Exp Astron (2013) 35:203–225
DOI 10.1007/s10686-011-9272-z

ORIGINAL ARTICLE

Automatic optimized discovery, creation and processing of astronomical catalogs

Hugo Buddelmeijer · Danny Boxhoorn · Edwin A. Valentijn

Received: 8 August 2011 / Accepted: 4 November 2011 / Published online: 7 February 2012

Abstract We present the design of a novel way of handling astronomical catalogs in Astro-WISE in order to achieve the scalability required for the data produced by large scale surveys. A high level of automation and abstraction is achieved in order to facilitate interoperation with visualization software for interactive exploration. At the same time flexibility in processing is enhanced and data is shared implicitly between scientists. This is accomplished by using a data model that primarily stores how catalogs are derived; the contents of the catalogs are only created when necessary and stored only when beneficial for performance. Discovery of existing catalogs and creation of new catalogs is done through the same process by directly requesting the final set of sources (astronomical objects) and attributes (physical properties) that is required, for example from within visualization software. New catalogs are automatically created to provide attributes of sources for which no suitable existing catalogs can be found. These catalogs are defined to contain the new attributes on the largest set of sources the calculation of the attributes is applicable to, facilitating reuse for future data requests. Subsequently, only those parts of the catalogs that are required for the requested end product are actually processed, ensuring scalability. The presented mechanisms primarily determine which

H. Buddelmeijer (✉) · D. Boxhoorn · E. A. Valentijn
Kapteyn Astronomical Institute, Postbus 800, 9747 AD,
Groningen, The Netherlands
e-mail: buddel@astro.rug.nl

D. Boxhoorn
e-mail: danny@astro.rug.nl

E. A. Valentijn
e-mail: valentyn@astro.rug.nl

catalogs are created and what data has to be processed and stored: the actual processing and storage itself is left to existing functionality of the underlying information system.

Keywords Data mining · Data lineage

1 Introduction

Billions of astronomical objects are detected in large astronomical surveys, for which thousands of properties are quantified. The classical way to handle catalog data produced by large surveys, is to create a static relational database with a direct interface to the user. This classical approach has several conceptual drawbacks. The catalogs are published in releases as is; there is very limited flexibility in the derivation of the data. Redoing part of the data reduction entails downloading a large part of the original data and reprocessing it offline. Scientists require knowledge about the internal representation of the data to access it.

Examples of such an approach are the Sloan Digital Sky Survey (SDSS) [5, 9] and the WFCAM Science Archive [6]. It is possible to create user-defined tables in the 'CasJobs' service[1] of SDSS. These are of a limited size and there is no facility to reprocess the data. This is an inherently *pushing* or *forward chaining* approach, that is, scientists create derived catalogs in a stepwise fashion—starting from the released catalogs—until they reach their required end product. The used database queries are stored within the tables, but there is no conceptual information about what the data in the catalogs represent.

This paper discusses the design of novel mechanisms to handle such large catalogs in Astro-WISE through request driven processing. That is, scientists request their required end product, and the information system autonomously determines the best way to provide this catalog. We achieve a high level of automation and implicit scalability, while enhancing flexibility in processing and sharing of data. This is done by using an object oriented data model that focuses on storing information about processing; storing the catalog data itself is of secondary importance.

1.1 Astro-WISE

The Astro-WISE consortium has designed a new paradigm and has implemented a fully scalable information system to overcome the huge information avalanche produced by wide-field astronomical surveys [7, 8]. This is achieved by capturing in a generic way the reality of end-to-end survey operations into a conceptual data model which is translated into hierarchical classes. The model

[1] http://casjobs.sdss.org/CasJobs/

maps all links between dependencies: objects are stored in the database, which links all data products to their dependencies. This creates a *dependency graph* with the *full data lineage* of the entire processing chain.

Astro-WISE uses the advantages of Object-Oriented Programming (OOP) to process data in the simplest and most powerful ways. In essence, it turns the objects that represent conventional astronomical science products, into OOP objects, called *process targets*. Every individual science product, such as frame or catalog, is an instantiation of a specific process target class. Each of these process target instances knows how to process itself to create the data product it represents. Each process target has associated *processing parameters*, which are configurable parameters that guide the processing of that target.

The most unique aspect of Astro-WISE is its ability to process data based on the final desired result to an arbitrary depth. This data pulling is the heart of Astro-WISE and is called *target processing*. Contrary to the typical case of forward chaining such as in the SDSS CasJobs service, the Astro-WISE database links allow the dependency chain to be examined from the intended *process target* all the way back to the raw data. A target's dependencies are checked to see if it is *up-to-date*: if there is a newer dependency or if the target does not exist, the target is (re)created.

1.2 A functional approach to catalogs as Process Targets

Target processing has been incorporated in the image reduction part of the Astro-WISE information system since its inception [8]. Originally, only a few classes were available in Astro-WISE to handle catalog data, of which the SourceList is the most prominent. The SourceList is primarily used to create catalogs with attributes derived from images and has limited functionality for creating new catalogs from existing catalogs. In particular such derived catalogs do not have full data lineage and can therefore not be pulled. Furthermore they can require large amounts of duplication of catalog data, leading to scalability problems.

This paper describes how data lineage and data pulling mechanisms are extended to cover astronomical catalogs with the design of process target classes—which we call *Source Collections*—for catalog data. A Source Collection instance represents a collection of sources (astronomical objects) with attributes (or parameters) that quantify physical properties. There are separate process target classes for different operations to create and manipulate catalogs (Section 3.1). The Source Collection classes take data pulling mechanisms to a higher level than is necessary for images; in particular it is not required to store the catalog data that a Source Collection represents in its entirety.

