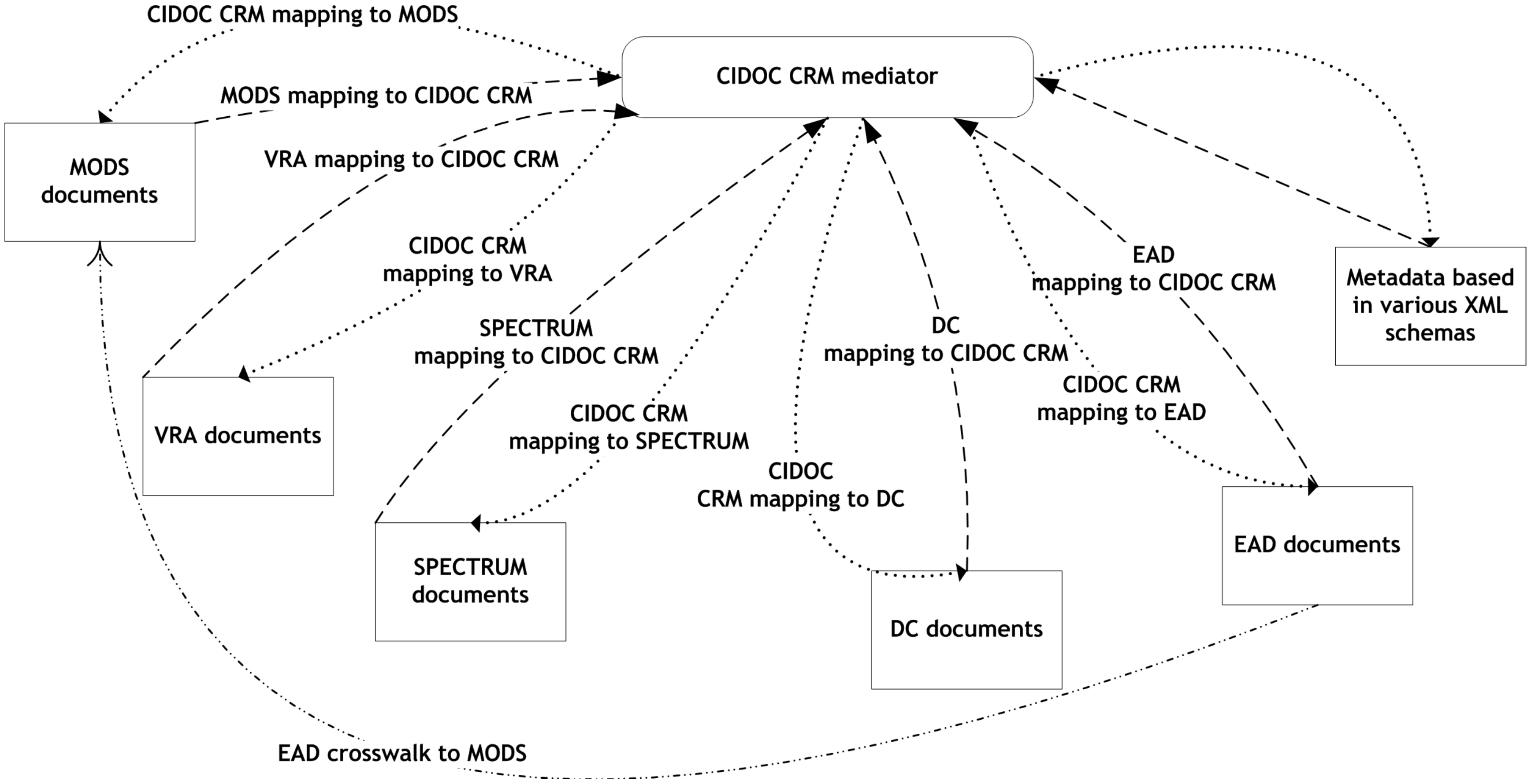
11-2-1945

E31 Document

"Yalta Agreement"

