International Journal of Cooperative Information Systems
© World Scientific Publishing Company

A DISTRIBUTED INFRASTRUCTURE FOR EARTH-SCIENCE BIG DATA RETRIEVAL

PANAGIOTIS LIAKOS*, PANAGIOTA KOLTSIDA† and GEORGE KAKALETRIS‡

Athena Research and Innovation Center, 15125 Maroussi, Greece
*{*p.liakos,‡gkakas}@di.uoa.gr, †p.koltsida@imis.athena-innovation.gr*

PETER BAUMANN

Jacobs University Bremen, 28759 Bremen, Germany
p.baumann@jacobs-university.de

YANNIS IOANNIDIS¶ and ALEX DELIS§

Athena Research and Innovation Center, 15125 Maroussi, Greece
University of Athens, 15784 Athens, Greece
¶yannis@athena-innovation.gr, §ad@di.uoa.gr

Received (Day Month Year)
Revised (Day Month Year)

Earth-Science data are composite, multidimensional and of significant size, and as such, continue to pose a number of on-going problems regarding their management. With new and diverse information sources emerging as well as rates of generated data continuously increasing, a persistent challenge becomes more pressing: to make the information existing in multiple heterogeneous resources readily available. The widespread use of the XML data-exchange format has enabled the rapid accumulation of semi-structured metadata for Earth-Science data. In this paper, we exploit this popular use of XML and present the means for querying metadata emanating from multiple sources in a succinct and effective way. Thereby, we release the user from the very tedious and time consuming task of examining individual XML descriptions one by one. Our approach, termed `Meta-Array Data Search (MAD Search)`, brings together diverse data sources while enhancing the user-friendliness of the underlying information sources. We gather metadata using different standards and construct an amalgamated service with the help of tools that discover and harvest such metadata; this service facilitates the end-user by offering easy and timely access to all metadata. The main contribution of our work is a novel query language termed `xWCPS`, that builds on top of two widely-adopted standards: `XQuery` and the `Web Coverage Processing Service (WCPS)`. `xWCPS` furnishes a rich set of features regarding the way scientific data can be queried with. Our proposed unified language allows for requesting metadata while also giving processing directives. Consequently, the `xWCPS`-enabled `MAD Search` helps in both retrieval and processing of large data sets hosted in an heterogeneous infrastructure. We demonstrate the effectiveness of our approach through diverse use-cases that provide insights into the syntactic power and overall expressiveness of `xWCPS`. We evaluate `MAD Search` in a distributed environment that comprises five high-volume array-databases whose sizes range between 20–100 `GB` and so, we ascertain the applicability and potential of our proposal.

Keywords: Array databases; declarative query language; scientific data.

1. Introduction

Contemporary scientific instruments not only offer extraordinary precision and yield high-quality acquisition data but also their machinery continues to improve. These advances inevitably lead to the creation of peta-scale data volumes and it is estimated that their size will approximately double every year[20]; the latter sets scientific investigation to be the predominant cause for large dataset generation. Although this generation is in general of great value, its expanding volume makes it very difficult to manage. This is especially the case for Earth-Science research, where the available data analysis tools are unable to keep pace with the unprecedented growth of data, and in an effort to do so, alienate their users with complicated interfaces[20].

Scientific data entails significant diversity in the data structures used. By and large, such data is usually multidimensional and complex in nature, its updates are mostly append-only, and it is accompanied with metadata[6]. In general, metadata is of utmost importance as it includes descriptive information on how the data was acquired, computed and/or measured. The heterogeneity and diversity of scientific data and its descriptive metadata representations, as well as the absence of a common information space, makes the retrieval of Earth-Science data difficult[24]. Exploiting such data calls for open, unified, and seamless access as well as ad-hoc analytics on multidisciplinary earth data repositories. Implementing interoperable infrastructures to provide access to Earth-Science data is a challenging problem that we believe can be addressed by exploiting advances in various domains.

Traditional database management systems (DBMSs) have been shown to be inefficient in supporting core scientific data types such as arrays[20]. This weakness has led to the development of array DBMSs including `rasdaman`[8] and `SciDB`[13]. These systems address the above requirement by extending the set of supported data structures in relational databases with multidimensional arrays of unlimited size. This enables the efficient storage of n-D spatiotemporal scientific data generated by satellites, telescopes and sensors. In general, this data-payload must be associated to its geospatial domain which in turn leads to the concept of *coverages* which are digital geospatial information representing space/time-varying phenomena[11]. Moreover, the *Open Geospatial Consortium (OGC)*[a] defines standards providing the directives to implement simple and easy-to-use web-based interfaces that offer operations atop an array-DBMS. To this end, search, retrieval and processing of array-data becomes more user-friendly. Finally, the *World Wide Web Consortium (W3C)*[b] provides standard definitions, such as XML[12] and XQuery[15] that jointly enable the efficient encoding and querying of metadata. To effectively re-

[a]`www.opengeospatial.org`
[b]`www.w3.org`

trieve scientific data, we must employ *W3C* standards for such an exercise would be meaningless without using the companion descriptive information.

Our approach harnesses both key features of contemporary array-databases and recent developments in *OGC* and *W3C* standards to tackle the issue of fusing cross-disciplinary scientific data with descriptive metadata. Figure 1 illustrates how `MAD Search` delivers a novel way in acquiring data from array and semi-structured data-sources.

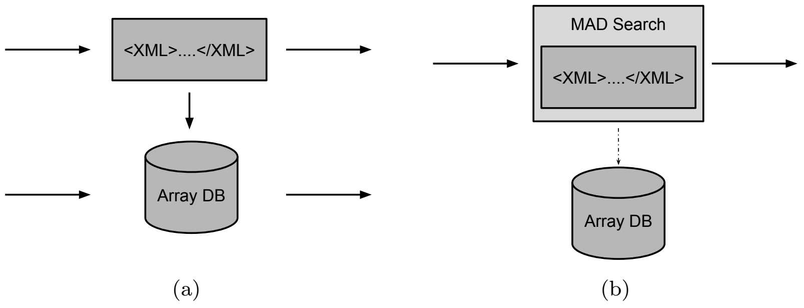


Fig. 1: Information retrieval over array- and semi-structured databases without (a) and with (b) `MAD Search`

Figure 1a illustrates how conventional querying of XML in combination with array-database information is now carried out: it is possible to directly query the array-database as long as we are well aware of the name/content of the *coverage* sought. However, this is a non-trivial proposition. Usually, we have to query the corresponding descriptive metadata in order to initially identify suitable *coverages* and subsequently, question the actual data resident in the array-database. It is obvious that the above means of querying is fragmented, cumbersome and rather ineffective as we end up with two individual sets of responses that the user has to manually combine. Our approach, depicted on Figure 1b, specifies a language that enables the composition of queries involving both types of data —meta and array-data— and offers a framework able to handle composite requests and deliver combined results. Our hypothesis is that the unification of `WCPS` and `XQuery`, two widely adopted standards targeting array and structured data respectively, may yield a powerful query language. The specification and subsequent implementation of `xWCPS` enables us to morph a complete array and semi-structured multi-database model for scientific data information retrieval. In summary, this paper makes the following contributions:

- Defines a query language designed to allow seamless integration of array data with metadata for search, retrieval, and processing applications.
- Presents an architecture for cooperative distributed information retrieval over array and semi-structured databases. We develop software components

that assist in acquiring resource descriptions, selecting the datasources to be queried, retrieving and presenting the desired information.

- Realizes a search engine to show the expressiveness of our language and evaluates the effectiveness of our architecture in an infrastructure consisting of voluminous array databases.
- Presents a number of use-cases that existing query languages operating atop array databases were unable to handle. Based on these use-cases, we demonstrate the syntactic effectiveness of our proposed language. With `xWCPS`, we can now query over array data and its accompanying metadata in a unified manner, retrieving results that previously were unreachable or required excessive effort. Moreover, `MAD Search` succeeds in offering search services in a distributed environment that entails both array and semi-structured databases. The overhead induced to automate this procedure is experimentally found to be insignificant.

The organization of this paper is as follows: Section 2 presents an overview of the query languages used as starting points for `xWCPS` as well as our motivation for the proposal of `MAD Search`. Section 3 discusses how resource descriptions and other available metadata are gathered while Section 4 specifies `xWCPS` and outlines its features. Section 5 details the `MAD Search` architecture and offers an overview of its core functionalities. In Section 6 we outline the implementation aspects of our work. A set of use-cases used to ascertain the expressiveness of `xWCPS` and an evaluation of `MAD Search` are presented in Section 7. Section 8 reviews the related work, and finally, concluding remarks and future work directions are found in Section 9.

2. Background and Motivation

The fundamental idea of this work is to merge two well-known standards into a unified query language in order to provide access to array data and its descriptive metadata seamlessly. In this section, we discuss the building blocks of our query language, i.e., the `XQuery` and `WCPS` languages and present their limitations.

2.1. *XQuery*

The use of XML data has exploded in recent years and now a colossal amount of information is stored in XML databases or filesystem repositories. `XQuery`[15] is a query language designed by the W3C to enable the efficient utilization of this information. `XQuery` allows the user to select the XML data elements of interest, reorganize and possibly transform them, and return the results in a structure of her choosing.

The basic structure of `XQuery` queries is the `FLWOR` (pronounced "flower") expression. The acronym derives from the keywords permitted in `XQuery` expressions, namely `for`, `let`, `where`, `order by`, and `return`. Figure 2 provides a sample query[c].

[c]See Figures 6 and 8 for a better understanding of the document structure. The *coverages*

```
for $c in /server//coverages/coverage
let $comp := $c//*[local-name()='composition']
where $comp//*[local-name()='lithology']/@*[local-name()='title'] = 'Clay'
order by $c//*[local-name()='CoverageId']
return $c//*[local-name()='CoverageId']/text()
```

Fig. 2: An `XQuery` example[c]

The *coverages* contained in an XML document are iterated using variable `$c` and a descendant of each of these elements is kept in variable `$comp` using the `let` clause. The *coverages* are then filtered with a condition that keeps only those whose value for the `lithology` attribute is *'Clay'*. Using `order by`, the elements are sorted according to the `CoverageId` element's value, which is also what is added to the results the query returns.

As XML is a widely used format for the descriptive metadata of Earth-Science data, `XQuery` provides the means to query and manage it efficiently. However, `XQuery` cannot be used to query and process array data.

