Conflict Reduction by Rulebased Postprocessing in C3PO Concept

Lukas Rötzer, Peter Schmidt

25.4.2013

1 Introduction

The goal is to develop and apply post-processing methods and tools to *FITS*[1] characterisation files within *C3PO*[2] to reduce the problem of "conflicting values" and significantly improve data quality.

The problem of conflicting values arises from the fact, that *FITS* uses several independent tools to gather information about the inspected files. This is necessary, because different tools retrieve information of different quality on the vast amount of file types and formats. The downside of this is the fact, that the tools sometimes don't agree on the value of metadata properties. Reasons for this disagreement are various and range from different wording ("Portable Document Format" vs. "PDF EXIM") to different format version recognition, or an insufficient mime-type specification ("text/plain" vs. "text/rtf").

While some of these conflicts are detected correctly and correspond to corrupted or incorrect data in the inspected file, a bigger portion of them could be avoided, if the specific weaknesses of the used tools were known to *C3PO*. To provide this ability, rules need to be defined by human experts, that know about the abilities and flaws of the tools. They then need to be applied automatically while parsing the data provided by *FITS*.

C3PO already provides a way to apply and execute post-processing rules. They are applied either while parsing single properties (pre-processing) or after all information is available (post-processing). We will use the built in mechanisms to apply our conflict resolution techniques. We provide a language for defining conflict resolution strategies and a mechanism to generate post-processing rules from these definitions. The rules can be exchanged and adapted and while some generic or proven rules could be packed into *C3PO* by default, every user of *C3PO* gains the possibility to add his own specific set according to the circumstances of his configuration.

Digital preservation depends strongly on data integrity and authenticity. Traceability of any action taken throughout the whole process is needed not only for testing and quality assurance, but also for provability and justification. To achieve this, we keep track of all rules and changes that were applied to the metadata during conflict resolution. This gives human experts the possibility to evaluate the set of rules used and to redefine them iteratively.

2 Problem Description

The File Information Tool Set (*FITS*), by its nature, uses different third-party tools to identify and validate files. It is able to extract various metadata, normalizes and consolidates the tools output and creates an XML characterisation profile per file. As long as there is not a single tool that is able to identify all files correctly and that can read all metadata related to them, this multi-tool approach is the only feasible way to characterise the files. But the problem with this approach is, that the used tools sometimes return different values on some properties for specific files, which leads to a conflict. When profiling a large amount of files, this influences the global view on the file set, because a huge amount of files is practically unknown from that perspective.

For example, when an XHTML file is evaluated, *JHove* returns "text/html"-"Hypertext Markus Language" as mime-type-format tuple, whereas *Droid* returns "application/xhtml+xml"-"Extensible Hypertext Markup Language". Beside the fact, that one tool is obviously wrong

[1] https://code.google.com/p/fits/
[2] http://ifs.tuwien.ac.at/imp/c3po

about the metadata, this semantically small misinterpretation lets the file been reported as conflicting. From the aggregated perspective of *C3PO*, you can no longer see, whether this file was plain text or binary or what format version was used. While single files do not impose a huge problem, the accumulation of errors can strongly skew the global view.

3 Current Workflow and Implementation

When profiling a file set, the single files need to be analysed using *FITS*, that generates an XML report for every single file. These reports are parsed by *C3PO* by calling its command line interface with the *-g* option to gather information. Upon initialization, the current configuration is loaded and several threads are spawned to parse the input data in parallel. Every thread uses an adaptor to handle the input data, according to its origin.

The *FITS* adaptor uses an Apache Commons Digester to parse the XML files. Therefore certain rules [3] are defined that are triggered on the occurrence of specific XML tags to copy the data into generated Java objects. To keep track of the objects during traversal of the file, the Digester uses an object stack. The adaptor pushes a *DigesterContext* object onto that stack that holds all collected data.

To parse the data, *C3PO* implements processing rules, that can be either applied to the data of single XML tags while reading the data (pre-processing). Afterwards, when the DigesterContext is completely built from the XML file, the post-processing methods are executed, where:

- the metadata properties are set
- the collection name of the database is set
- if desired, date values can be obtained from the element name
- at last, the post-processing rules are called

Finally, the generated object is persisted in the database and the next XML file is fetched form the input.

4 Approach

Conflicts arise from multiply detected properties of varying value by different analysis tools, and therefore a conflict implies that the certain property is defined multiple times. To resolve a conflict, a rule would look out for a set of properties in the current element and remove or change some values under defined conditions.

4.1 Basic Rules Structure

A rule is defined on a single property and will be triggered if that property is reported to be conflicting.