P94 has created

The integration architecture

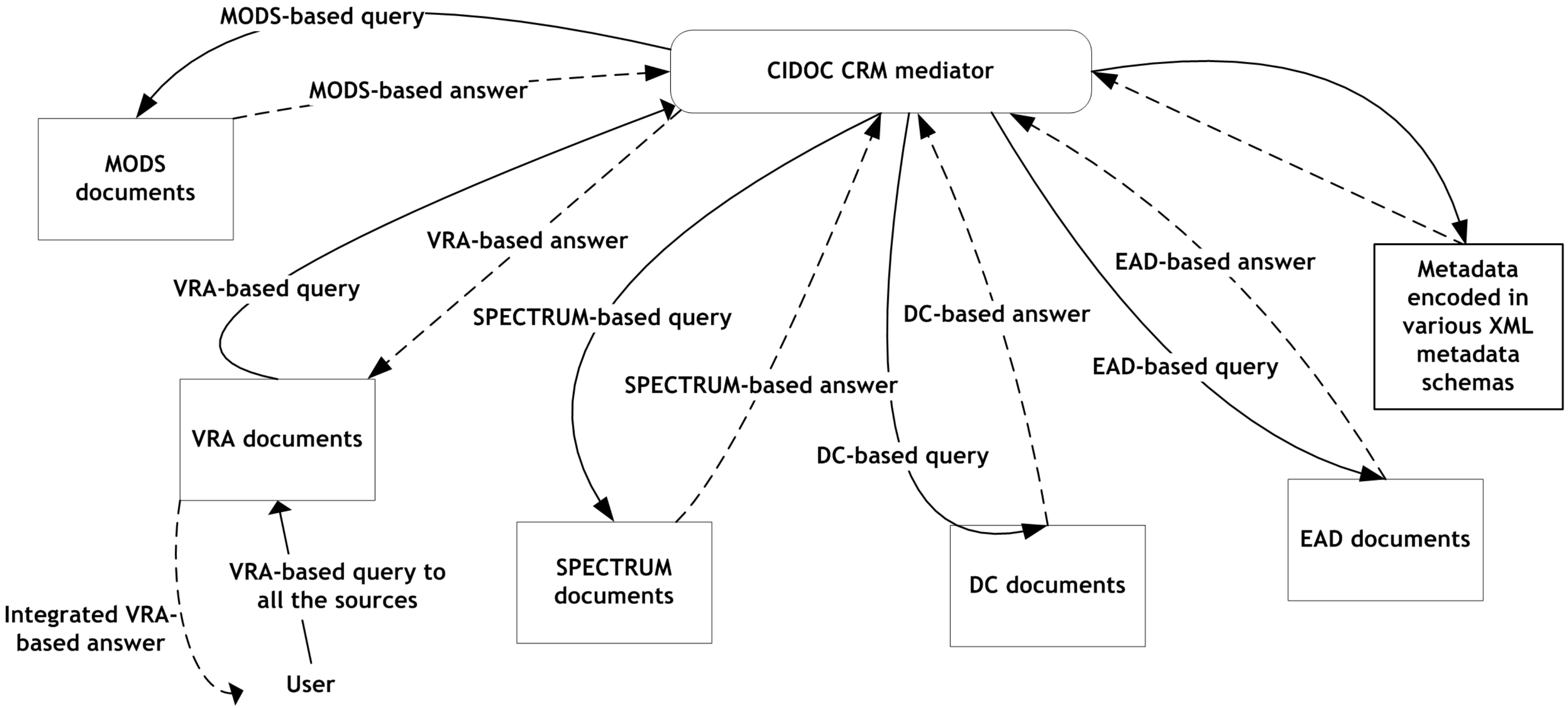


- Local metadata sources; XML-encoded metadata
- Mediator: CIDOC CRM; various roles, depending on 4 integration scenaria
- Each schema is semantically mapped to the mediator (which may retain its own database of metadata encoded in CIDOC CRM)
- There exist also crosswalks between the local metadata

Integration scenaria

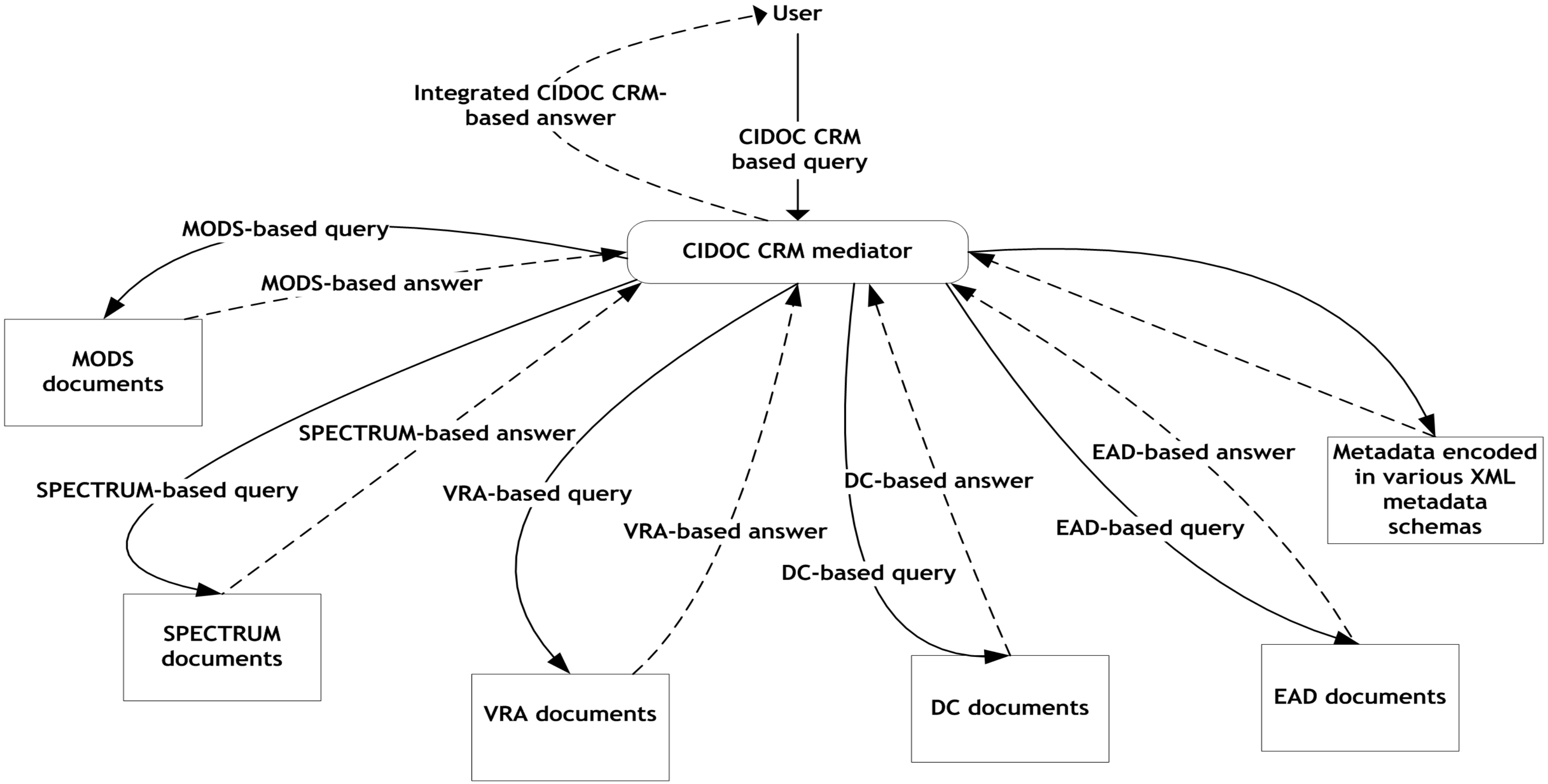
- Based on this integration architecture:
  - We present four integration scenaria based on the semantic mappings of various metadata schemas to CIDOC CRM
  - We define the ***Mapping Description Language*** to formally describe the mappings between the XML-based metadata and the ontology
    - We present a mapping from EAD to CIDOC CRM
  - We present an algorithm to transform EAD metadata to CIDOC CRM
  - We present an algorithm to transform XPath queries to queries expressed in terms of the ontology