The full data lineage allows any target to be processed at any time for any reason, since the process parameters unambiguously define how to do so. Ultimately, this means that it is not necessary to process a target completely, or at all. In a sense, this turns the Object-Oriented approach into a Functional one: A process target can also be seen as a representation of the operation that is used derive the science product, in addition to seeing it as a representation

of the result itself. The actual processing of the object and storing the result is then optional. These two viewpoints are equivalent and interchangeable and the contributions in this work stem from this dual perspective:

1. We allow Source Collections to be created—and used as a dependency for other process targets—by specifying their data lineage, without requiring them to be processed, unlike other process targets in Astro-WISE.
2. Dependency graphs of Source Collections are created automatically through data pulling. These mechanisms create new Source Collections in a way that maximizes their reusability for future data pulling requests.
3. We present a novel way to process only the part of a Source Collection that is required for the last process target in a dependency graph. This is done by using the power of backward chaining to temporarily optimize the dependency graph.
4. We use a novel algorithm (Buddelmeijer et al. [1], hereafter Paper II) to infer the logical relationships between catalogs from their data lineage directly. This is required because the exact set of sources that a catalog represents might not be evaluated. This algorithm is used to find Source Collections and for the optimization of dependency graphs.
5. The methods to calculate new attributes from existing attributes are decoupled from their application. This offers scientists flexibility in implementing their own methods while reinforcing the principles of data pulling.
6. The catalog objects and data pulling mechanisms are designed to be used in query driven visualization [2]. The high level of automation allows the data pulling to be abstracted, which implicitly minimizes the processing required to create the visualized datasets.

1.3 Outline

The remainder of the paper is structured as follows. The Source Collection concept is introduced and demonstrated with an example in Section 2. This is followed by a short description of the different Source Collection classes that are implemented in Astro-WISE in Section 3 and a discussion about storing Source Collections and the catalog data they represent in Section 4. Subsequently the concept of *dependency graphs* is explained in Section 5 and their automatic creation through data pulling in Section 6. The optimization of dependency graphs is discussed in Section 7 and their processing in Section 8. A summary and conclusion is provided in Section 9.

2 Introducing Source Collections

A Source Collection is an Astro-WISE process target (Section 1.1) for the handling of astronomical catalogs. These catalogs consist of sets of sources and attributes that quantify properties of the sources. The exact set of sources and

the values of the attributes is determined by processing a Source Collection. When we refer to *catalog data*, we mean this processing result. A Source Collection can also be seen as a representation of the action required to derive the catalog data, since a Source Collection can be created without being processed. We refer to this action in a conceptual sense as the *operator* of a Source Collection and define separate process target classes for different operations on catalogs (Section 3.1).

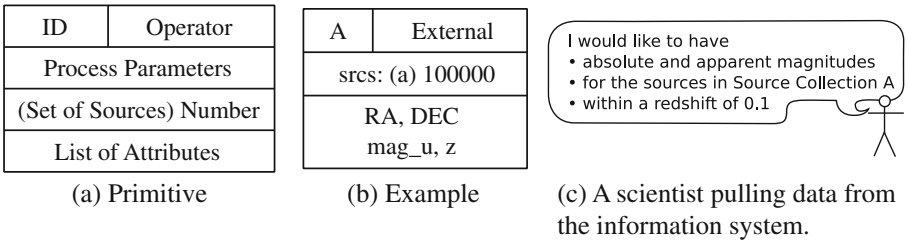
Every source in a Source Collection has a unique identifier and two Source Collections are considered to represent the same sources if and only if the identifiers of their sources are identical. A source itself can be seen as an object in the computer science reading of the term. A parametrized property of a source can then be seen as an attribute of such an object. We will use the term *attribute* instead of *parameter*, which originates from this object oriented approach. Attributes quantify physical properties of the sources in a Source Collection and the set of attributes forms the Source Collection dimensions. The label of an attribute only describes what physical property is represented by the attribute. This labeling could be standardized, for example with Unified Content Descriptors; for the scope of this paper we will refer to attributes by their name only.

Every Source Collection instance is linked to its dependencies, forming a dependency graph all the way to the raw data. Dependencies of a Source Collection that are Source Collections themselves are also called its *parents*, because the catalog represented by the Source Collection is derived from them. Such a dependency graph can be visualized as interconnected nodes. In the figures in this paper, the dependencies of a process target are shown above it. Therefore the data processing runs from top to bottom and the data lineage from bottom to top. Such a dependency graph can conceptually be extended in both directions. The top nodes will contain photometric attributes and can be connected to nodes representing frames that were used to measure these attributes. The bottom nodes can be connected to nodes representing hypothetical process targets for graphs or other analysis products.

2.1 Source Collection example

We demonstrate the Source Collection concept with a simplified example of data pulling. We assume the existence of a Source Collection (labeled A, Fig. 1) that contains apparent magnitudes and redshifts for a large set of galaxies. A scientist pulls a dataset with both absolute and apparent magnitudes for nearby galaxies. First, the scientist formulates a data pulling request (Fig. 1) in which three pieces of information are specified:

- The data set from which the sources should be selected: Source Collection A.
- The selection criterion for the sources: a redshift below 0.1.
- The required attributes: absolute and apparent magnitudes.



(a) Primitive (b) Example (c) A scientist pulling data from the information system.