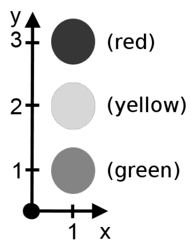
2.2. *OGC Standards*

Raster data acquired by satellites, sensors or telescopes are received as uncalibrated raw data to be turned into higher level products with the use of science data-processing algorithms. An example of a *coverage* representation is given in Figure 3. We consider the traffic light of Figure 3a and to represent it we use the 1×3 array of 3b. Each element of the array holds the `RGB` value of the color of the traffic light in the corresponding position. The array itself is of little use. However, a *coverage* also holds metadata information that enables the interpretation of the array's content. A typical real-life *coverage* usually involves relative geographic positions that are mapped to values corresponding to temperature measurements, in a similar, yet significantly larger, array, as well as descriptive information regarding the actual geographic location of the array contents, the time the measurements took place or the instruments that were used. This descriptive information is stored alongside the array data as accompanying metadata[d]. This duality of formats hampers the querying of *coverages* and existing solutions handle array data and metadata separately.

OGC is a voluntary-based standards organization aiming towards composing publicly available interface standards. OGC Standards detail interfaces and encodings for developing interoperable, Earth-Science-related solutions over the web. We

nodeset of Figure 6 has *coverage* elements as its children that hold a response similar to that of Figure 8.

[d]In `rasdaman`, the descriptive metadata of the array database used are XML documents stored in a separate table of a relational database. Figures 6 and 8 are examples of such metadata.



(a)

$$\begin{array}{ccr} (1,1) & \longrightarrow & (0,255,0) \\ (1,2) & \longrightarrow & (255,255,0) \\ (1,3) & \longrightarrow & (255,0,0) \end{array}$$

(b)

Fig. 3: A traffic light (a) and its *coverage* representation (b)

outline below the `Web Coverage Service (WCS)` and `Web Coverage Processing Service (WCPS)` standards that enable querying of *coverages*.

2.2.1. *OGC WCS*

The OGC Web Coverage Service (`WCS`)[10] supports retrieval of geospatial data as *coverages*. The `WCS` standard defines a data access service that allows *coverages* to be queried through HTTP based interfaces. The response to a WCS request includes data with its original semantics, which may be interpreted, extrapolated or processed in general, instead of just being portrayed. `WCS` allows retrieval based on subsetting[e], scaling, and re-projection. Additionally, requests that allow the retrieval of descriptive metadata accompanying the actual data are also defined. However, extracting values from metadata is not possible through `WCS`. Each of `WCS` requests regarding the metadata part of a *coverage* can only deliver the whole corresponding element as a response.

2.2.2. *OGC WCPS*

To further enhance `WCS` with *coverage* processing capabilities, OGC has defined `WCPS` that extends the OGC `WCS` by adding the `ProcessCoverage` request. This operation allows a client to request the processing of *coverages* using the WCPS language. `WCPS`[9] can be characterized as the "SQL for *coverages*" as it specifies a syntax for retrieval and processing of *coverages*. The reference implementation of `WCPS` is provided by `rasdaman`, the array database used in the context of this work.

The primary structure of the `WCPS` language comprises the `for - where - return` clauses. The `for` clause specifies the set of *coverages* that will be processed by the query. The `return` clause specifies the potential output that may be appended to the list of results in each iteration determined by the `for` clause. Criteria used for determining if the output of `return` is actually appended are specified

[e]Subsetting is the process of retrieving the parts of interest from large files.

```
for $s in ( A, B, C )
where some( $s.red > 127 )
return encode( $s , "png" )
```

Fig. 4: A WCPS query example

by the `where` clause. Towards enhancing the expressiveness of the language, WCPS defines *coverage* processing expressions and functions employed in the `where` and `return` clauses, such as *some()*, *max()* and *encode()*.

An example that offers insight into the WCPS structure is given in Figure 4. Consider a WCPS server offering the satellite images A, B and C and a case where the user wants to retrieve those that their red channel exceeds 127 somewhere. The query results in a list of at most three PNG-encoded *coverages* for A, B and C, depending on the evaluation of the `where` condition. As this example suggests, selecting, filtering, and processing array data is straight-forward through WCPS. However, the WCPS language does not provide any means of utilizing the corresponding descriptive metadata of the array data involved.

2.3. *Motivation*

In this subsection, we argue that the use of XQuery and WCPS presents limitations when querying array-data along with corresponding metadata and outline salient features of our proposal that address these constraints. We assume a use case in which we wish to retrieve the temperature and the geographic location of all *coverages* of a service provider that include a specific coordinate.

A key aspect of a database language is that a user must be only aware of the schema before she launches any queries. In our use case, we aim at selecting all *coverages* satisfying certain criteria without knowing additional details. However, WCPS requires the specification of *coverage* names in selection queries. These names can be retrieved only after issuing a WCS operation. Therefore, one has to request and scan through a usually large resource description enlisting thousands of *coverages* to discover what names should be used *before* issuing any WCPS query. This precursory step introduces overhead in the querying process, which significantly constrains the user-friendliness of the query language and undermines the overall user experience. Moreover, as the specification of the desired data is possible only with the use of *coverage* names, the ability to query over a distributed environment is limited. For example, there is no way of distinguishing two or more *coverages* bearing the same name. Therefore, eliminating the need for stating names in queries and providing enriched syntax for specifying the desired *coverages* would significantly ease the task of querying Earth-Science data.

Another very common feature a query language must offer is that of filtering results against user-designated criteria. Our example considers the case in which the user wishes to retrieve *coverages* that include a geographic coordinate. However, when a user asks for array data with `WCPS`, she is not able to specify conditions regarding their accompanying metadata. The only workaround to this problem is to fetch metadata of all involved *coverages*, filter them manually, and finally, issue `WCPS` requests only for the *coverages* that do actually satisfy the user-imposed criteria. Evidently, this course of action is rather impractical. An extension of the language, enabling the specification of condition criteria on both data and metadata, would allow the users to automatically filter results in much more meaningful ways than what is now possible.

Finally, when querying, it is extremely important that the user can include in the results all available information from the underlying databases. For instance, for our use case both array data (temperature) and metadata (geographic location) is required as an output. `WCS` operations allow the retrieval of metadata, but the process of acquiring them is not part of the `WCPS`, which fetches only array data. Therefore, manual processing is again required if the user aims at results containing *both* data and metadata. Ideally, this functionality should also be offered in an automated fashion through extensions of `WCPS`.

The current approach of using `WCS` and `WCPS` suffers from all the aforementioned limitations. This has motivated us to propose `xWCPS` that effortlessly handles the three issues discussed above through respective enhanced syntactic additions. Furthermore, the corresponding engine executing the actual queries is designed in order to service requests in the most effective way. Queries such as the one of our use case are easily expressed through `xWCPS` and the desired output is produced by our engine.

3. MAD Search Catalog

The `OGC WCS` standard describes operations that allow the retrieval of resource descriptions through HTTP requests. Additionally, metadata relevant to the content of a database are available in other resources. `MAD Search` employs a number of harvesting components in order to collect resource descriptions for the databases it covers, as well as available metadata that can be located alongside the actual data or in external sources. The goal of the `MAD Search Catalog` is to amalgamate all this information and make it available through a more effective interface that takes full advantage of the available metadata.

In this section, we first discuss on the library that handles the transactions with an XML database used to hold the metadata and the generic schema used. Then, we elaborate on how we gather information that is necessary in order to efficiently route queries in a distributed environment, instead of sending requests to irrelevant databases. Next, we outline the different ways in which we build a central metadata collection from various sources. Figure 5 illustrates the actions of `MAD Search` that are related to metadata harvesting. In particular, we collect resource descriptions

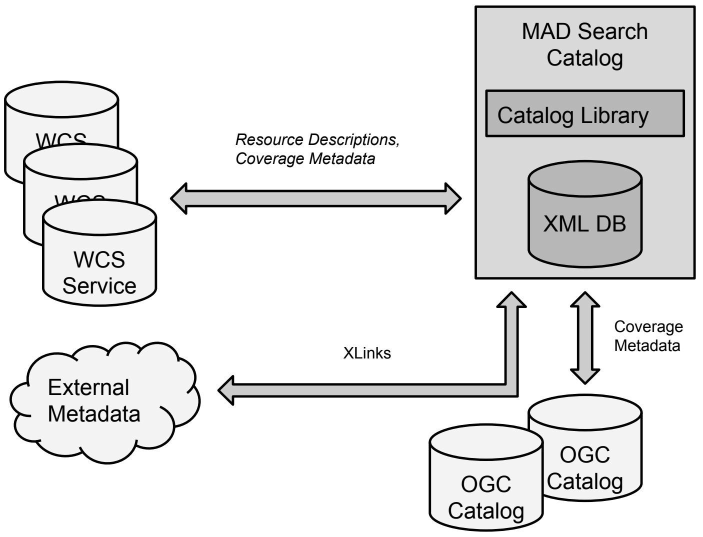


Fig. 5: The Metadata Harvesting Architecture

as well as metadata from `WCS` endpoints, and we complement this metadata from additional resources, i.e., `OGC Catalogs` and external `URLs`.

All the `OGC WCS` requests described in the following sections use the `HTTP GET` method.

3.1. *Catalog Library and Schema*

An essential part of `MAD Search` is the `Catalog Library`. In particular, this is the library that interacts with an `XML` database and brings out all the transactions with it. The requests are handled by issuing `XQuery` queries to the database, to ensure that the transactions are `ACID` safe and efficient. Moreover, the `Catalog Library` offers `APIs` for adding, modifying, deleting and locating `WCS` resources.

In order to hold all the available meta-information in a central location, we opted for certain constraints on the structure of the documents that are added to the `MAD Search Catalog`. The key elements of this generic schema are visible in Figure 6. For every co-operating database a document, such as the one of this figure, is hosted in the catalog, with the *endpoint* attribute of the *server* element specifying its endpoint. The schema of this document reserves a number of placeholders for all the available metadata accompanying the array data, as well as other types of information such as user annotations. Thereby, `MAD Search` is able to discover all this information in a central location by simply following some structure related conventions regarding these documents. In particular, the *capabilities* element holds the resource description of the database (see Section 3.2), *coverage* elements hold *coverage*-specific metadata (see Section 3.3), and the *user* element holds user defined metadata. Using the *harvestinterval* element we can specify the harvesting period, although all harvesters may also be triggered manually. As Earth-Science data has low update rates and most updates only append information[6], inconsistencies between harvestings rarely occur.