Local query manipulation scenario



- Users can pose their queries to the local data sources following the format of the queries specified by these sources
- The local query engine promotes the query to the mediator, which based on the defined mappings translates the query to forms suitable to be answered by the other local sources.
- The results are collected and returned to the user

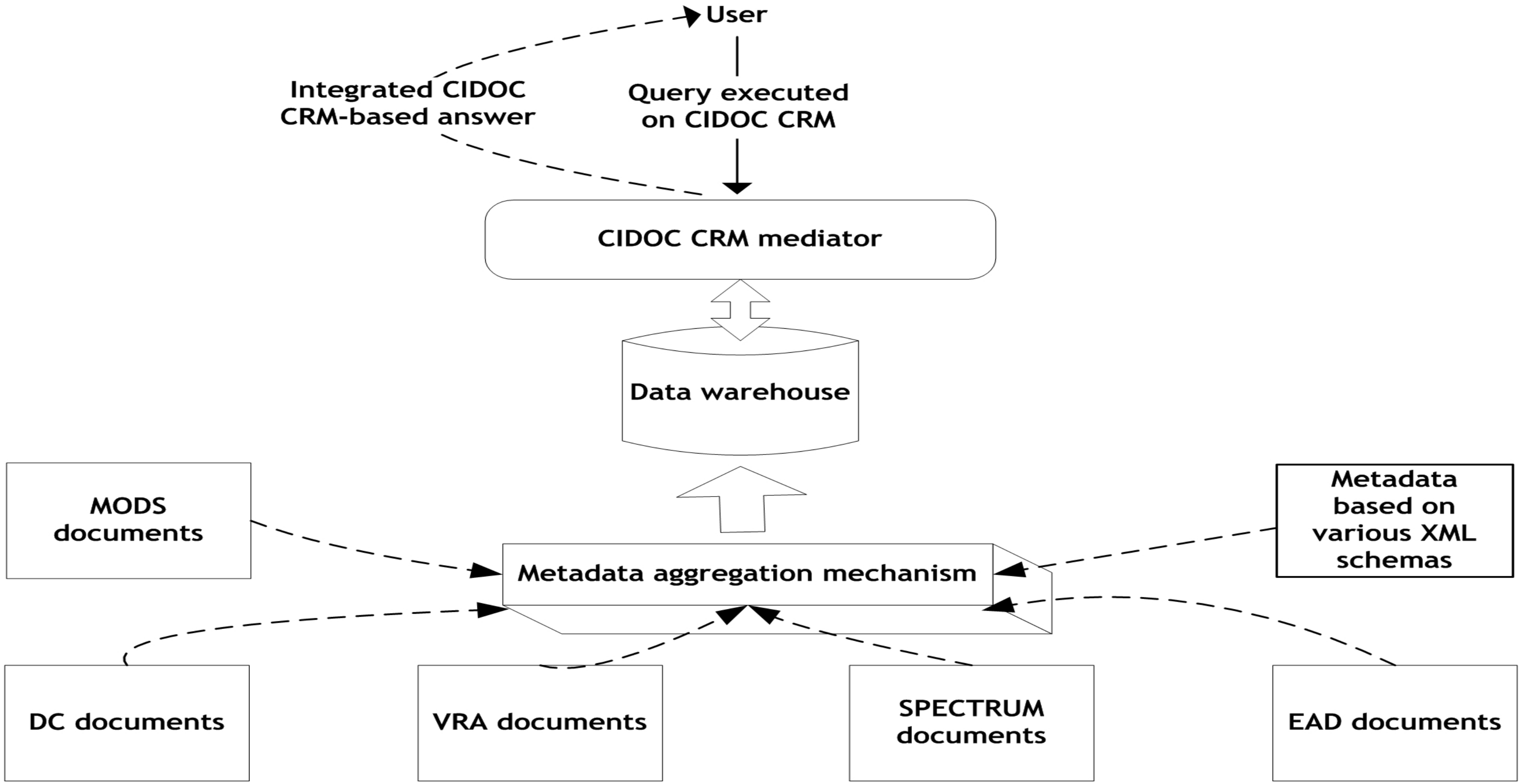
Global query manipulation scenario



- The user poses queries in CIDOC CRM terms to the mediator, which also acts as the query schema
- The mediator returns the result sets from its database (in case it provides) and translates/promotes the query to the local XML data sources
- The local query engines respond to the query and return the result sets to the mediator
- The mediator translates the result in terms of the ontology and returns them to the user