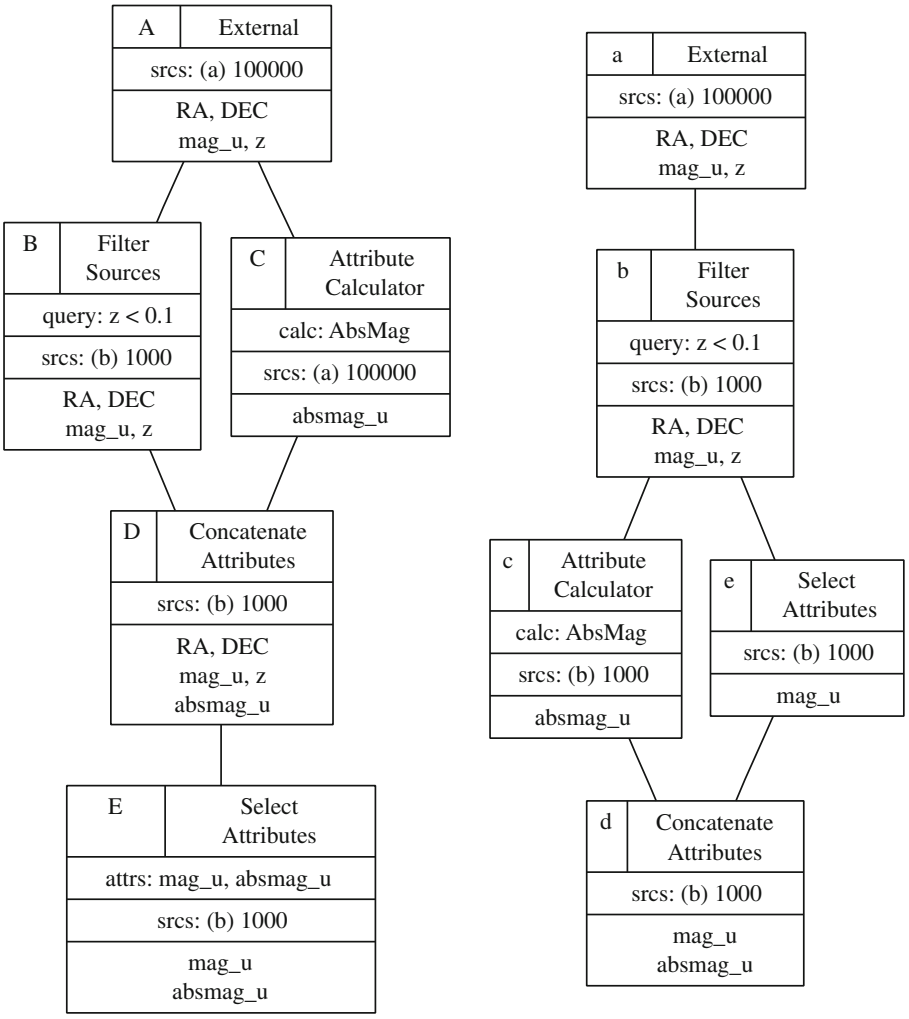
Fig. 1 **a** A Source Collection primitive, **b** a representation of a Source Collection used in the example and **c** the scientist formulating a data pulling request. The following elements can be seen in the Source Collection representation: *Top left*: a unique identifier of this Source Collection. *Top right*: the operator of the Source Collection. The *second row* represents the process parameters, if any. The *next row* describes the sources of the Source Collection. The number on the *right* is the number of sources and the letter between parenthesis represents the exact set of sources. Source Collection with the same symbol represent the exact same set of sources; different symbols might represent different sets. At the *bottom:* the names of the attributes that are represented by this Source Collection; in this case celestial coordinates, an apparent magnitude in the *u* band and a redshift

Subsequently, the information system creates the required Source Collections (Fig. 2a):

- Source Collection *B* is created to select all sources that match the given selection criterion.
- The information system determines that no absolute magnitudes have been defined for these sources and it creates Source Collection *C* to calculate absolute magnitudes from apparent magnitudes.
- The information system determines that the calculation can be performed on all sources in Source Collection *A*. Therefore, it optimizes for generality and uses Source Collection *A* as dependency for Source Collection *C*, instead of *B*. The Source Collection is not yet processed at this stage.
- Source Collections *D* and *E* are created to combine the attributes represented by different Source Collections and select the required ones.

Finally, the created dependency graph is optimized and processed (Fig. 2b):

- The information system creates a temporary copy of the dependency graph in order to optimize it for scalability to fulfill the request as quickly as possible.
- It reorganizes the dependency graph to minimize the required processing by placing the selection of sources before the calculation of absolute magnitudes.
- The information system retrieves the data of Source Collection *b* and uses this to process Source Collection *c* completely. The calculated attributes will be stored for future requests as part of Source Collection *C*, because they cannot be derived on the fly.



(a) Persistently Stored Source Collections

(b) Transient Source Collections used for Processing

Fig. 2 Two dependency graphs of Source Collection, generated by the information system. Every box represents a Source Collection. The Source Collections on the *left* are persistently stored, where the Attribute Calculator is defined as general as applicable, to facilitate reuse. The Source Collections on the *right* are temporary and transient, where the Attribute Calculator is defined as specific as possible, to minimize the required processing

– The other Source Collections are processed on the fly while retrieving the catalog data of Source Collection *e*. The catalog data is subsequently returned to the scientist.

2.2 Key features in example

The example in Section 2.1 highlights the key aspects of the Source Collection:

- Catalog data is pulled and new Source Collections are created to compute attributes that do not yet exist (Section 6).
- The final catalog has full data lineage: any attribute value can be recalculated and the selection criterion is stored (Section 4).
- Calculations are defined to be as general as applicable. Source Collection C can be reused if at a later stage absolute magnitudes are requested for another subset of Source Collection A (Section 6).
- The information system reorganizes the order of the Source Collections to prevent the calculation of unnecessary data (Section 7). The algorithm to determine logical relationships between sets of sources of Paper II is used for more complex dependency graphs.
- Source Collection C is processed partially by processing its smaller copy c entirely and sharing the result (Sections 4 and 7).
- The calculation of the absolute magnitudes can be performed on the workstation of the scientist or on a distributed computing cluster, while the selection of data can be performed on the database (Section 7).

3 Source Collection classes: elementary operations on catalogs

Many of the novel features of the Source Collections originate from the ability of the information system to assess aspects of the catalogs by inspecting only the data lineage. This is achieved by having a predefined set of operations that can be used to process a Source Collection. Separate process target classes are designed for the different operations. We use the term *operator* to refer to the action required to create the catalog data.

These operators are designed to be as elementary as possible in order to maximize the information that can be inferred from the data lineage directly. Therefore, there are no Source Collection operators that are entirely user-defined. However, the behavior of Source Collections can be influenced by setting the process parameters. For example, we do define an operator to calculate new attributes of sources from existing attributes (Section 3.2). This allows scientist to specify their own calculation method as a process parameter.

There are two main effects of the elementary operators: firstly, they allow the information system to determine whether a Source Collection can be used in the construction of a dependency graph (Section 6). Secondly, they allow efficient reorganization of the dependency graph, e.g. for partial processing (Section 7).

Most operators we define are modeled after relation operations [3] to allow them to be evaluated on the Astro-WISE database. In essence, we extend SQL commands to target processing, although this is not directly our goal. The important aspect in the design of the operators is maximizing the information

that can be inferred from the data lineage. Not all operators we describe can be evaluated on SQL and vice versa, most operators can be evaluated in the Astro-WISE Python environment as well.

