AONS System Documentation

Revision 169 2006-09-29

Joseph Curtis, The Australian National University

http://www.apsr.edu.au

September 2006

Contents

Introduction

AONS (Automated Obsolescence Notification System) is a system to analyse digital repositories and determine whether any digital objects contained therein may be in danger of becoming obsolescent. It uses perservation information about file formats and the software which supports these formats to determine if the formats used by digital objects are in danger. The information used is gleaned from various registries (PRONOM[1] and LCSDF[2]) and stored in a MySQL[3] database. AONS will then periodically check the contents of the repository for formats in danger of becoming obsolescent. When the repository is found to contain such formats a notification report will be sent to the repository manager responsible. AONS supports multiple repositories with fully independent scheduling. Currently AONS only supports the DSpace[4] digital repository; however, the interface between the repository and AONS is simple and well defined and it shouldn't be too hard to extend to other repositories in the future.

Towards the end of development there was an attempt to create an AONS interface with the Fedora/Fez[5] digital repository. The attempt was largely successful and a quick proof of concept was developed which would produce a format summary compatible with AONS. Fedora/Fez is structured in a way different to DSpace. Fedora by itself doesn't specifically define things such as communities and collections, it is very flexible. This highlights the need for AONS to have a repository agnostic-format summary format if it is to become more widely used in the digital repository community.

The context of AONS is shown in Illustration 1

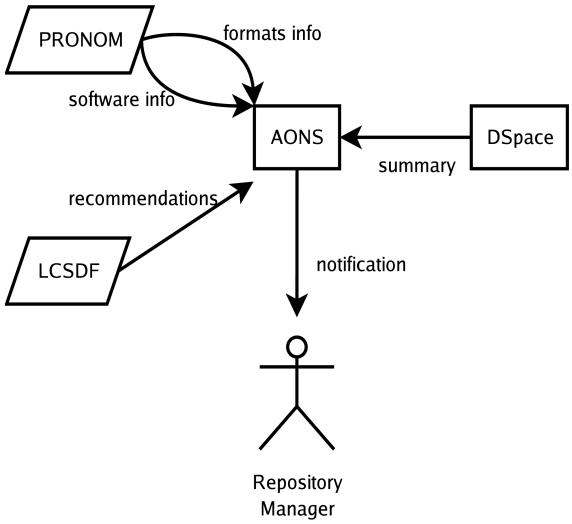

Illustration 1: AONS Context

1 http://www.nationalarchives.gov.uk/pronom/
2 http://www.digitalpreservation.gov/formats/
3 http://www.mysql.com/
4 http://dspace.org/
5 http://dev-fez.library.uq.edu.au/

Architecture

AONS is comprised of three main parts (encapsulated in Java packages):

- Repository interface (`aons.repository`package):

  Scans a DSpace repository and provides an XML summary of all the different formats in the repository as determined by DROID (Digital Record Object Identification)[6].

- AONS Core (`aons`package):

  Uses the format and software information provided by the Registries interface to determine the obsolescence of the formats contained in a digital repository provided in the repository summary.

- Registries interface (`aons.registry`package):

  Retrieves raw formats, software and recommendations information from the associated registries (PRONOM, LCSDF) and processes them such that they can be added to the MySQL registry database for use by AONS.

These three parts interact as shown in the high-level system architecture, Illustration 2.