Then the circumstances need to be tested, where a set of conditional operations comes into play. These operations allow comparison of metadata properties. In the context of conflict

[3] http://commons.apache.org/proper/commons-digester/guide/core.html#doc.Rules

resolution, a property is not just a key-value pair, but actually a triple consisting of its name, value and defining source (the tool name and its version). The operations support access to these pieces of information, testing for their existence, their values and their relation. Numerical expressions need not only to be tested for equality, but also for their relation. This allows defining rules for certain versions of a tool, not by listing all known versions, but declaring a minimum or maximum version, that this rule applies to. For string values an operator to compare it for occurrence within a list of given strings (e.g. *fileformat IN ("XHTML", "XML")*) will ease the definition of rules and improve readability.

All operators evaluate to boolean values that can be combined with logical expressions like NOT, AND, OR.

If a rules condition eventually evaluates to true, an action needs to be performed. Possible actions are the removal of one or multiple properties, that match a certain criteria (originating from a specific tool or not having a specific value), removing all but one conflicting values or changing of a value in case of naming conflicts (see below).

4.2 Applying Rules

C3PO already provides a way to apply and execute processing rules to refine data input. They need to be added to the Controller that is responsible for parsing the information about the files provided by *FITS*. The rules are applied either while parsing single properties (pre-processing) or after all information is available (post-processing). We will use the latter to apply our conflict resolution techniques. The idea is to implement a framework for generic, configurable rules. A way to specify them could be one or several XML files, which are read and parsed during initialization.

Whenever a resolution strategy got triggered and successfully removed a conflicting value, the status of the property needs to be recalculated. When looking at the structure of the XML files provided by *FITS*, this is would work for leave elements or attributes of the XML tree. But for hierarchical properties like the triple of mime-type, file-format and version, this becomes more complex. Two tools could agree on the format version value, but define different mime-types. In the case of conflicting mime-types, *C3PO* marks all three properties of the triple as conflicting, so resolving a mime-type conflict makes recalculating the other property states necessary.

To ensure the traceability of the rules, and to confirm, that no valuable knowledge is lost, each change to an element is logged:

- Which rules has been applied and in which order were they executed
- Which values were changed or removed

This information will be stored in the database, along with a copy of the representation of the rule for reference. This is necessary, because the original definition of the rule in the XML file could be changed or removed on the file system. Keeping the original values further gives the option to revert the changes applied to one or all objects by this rule. Also, applying rules to objects in the database after parsing *FITS* input would be possible, but these features are most likely out of the scope of this project.

4.3 Types of Conflicts

Different sources of conflicts need different resolution strategies. Some possibilities, on how rules could work on those types of conflicts are explained in the following sections.

4.3.1 Naming Conflicts

Often different tools, that are used in *FITS*, have just different names for each and the same file format. For example, *Exiftool* classifies some PDF files as "PDF EXIF", while *Droid*, *Jhove* along others identify them as "Portable Document Format".

For known cases, where specific tools use a wrong or unusual naming for e.g. file formats, one could implement some easy matching rules, which compare the different values and, if they are found as equal, will be set to the specified value. This is the only case where setting a value instead of deleting it is useful. Actually this case would be even interesting, if no conflict was detected, but the single tool providing the value is known to be erroneous.

4.3.2 Type Hierarchies

Sometimes one tool can make a more concrete proposition about the format or mime-type of a file than others. For example, a CSV file is recognized by *Jhove* as a simple plain text file, where *Droid* rightly identified it as "Comma Separated Values".

An approach could be, to prioritize more specific values over less detailed ones. For example a rule action could remove all occurrences of "plain text" for the file type property, if a value of "Comma Separated Values" is present. Depending on further conditions these rules can be defined globally or only on specific tools involved (e.g. only if *Droid* claims it to be a CSV file).

4.3.3 Explicit Values

If the element is in conflicted state, it could also possible to look for the existence of other properties or even specific values, that are known to exist just on a specific file type or format. From this value one could derive the right format or mime-type. These rules can be rather complex but allow a detailed level of conflict resolution.

5 Adapting C3PO

As mentioned above, *C3PO* already provides an interface and mechanism to implement and call post-processing rules. We will use this method to add our own conflict resolution rules and therefore keep the impact on the existing code as minimal as possible. Still, there are some modifications that need to be applied.

In the c3po-cmd module, we will likely have to add additional functionality to control rule execution from the command line.

The core module will be slightly extended. The Controller gains the ability to check and parse the post-processing rules and check their syntactic validity. This will be done in the "checkConfiguration()" method. Further the rules will have to be loaded into the rule set by extending the existing "getRules()" method.

The data model needs to be extended by adding a backlog of changes to the elements saved in the database. The backlog combines a reference to the applied rules with the original values of changed properties. This feature assures the traceability of changes applied to the data.

Additionally to these adoptions of the existing code, the newly added code *C3PO* is reduced to a utility for parsing the rules from their textual definition and a mechanism to persist the rules in the database. Persisting the rules is necessary to assure traceability of changes, as the textual definition of the rules can be altered easily.