3.1 List of classes

We summarize the operators that are most important for our research:

- **Select Attributes**: Selects a subset of attributes from a parent Source Collection.
- **Concatenate Attributes**: Combines the different attributes from several parent Source Collections that represent the same sources.
- **Rename Attributes**: Renames attributes of a parent Source Collection.
- **Filter Sources**: Selects a subset of sources from a parent Source Collection by evaluating a selection criterion.
- **Select Sources**: Selects a subset of sources from a parent Source Collection by listing the required sources explicitly.
- **Concatenate Sources**: Combines the different sources of several parent Source Collections that represent the same attributes.
- **Relabel Sources**: Changes the source identifiers of a parent Source Collection.
- **Attribute Calculator**: Calculates new attributes from existing attributes for the sources in a parent Source Collection (Section 3.2).
- **External**: Represents a catalog without data lineage.
- **Pass**: Represents the exact same catalog as its parent.
- **SourceList Wrapper**: A special Source Collection to use the Astro-WISE SourceList class as a Source Collection. The SourceList class is used to detect sources from images and measure photometric and related attributes.

3.2 Generic operator for attribute calculation

A special Source Collection class is designed for the calculation of new attributes of sources from existing attributes. The calculation performed by a Source Collection of this class, is decoupled from the definition of the class and is stored as another persistent object, which can be created by scientists themselves.

This auxiliary object is called an *Attribute Calculator Definition* and contains both information about how to perform the calculation as well as information about the calculation itself: which attributes are calculated, which attributes are required and which process parameters can be set. This allows the information system to discover attribute derivation methods in order to instantiate Source Collections to calculate these attributes for a requested set of sources. This offers scientists flexibility in implementing their own methods while reinforcing the principles of data pulling.

Multiple Attribute Calculator Definitions might exist for the calculation of the same attribute, for example through different methods or different versions of the same method. Astro-WISE has functionality to indicate that stored objects should not be used anymore by invalidating them, for example when a newer version of the object exist. This is used within the Source Collections to indicate that newer versions of Attribute Calculator Definitions exist. This allows existing functionality to be used for ensuring that catalogs are always created with the latest method and that out-dated catalogs are flagged for possible recreation.

4 Storing data lineage instead of tables

SourceCollections can be created and stored by specifying their data lineage only; it is not required to process them. That is, the actual determination of the exact composition of sources in a catalog, and the calculation of the values of their attributes, is delayed as long as possible. Furthermore, the result of the processing is stored only if necessary for performance reasons and the results can be shared between Source Collections. We summarize the benefits of this approach:

- Different Source Collections can represent partially identical catalogs without any duplication of stored data.
- The processing of intermediate Source Collections can be limited to those subsets that are required for the end node of a dependency graph. Source Collections can therefore be created with arbitrary sizes without performance penalties. This ensures maximum reusability of the created Source Collections.
- No results have to be stored at all for Source Collections that can be processed on the fly.

4.1 Source Collection persistent properties

The *persistent properties* of a process target are the properties of the object that are stored in a database. These properties can be grouped in the following types, a categorization that is especially important for Source Collections:

- **Data Lineage**: Properties that define the catalog that is represented by the Source Collection. These are dependencies and process parameters. Dependencies are other process targets from which the catalog represented by this Source Collection is derived, often Source Collections as well. Process parameters influence the processing and are defined by the class of the Source Collection. The dependencies and process parameters together unambiguously define the catalog that the Source Collection represents.
- **Processing Results**: Results of processing the Source Collection, detailed in Section 4.2.

– **Other Properties**: Properties that do not refer directly to the processing or the processing results. These include identifiers of the object, a human readable name of the Source Collection, a reference to its creator, status of the processing, etc. Some of these can be specified by the user, others are set automatically by the information system.

4.2 Processing results

The result of processing a process target instance (Section 1.1) can be stored persistently. The processing results of image classes are primarily the values of the pixels of the image, which in Astro-WISE are stored as FITS files on the dataserver. For Source Collections the primary result is the catalog data it represents, which in Astro-WISE is stored in the database.

The Source Collection classes are designed to allow partial processing of objects, for example because only a part of the catalog data is required at a specific moment. The processing results are split up in distinct components in order to achieve this. These components can, in principle, be processed separately. The following results can be distinguished:

– The catalog the Source Collection represents: the values of all the attributes for all the sources. This is the primary processing result and can be decomposed in the partial results that follow.
– A partial catalog: the values of the attributes for a subset of the sources or attributes.
– The set of sources the Source Collection represents, which can be seen as a list of identifiers of the sources. This can be further split up into the number of sources, or an identification of the set without actually enumerating all the sources individually.
– The set of attributes of the sources. That is, which physical properties the Source Collection represents, not the actual values of the attributes.

To process a Source Collection partially, a new process target is created that only represents the required component, which is subsequently processed in its entirety. Such a component is either stored in its entirety or not at all, and can be shared between Source Collections.

The sharing of processing results leads to multiple paths to the same stored data. The dependency graphs representing these different paths are only created automatically by the information system through modifications of existing dependency graphs (Section 5.1). The information system ensures that the different paths are equivalent by only performing modifications where this is guaranteed.

5 Source Collection dependency graphs

A Source Collection represents a catalog that is derived from its dependencies, which again have dependencies themselves. These dependencies chain

a Source Collection back to the raw data and form a graph of process targets. The term *dependency graph* is used to refer to this complete set of dependencies of a Source Collection. These graphs are *directed acyclic graphs*, or *acyclic digraphs*, because there are no cyclic dependencies [10].

In the figures depicting dependency graphs in this paper, the dependencies of a Source Collection are shown above it. Therefore the data processing runs from top to bottom and the data lineage from bottom to top. There are no arrows on the shown edges, because the preferred direction is dependent on context. This paper only treats the part of such a dependency graph that considers Source Collections.

5.1 Modifications of dependency graphs

The information system can modify dependency graphs of Source Collections, e.g. while constructing new ones or when optimizing existing ones as discussed in the next sections. All modification steps in the following algorithms are performed by replacing a Source Collection with another one. There are two ways to do this:

- Replacing a Source Collection with another one that represents the exact same catalog. This is the only mechanism that is used in the dependency graph optimization (Section 7).
- Replacing a Source Collection with one that represents a different catalog. This is only performed during the creation of new dependency graphs (Section 7) and only on dependencies of the Pass Source Collections at the end of the graph.