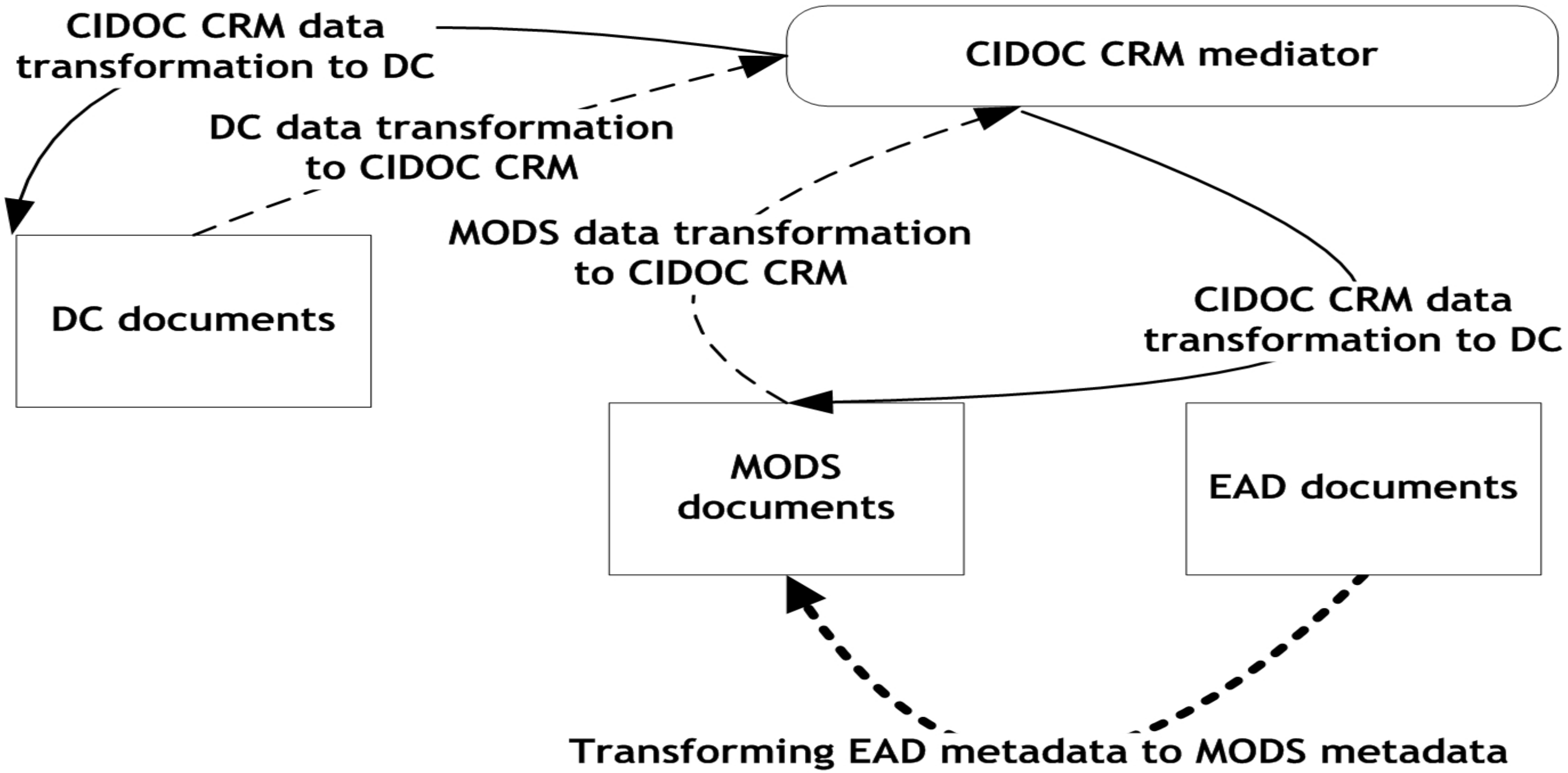
DBIS
database & information systems group
ionian university

Metadata warehouse scenario


User
Integrated CIDOC CRM-based answer
Query executed on CIDOC CRM
CIDOC CRM mediator
Data warehouse
MODS documents
Metadata aggregation mechanism
Metadata based on various XML schemas
DC documents
VRA documents
SPECTRUM documents
EAD documents

- The metadata from the local XML data sources are collected from time to time, translated (according to the predefined mappings) into CIDOC CRM terms and, finally stored in the mediator's data warehouse
- The user poses a query to the mediator, acting as the query schema, and the mediator's search engine returns the result sets collected from its data warehouse, without promoting the query to the local sources

Metadata exchange scenario


CIDOC CRM data transformation to DC
CIDOC CRM mediator
DC data transformation to CIDOC CRM
MODS data transformation to CIDOC CRM
DC documents
CIDOC CRM data transformation to DC
MODS documents
EAD documents
Transforming EAD metadata to MODS metadata

- Handle the transformation of a metadata to another metadata set
- Directly between the local data sources for which Crosswalks have been defined
- Between the local XML data sources and the ontology, for cases that either there are no Crosswalks between the local XML data sources or there is a growing number of metadata schemas that have to be mapped to each other
- Needed even inside a single institution

The Mapping Description Language (MDL) (1)

- The ***Mapping Description Language (MDL)*** is a formal language that expresses the mapping rules between a source schema and a target schema, based on a *path-oriented approach*.
  - We map the paths of the source schema to paths of the target schema
- Metadata are XML-based, hence source paths are *XPath location paths* enriched with *variables* and *stars*.
- The ***MDL rules*** have the following syntax:
  - *Left part*: extension of XPath
  - *Right part*: sequence of CIDOC CRM class/property path(s)
  - *variables*: declare and refer to branching points
  - *stars*: declare the transfer of value from the XML element/attribute to the corresponding class' instance

The Mapping Description Language (MDL) (2)