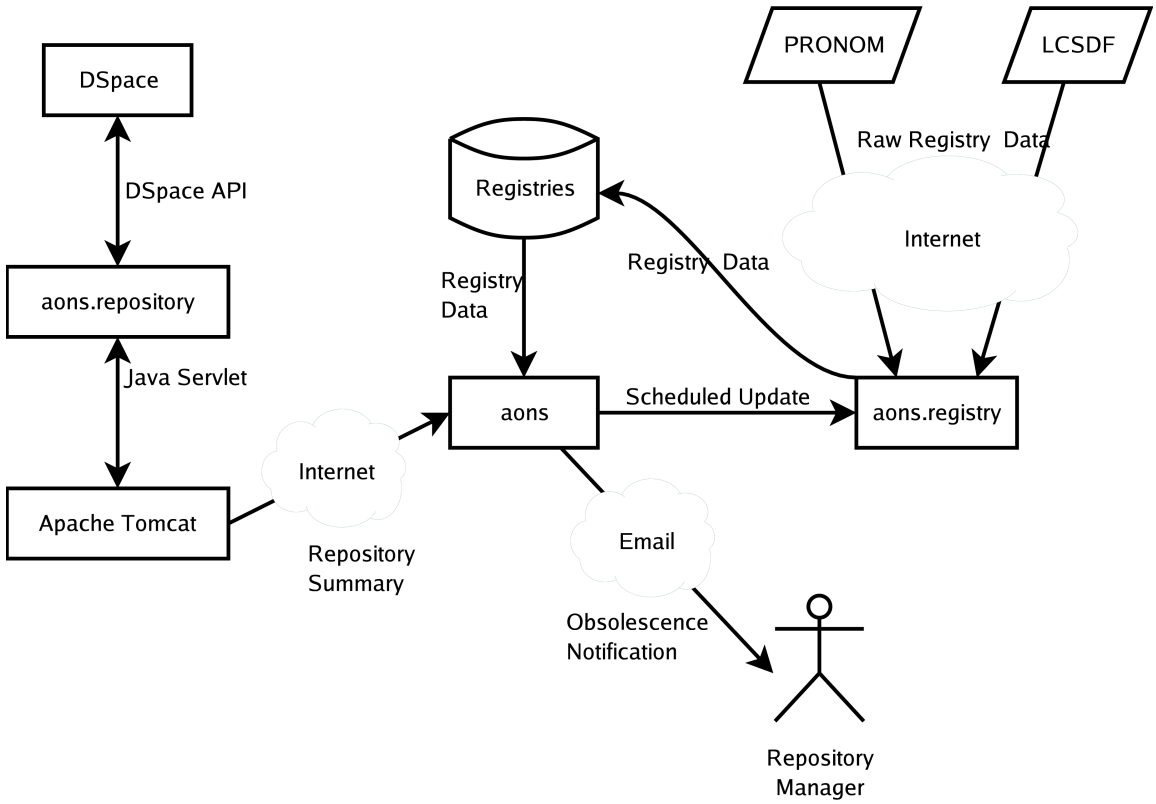

Illustration 2: AONS High-Level Architecture

6 http://droid.sourceforge.net/

Repository Interface Overview

The repository interface is implemented as a Java Servlet on top of DSpace. DSpace itself is comprised of many Servlets and so this interface integrates very easily with DSpace. The repository interface communicates with DSpace via DSpace's API. Fortunately the DSpace API is complete and well documented so accessing the required data in DSpace for AONS to process the files wasn't too hard. One small modification was made to one of the DSpace source files to allow direct access to a DSpace bitstream (the raw bytes which make up a file in DSpace). This was necessary because by default it is not possible to obtain the raw location (as a filesystem path) of a DSpace bitstream, only an `InputStream` from which the data can be accessed indirectly. This approach was used at first but was found to be far too slow to be practical. The modified source file is included with AONS and no modification to existing DSpace installations is required.

The interface is accessed via a URL through the Java Servlet container in which DSpace is running (most likely Apache Tomcat[7]). A small modification to the DSpace Tomcat configuration is required, this is addressed in further detail in the installation guide.