3.2. *Collecting Resource Descriptions*

In a multi-database model, a central site is responsible for gathering descriptions regarding the content of each disparate collection, in order to allow the resource selection process to efficiently route queries to suitable collections. Depending on whether the involved data sources are publishing information with regard to their content, this entity should collect the descriptions of these sources, or use query probing techniques[19] to create them.

In the scope of this work, we concentrate on raster data residing in `rasdaman` databases. Since `rasdaman` fully supports the `OGC WCS` standard (Section 2.2.1) every database participating in our model already offers descriptive information for its data. In particular, `WCS` defines the `GetCapabilities` operation, which allows the retrieval of a list of the server's data, as well as the WCS operations and parameters that are valid. Figure 7 illustrates part of the response to a `GetCapabilities` request[f], featuring the different encoding formats supported by this service provider, as well as identification information of all the *coverages* populating this database. As shown, this specific server contains a total of three *coverages*, namely `GTOPO30_FSC`, `WFD_RBD_f1v4` and `FSC_PanEuropean`, all of which are of type `RectifiedGridCoverage`. Knowing the names of all the *coverages* for each `rasdaman` installation of our multi-database model allows for accurate resource selection, since every query can be routed exactly to those servers that contain the related data.

3.3. *Collecting Related Metadata*

The information provided through operations defined in the `WCS` and `WCPS` standards helps users compose valid queries. A typical use case scenario includes a `DescribeCoverage` request before a `GetCoverage` or a `ProcessCoverage` request is issued. On the contrary, `MAD Search` gathers all available metadata and exposes them directly to the users, enabling simultaneous searching over them and the actual data.

In order to collect as much metadata as possible, we do not rest on acquiring the `DescribeCoverage` response but further look for information residing in external links or in OGC Catalogs. In the following, we outline the details of our approach on metadata gathering.

[f]Information regarding the service provider and the operations permitted were omitted for space saving reasons.

```
<server endpoint="wcs_endpoint" type="wcs_type">
   <system>
      <general>
         <accessmethods><method /></accessmethods>
         <statistics/>
         <geodeticsystem>CRS name</geodeticsystem>
      </general>
      <harvester><harvestinterval/></harvester>
   </system>
   <user>
      "User defined metadata"
   </user>
   <formal>
      <lastimage timestamp="time" active="boolean">
         <harvested>
            <capabilities>
               "GetCapabilities response"
            </capabilities>
            <coverages>
               <coverage active="true" id="coverage_id">
                 <metadata>
                    <descriptiveMetadata>
                       <external />
                       <geodeticsystem crsname="" />
                    </descriptiveMetadata>
                    <usermetadata>
                       <dcmetadata />
                       <userranking />
                       <domain userid="" />
                       <generalannotations>
                          <item userid="" />
                       </generalannotations>
                    </usermetadata>
                 </metadata>
                 <coveragedescription>
                   "DescribeCoverage response"
                 </coveragedescription>
               </coverage>
               <coverage>
                 ...
               </coverage>
            </coverages>
         </harvested>
      </lastimage>
   </formal>
</server>
```

Fig. 6: MAD Search Catalog Schema

3.3.1. *DescribeCoverage requests*

The `DescribeCoverage` request helps gather additional information regarding a *coverage* such as its coordinate reference system (CRS), its range or its domain

```
<wcs:Capabilities>
  <ows:ServiceIdentification xmlns="http://www.opengis.net/ows/2.0">
  ...
  </ows:ServiceIdentification>
  <ows:ServiceProvider xmlns="http://www.opengis.net/ows/2.0">
  ...
  </ows:ServiceProvider>
  <ows:OperationsMetadata xmlns="http://www.opengis.net/ows/2.0">
  ...
  </ows:OperationsMetadata>
  <wcs:ServiceMetadata xmlns="http://www.opengis.net/ows/2.0">
    <wcs:formatSupported>application/netcdf</wcs:formatSupported>
    <wcs:formatSupported>image/tiff</wcs:formatSupported>
    <wcs:formatSupported>image/jp2</wcs:formatSupported>
    <wcs:formatSupported>application/gml+xml</wcs:formatSupported>
  </wcs:ServiceMetadata>
  <Contents xmlns="http://www.opengis.net/wcs/2.0">
    <CoverageSummary>
      <CoverageId>GTOPO30_FSC</CoverageId>
      <CoverageSubtype>RectifiedGridCoverage</CoverageSubtype>
    </CoverageSummary>
    <CoverageSummary>
      <CoverageId>WFD_RBD_f1v4</CoverageId>
      <CoverageSubtype>RectifiedGridCoverage</CoverageSubtype>
    </CoverageSummary>
    <CoverageSummary>
      <CoverageId>FSC_PanEuropean</CoverageId>
      <CoverageSubtype>RectifiedGridCoverage</CoverageSubtype>
    </CoverageSummary>
  </Contents>
</wcs:Capabilities>
```

Fig. 7: Part of the response to a `GetCapabilities` request

as well as related metadata. For instance, in cases where the data residing in the database are geological the Geoscience Markup Language (`GeoSciML`) can be used to offer a full description of a *coverage*. `MAD Search` issues a `DescribeCoverage` request for every *coverage* specified in the corresponding `GetCapabilities` response.

Figure 8 illustrates part of a response to a `DescribeCoverage` request on a given *coverage*. It contains general information regarding this *coverage*, including its bounding box and a short description. It also may feature more specialized information such as the description of the *Geologic Unit* it underlies and the description of its lithology. By issuing a `DescribeCoverage` request for every *coverage* populating the databases covered in our engine, and inserting the corresponding responses into a central `XML` database, we attain the exposure of all related information collectively and its exploitation within our searching facilities.

```xml
<wcs:CoverageDescriptions xsi:schemaLocation="http://www.opengis.net/wcs/2.0 ...">
  <wcs:CoverageDescription gml:id="glasgow_karn_b"
      xmlns="http://www.opengis.net/gml/3.2">
    <boundedBy>
      <Envelope srsName="..." axisLabels="E N" uomLabels="..." srsDimension="2">
        <lowerCorner>254750 659824.9</lowerCorner>
        <upperCorner>265250 670024.9</upperCorner>
      </Envelope>
    </boundedBy>
    <wcs:CoverageId>glasgow_karn_b</wcs:CoverageId>
    <gmlcov:metadata>
      ...
        <gsmlst:Contact gml:id="KARN-BASE">
          <gml:description>KILLEARN SAND AND GRAVEL MEMBER (BASE)</gml:description>
          <gsml:observationMethod>...</gsml:observationMethod>
          <gsml:purpose>instance</gsml:purpose>
          <gsml:occurrence>...</gsml:occurrence>
          <gsml:relatedFeature><gsmlst:BoundaryRelationship gml:id="LOCAL_ID_1">
            ...
          </gsmlst:BoundaryRelationship></gsml:relatedFeature>
        </gsmlst:Contact>
      ...
    </gmlcov:metadata>
    <domainSet><RectifiedGrid dimension="2" gml:id="glasgow_karn_b-grid">
        <limits><GridEnvelope>
            <low>0 0</low>
            <high>104 101</high>
        </GridEnvelope></limits>
        <axisLabels>E N</axisLabels>
        <origin><Point gml:id="glasgow_karn_b-origin" axisLabels="E N"
             uomLabels="...">
            <pos>254800 669974.9</pos>
        </Point></origin>
    </RectifiedGrid></domainSet>
    <gmlcov:rangeType>...</gmlcov:rangeType>
    <wcs:ServiceParameters>...</wcs:ServiceParameters>
    <gmlcov:metadata>
      ...
        <gsmlgu:composition>
          ...
            <gsmlem:lithology xlink:title="Clay"/>
          ...
        </gsmlgu:composition>
      ...
    </gmlcov:metadata>
  </wcs:CoverageDescription>
</wcs:CoverageDescriptions>
```

Fig. 8: Part of the response to a `DescribeCoverage` request

.

```
<gmlcov:metadata>
  <gmlcov:Extension>
    <PlanetServerPDSMetadata
         xlink:href="http://oderest.rsl.wustl.edu/live/?query=product&
    amp;results=m&amp;output=XML&amp;pdsid=frt00017ae0_07_if165l_trr3"
         xlink:type="simple"/>
  </gmlcov:Extension>
</gmlcov:metadata>
```

```
<ODEResults>
  <Status>Success</Status>
  <QuerySummary>
    <Date>2014-07-18T03:16:51.405</Date>
    <target>MARS</target>
    <query>PRODUCT</query>
    <End>End</End>
  </QuerySummary>
</ODEResults>
```

Fig. 9: XLink pointing to external metadata and a sample for a given *coverage*. The external metadata is kept in designated placeholders and the original hierarchy is maintained, making the construction of paths to the new XML tree a trivial procedure.

3.3.2. *External Metadata Harvesting*

Responses to `DescribeCoverage` requests may potentially include links to external documents with the use of the `XLink` specification. In such cases, we can further enrich our metadata collection by following the links and gathering this additional information in specific designated placeholders. Thereby, constructing paths with the new XML tree is trivial, as we maintain the original hierarchy of the external metadata and its position in the XML tree is predefined. Figure 9 presents how a `DescribeCoverage` response encodes this pointer to external metadata and the actual metadata for a given *coverage*.

3.3.3. *OGC Catalog Harvesting*

Collections of descriptive information about geospatial data, services and related resources are also published and searched with the OpenGIS® Catalogue Services (OGC) Interface Standard. Therefore, our approach extracts information from such catalogs and stores it alongside the metadata of the *coverage* in question, in an effort to collect as much as possible of the available information.

4. xWCPS

In this section, we define and present the `xWCPS` query language, the main contribution of this work. We provide the syntactic elements of our newly defined language and we give insights into the architecture of its prototype implementation.

4.1. *xWCPS Definition*

The `xWCPS` query language aims towards merging the path two widely adopted standards, namely `XQuery` and `WCPS`, have paved, into a new construct, which enables simultaneous search on both XML-encoded metadata and OGC *coverages*. Moreover, our novel language offers a more expressive way of querying array data, as well as federated search over multiple data sources. Domain Specific Languages (DSLs) enable experts of a particular area to develop their own solutions and enhance their productivity[27]. However, the cost of designing and implementing a new DSL, as well as educating its users is significant[27]. Therefore, we opted to build `xWCPS` by extending `WCPS` with `XQuery` features, due to the large number of users working with array data that are already familiar with `WCPS`.