R ::= Left ‘- -’ Right

Left ::= A_{Path} | V_{Path}

A_{Path} ::= ∈ | ‘/’ R_{Path}

R_{Path} ::= L | L `*’ | L `{’ V_i`}’ | L `*’ `{’ $V_i$`}’ (L represents an XPath location path, V_i represents a location variable, declaring the new branches in the XPath paths; ‘*’ denotes a class instance; a variable name enclosed in curly brackets “{...}”, means that the path defined by the left part of this rule is assigned to this this variable,)

V_{Path} ::= ‘$’ V_i ‘/’ R_{Path} |`$’ $V_i$ `{’ V_i`}’ (a path starting with a variable name V prefixed with the dollar symbol (‘$’), denotes that a path fragment is prefixed by another path represented by the value of the variable V)

Right ::= E_e | E_e ‘→’ O | `$’ $V_c$ ‘→’ O | `$’ V_p ‘→’ ‘E55’ (Vc: it represents a class variable declaring that one ore more CIDOC CRM paths may begin with a class; Vp represents a property variable)

O ::= P_e ‘→’ E_e

O ::= O ‘→’ O

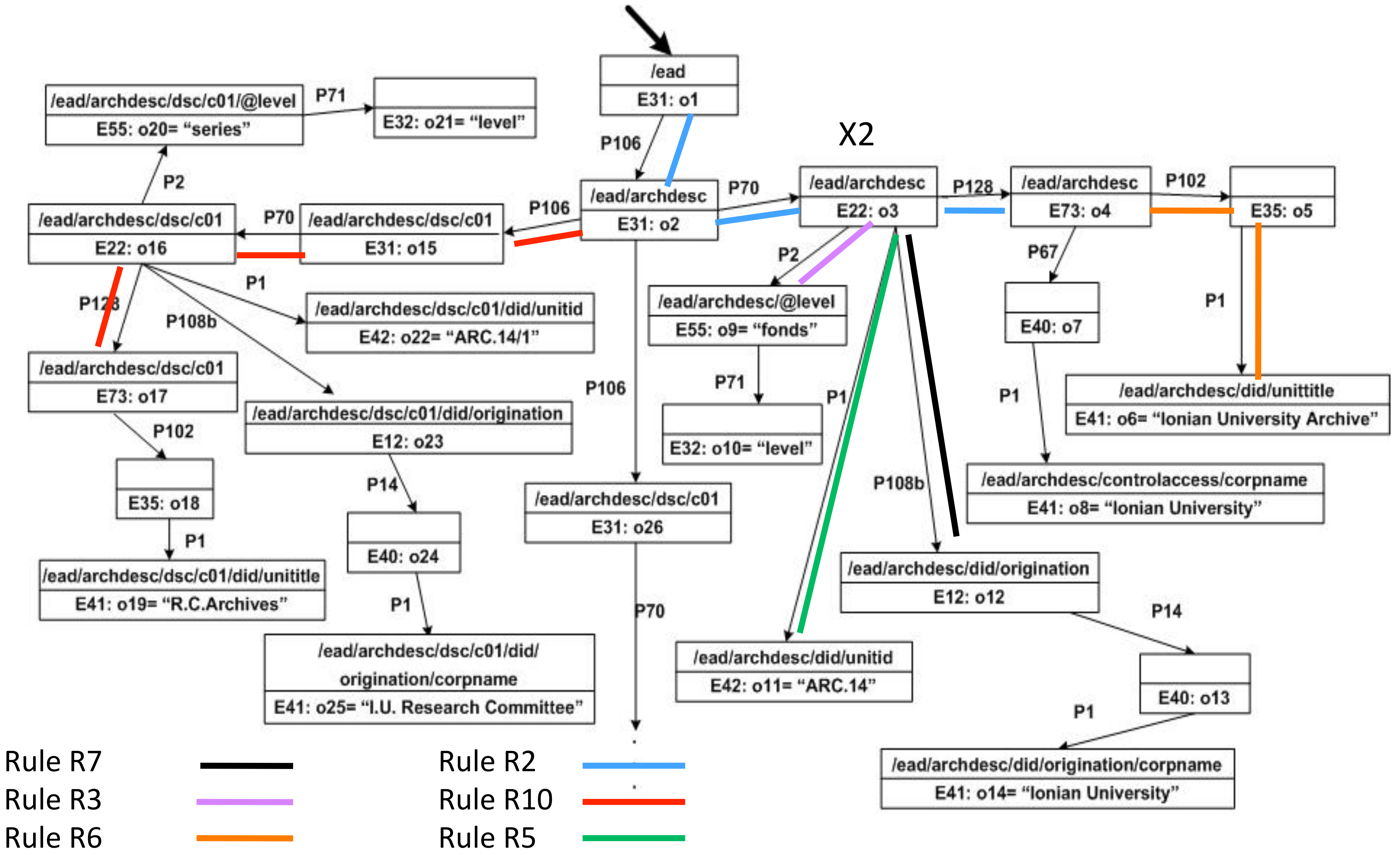
E_e ::= E | E `{’ $V_c$ `}’ | E `{=’ String `}’

P_e ::= P | P `{’ $V_p$ `}’

Mapping Rules in MDL: the EAD to CIDOC CRM case

Rule number	XPath location paths	CIDOC CRM paths
R1	/ead{X0}	E31{D0}
R2	$X0/archdesc{X2}	$D0->P106->E31{D2} ->P70->E22{A0} ->P128->E73{I0}
R3	$X2/@level*{Y2}	$A0->P2->E55{A01}
R4	$Y2	$A01->P71->E32 (= level)
R5	$X2/did/unitid*	$A0->P1->E42
R6	$X2/did/unittitle*	$I0->P102->E35->P1->E41
R7	$X2/did/origination{X22}	$A0->P108b->E12{A03}
R8	$X22/corpname*	$A03->P14->E40->P1->E41
R9	$X2/controlaccess/corpname*	$I0->P67->E40->P1->E41
R10	$X2/dsc/c01{X24}	$D2->P106->E31{D3}->P70->E22{A1} ->P128->E73{I1}
R11	$X24/@level*{Z2}	$A1->P2->E55{A11}
R12	$Z2	$A11->P71->E32 (= level)
R13	$X24/did/unittitle*	$I1->P102->E35->P1->E41