AONS Core Overview

The AONS Core is the part which performs the obsolescence detection functions. It also contains the main function which is executed when AONS begins and parses the command line arguments. This part contains the scheduling logic which determines when to run the obsolescence detector and also when to update the registries from their web sources. The AONS Core takes a format summary from the DSpace repository, iterates through every format therein and compares it to format,

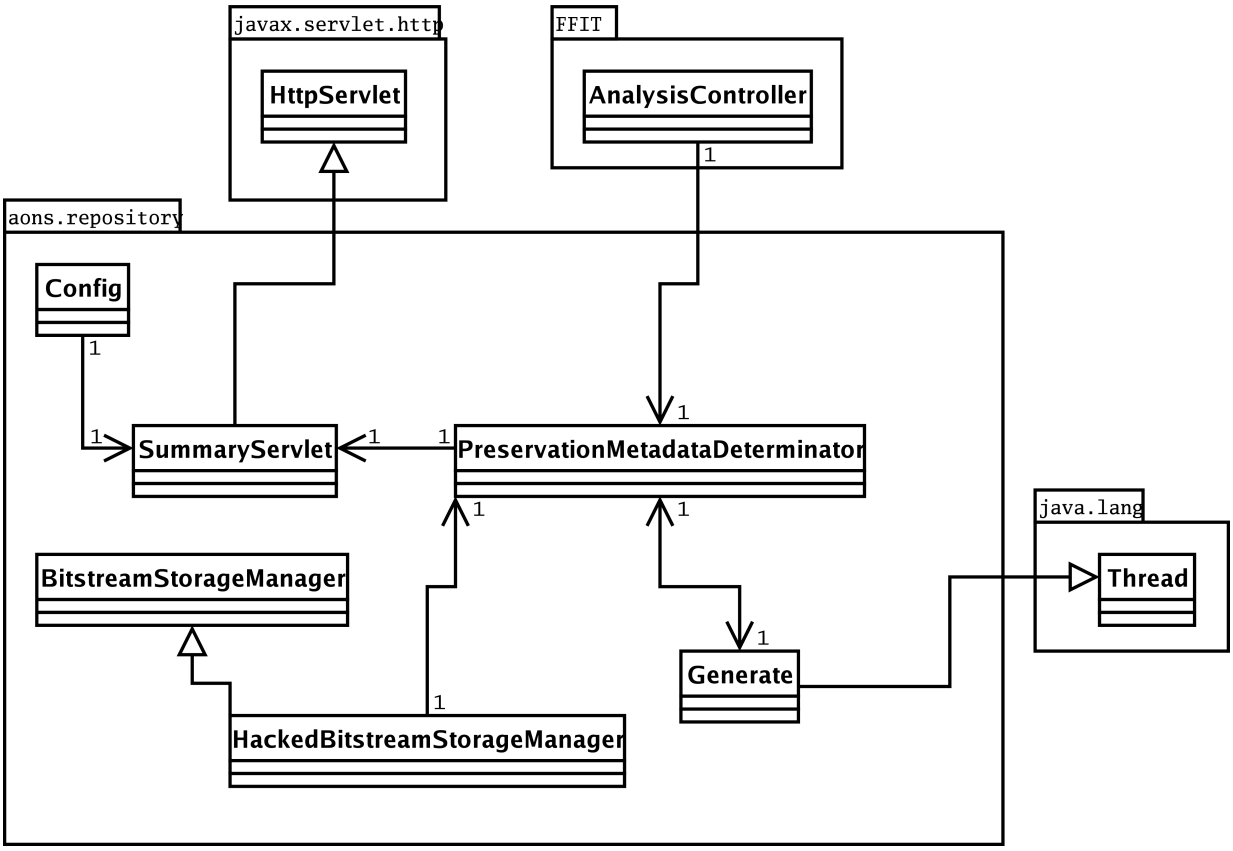

Illustration 3: aons.repository package class diagram

7 http://tomcat.apache.org/

software and recommendation data in the registries. It then tries to determine if that particular format is in danger of becoming obsolescent. If obsolescent formats are discovered during this process then the AONS core has the responsibility of notifying the repository manager of the repository currently being scanned.

Registry Interface Overview

The registry interface has the job of retrieving the raw registry data from the two web sources (PRONOM and LCSDF) and processing it such that it can be used by AONS. The registry interface first obtains the raw data from a web source. In the case of PRONOM, this data is already formatted in XML and so fairly minimal processing is required to use it. In the case of LCSDF the data is stored by the Library of Congress as static HTML pages. This requires considerable processing to extract this data and format it logically in XML. After this data is extracted from the web sources and formatted in XML, it is then processed further to map format data obtained from PRONOM to the corresponding data obtained from LCSDF. Once the data has been processed it is converted into SQL which is then inserted into the AONS registries MySQL database. An explanation of the database structure can be found later in Appendix 1.

