DatSha : A Data Sharing Algebra for Access Control Plans

Athanasia Katsouraki[1,2], Luc Bouganim[1,2],
Cedric Eichler[3], *Benjamin Nguyen*[2, 3]

[1]DAVID, [2]Inria Saclay, [3]LIFO / INSA CVL

in *EDBT* 2015
ISN Valdo Project

Context

- Data sharing in Online Social Networks (OSN) is limited
  - Possibilities to define different circles
  - Publishing limited to certain circles

→ Difficulty to publish simultaneously to several circles with different granularities

Our objective : **Define an algebra to**
- **Define Access Control Plans (ACP)**
- **Modify, Combine, Factorise and Share ACPs**

→ **Algebra Definition**

→ Let advanced users share their ACPs with neophyte users
→ Let advanced users define their ACPs with XQuery (3.0)
→ **Let neophyte users better understand access control !! (cf AC by example)**

Example :

Alice wants to share a set of photos with her family,
photos with no metadata with her close friends,
photos without faces (and without metadata) in a reduced definition with her acquaintances,
and does not want to share anything with anyone else.

Application : Monetize Personal Data

Another application : the ACP marketplace

- *Data Consumers* (e.g. private companies) are interested in specific queries (e.g. relational or XQueries)

- *Running example* :

  Find the most photographed place on earth. ← this intension is described by the data consumer

→ *DC* post queries and users decide (or not) to answer.

The ingredients

- An AC model
- The difficulty of writing AC rules
- The XQuery language, and an intensional description of queries

=

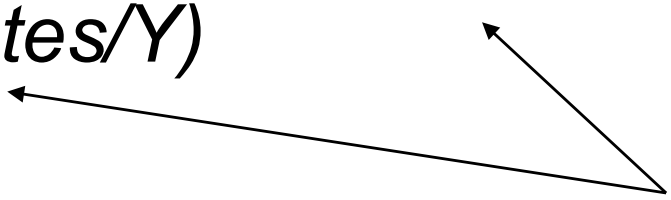
An Access Control Plan (ACP) defined by a succession of polymorphic operators on an initial XML document.

What kind of access is given ?

- Access to atomic information = a document path + an XPath
- Access to an ACP
- The possibility to construct a new ACP using existing access + operators (including calls to external functions).

  Computation of functions is assumed safe.

E.g. *Alice grants to bob the right to call GPS2Country(//GPSCoordinates/X, //GPSCoordinates/Y)*



The ACP marketplace

- Users publish ACPs (=workflows)
- Creator explains the ACP
- Users (maybe others comment on the ACPs + ACP intension)
- Users rank ACPs