The main objectives of `xWCPS` are to:

- Allow seamless integration of multidimensional *coverage* data with descriptive metadata for search retrieval and processing applications.
- Semantically cover existing capabilities of `WCPS` and avoid alienating its users, by opting to extend this language instead of `XQuery`.
- Enhance `WCPS`'s `for-where-return` structure with `XQuery`'s `let-order by` clauses.
- Widen the horizon of `WCPS` as far as diversity is concerned.
- Bring the `WCPS` standard closer to an existing language specification.

In the next Section, we discuss the elements of `xWCPS` and their novelty.

4.2. *xWCPS Language Elements*

`XQuery` follows the `FLWOR` expression syntax and therefore, it offers two additional clauses in comparison with `WCPS`, namely `let` and `order by`. `xWCPS` uses all five clauses, but the optional nature of the additional ones allows it to subsume `WCPS`. The top level production rules of the *Extended BackusNaur Form* (EBNF) that expresses the `xWCPS` grammar are outlined below:

```
xwcps : (xwcpsForClause | xwcpsLetClause)+
        (xwcpsWhereClause)?
        (xwcpsOrderByClause)?
        xwcpsReturnClause;
xwcpsForClause : "for" var "in" ("(" coverageList ")" | xQeuryExpr);
xwcpsLetClause : "let" var ":=" (coverageExpr | xQeuryExpr);
xwcpsWhereClause : "where" xwcpsORexpression;
```

```
xwcpsORexpression : xwcpsANDexpression ("or" xwcpsANDexpression)*;
xwcpsANDexpression : xwcpsAtom ("and" xwcpsAtom)*;
xwcpsAtom : comparisonExpr
        | booleanScalarExpr
        | "(" xwcpsAtom ")"
        | "(" xwcpsANDexpression ")"
        | "(" xwcpsORexpression ")";
xwcpsOrderByClause : ("order by" | "stable order by") orderSpecList;
xwcpsReturnClause : "return" (exprSingle | processingExpr);
```

The production rules for the complete EBNF of `xWCPS` extend to over 1,800 lines and feature the grammars of `XQuery`[3] and `WCPS`[2].

We see that clauses `for` and `let` can appear any number of times in any order. The `where` and `order by` clauses are optional and every query ends with a `return` clause. The increased syntactic power of the `xWCPS` clauses is outlined below:

- `for`: this clause can create iterator variables holding *coverages* through the `WCPS` syntax or metadata, which can potentially be *coverages* as well, through the `XQuery` syntax.
- `let`: as the case is with `for`, the `let` clause can initialize variables following either of the two syntaxes in the assignment expression. The use of the `let` clause can greatly reduce repetitiveness, making `xWCPS` extremely less verbose than `WCPS`.
- `where`: the expressions provided through this clause enable the specification of criteria on both data and metadata, thus enriching significantly the selecting and filtering capabilities.
- `order by`: allows the ascending or descending ordering of the results depending on the expression provided.
- `return`: supports both `XQuery` and `WCPS` expressions and additionally enables the creation of *mixed results*, through a newly defined function, viz. `mixed()`. The currently available formats of *mixed results* offered by the `xWCPS Engine` are `XML`, `HTML` and `ZIP`.

Metadata values can be used directly in `WCPS` expressions as well, with the help of the `xquery()` function. This function permits the injection of `XPaths` inside `WCPS` expressions and thus, allows the use of metadata while composing `WCPS` conditions or processing actions. Pure `XQuery` expressions do not need to use this function.

Furthermore, it is worth noting that the `xWCPS Engine` is completely agnostic of the underlying metadata structure. To compose queries that showcase our approach we rely on the schema outlined in Section 3.1. However, the language and our engine do not depend on a particular `XML` hierarchy and is able to execute queries using different schemas.

4.3. *xWCPS Implementation*

The architecture of our prototype for `xWCPS` query execution is illustrated in Figure 10. The core of this architecture is the language component that consists of a *parser* for the language, a *planner*, and the libraries that enable the execution of the queries. To answer the queries, the language component pulls metadata from the `MAD Search Catalog` and data from various array databases (`WCPS` services). In addition to this, a local `WCPS` service is used as temporary storage when processing of `coverages` residing in different databases is requested.

More specifically, each time a query is submitted it passes through the *parser*. There, it is analyzed in its sub-clauses, which can follow the syntax of either of `XQuery` and `WCPS`. In case the query contains syntax errors it is discarded and an informative message is returned as a response. Next, the *planner* uses the sub-clauses generated during the parsing step to create an internal structure. With this structure it then generates the corresponding `XQuery` and/or `WCPS` sub-queries that need to be issued to the `MAD Search Catalog` and the `WCPS` endpoints. The planner ensures that only the necessary requests will be made by discovering the location of the data in question. In case a `WCPS` sub-query involves more than one `WCPS` service, the planner imports the data involved in the query into a local array database in order to execute the query locally[g]. The *executive* receives concrete query actions from the planner and sends them to `XML` and array databases so that they will be finally evaluated. The results that are produced in this step are merged and passed to the next phase. Finally, the *projector* is responsible for the delivery of the data and/or metadata into forms compatible with the requester's requirements.

5. MAD Search Architecture for Retrieval

Based on the concepts introduced so far we have implemented a prototype system that enables distributed information retrieval over array and semi-structured databases. In this section, we present how `MAD Search` exposes its functionality and we outline the rest of the related co-operating components it consists of.

5.1. *Architecture*

Our approach employs a distributed architecture with loosely coupled services that interoperate through standards. Figure 11 depicts an abstract view of our architecture.

In Sections 3 and 4, we elaborate on the most significant parts of `MAD Search`, the `xWCPS Component` and the `MAD Search Catalog`. The `xWCPS Component` enables querying using the `xWCPS` language. Requests are transformed into the corresponding `XQuery` and `WCPS` sub-queries, which are routed to the catalog and the appropriate `WCPS` endpoints, respectively. The results are processed according to

[g]An example of such a query is given in Section 7.4.

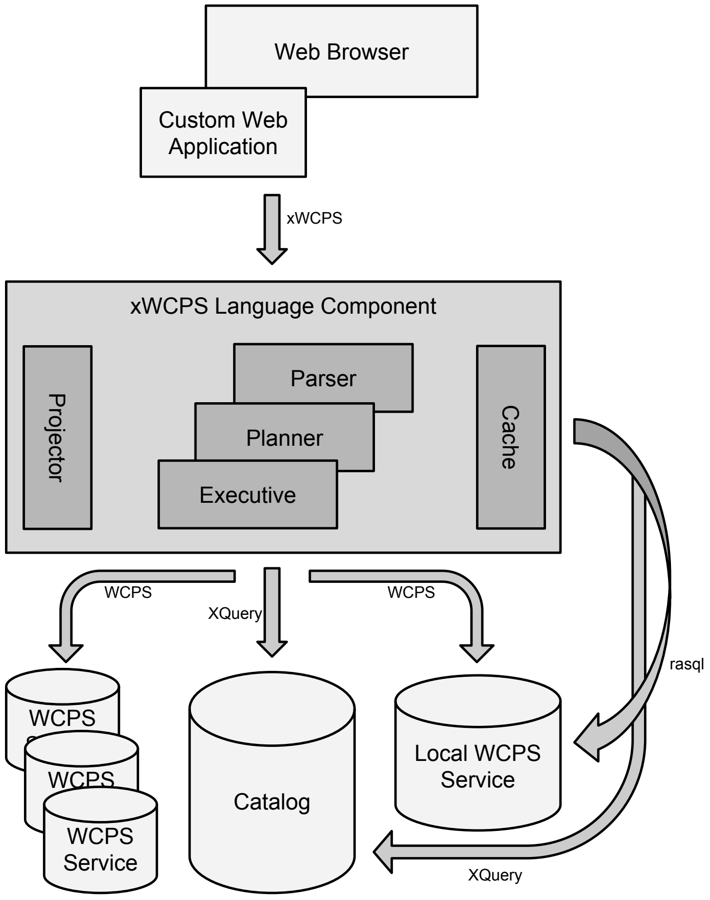


Fig. 10: The `xWCPS` Component Architecture

the user's request to produce the final response. The `MAD Search Catalog` is responsible for holding resource descriptions and metadata collected by the harvesting services. The information is stored in an XML database and is accessible and modifiable through the `Catalog Library`. In particular, this library offers `API`s that can serve the `XQuery` queries produced from `xWCPS` queries or targeted directly to the `MAD Search Catalog`.

Access to `MAD Search` is provided through a number of services that are discussed in Sections 5.2 and 5.3. The `MAD Search Portal` is the main gateway of `MAD Search`, since it gathers a number of related services under a common interface. The `MAD Search Servlets` offer access to the metadata of the `Catalog` through alternative interfaces or protocols.

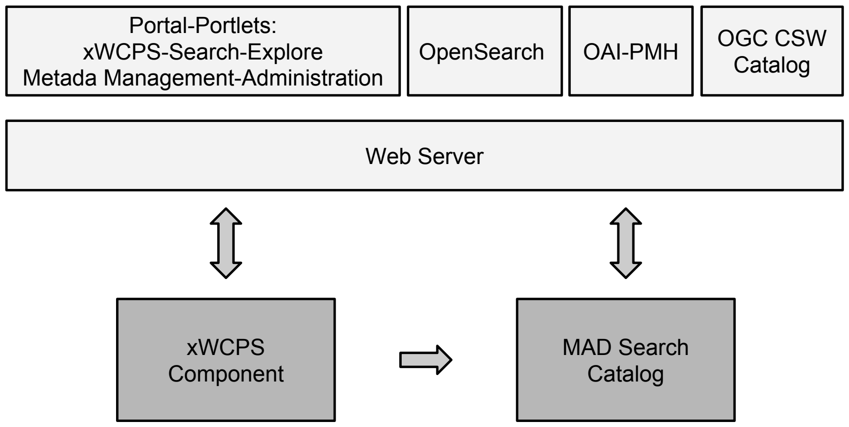


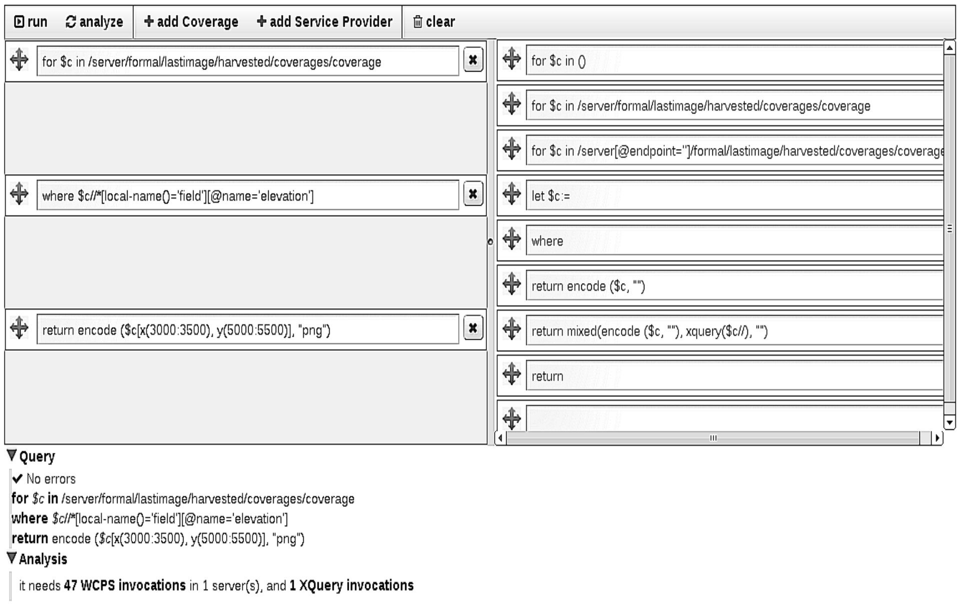
Fig. 11: The `MAD Search` Architecture for Searching. Detailed views of the `MAD Search Catalog` and the `xWCPS Component` are presented in Figures 5 and 10, respectively