The individual modifications themselves are designed in a way that separates the knowledge of *how* to perform a modification and *why* to do so. How to perform a modification is part of the definition of the Source Collection classes. Whether a specific modification should be applied is the responsibility of the part of the information system that governs the entire dependency graph. Therefore, all modifications are between a Source Collection and its direct dependencies, because an individual Source Collection has no knowledge of other objects.

A specific kind of modifying a dependency graph is ‘moving’ Source Collections through the graph. The way this should be interpreted—in simplified form—is that copies of a Source Collection and its parent are created, but with their dependencies swapped. The original Source Collection is then replaced by these copies. As a result, Source Collections can only be moved ‘up’ the graph. To move a Source Collection down, the Source Collection with that Source Collection as a parent should be moved up.

Some modifications can only be performed if the relationship between the sets of sources of the involved Source Collections is known. The information system uses the algorithm of Paper II to provide this information to the individual Source Collections.

6 Pulling catalogs

The 'pushing' way to use catalogs such as represented by the Source Collections is to define the catalog, process and store it, and then request subsets of the catalog. This order is changed with target processing [8].

Source Collections are primarily created automatically by pulling data, which means that the evaluation of processing starts at the end of the chain by requesting the final catalog that is required. The information system will autonomously create a dependency graph of Source Collection which ends with a Source Collection that represents the requested catalog (Algorithm 1).

There are two main goals of the data pulling mechanisms with respect to the creation of the dependency graph: Firstly, they ensure that existing Source Collections will be reused as much as possible and secondly, new Source Collections are created in a way that maximizes their reusability.

6.1 Data pulling: formulating a request

The pulling of data starts with a request for a specific dataset. In our research we have limited such requests to three pieces of information:

- A starting Source Collection from which a selection is made.
- A list of required attributes, not necessarily represented by the starting Source Collection.
- Optionally, a selection criterion for the sources.

Algorithm 1 Creating Target Dependency Graph

```
1: Receive and parse a request for catalog data.
2: Instantiate the starting Source Collection.
3: Create a Select Attributes that selects an empty attribute list from this
   Source Collection.
4: Create a Pass Source Collection with the Select Attributes as parent.
5: if a selection criterion is specified then
6:     Select the right sources (Algorithm 2).
7: end if
8: for all requested attributes do
9:     Add the attribute to the Pass Source Collection (Algorithm 3).
10: end for
```

6.2 Data pulling: derivation preferences

The information system will use existing and newly created Source Collections to create a dependency graph which ends with a Source Collection representing the requested catalog. The information system is able to autonomously decide how to proceed if there are multiple Source Collections that can be used to fulfill a particular dependency. This is done by applying a ranking

function to all Source Collections that can be used and select the one with the highest ranking. Scientists can influence this process by specifying their own ranking function or by overruling the choices made by the information system manually.

6.3 Data pulling: selecting sources

Fulfilling a request for a catalog begins with creating a Source Collection with the correct the composition of sources (Algorithm 2). The resulting Source Collection will only represent the selected set of sources at this stage, without attributes.

Algorithm 2 Selecting Sources

1: Search for all Source Collections representing the original sources.
2: Search for all Filter Sources with one of these Source Collections as parent and the specified criterion as parameter.
3: Rank all found Source Collections.
4: **if** a suitable Source Collection is found **then**
5: Use the highest ranking Source Collection to represent the sources.
6: **else**
7: Use Algorithm 1 to create a Source Collection with all attributes referenced in the selection criterion.
8: Create a new Filter Sources to represent the sources.
9: **end if**
10: Create a Select Attributes to select no attributes from the Source Collection representing the sources.
11: Create a Select Sources with the original Source Collection as parent and the Select Attributes to specify the selected sources.
12: Use the Select Sources as the parent of the final Pass Source Collection.

In this paper we restrict ourselves to requesting subsets of sources that are already represented by an existing Source Collection, because our focus is on operations on catalogs. In particular we assume the existence of Source Collections with photometric and related attributes derived from images. These catalogs could be created through pulling mechanisms as well; this is beyond the scope of this paper.

The logical relations algorithm of Paper II is used to search for an existing Source Collection that represents the requested selection. First all Source Collections that represent the same sources as the original Source Collection are found. Subsequently a Filter Sources is sought, one with the specified selection criterion and with one of these Source Collections as parent. New Source Collections are created to select the required sources if no suitable Source Collection is found. This might require more than only a single Filter Sources Source Collection because the information system has to ensure that the attributes used in the selection criteria are available.

For example, the specified selection criterion in the example in Section 2.1 depends on the availability of the redshift attribute. The information system would have tried to find this attribute if it would not have been included in Source Collection A.

A Select Attributes Source Collection is created to select no attributes from the found or created Source Collection with the sources. The required attributes are subsequently added to this new Source Collection with only sources (Section 6.4).

6.4 Data pulling: selecting attributes

A catalog pulling request should contain a list of required attributes. For every requested attribute, the information system will search for an existing Source Collection that represents this attribute for the requested sources. A hierarchy of Select Attributes and Concatenate Attributes Source Collections is created to add the attribute to the Pass Source Collection already representing the sources (Algorithm 3).

Algorithm 3 Adding Attributes

1: Search for all Source Collections representing the attribute.
2: Rank all found Source Collections.
3: **if** a suitable Source Collection is found **then**
4: Use the highest ranking Source Collection to represent the attribute.
5: **else**
6: Create an Attribute Calculator to represent the attribute (Algorithm 4).
7: **end if**
8: Create a Select Attributes that selects the requested attribute from this Source Collection.
9: Create a Concatenate Attributes with the original parent of the final Pass Source Collection and the new Select Attributes as parents.
10: Use the Concatenate Attributes as new parent of the final Pass.

Requested attributes for which no suitable Source Collections can be found, are derived with new Source Collections (Algorithm 4). In this paper we limit ourselves to attributes that are derived from other attributes using Attribute Calculators Source Collections. The calculation performed by an Attribute Calculator is specified through a process parameter referencing an Attribute Calculator Definition object. New Attribute Calculator Source Collections are instantiated for all Attribute Calculator Definitions that can be used to derive the requested attribute. The search for attributes is applied recursively if more attributes are required for the derivation of the requested attributes, as specified by the Attribute Calculator Definition.