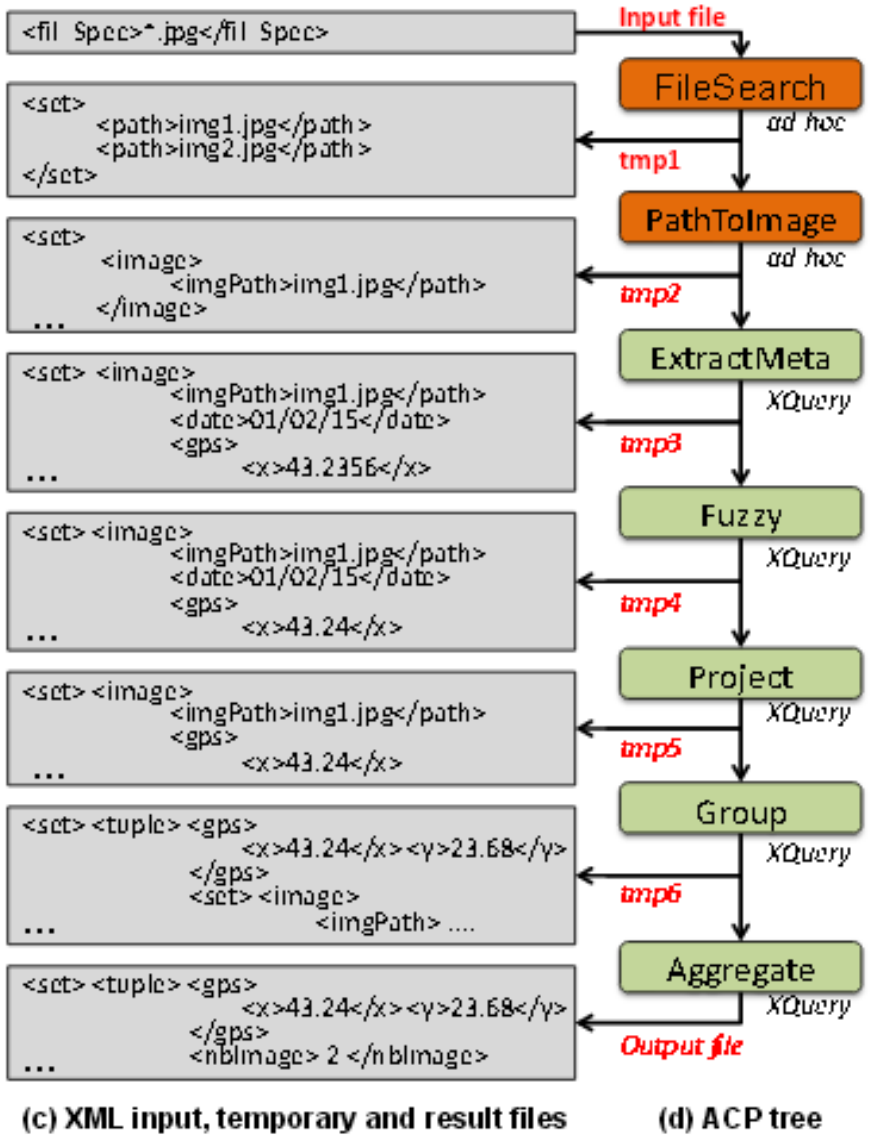
=

Simplify AC definition for neophyte users.

→ easier to understand ACPs

→ easier to reuse existing ACPs (if they trust ACP owner)

ACP Example



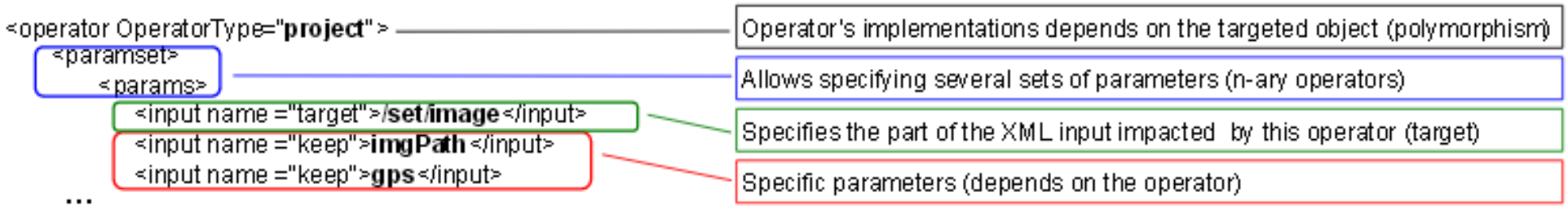
Alice wants to participate in a survey to determine the most photographed place on Earth, which can be done by computing a "fuzzy" location of all her photos, where the "fuzzy" location is defined by GPS coordinates and an error bar

e.g. X=45.23+/-0.01 Y=27.67+/-0.01.

ACP definition

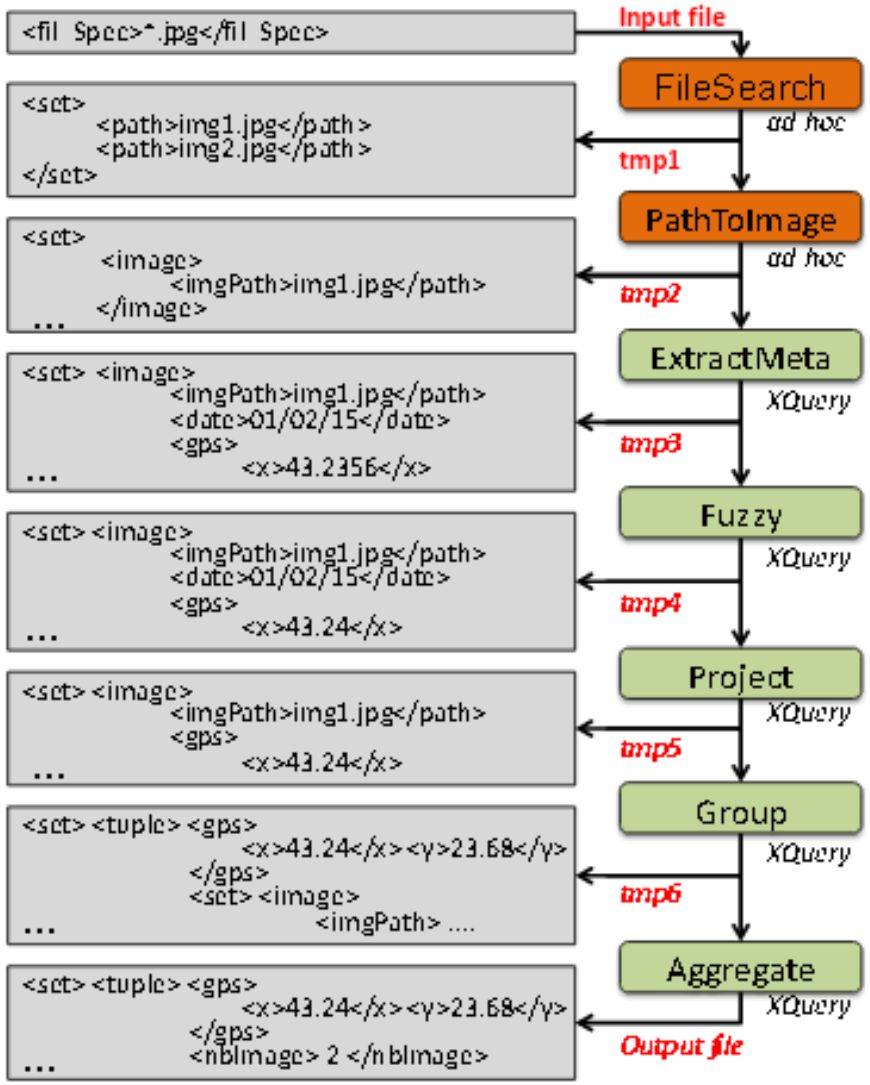
- An ACP = a sequence of operators (there is not necessarily just one linear sequence)
- Operator signature (Typechecking is possible) :
  - Input = xml file
  - Output = xml file
- Operator implementation
  - Ad hoc
  - XQuery 3.0
  - Combination operators (e.g. n-ary join : takes *n* files and produces 1 output)