5.2. *MAD Search User Interface (UI)*

In order to provide a rich user interface, `MAD Search` uses the `Liferay` portal framework [h] to act as the gateway to its features. The `MAD Search UI` hosts a number of *portlets* that serve as access points to the information residing in our system and its services. The following list outlines the features of the most significant *portlets*.

- `xWCPS Editor`: This portlet features a query editor that makes `xWCPS` query construction a trivial procedure, by providing assistance to their composition. In particular, the query editor offers a step by step build up of the query using drag-and-drop of frequently used query parts, on-the-fly syntax checking to alert for possible errors in the query, analysis of the actual cost of the query, and, of course, execution of the `xWCPS` query. Depending on their format, the results are returned to the user as HTML links to binary, `XML`, `HTML` and `ZIP` files or raw text. The predefined query parts on the right side were determined after analysing query logs and extracting the most frequently used expressions. As it can be seen in the left side of Figure 12, the user has dragged the 2nd, 5th and 6th predefined query parts and complemented them only where necessary. The query to be executed is shown below and the syntax check has found no errors. The analysis results illustrated in the lower part of the Figure, indicate that there are 47 *coverages* satisfying the desired criteria in the underlying databases. This informs the user that the total number of `WCPS` queries that needs to be

[h] `http://www.liferay.com/`.



```
for $c in /server/formal/lastimage/harvested/coverages/coverage
where $c//*[local-name()='field'][@name='elevation']
return encode ($c[x(3000:3500), y(5000:5500)], "png")
```

```
for $c in ()
for $c in /server/formal/lastimage/harvested/coverages/coverage
for $c in /server[@endpoint='']/formal/lastimage/harvested/coverages/coverage
let $c :=
where
return encode ($c, "")
return mixed(encode ($c, ""), xquery($c//), "")
return
```

Query

✓ No errors

```
for $c in /server/formal/lastimage/harvested/coverages/coverage
where $c//*[local-name()='field'][@name='elevation']
return encode ($c[x(3000:3500), y(5000:5500)], "png")
```

Analysis

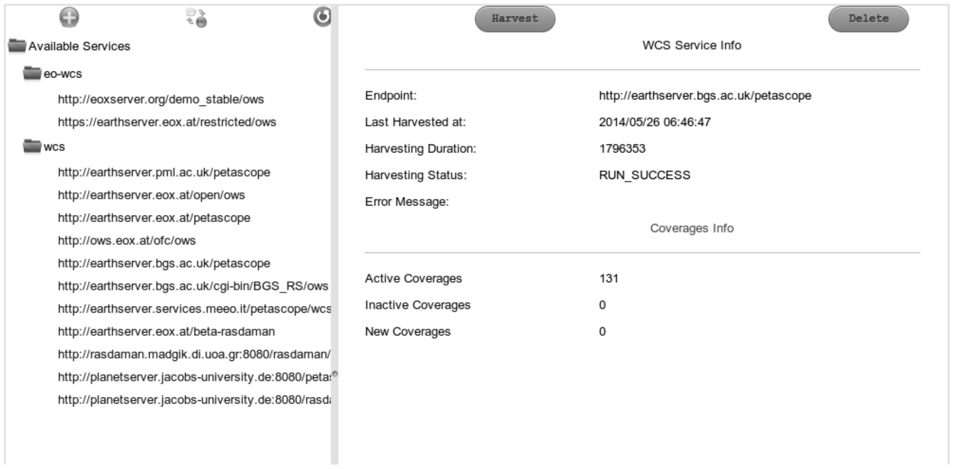
it needs **47 WCPS invocations** in 1 server(s), and **1 XQuery invocations**

Fig. 12: The `xWCPS Engine` query editor *portlet*

performed is not prohibitive and provides her with an estimation on the execution time of the `xWCPS` query.

- `Search Catalog`: this *portlet* offers full-text and `XQuery` search over the metadata. Users can either search with simple text terms and retrieve formatted results, or submit an `XQuery` and retrieve `XML` results. The queries are sent to the `XML` database through the `Catalog Library`. The functionality of this *portlet* targets users only interested in the metadata and uses `XQuery` directly to avoid the parsing overhead of `xWCPS`.
- `Explore Catalog`: using this portlet a user can explore the available `WCS` services and their metadata. More specifically, for each service:
  - A list with the service's *coverages* is available.
  - The available metadata is displayed.
  - Map visualization of selected *coverages* is available.
  - Geospatial search over the bounding boxes of the registered `WCS` services, supporting the `contains`, `intersects` and `is_contained` relations is also supported. Using this functionality the identification of specific *coverages* becomes straightforward.
- `Metadata Management`: This *portlet* allows users to see and add or edit annotations and `Dublin Core` (DC)[i] metadata to the available *coverages*,

[i]Dublin Core Metadata Initiative: `http://dublincore.org/`.



Available Services
- eo-wcs
  - http://eoxserver.org/demo_stable/ows
  - https://earthserver.eox.at/restricted/ows
- wcs
  - http://earthserver.pml.ac.uk/petascope
  - http://earthserver.eox.at/open/ows
  - http://earthserver.eox.at/petascope
  - http://ows.eox.at/ofc/ows
  - http://earthserver.bgs.ac.uk/petascope
  - http://earthserver.bgs.ac.uk/cgi-bin/BGS_RS/ows
  - http://earthserver.services.meeo.it/petascope/wcs
  - http://earthserver.eox.at/beta-rasdaman
  - http://rasdaman.madgik.di.uoa.gr:8080/rasdaman/
  - http://planetserver.jacobs-university.de:8080/petas
  - http://planetserver.jacobs-university.de:8080/rasd

WCS Service Info

Endpoint:	http://earthserver.bgs.ac.uk/petascope
Last Harvested at:	2014/05/26 06:46:47
Harvesting Duration:	1796353
Harvesting Status:	RUN_SUCCESS
Error Message:	

Coverages Info

Active Coverages	131
Inactive Coverages	0
New Coverages	0

Fig. 13: The `Administration` *portlet*

thereby enriching their overall metadata.

- `User Profile Manager Portlet`: Each user has a profile stored in a special collection in the system's registry. This profile contains a user's favorite `xWCPS` and `XQuery` queries to be exploited in future use and her most frequently used *coverages*, making it easy to exploit them in other *portlets*.
- `Administration`: This *portlet* is used for monitoring the current condition of the `MAD Search Catalog`. As it can be seen in Figure 13, it displays all the registered services, their status since the last harvest and the number of active and inactive *coverages*. Among the available actions of this *portlet* are the following:
  – Adding a new `WCS` service endpoint to the catalog.
  – Removing an existing `WCS` service from the catalog.
  – Starting/Stopping the harvesting procedure.
  – Harvesting a specific `WCS` service on demand.

5.3. *MAD Search Servlets*

In order to enhance the access to the data residing in the `MAD Search Catalog` and with interoperability in mind, we provide a number of alternative ways in which this information is made available, that are summarized below:

- `OpenSearch`[j] provides a way for websites and search engines to publish search results in a standard and accessible format. It is a collection of technologies that allow publishing of search results in a format suitable for

[j]`http://www.opensearch.org/`.

syndication and aggregation. Our system utilizes the `OpenSearch` specification to expose the `MAD Search Catalog` through this *servlet* and uses `RSS` as its response format.

- The Open Archives Initiative Protocol for Metadata Harvesting (`OAI-PMH`) provides an application-independent interoperability framework for metadata harvesting. We provide a *servlet* that enables interaction with our catalog over the `OAI-PMH` protocol in order to allow any client application issuing `OAI-PMH` requests to harvest its content. The `OAI-PMH Servlet` supports both `POST` and `GET` requests and follows the *must* functionality as defined by the protocol, i.e., the mandatory operations.
- Catalog Service for the Web (`CSW`) is a standard for exposing a catalog of geospatial records on the Internet. It is the part of the `OGC Catalog Service` that defines common interfaces to discover, browse, and query metadata about data, services, and other potential resources. The `OGC CSW Catalog Servlet` facilitates interaction with our `Metadata Catalog` over the `CSW` standard. The `CSW Servlet` supports all the mandatory methods of the standard through `GET` and `POST` requests.

6. Implementation Aspects

We outline here the implementation choices we made while developing `xWCPS` and the pertinent infrastructure.

The harvesting components are written in `Java` and that is mostly the case with the libraries developed for `xWCPS` query execution. However, the planner of the `xWCPS Component` uses the `ANTLR` framework[1] to generate `Java` code, by specifying the language with a context-free grammar, which was expressed using EBNF. The syntactic rules of `XQuery` and `WCPS` were extracted from previous efforts[3,2]. Concerning the actions of the *harvesters*, the `WCS` queries are issued as `HTTP GET` requests to `OGC WCS` interfaces and the `XQuery` insertions are issued with the use of `Java` libraries implementing the `XQJ API`, the standard `Java` interface to XML datasources. These libraries are the core of the `Catalog Library` implementation. In the scope of this work, libraries for both BaseX[4] and eXistdb[23] databases were developed. `xWCPS` queries are analyzed in `WCPS` and `XQuery` sub-queries. The `WCPS` sub-queries issued are also `HTTP GET` requests, while the `XQuery` sub-queries are executed with the help of the `Catalog Library` as well.