Source Collections that require the calculation of new attributes will automatically be defined to operate on the largest dataset the calculation is applicable for. This is done by giving Source Collections which represent a larger set of sources a higher ranking when searching for attributes.

Algorithm 4 Instantiate Attribute Calculators

1: Search for all Attribute Calculator Definitions that can be used to calculate the required attribute.
2: **for all** found Attribute Calculator Definitions **do**
3: Create a Source Collection with all the attributes required by the Attribute Calculator Definition (Algorithm 1).
4: Create an Attribute Calculator with that Source Collection as parent, using the Attribute Calculator Definition.
5: **end for**
6: Rank all created Attribute Calculators.
7: Use the highest ranking Attribute Calculator to represent the attribute.

6.5 Data pulling: storing Source Collections

The result of data pulling is the creation of a dependency graph that ends with a Source Collection that represents the requested catalog. The Source Collections in this dependency graph might be stored persistently if necessary. The information system will subsequently optimize this graph to process it in the most optimal way (Section 7).

7 Optimization of dependency graphs and processing

The information system will optimize the dependency graph of a Source Collection before processing the Source Collections that it contains. There are two goals to these optimizations: minimization of the required processing and optimization of the processing itself. These optimizations are performed on a temporary transient copy of the dependency graph, which can be discarded after the processing is completed.

Reducing the necessary processing to the minimum required for the last Source Collection is the primary goal of this paper. In essence this is done by placing filtering Source Collections before Source Collections that create new attributes and removing parts of the dependency graph that are not necessary for the final result. This will ensure that the Source Collections in the dependency graph only represent data that is required for the requested catalog data. These mechanisms allow the information system, or the scientist, to create and store Source Collections instances in their most general and reusable form, (e.g. as in Fig. 2a), because the creation and storage of the catalog data is minimized automatically (e.g. as in Fig. 2b).

Optimization of the processing itself is a secondary goal of this paper. This is done by reorganizing the dependency graphs such that the processing can be performed on the most suitable subsystem of the information system. For example, Source Collections that can best be processed on the database are placed such that they can be combined into one SQL query and processed

together. Parts of the dependency graph can be parallelized in order to process large Source Collections on a distributed cluster, especially those that cannot be processed on the database.

Optimizing the dependency graph of the example in Fig. 2 is depicted in Fig. 3. The required processing in this example is dominated by a calculation of absolute magnitudes. Without optimization, absolute magnitudes have to be calculated for 100,000 sources; with optimization the calculation is only performed for the 1,000 sources that are actually requested, resulting in a factor 100 increase in performance. The optimizations required to determine the exact set of sources in a Source Collection is depicted in Fig. 4. In this case, the calculation of absolute magnitudes is removed from the dependency graph entirely and the entire graph can be processed on the database.

7.1 Dependency optimization: strategy

The best strategy for the optimization of dependency graphs depends on many factors, such as the size of the catalogs, how they will be processed, etc. Therefore it is not possible to give a one-size-fits-all optimization strategy. Algorithms 5 and 6 are procedures that cover most scenarios, they can be adjusted for particular cases. The steps described in the algorithms are detailed in the rest of the section.

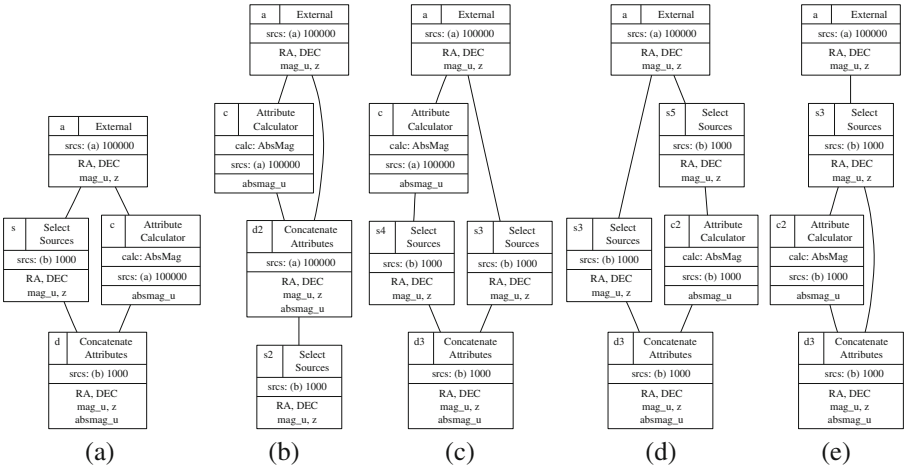


Fig. 3 Source limiting optimizations for before processing. On the *left* a transient copy of the Source Collections of Fig. 2a, without the final Select Attributes. The criterion of the Filter Sources has been evaluated so its sources are known and is converted in a Select Sources, which explicitly lists the sources. The Select Sources can move through the dependency graph freely, since its operator is not dependent on the attributes of the parent anymore. The Select Sources is moved down to the end of the dependency graph in **b** and subsequently back up in **c** and **d**. In **c**, two copies of the Select Sources are created, which are merged back into one in **d**. The temporary transient copy Attribute Calculator *c*2 describes a subset of the original Source Collection *C*, thereby reducing the required processing