EAD to CIDOC CRM mapping: Branching variables



DBIS
database & information systems group
ionian university

Transforming EAD to CIDOC CRM

- Input: An EAD Document D and A set M of mapping rules
- Output: A CIDOC CRM graph G
- The XML tree of the source metadata is traversed in a depth-first, left-to-right manner
- The path fragments created during the tree traversal are checked for matching against the left part of each MDL rule
- If such a rule is detected, then the right part of this rule is applied in order to create a part of the CIDOC CRM graph
- The above process is applied repetitively for all sub- trees of the original tree, until the full traversal of the original tree is completed

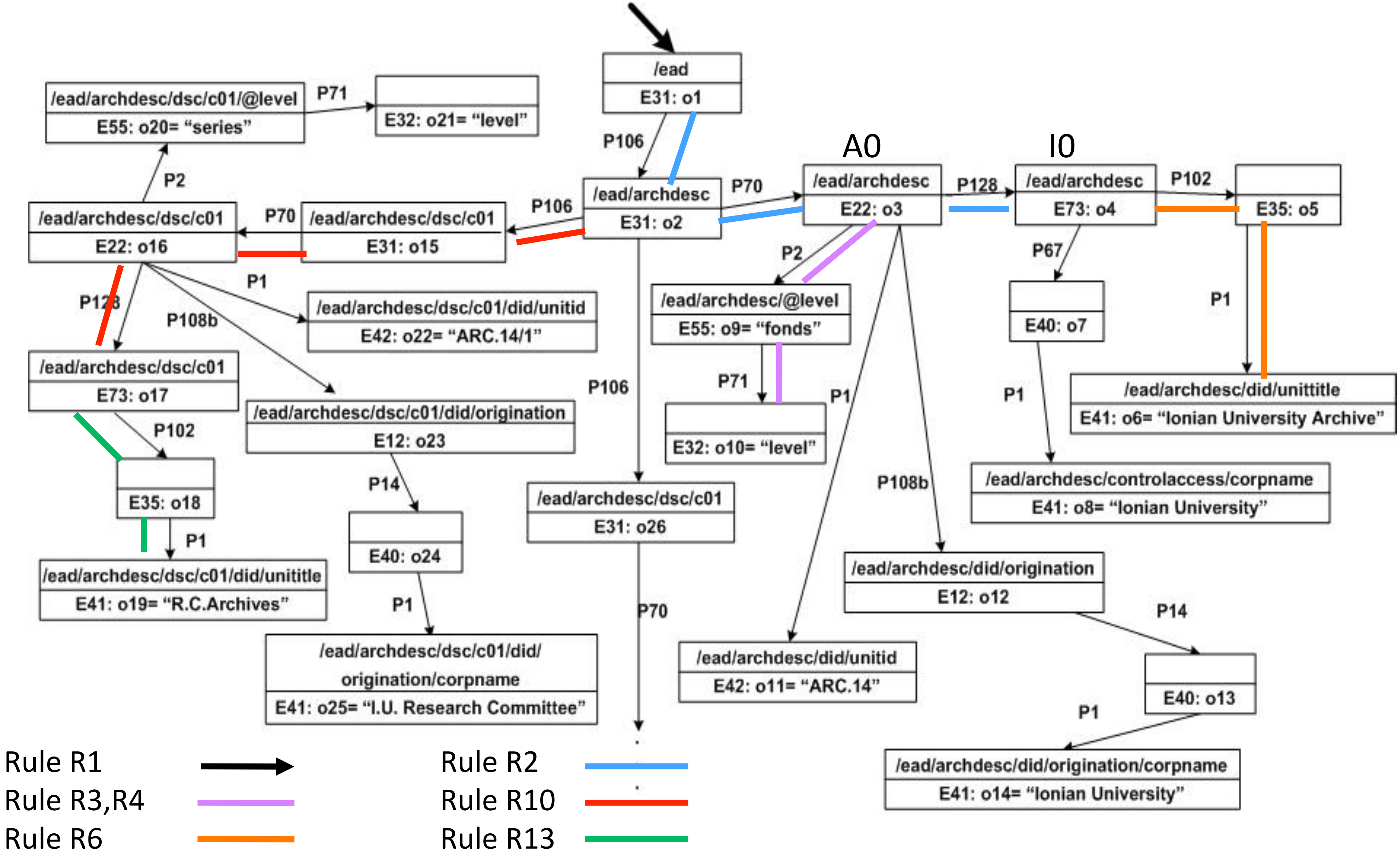
An EAD document

```
<ead>
  <eadheader>...</eadheader>
  <archdesc level="fonds"> (R1 – R4)
            <did>
                          <unitid countrycode="GR" repositorycode="IU">ARC.14</unitid>
                          <unittitle>Ionian University Archive</unittitle> * (R6)
                          <unitdate>1984 - 2007</unitdate>
                          <origination>
                                       <corpname>Ionian University</corpname>
                          </origination>
            </did>
            <bioghist>
                          <p>The Ionian University was founded in 1984...</p>
            </bioghist>
            <controlaccess>
                          <corpname>Ionian University</corpname>
                           <corpname>Ministry of Education</corpname>
            </controlaccess>
            <dsc>
                          <c01 level="series">
                                       <did>
                                         <unitid countrycode="GR“ repositorycode="IU">ARC.14/1</unitid>
                                         <unittitle>R. C. Archives</unittitle>
                                         <unitdate>1998 - 2007</unitdate>
                                         <origination><corpname>I.U. Research Committee</corpname></origination>
                                       </did>
                          </c01>
                          <c01 level="..."> ... </c01>
            </dsc>
  </archdesc>
</ead>
```