Regarding the portal, the open source portal framework `Liferay`[h] was used. `Liferay` is a web platform that provides features such as enabling the build-up of a portal as an assembly of portlets sharing common navigation. For the portlets we utilized the `Google Web Toolkit (GWT)`[k], that offers development tools for building and optimizing complex browser-based applications. All services offered in the context of `MAD Search` are exposed either as `RESTful Web Services` or

[k]`http://www.gwtproject.org/`.

`Java Servlets`, following the Java API for RESTful Services[l] and the Java Servlet Specification, respectively. Thereby, they are easily utilized by the portal but they can also be accessed independently.

7. Use Cases and Evaluation

In the context of the `EarthServer` project[m] we have tested `MAD Search` by searching over five high-volume array databases and maintaining their respective metadata in the `MAD Search Catalog`. Each one of the array databases serves at least 20 `GB` of data, and the size of `XML` documents held in the catalog is 1.6GB. The documents of the catalog contain all metadata available in the array databases, as well as information pulled from external resources and `OGC Catalogs`. Our system can be publicly accessed[n].

The features and functionality introduced with `xWCPS` are presented in this section through a number of use cases. The queries in each one of the cases demonstrate the expressive power of our language and its superiority over `WCPS` in array database search. We also evaluate the performance of our approach by measuring the overhead induced when querying with `xWCPS`. Our project partners, forming a very experienced in the earth data domain evaluation team, have verified the effectiveness and user-friendliness of our system.

7.1. *Using metadata to select* coverages

Figure 14 shows a query that uses metadata for the selection of *coverages* to be included in the results. The `XPath` in the `for` clause returns a node set holding *coverages*. The condition included in the `where` clause restricts the returned *coverages* to those that have a bounding box with upper corner latitude greater than 72. The `return` clause encodes the *coverages* as `PNG` images. Sample results are shown in Figures 16a and16b.

It is noted here that the inability to access metadata with other array query languages resulted in impractical solutions, such as retrieving data descriptions and manually filtering the ones that satisfy the desired criteria in order to submit queries for them later.

7.2. *Building complex queries with the ehnanced syntax of xWCPS*

The query in Figure 15 demonstrates the convenience offered to the user through the `let` clause, which allows assigning complex expressions to variables and using them for subsequent references. Moreover, the extended power of the `for` clause is shown, since both `XQuery` and `WCPS` expressions are used. In particular, the query

[l]We take advantage of the widely used Jersey framework: `http://jersey.java.net/`.
[m]`http://www.earthserver.eu/`.
[n]`http://earthserver1.madgik.di.uoa.gr/`

```
for $c in /server/formal/lastimage/harvested/coverages/coverage
where $c//boundingbox/uppercorner/lat >= 72
return encode($c,"png")
```

Fig. 14: An `xWCPS` query which filters *coverages* that have a bounding box with upper cornet latitude greater than 72 and encodes them as `PNG` images.

```
for $c in (os_dtm)
for $t in /server[@endpoint='WCS_PROVIDER_1']//coverage
for $comp in $t/coveragedescription//*[local-name()='composition']
let $u:=scale($t,{x:"CRS:1"(0:419), y:"CRS:1"(0:407)},{})
let $d:=trim($c,{x:"http://www.opengis.net/def/crs/EPSG/0/27700"
 (254750:265250), y:"http://www.opengis.net/def/crs/EPSG/0/27700"
 (659824.9:670024.9)})
let $e:=scale($d,{x:"CRS:1"(0:419), y:"CRS:1"(0:407)},{})
let $g:=($e*(not($u=-340282346638528859811704183484516925440.0)))
let $h:=($u*($u=-340282346638528859811704183484516925440.0))
let $i:=($g+$h)
let $f:=($i-$u)

where $comp//*[local-name()='lithology']/@*[local-name()='title']='Sand'
 and $comp//*[local-name()='proportion']//*[local-name()='lowerValue']>=25
 and max($f)<=30

return encode((char) (($f-min($f))*255)/(max($f)-min($f)),"png")
```

Fig. 15: A complex `xWCPS` query for the retrieval of surfaces underlying sand units within 30 meters of surface. This is an example of how `xWCPS` helps the user avoid repetitiveness.

features three `for` clauses. The first holds a `WCPS` expression while the other two follow the `XQuery` syntax. However, the first `XQuery` expression actually describes *coverages* instead of a node set, as the second `XQuery` expression does. A number of `let` clauses follow, all of which are `WCPS` expressions, specifying operations that could only be expressed in the `return` clause of a `WCPS` query. Considering that the `return` clause uses the last `let` assignment, which essentially holds all the previous expressions, a number of times, we understand that the equivalent `WCPS` query would be extremely cumbersome. Finally, the `where` clause features a mixture of `XQuery` and `WCPS` conditions, and the `return` clause is pure `WCPS`, specifying the type of the results as `PNG` images.

The aim of the query is the retrieval of surfaces underlying sand units within 30 meters of surface and sample results are shown in Figures 16c and 16d. However, the `WCPS` expressions that achieve this outcome are outside the scope of this paper and therefore we will not go into details about the specific clauses that achieve this result. Our intention is to illustrate how `xWCPS` avoids repetitiveness and reduces the number of query lines for such complex queries.

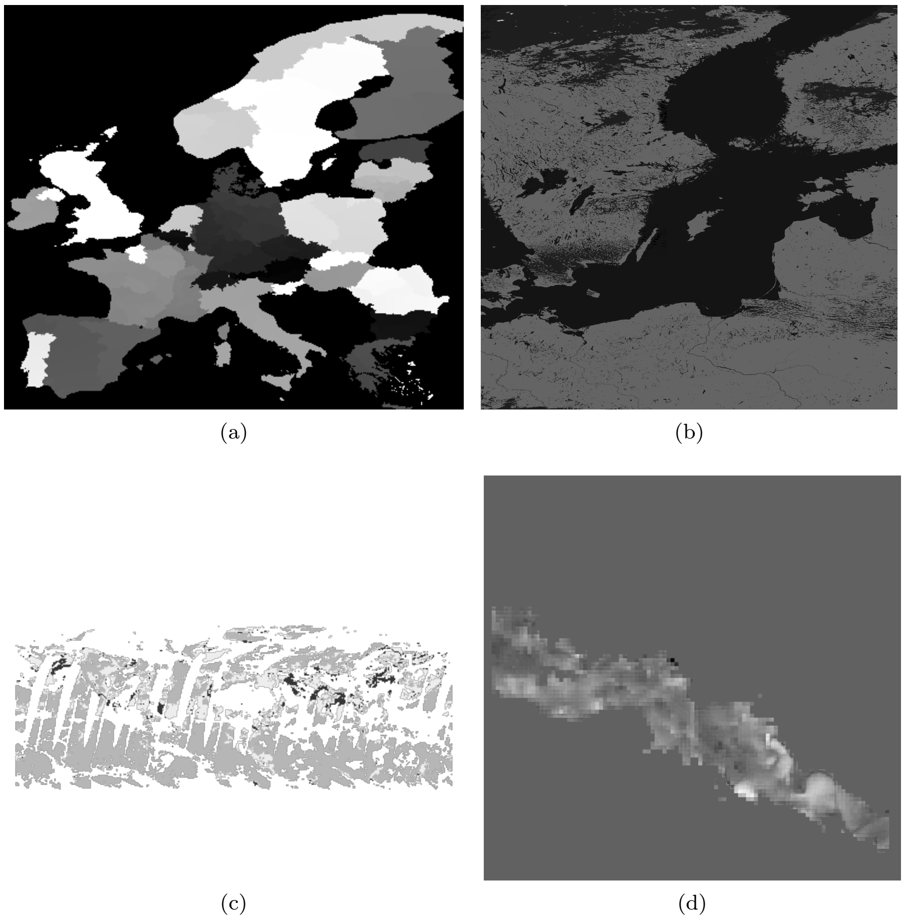


Fig. 16: Sample results for the queries illustrated in Figures 14 (a,b) and 15 (c,d)

7.3. *Retrieving data accompanied with related metadata*

As discussed in Section 4.2, `xWCPS` allows the use of a new function named `mixed`. This extension over `WCPS` enables the user to request a mixture of data and metadata as a result of her query. This feature is not offered through existing languages and is an extremely important one since it allows access to all the information residing in the cooperating databases directly through `xWCPS` queries. The format of the result can be one of the following: XML, HTML and ZIP.

In the query of Figure 17, the user requests all the *coverages* satisfying certain criteria, i.e., those that have *'Clay'* as the title for their *lithology* element and do not have a value greater than 50 in their color channels, should be encoded as `TIFF` images. However, the final outcome is specified as XML. In particular the

```
for $c in /server//coverage
where $c//*[local-name()='lithology']/@*[local-name()='title'] = 'Clay'
 and max($c)<50
return mixed(encode($c, "tiff"), xquery($c//boundingbox), "xml")
```

Fig. 17: An `xWCPS` query that filters *coverages* that have *'Clay'* as the title for their *lithology* element and do not have a value greater than 50 in their color channels, encodes them as `TIFF` images, and creates an XML with additional metadata information. This query is an example of the use of function `mixed()`.

```
<document xmlns:xlink="http://www.w3.org/1999/xlink">
  <result xlink:href="http://earthserver1.madgik.di.uoa.gr/xwcps/glasgow_liwd.tiff">
    <boundingbox>
      <lowercorner>
        <lat>254750.0</lat>
        <long>659824.9</long>
      </lowercorner>
      <uppercorner>
        <lat>265250.0</lat>
        <long>670024.9</long>
      </uppercorner>
    </boundingbox>
  </result>
</document>
```

Fig. 18: The result of the mixed `xWCPS` query of Figure 17

user asks for XML documents that will hold links to the encoded *coverages* as well as the results of an `XQuery` asking for the *boundingbox* element of each *coverage*. `MAD Search` places the `WCPS` part of the result as an `XLink` in the resulting XML document. A sample outcome is illustrated in Figure 18.