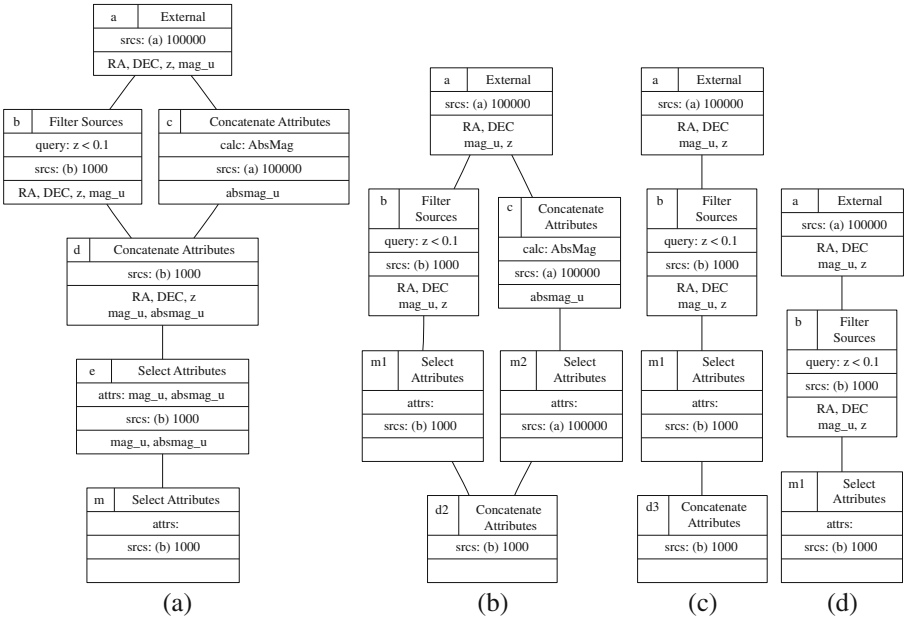


Fig. 4 Evaluation of the exact set of sources in a Source Collection. On the *left* a transient copy of the Source Collections of the example in Fig. 2a. A Select Attributes that selects no attributes is placed at the end of the dependency graph, which is subsequently moved up in **b**. One of the parents of the Concatenate Attributes can be removed, because the algorithm from Buddelmeijer et al. [1] is used to infer that it is not required anymore. The Concatenate Attributes itself is removed because it has only one parent. The final dependency graph does not involve any calculation and can be evaluated quickly

Algorithm 5 Optimization for Processing

1: Create transient copy of the involved Source Collections.
2: Simplify the dependency graph (Algorithm 6). Perform this step after every movement-step.
3: Move all Select Attributes up the graph, to remove parts of the graph.
4: Convert all Filter Sources to Select Sources, to move them through the graph.
5: Move all Select Sources down, to copy them to every part of the graph.
6: Move all Select Sources up the graph, to limit processing to the required subset.
7: Move all Select Attributes up the graph once more, to simplify the graph further.
8: Move all Select Sources up the graph in order to combine them.

Algorithm 6 Simplification for Processing

1: **repeat**
2: Convert processed Source Collections to External Source Collections.
3: Remove parts of the graph with unnecessary dependencies.
4: Remove Source Collections that are essentially a Pass Source Collection.
5: Integrate Source Collections and their parents if possible.
6: Unite identical Source Collections, especially those with the same parents.
7: **until** no more of these modifications are possible.

7.2 Dependency optimization: simplifications

A dependency graph of Source Collections can be simplified as part of the optimization routines (Algorithm 6). A Source Collection that has already been fully processed does not have to be processed again. All these Source Collections can be substituted with an External Source Collection that represents the same catalog. Furthermore, the complexity of a dependency graph can be reduced by combining operators or removing redundant ones.

For example, the initial Source Collection in Fig. 2 is an External Source Collection for simplicity. In a realistic scenario, this Source Collection would have dependencies of its own and would only be substituted with an External Source Collection just before processing. In Fig. 4a two serial Select Attributes Source Collections are combined and in Fig. 3d two parallel Select Sources Source Collections are combined.

7.3 Dependency optimization: removing dependencies

Unnecessary parts of a dependency graph can be removed by moving Select Attributes Source Collections up in the graph (Algorithm 5). The result of moving a Select Attributes up past a Concatenate Attributes, might be that one of the dependencies of the Concatenate Attributes does not represent attributes anymore. The part of the graph that ends with this dependency might be removed from the graph in its entirety.

The set of sources of a Concatenate Attributes is the intersection of the sets of sources of its parents. Therefore it is only possible to remove this part of the dependency graph if doing so does not influence the selection of sources. The logical relations algorithm of Paper II is used to determine whether this is the case.

7.4 Dependency optimization: sources limitation

Select Sources Source Collections are moved through the dependency graph to ensure that only those parts of the Source Collections are processed that

are required to create the catalog data of the end node (Algorithm 5). Filter Sources Source Collections first have to be converted into a Select Sources. Before moving the Select Sources Source Collections up the dependency graph, they are moved down in order to copy them to all parts of the dependency graph they are applicable to.

In Fig. 3 the Select Sources Source Collection is moved up through the Attribute Calculator Source Collection. This creates a copy of the Attribute Calculator that represents a subset of the sources of the original.

7.5 Dependency optimization: parallelization

The Source Collections are well suited for parallelization because they are processed on a per-source basis. A Source Collection can be parallelized by creating a set of Select Sources (or Filter Sources) Source Collections that each select a subset of the original target, such that all sources are selected exactly once. The set of Select Sources Source Collections is then combined with a Concatenate Sources Source Collection which can replace the original Source Collection in the dependency graph. Further optimization can move the Select Sources up to parallelize the entire graph. The parallelization algorithm is currently not implemented in Astro-WISE.

8 Processing and storage

The result of the dependency graph optimization is a set of Source Collections that requires the least amount of processing to create the requested catalog data. The information system will recursively process the Source Collections and store the results if necessary.

The mechanisms designed for the research presented in this paper are intended be used in conjunction with existing large-scale data storage and processing facilities. Therefore, the precise way catalog data is processed and stored is largely beyond the scope of this paper and will depend on what is available in the information system. We give a general discussion of how the processing and storage could be achieved and highlight how this is implemented in Astro-WISE.

In particular, for this paper we assume the existence of mechanisms for authentication of users, privilege management and for queuing requests when processing Source Collections on shared resources.

8.1 Processing: processing Source Collections

The information system can process the Source Collections on the most suitable subsystem to achieve scalability for large scale catalogs and real time

interaction for small scale catalogs. We describe the different subsystems to evaluate the operators on:

- **Database**: The selection and combining operators are designed to be evaluated on a database. The operators of consecutive Source Collections can be combined into one database query. The database can create indexes on columns containing attributes that are frequently used in selection criteria and can automatically cache the results. Some Source Collections, in particular Attribute Calculators, will not be suitable to be processed on the database.
- **Workstation**: The processing can be performed on the workstation of the scientist for Source Collection that cannot be processed on the database. Furthermore, all the relational operators should also be evaluated on the local machine during interactive visualization of small datasets. The latency of a round trip to the database or distributed computing facility is too large for responsive interaction. Such a local implementation of the Source Collections holds all the catalog data in memory or in files.
- **Distributed Computation**: Operators that require large computations can be performed on a distributed processing cluster. This is done by parallelizing the respective parts of the dependency graph and evaluating each sub-graph on a cluster node.