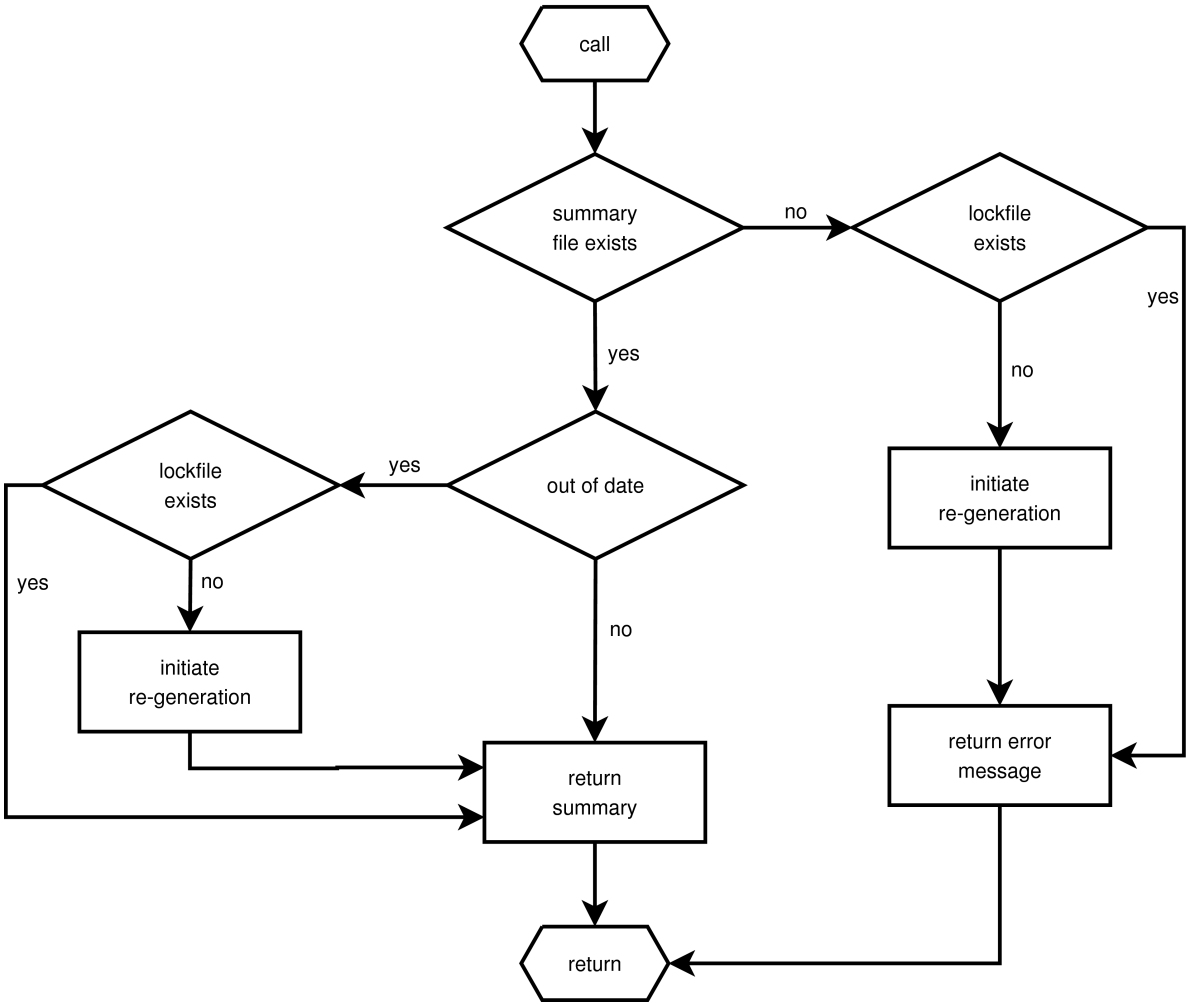

Illustration 4: `getSummary()` *method flowchart*

Repository Interface

Execution

The repository interface is special amongst the different parts of AONS since it is to be run separately from the main AONS application. If DSpace stored sufficient preservation metadata with its digital objects then this part of AONS would be rendered obsolete. DSpace in fact stores practically no preservation metadata apart from a simple MIME-type. This dearth of metadata was not sufficient for AONS to make sensible decisions about format obsolescence since it couldn't accurately determine exactly what formats an object had. Thus, this interface exists which provides an easy method for AONS to access the preservation metadata for objects in a DSpace repository. It does this by scanning DSpace bitstreams with DROID and providing a summary (in XML format) at a URL for easy access. For this to be possible the interface must co-exist on the same machine as DSpace so it can access DSpace via its API.

Below (Illustration 3) is a class diagram for the `aons.repository` Java package. For an in-depth explanation of how each class works internally, please see the source code or the JavaDocs (in Appendix 1).

The repository interface is invoked through a Java servlet contained in the same servlet container as DSpace. When the servlet is accessed the `doGet()` method is called. This method parses the configuration file and creates a `PreservationMetadataDeterminator` object to which it passes the configuration object. The `PreservationMetadataDeterminator` object is then invoked by means of the `getSummary()` method. The `doGet()` method sends whatever the `getSummary()` method returns back to whomever accessed the servlet by writing it to the servlet response output stream. The `doGet()` method can return either a format summary, a message informing the requester that a summary is not available and is being generated, or an error message if the metadata determination process fails.