Mapping Rules in MDL: indicative fragment for the EAD case

Rule number	XPath location paths	CIDOC CRM paths
R1	/ead{X0}	E31{D0}
R2	$X0/archdesc{X2}	$D0->P106->E31{D2} ->P70->E22{A0} -> P128->E73{I0}
R3	$X2/@level*{Y2}	$A0->P2->E55{A01}
R4	$Y2	$A01->P71->E32 (= level)
R5	$X2/did/unitid*	$A0->P1->E42
R6	$X2/did/unittitle*	$I0->P102->E35->P1->E41
R7	$X2/did/origination{X22}	$A0->P108b->E12{A03}
R10	$X2/dsc/c01{X24}	$D2->P106->E31{D3}->P70->E22{A1} ->P128->E73{I1}
R11	$X24/@level*{Z2}	$A1->P2->E55{A11}
R12	$Z2	$A11->P71->E32 (= level)
R13	$X24/did/unittitle*	$I1->P102->E35->P1->E41

A fragment of the EAD to CIDOC CRM transformation



database & information systems group
ionian university

Query Transformation Algorithm

- MDL rules are used to transform XPath queries posed on XML-based metadata to equivalent queries on CIDOC CRM, written in an RQL-like syntax
- XPath queries allow the *child axis* and *predicates.*
- The RQL-like syntax allows queries in the form of s*elect-from-where* clauses
  - *select*: the requested variables
  - *from*: data paths expresses as triples of the form class – property –class
  - *where*: used for data filtering
- The transformation algorithm includes two phases

Generalized CIDOC CRM data path expressions

- The input of Phase 1 consists of a) an XPath query ***X***, and b) the mapping rules ***M***
- The output is a sequence ***S*** of ***generalized CIDOC CRM data path expressions*** (***GDPEs***)

$$CE_0 \text{ -> } P_1 \text{ -> } CE_1 \text{ -> } P_2 \text{ -> } \ldots \text{ -> } P_n \text{ -> } Ce_n$$

- A class expression CEj consists of a class id E followed by an optional class variable definition. CE_0 may also be of the form $Vc, where Vc is a class variable
- CE_n may be followed by a \`+' sign and/or a predicate
- Class variable definitions introduce linking points between GDPEs, while class variables prefixed by \`$' refer to these linking points
- The symbol \`+' marks the class whose instance corresponds to the query result.
- Predicates use comparison operators (=, >, >=, <, =) *to impose constraints* on the values of class instances

Query Transformation – Phase 1

- Given an XPath query ***X***, and the mapping rules ***M***
- Phase 1 traverses ***X depth-first-left-to-right*** and decomposes it into smaller paths matching the left parts of rules in ***M***
- Based on the right part of the rules, the corresponding ***GDPEs*** are constructed

A simple XPath query

- *"Find the title of the archive"*

XPath: /ead/archdesc/did/unittitle

- Applying Phase 1, the following ***GDPEs*** are produced:

/ead	E31{D0}	R1
/archdesc	$D0->P106->E31{D2} ->P70->E22{A0} -> P128->E73{I0}	R2
/did/unittitle	$I0->P102->E35->P1->E41+	R6

- The symbol "+" marks the class whose instance corresponds to the query result

Query Transformation – Phase 2

- ***GDPEs*** obtained in Phase 1 are transformed in RQL-like form.
  - The variable related to the class marked with “+” appears in the ***select***-part
  - To construct the ***from***-part,
    - a fresh data variable is associated with each class appearing in ***GDPEs***
    - triples of the form {X1;E1}P{X2;E2}, corresponding to GDPEs' sub-paths, are inserted, where the pairs {X1;E1} and {X2;E2} describe the domain and the range of the property P
  - The constraints appearing in ***GDPEs*** are used to construct the ***where***-part of the query

A simple XPath query

- *"Find the title of the archive"*

XPath: /ead/archdesc/did/unittitle

- Applying Phase 1, the following ***GDPEs*** are produced:

/ead	E31{D0}	R1
/archdesc	$D0->P106->E31{D2} ->P70->E22{A0} -> P128->E73{I0}	R2
/did/unittitle	$I0->P102->E35->P1->E41+	R6

  - The symbol "+" marks the class whose instance corresponds to the query result

```
select X6
from
{X1;E31_Document}P106_is_composed_of{X2;E31_Document},
{X2;E31_Document}P70_documents{X3;E22_Man-Made_Object},
{X3;E22_Man-Made_Object}P128_carries{X4;E73_Information_Object},
{X4;E73_Information_Object}P102_has_title{X5;E35_Title},
{X5;E35_Title}P1_is_identified_by{X6;E41_Appellation}
```