7.4. *Processing over multiple service providers*

Figure 19 presents a relatively simple query written in what seems to be pure `WCPS`. The result of a subtraction of two aggregate functions that are applied on two *coverages* is encoded in `CSV` format and returned to the user. However, this query can only be executed through `MAD Search`. The reason is that the *coverages* assigned to the two variables in the `for` clauses of the query are hosted in different array databases. `MAD Search` identifies the location of the *coverages* with the use of metadata. Therefore, it can execute the query and retrieve the desired result following two different strategies. If the sought information coming from one of the remote service providers is a scalar value, e.g., `max($c)` in the query of Figure 19, the engine performs an additional query to retrieve this value and transforms the original query accordingly. If the complete *coverage* is needed, the engine imports it in a local array database and executes the query there. However, since *coverages*

```
for $c in (GTOPO30_daily_FSC_Baltic_Optical)
for $d in (frt0000683f_07_if163l_trr3_1_01)
return encode(max($c)-min($d[E(-492685.9203:-492685.92029),
 N(-3117164.165:-3117164.1649)]),"csv")
```

Fig. 19: An `xWCPS` query that subtracts two aggregate functions that are applied on two *coverages* and encodes the result in `CSV` format. The two *coverages* are in different service providers.

```
for $c in /server[@endpoint="http://earthserver.bgs.ac.uk/rasdaman/ows"]/
  formal/lastimage/harvested/coverages/coverage[@id="os_dtm"]
for $t in /server[@endpoint="http://earthserver.bgs.ac.uk/rasdaman/ows"]/
  formal/lastimage/harvested/coverages/coverage
let $u:=scale($t,{E:"CRS:1"(0:419), N:"CRS:1"(0:407)},{})
let $d:=trim($c,{E:"http://www.opengis.net/def/crs/EPSG/0/27700"(254750:265250),
  N:"http://www.opengis.net/def/crs/EPSG/0/27700"(659825:670025)})
let $e:=scale($d,{E:"CRS:1"(0:419),N:"CRS:1"(0:407)},{})
let $g:=($e*(not($u=-340282346638528859811704183484516925440.0)))
let $h:=($u*($u=-340282346638528859811704183484516925440.0))
let $i:=($g+$h)
let $f:=($i-$u)
where $t//boundingbox/lowercorner/lat > THRESHOLD
return encode((unsigned char)(($f-min($f))*255)/(max($f)-min($f)),"png")
```

Fig. 20: The query we used for the evaluation of `MAD Search`. We varied the value of `THRESHOLD` to progressively relax the metadata criteria

range into Terabyte sizes, this is efficient only for 2-dimensional *coverages*. Whatever the case, the actions taken by the engine are completely transparent to the user.

7.5. *Evaluation*

`MAD Search` and `xWCPS` enable users to take full advantage of metadata information when composing queries. In order to examine the performance of our approach, we present in this section the cost of the individual operations performed during the execution of an `xWCPS` query. This gives a clear view on the overhead that is induced to allow seamless integration of array data with metadata.

We used two identical virtual machines (VMs) using CentOS 5.7 on a server located in Greece, with two Intel® Xeon® E5620 @ 2.40GHz CPUs and 4GB of RAM each, to host the `xWCPS` component and the `Catalog`, respectively. The `Catalog` contained 1.6GB of metadata for five array databases, but for this example we only targeted one `WCPS` service provider, located in Germany. We executed the query of Figure 20 multiple times, varying the value of `THRESHOLD` to progressively relax the metadata criteria.

In Figure 21, we present timing results for this query. We note that parsing, planning, merging and projecting are operations executed by the `xWCPS` component (first VM), execution of XQuery queries is handled by the Catalog (second VM) and execution of WCPS queries is handled by WCPS service providers. As it can be seen, most of the total execution time is consumed during WCPS queries. These operations are responsible for the processing of array data. We also show that the execution time increases linearly with the number of WCPS queries, which is expected as our planner schedules the latter for serial execution to avoid thrashing. WCPS queries from different xWCPS queries are, of course, executed in parallel. It is worth noticing here, that rasdaman is not able to produce results for more than one *coverage* per query, even though the WCPS syntax allows it.

WCPS operations are essential even outside the context of MAD Search. Therefore, it is safe to assume that there is no overhead regarding their execution in the context of MAD Search. Moreover, for all queries of Figure 21, a total of 2 XQuery queries were executed to pull information from the MAD Search Catalog. Without MAD Search, users would have to spend hours retrieving the resource description of a database to discover its content (*coverage* names), and the description of every *coverage* to filter based on metadata. The overhead induced to achieve an automation of these operations comprises the cost of XQuery execution, parsing, planning, merging and projecting. As we can see in Figure 21, the cost of parsing and XQuery execution is extremely small, whereas the cost of planning, merging and projecting is fairly insignificant relative to the cost of WCPS query execution and remains constant as the number of WCPS queries grows.

8. Related Work

The use of traditional databases for scientific data is relatively limited. The reasons behind the reluctance of scientists in utilizing these tools, are usually their lack of support for their data types and access patterns as well as their speed when handling this kind of data. The incapability of commercial data-management tools to support the unprecedented scale, rate, and complexity of scientific data has intensified the research effort expended on developing array DBMSs. The latter provide database services specifically designed for arrays (raster data), i.e., homogeneous collections of data items (often called pixels or voxels), sitting on a regular grid of one, two, or more dimensions. Arrays are used to represent sensor, simulation, image, or statistical data. Array databases aim at offering flexible, scalable storage and retrieval of such data.

In terms of Array DBMS implementations, the rasdaman[8] system coined the field of array databases and has the longest implementation track record of n-D arrays with full query support. The Raster query language (rasql) enables querying directly over array structures but rasdaman also offers the reference implementation of the Web Coverage Processing Service (WCPS) standard, which defines a protocol-independent language for the extraction, processing, and analysis of multi-dimensional gridded *coverages*. WCPS is one of the building blocks of the language presented in this paper. The Oracle GeoRaster[28] is a feature of Oracle Spatial that

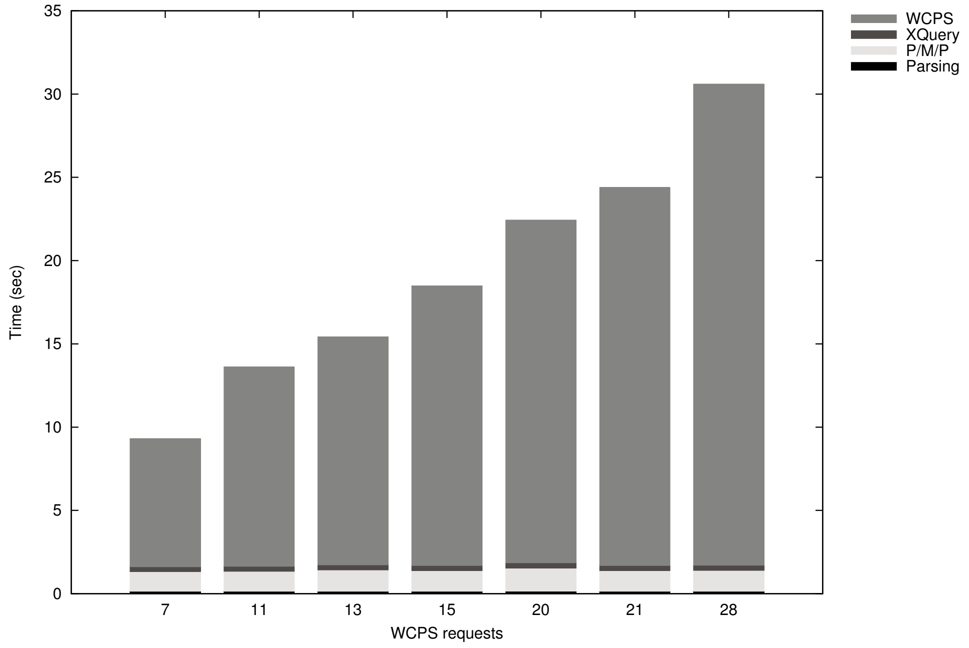


Fig. 21: Measuring the overhead of `xWCPS` by applying a query that filters *coverages*. The Figure illustrates the time needed for parsing (Parsing), planning, merging and projecting (P/M/P), and executing `XQuery` (XQuery) and `WCPS` (WCPS) queries, during the execution of `xWCPS` queries. We progressively relax the metadata criteria to increase the number of needed `WCPS` requests.

enables users to store, index, query, analyze, and deliver raster image and gridded data and its associated metadata. `GeoRaster` provides Oracle spatial data types and an object-relational schema, while access is possible through `PL-SQL`. On the downside, GeoRaster calls are not embedded into SQL and optimization capabilities are restricted as opposed to the `rasdaman` approach. `TerraLib`[14] is an open-source GIS framework that extends object-relational DBMS technology to handle spatio-temporal data types. While its main focus is on vector data, there is also some support for rasters. Starting with version 2.0, `PostGIS`[25] embeds raster support for 2-D rasters. The goal of `PostGIS Raster`[5] is to implement the `RASTER` type in a way equivalent to the implementation of the `GEOMETRY` type in `PostGIS` and to offer a single set of overlay SQL functions operating seamlessly on vector and raster *coverages*. `SciQL`[29] is an SQL-based query language with both tables and arrays as first class citizens which aims at a true symbiosis of the relational and array paradigm. In order to underpin its storage layer implementation `SciQL` uses MonetDB[o]. `SciDB`[13] is a relatively recent initiative to establish array database support. Like `SciQL`, arrays are seen as an equivalent to tables, rather than a new attribute type as in `rasdaman` and `PostGIS`.

Multi-database models offering distributed information retrieval has been an active field of research since the early 1980's. Motivated by this need, Marcus[22] deals with the issue of resource description and selection. The data available at the time did not allow the study of issues such as handling resources that are distributed geographically, are managed by many parties, or scale to large numbers. However, this work sets the ground for automatic data-base selection. In a cooperative environment there is a need to store information concerning the available collection in order to route queries effectively. Gravano et al.[18] define `STARTS`, a protocol for internet retrieval and search, which allows the exchange of information regarding the contents of a collection. Among the things that need to be shared are statistics concerning the presence of data in the collection as well as other metadata such as the size of the collection. In cases where the collections do not publish any information regarding their contents other methods must be followed. In a later effort, Gravano et al.[19] propose a technique called query probing, which adaptively sends queries derived from document classifiers to databases in order to automate their classification. Since previous efforts failed covering low-frequency words, Ipeirotis and Gravano[21] complement the summaries of topically related databases to increase their coverage, and compare them to each submitted query to estimate the relevance of the collection properly.