Operator definition



The *project* operator, defined in the DatSha language, implemented using XPath & XUpdate

Operator Details



(c) XML input, temporary and result files (d) ACP tree

FileSearch: replaces a <fileSpec/> with jokers in a set of file paths looking recursively or not (input "mode") in the directory indicated by input "target" every <fileSpec/> should be replaced by a set of path.

PathToImage: is an operator that replaces every occurrence of a path by an image (an xsd type). An image is at least a <imgPath/> to an "image" file, i.e., a jpg, png, gif, etc.... Initially the image type only includes the <imgPath/> but metadata can be added using the ExtractMeta operator.

ExtractMeta: replaces every occurrence of an image by the same image (every field is copied), and adds metadata that can be extracted from the actual file (e.g. location information embedded in the image).

Fuzzy: is an operator that can be applied to many types. The global behavior is to replace any occurrence of the target by fuzzy values, the precision being informed by the "precision" input, which can be an XPath.

Project: this operator is used like the relational algebra Π operator. It replaces the target subtree by the same subtree in which it keeps only the elements (or subtrees) that are mentioned in the "keep" parameters.

Group: replaces the "target" subtree by a restructured one which must be a set. It constructs a <set> of <tuple>s, each containing n+1 elements (where n is the number of "groupBy" elements in the operator specification, in this example, n = 1). The last element of the tuple is a set of elements that share the same value of groupBy (here a set of image having the same GPS value). This operator is implemented by XQuery 3.0. Group By.

Aggregate : The aggregate operator replaces a set of elements ("target" input) by an aggregate value having the "AggName" name and applying the "AggOperation", which in this case is the XQuery function fn:count().

SEE ACP.XML FILE

Impact

- Simplification of the definition and combination of operators
- Possibility to *monetize* some ACPs = pay for the results of users who execute a given ACP
-

What about privacy ??

- OK to execute monetized ACPs but I want to protect my data using an anonymity model (e.g. *k*-anon)

→ Easy to compose ACPs with privacy preserving workflows (defined & scored by users)

What metadata for an ACP ?

- Precise Information value
- Privacy score
- Anonymization and degradation schemes (both for data & pricing)

ACP modification

- Due to the fact operators form an algebra, we can statically propose ACP modifications that :
  - Degrade (or not) the result
  - Improve the privacy of the computation or the result
- Each user can define locally ACPs (as in virtual private DBs) that will be executed before any other ACP, or that will be executed before specific function calls
- It is also possible to define global constraints (queries) that will restrict the publication of data

→ We want to be able to compute the quality of the result (for pri***cing***)

→ We want to be able to compute the quality of the anonymisation (for pri***vacy***)

→ The computation must take into account all the participants

How to correctly compute all this ?

- *Example* :
  Each user has a HAVING COUNT(*) > K_i condition

- Reverse evaluation :
  - Suppose all individuals answer and remove all data that does not respect the conditions
  - Iterate until fixedpoint is reached

  - Works if : monotonicity (condition must stay false when tuples are removed)

How to securely compute all this ?

Use Trusted Cells paradigm ☺

(See previous SMIS presentations)

Open questions

- Data & query pricing
- Risk evaluation
- ACP optimisation
- Multi-criteria optimization (data + privacy)

Questions ?