Within Astro-WISE, most operators can be performed on both the Oracle 11g database or in the Python. Database queries usually scale linearly; requests similar to the example of Section 2.1 are typically delivered with speeds of 100,000 source attributes per second in the current setup. There is currently no explicit functionality to process Source Collections on the distributed processing cluster.

8.2 Processing: storing catalog data

The result of processing a Source Collection—the exact set of sources and the values of the attributes—only has to be stored if this is necessary for performance reasons. Therefore we make no explicit distinction between storing and caching of catalog data.

The optimization process (Section 7) creates copies of Source Collections that represent subsets of the originals. If the information system decides to store the processing result of such a copy, it will append the catalog data of the copy to that of the original. Deciding what should be stored is primarily the responsibility of the information system. The decision should be made for individual processing results.

For example, it can be useful to store the identifiers of the sources without storing the values of the attributes in order to store the result of evaluating a complex selection criterion. Different Source Collections that represent the same sources can share this processing result.

A key principle of the presented research, inherited from Astro-WISE, is that a Source Collection cannot be altered once stored. Therefore, any stored

catalog data of a Source Collection cannot change either. Reliability of the data storage, e.g. ACID properties [4], is automatically achieved as a result. For example, an incomplete database transaction will not leave the database in an invalid state because these will only append data that could also be ingested partially in the first place.

Stored catalog data can, in principle, also be deleted at any time, since it can always be recreated. A deletion mechanism is currently not incorporated in Astro-WISE; instead, all catalog data is backed up regularly to be able to recover quickly from database failures.

8.3 Processing: retrieving catalog data

The last node in the dependency graph represents the requested catalog. Once it has been processed, the catalog data can be returned to the scientist or used for further analysis or visualization. Any temporary transient Source Collections instantiated for the processing are discarded.

9 Summary and conclusions

The presented work shows a novel approach for the handling of source catalogs, as incorporated in Astro-WISE. The core difference between the Astro-WISE approach and the way astronomical catalogs are traditionally disclosed, is that the user works with data models rather than a set of tables in a relation database. We showed how data pulling is extended to source catalogs, a first step to data pulling and data lineage in the analysis domain. A process target—labeled a Source Collection—is designed to represent catalog data and operations thereon. We summarize the key features of our design:

- Source Collections are primarily created automatically by the information system through data pulling. Source Collections that derive new data are created as general as possible in order to facilitate reuse and to prevent duplication of data.
- Source Collections allow a functional approach to target processing: they can be seen as the operation to create the catalog data. A Source Collection is only processed when this is required, not necessarily at the moment it is created. Every Source Collection class correspond to an elementary operation on catalogs; complex operations should be split over multiple Source Collection instances.
- The Source Collection have full data lineage, which allows the information system to assess aspects of the catalogs without processing them. For example, it allows the information system to optimize the a dependency graph of Source Collections before processing it.
- A Source Collection is processed by creating temporary copy of the dependency graph and reordering the dependencies so they are as specific as possible in order to minimize the required processing.

– A generic Source Collection class is designed for the calculation of new attributes from existing attributes. This offers a framework for scientists to implement their own methods while enforcing the benefits of full data lineage and data pulling.

The Astro-WISE way of handling astronomical catalogs takes care of most of the administrative tasks automatically. Discovery of existing catalogs and creation of new catalog is done in the same way, by requesting the required end product. Catalogs are shared implicitly, because existing catalogs are discovered automatically. New catalogs are automatically created in their most general form, but only the necessary parts are processed. Together, this allows scientists to focus on the data itself and the science they want to perform instead of how the data is handled.

Acknowledgements This research is part of the project "Astrovis", research program STARE (STAR E-Science), funded by the Dutch National Science Foundation (NWO), project no. 643.200.501.

Astro-WISE is an on-going project which started from a FP5 RTD programme funded by the EC Action "Enhancing Access to Research Infrastructures".

References

1. Buddelmeijer, H., Valentijn, E.A.: Leveraging data lineage to infer logical relations between sets. Paper II. Exp. Astron. (2011a). doi:10.1007/s10686-012-9288-z
2. Buddelmeijer, H., Valentijn, E.A.: Query driven visualization of astronomical catalogs. Exp. Astron. (2011b). doi:10.1007/s10686-011-9263-0
3. Codd, E.F.: A relational model of data for large shared data banks. Commun. ACM **13**(6), 377–387 (1970)
4. Gray, J.: The transaction concept: virtues and limitations. In: Proceedings of the 7th International Conference on Very Large Databases, pp. 144–154 (1981)
5. Gray, J., Szalay, A.S., Thakar, A.R., Kunszt, P.Z., Stoughton, C., Slutz, D., van den Berg, J.: Data Mining the SDSS SkyServer Database. ArXiv Computer Science e-prints (2002)
6. Hambly, N.C., Collins, R.S., Cross, N.J.G., Mann, R.G., Read, M.A., Sutorius, E.T.W., Bond, I., Bryant, J., Emerson, J.P., Lawrence, A., Rimoldini, L., Stewart, J.M., Williams, P.M., Adamson, A., Hirst, P., Dye, S., Warren, S.J.: The WFCAM science archive. Mon. Not. R. Astron. Soc. **384**, 637–662 (2008)
7. McFarland, J.P., Verdoes-Kleijn, G., Sikkema, G., Helmich, E.M., Boxhoorn, D.R., Valentijn, E.A.: The Astro-WISE optical image pipeline development and implementation (2011). doi:10.1007/s10686-011-9266-x
8. Mwebaze, J., Boxhoorn, D., Valentijn, E.A.: Astro-wise: tracing and using lineage for scientific data processing. In: Proceedings of the 2009 International Conference on Network-Based Information Systems, NBIS 09, pp. 475–480. IEEE Computer Society, Washington (2009)
9. Szalay, A.S., Gray, J., Thakar, A.R., Kunszt, P.Z., Malik, T., Raddick, J., Stoughton, C., vandenBerg, J.: The SDSS SkyServer: Public Access to the Sloan Digital Sky Server Data. ArXiv Computer Science e-prints (2002)
10. Thulasiraman, K., Swamy, M.N.S.: Graphs: Theory and Algorithms. Wiley-Interscience, New York (1992)