Providing integrated access to heterogeneous data sources remains a major challenge. Several approaches aiming towards this direction exist in the literature. The one mostly related to our work is that of Akhtar et al.[7], who address the issue of querying both XML and RDF/OWL data, by using `XQuery` and `SPARQL` as starting points to introduce the unified query language `XSPARQL`. The use of a common framework to handle both formats proves to be a significant advantage which eliminates several unnecessary steps that were previously mandatory. Essid et al.[17] propose the first geographic data mediation system combining geographic standards with W3C ones. In addition, an `XQuery` based language is presented that allows spatial complex queries over GML data sources. Zhao et al.[30] present a method to enable combined ontology queries on spatial data available from OGC WFS services and data stored in databases. User queries are rewritten to WFS `GetFeature` requests and SQL queries. An ontology-based approach to tackle the problem of heterogeneous Earth-Science data integration and interoperability is presented by Cruz and Xiao[16]. Both schematic and semantic heterogeneities are taken into account. Finally, Ternier et al. [26] develop a query language for searching distributed repositories of learning objects. However, with respect to Earth-Science data federation little work is reported.

To the best of our knowledge, the effort presented here is the first approach

[o]`http://monetdb.cwi.nl/`.

that attempts to bridge the gap between array databases and metadata by merging two well established query languages to compose a unified one, that provides access to the underlying data sources in a uniform manner. Our approach does not deal with the representation of array data, but focuses on offering distributed information retrieval over array databases and seamless integration of scientific data with metadata. In the context of this work, all involved parties use the same software and parameter settings and are responsible for publishing resource descriptions for their databases. Therefore, techniques related to query based sampling, whose applicability with scientific data is not as successful as with text databases, are not needed.

9. Conclusions and Future Work

We have presented a novel approach that enables distributed information retrieval over array and semi-structured databases. `MAD Search` provides a number of services to retrieve resource descriptions from array databases, build a metadata catalog, offer access through a variety of established protocols, and uniformly search the available data through `xWCPS`. This is a unified query language defined through the merging of `XQuery` and `WCPS`, two widely-used standards, and helps bridge the existing gap for integrated retrieval involving scientific data and respective metadata. By electing to use `XQuery` and `WCPS` as building blocks of our proposed query language, we have preserved usability and avoided alienating users already familiar with `WCPS`. We have further extended our language with the query facilities of `XQuery` to further benefit the user community. This extended `xWCPS` functionality offers a rich set of features that make searching over array databases substantially more effective and user-friendly. The discovery of *coverages* is enhanced through `xWCPS` as users are now able to specify their selection through `XPath` expressions and compose complex conditions featuring both data and metadata criteria. In addition, the outcome of `xWCPS` queries can be a pure `XQuery` or `WCPS` result as well as a mixture of the outcomes of the two languages, expressed in `XML`, `HTML` and `ZIP` format.

We have exhaustively investigated numerous use-cases for the proposed `xWCPS` and showcase a number of such examples that demonstrate its salient features. This functionality that entails retrieval over multiple data source providers and the simultaneous querying of both data and its respective metadata was simply not available before. Key features also include injection of `XPaths` inside `WCPS` expressions and simplicity of `xWCPS` queries over their equivalent `WCPS` counterparts. `MAD Search` is also capable of cooperating with some of the most extensively used array and XML database tools to render searching for scientific data often resident in heterogeneous resources much simpler. In this paper, we demonstrate both the utility and effectiveness of our approach by experimenting with a distributed environment consisting of five high-volume array databases whose data ranges between 20–100 `GB`.

In the future, we plan to pursue a number of promising directions that may strengthen our work on `xWCPS`. There are at least three distinct areas that we would like to further investigate. They include: 1) the implementation of the `xWCPS` engine can receive a number of optimizations to realize an enterprise-grade software facility, 2) the potential security issues that may arise over time from the various interfaces involving the layers of our proposed architecture; We plan to formally analyze their behavior and address derived issues, and finally, 3) the extension of `xWCPS` with functions for the handling of spatio-temporal data encoded in XML formats like the Geography Markup Language (GML); the latter are methods such as *contains()*, *intersects()*, and *overlaps()* that facilitate the filtering of *coverages* according to geospatial criteria.

Acknowledgements: We would also like to express our gratitude to all the anonymous reviewers who helped us improve the presentation of our work with their insightful comments. This research has been partially supported by *EC FP7 Workprogramme (2007-2013)* under grant agreement n. 283610 *"European Scalable Earth-Science Service Environment (EarthServer)"* and *EC H2020* under grant agreement n. 654367 *"Agile Analytics on Big Data Cubes (EarthServer 2)"*.

References

1. ANTLR. `http://www.antlr.org/`. [Online; accessed 19-July-2014].
2. ANTLR 3 grammar for WCPS - rasdaman source code. `http://www.rasdaman.org/browser/applications/petascope/src/main/java/petascope/wcps/grammar/`. [Online; accessed 19-July-2014].
3. ANTLR 3 grammar for XQuery with Update, Scripting and Full-Text extensions. `http://code.google.com/p/xqgrammar/source/browse/trunk/xqgrammar/src/main/antlr3/`. [Online; accessed 19-July-2014].
4. BaseX. `http://basex.org/`. [Online; accessed 19-July-2014].
5. PostGIS Raster Home Page. `http://trac.osgeo.org/postgis/wiki/WKTRaster/`. [Online; accessed 19-July-2014].
6. A. Ailamaki, V. Kantere, and D. Dash. Managing scientific data. *Commun. ACM*, 53(6):68–78, June 2010.
7. W. Akhtar, J. Kopeck, T. Krennwallner, and A. Polleres. XSPARQL: Traveling between the XML and RDF worlds and avoiding the XSLT pilgrimage. In *The Semantic Web: Research and Applications*, volume 5021 of *Lecture Notes in Computer Science*, pages 432–447. 2008.
8. P. Baumann. Management of multidimensional discrete data. *The VLDB Journal*, 3(4):401–444, 1994.
9. P. Baumann. The OGC web coverage processing service (WCPS) standard. *Geoinformatica*, 14(4):447–479, Oct. 2010.
10. P. Baumann. OGC® WCS 2.0 Interface Standard - Core. OGC 09-110r4, version 2.0, July 2012.
11. P. Baumann, J. Yu, and S. Meissl. OGC® GML Application Schema – Coverages. Apr 2012.
12. T. Bray, J. Paoli, C. M. Sperberg-McQueen, E. Maler, and F. Yergeau. Extensible markup language (XML) 1.0 (fifth edition). Technical report, World Wide Web Consortium, Nov. 2008.

13. P. G. Brown. Overview of sciDB: Large scale array storage, processing and analysis. In *Proc. of the 2010 ACM SIGMOD Int. Conf. on Management of Data*, pages 963–968, Indianapolis, Indiana, USA, June 2010.
14. G. Câmara, L. Vinhas, K. R. Ferreira, G. R. D. Queiroz, R. C. M. D. Souza, A. M. V. Monteiro, M. T. Carvalho, M. A. Casanova, and U. M. D. Freitas. Terralib: An open source GIS library for large-scale environmental and socio-economic applications. In *Open Source Approaches in Spatial Data Handling*, volume 2 of *Advances in Geographic Information Science*, pages 247–270. Springer, 2008.
15. D. Chamberlin. XQuery: A query language for XML. In *Proc. of the 2003 ACM SIGMOD Int. Conf. on Management of Data*, pages 682–682, San Diego, California, June 2003.
16. I. F. Cruz and H. Xiao. Data integration for querying geospatial sources. In *Geospatial Services and Applications for the Internet*, pages 110–134. Springer, 2008.
17. M. Essid, F.-M. Colonna, O. Boucelma, and A. Bétari. Querying mediated geographic data sources. In *Proc. of Advances in Database Technology - EDBT 2006, 10th Int. Conf. on Extending Database Technology*, pages 1176–1181, Munich, Germany, Mar. 2006.
18. L. Gravano, C.-C. K. Chang, H. García-Molina, and A. Paepcke. STARTS: Stanford proposal for internet meta-searching. In *Proc. of the 1997 ACM SIGMOD Int. Conf. on Management of Data*, pages 207–218, Tucson, Arizona, USA, June 1997.
19. L. Gravano, P. G. Ipeirotis, and M. Sahami. Qprober: A system for automatic classification of hidden-web databases. *ACM Trans. Inf. Syst.*, 21(1):1–41, Jan. 2003.
20. J. Gray, D. T. Liu, M. A. Nieto-Santisteban, A. S. Szalay, G. Heber, and D. DeWitt. Scientific data management in the coming decade. *ACM SIGMOD Record*, 34(4):35–41, January 2005. also as MSR-TR-2005-10.
21. P. G. Ipeirotis and L. Gravano. When one sample is not enough: Improving text database selection using shrinkage. In *Proc. of the 2004 ACM SIGMOD Int. Conf. on Management of Data*, pages 767–778, Paris, France, June 2004.
22. R. S. Marcus. An experimental comparison of the effectiveness of computers and humans as search intermediaries. *Journal of the American Society for Information Science*, 34(6):381–404, 1983.
23. W. Meier. eXist: An open source native XML database. In *Web, Web-Services, and Database Systems*, volume 2593 of *Lecture Notes in Computer Science*, pages 169–183. Springer, 2003.
24. K. Saidis, Y. Smaragdakis, and A. Delis. DOLAR: virtualizing heterogeneous information spaces to support their expansion. *Softw., Pract. Exper.*, 41(11):1349–1383, 2011.
25. C. Strobl. Postgis. In *Encyclopedia of GIS*, pages 891–898. Springer, US, 2008.
26. S. Ternier, E. Duval, D. Massart, A. Campi, S. Guinea, and S. Ceri. Interoperability for Searching Learning Object Repositories: The ProLearn Query Language. *D-Lib Magazine*, 14(1/2), 2008.
27. A. Van Deursen, P. Klint, and J. Visser. Domain-specific languages: An annotated bibliography. *Sigplan Notices*, 35(6):26–36, 2000.
28. Q. Xie. Oracle spatial, raster data. In *Encyclopedia of GIS*, pages 826–832. Springer US, 2008.
29. Y. Zhang, M. L. Kersten, and S. Manegold. SciQL: array data processing inside an RDBMS. In *Proc. of the 2013 ACM SIGMOD Int. Conf. on Management of Data*, pages 1049–1052, New York, NY, USA, Jun 2013.
30. T. Zhao, C. Zhang, M. Wei, and Z.-R. Peng. Ontology-Based Geospatial Data Query and Integration. In *Proc of the 5th Int. Conf. on Geographic Information Science*,

pages 370–392, Park City, UT, USA, Sep 2008.