An XPath query

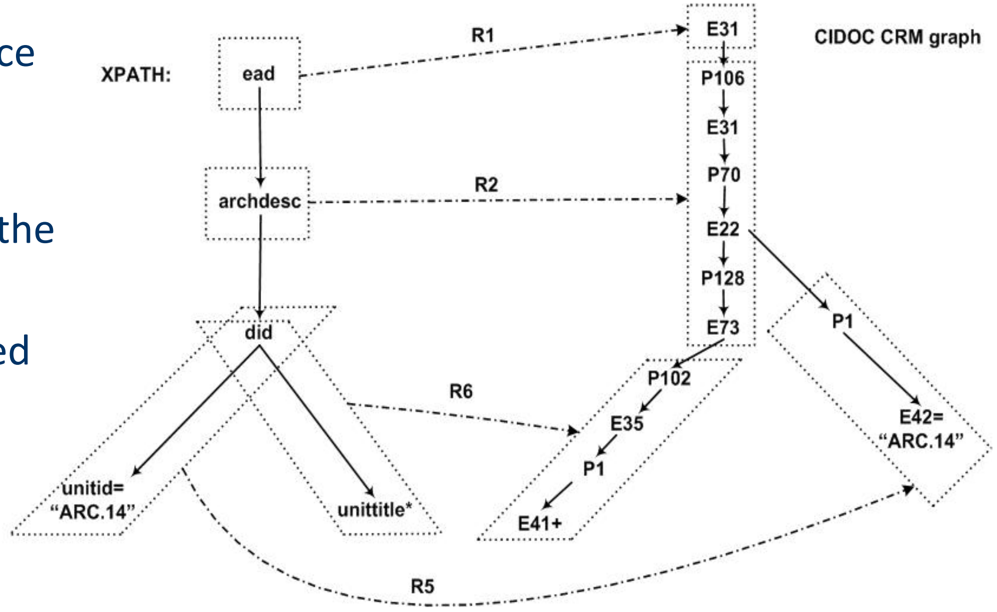
- *“Find the title of the archive identified as “ARC.14”on the EAD document.”*

  XPath: /ead/archdesc/did[unitid=“ARC.14”]/unittitle

- Applying Phase 1, the following ***GDPEs*** are produced:

/ead	E31{D0}	R1
/archdesc	$D0->P106->E31{D2} ->P70->E22{A0} -> P128->E73{I0}	R2
/did/unittitle	$I0->P102->E35->P1->E41+	R6
/did/unitid=“ARC.14”	$A0->P1->E42=“ARC.14”	R5

- The symbol “+” marks the class whose instance corresponds to the query result.
- Comparison operators (=, >, >=, <,<=) impose constraints (inherited from the predicates of the XPath query) on the values of class instances.
- Notice that the XPath has two branches rooted at the did on which we apply R5 and R6 resp.



The RQL – like query produced

“Find the title of the archive identified as “ARC.14”on the EAD document.”

XPath: /ead/archdesc/did[unitid=“ARC.14”]/unittitle

```
select X6
from
{X1;E31_Document}P106_is_composed_of{X2;E31_Document},
{X2;E31_Document}P70_documents{X3;E22_Man-Made_Object},
{X3;E22_Man-Made_Object}P128_carries{X4;E73_Information_Object},
{X4;E73_Information_Object}P102_has_title{X5;E35_Title},
{X5;E35_Title}P1_is_identified_by{X6;E41_Appellation},
{X3;E22_Man-Made_Object}P1_is_identified_by{X7;E42_Identifier}
where X7=“ARC.14”
```

Example 2 - Phase 1

"Find the identifier of the archive whose controlled access headings are corporate names having the values "Ionian University" and "Ministry of Education""

XPath : /ead/archdesc[controlaccess[corpname="Ionian University"][corpname="Ministry of Education"]]/did/unitid

Applying Phase 1, the following ***GDPEs*** are produced:

/ead	E31{D0}	R1
/archdesc	$D0->P106->E31{D2} ->P70->E22{A0} -> P128->E73{I0}	R2
/did/unitid	$A0->P1->E42+	R5
/controlaccess/corpname	$I0->P67->E40->P1->E41="Ionian University"	R9
/controlaccess/corpname	$I0->P67->E40->P1->E41="Ministry of Education"	R9

Inside the EAD document there are two `corpname` elements under the `controlaccess` element, hence *the R9 will be used twice*

Example 2 – Phase 2

```
select X9
from
{X1;E31_Document}P106_is_composed_of{X2;E31_Document},
{X2;E31_Document}P70_documents{X3;E22_Man-Made_Object},
{X3;E22_Man-Made_Object}P128_carries{X4;E73_Information_Object},
{X4;E73_Information_Object}P67_refers_to{X5;E40_Legal_Body},
{X4;E73_Information_Object}P67_refers_to{X7;E40_Legal_Body},
{X5; E40_Legal_Body}P1_is_identified_by{X6;E41_Appellation},
{X7; E40_Legal_Body}P1_is_identified_by{X8;E41_Appellation},
{X3;E22_Man-Made_Object}P1_is_identified_by{X9;E42_Identifier}.
where X6 = “Ionian University”
          X8 = “Ministry of Education”
```

Query Transformation – Phase 1

- A common fragment of different branches in the XPATH query matches with the same rule
- A rule may be used multiple times
- Keep track of the class variable definitions and associate them with appropriately chosen data variables
    - a *variables' renaming mechanism* provides appropriate names to the class variables of the right hand side of this rule, ensuring the consumption of the correct version of them
    - A rule counter C(R) is associated to each R in M. Each time a specific rule R is selected, then C(R) increases by 1
    - If C(R) > 1 then a fresh variable constitutes the class variables appeared in R

Conclusions and Outlook

- Conclusions
  - MDL: part of a CIDOC CRM-based metadata integration scenario
  - The methodology proposed transforms widely known XML-based metadata, such as VRA and EAD, to CIDOC CRM
  - Development of a demo that generates RQL queries and PROLOG queries
- Future work
  - Expansion to other XPath axes and expressions, such as wild characters, etc.
  - Expansion to other ontology query languages such as SPARQL