The `PreservationMetadataDeterminator.getSummary()` method acts in a slightly peculiar manner. The behaviour of this method is described by the flowchart Illustration 4. A new thread must be spawned so the summary can be re-generated without tying up the whole servlet. Also an informative message needs to be sent to the requester telling them that the summary re-generation is under way and to check back in a few hours. The `Generate` class provides this thread. When the format summary needs to be re-generated a new `Generate` class is instantiated and passed the current `PreservationMetadataDeterminator` object. When the `Generate` thread is then run, it calls its `PreservationMetadataDeterminator.determinate()` method which then runs as usual. The `Generate` object creates a lock file whilst the format summary generation is under way and removes it once the generation is complete. This is how `getSummary()` determines if a summary is currently being generated or not. Once the generation is complete and the lock file removed then any subsequent calls to `getSummary()` will be able to return the actual summary and not an in progress message.

Transformations

The `PreservationMetadataDeterminator` class generates two temporary XML documents, the DROID results and the DROID mapping. These documents are transformed as in Illustration 5.

Data Transformation Diagram Key used by all data transformation diagrams:

- **Boxes:** XML data. 

The text inside the box indicates which DTD this data conforms to.

- **Rounded boxes:** Transformation by an XSLT stylesheet.
- **Slanted boxes:** Transformation by a Java class.
- **Angled boxes:** SQL data.
- **Vertical cylinder:** MySQL database

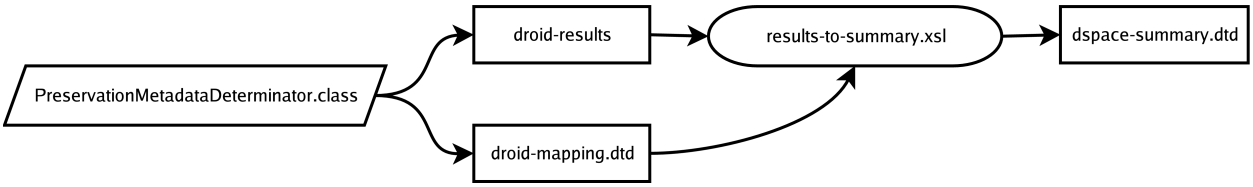

Illustration 5: aons.repository data transformations

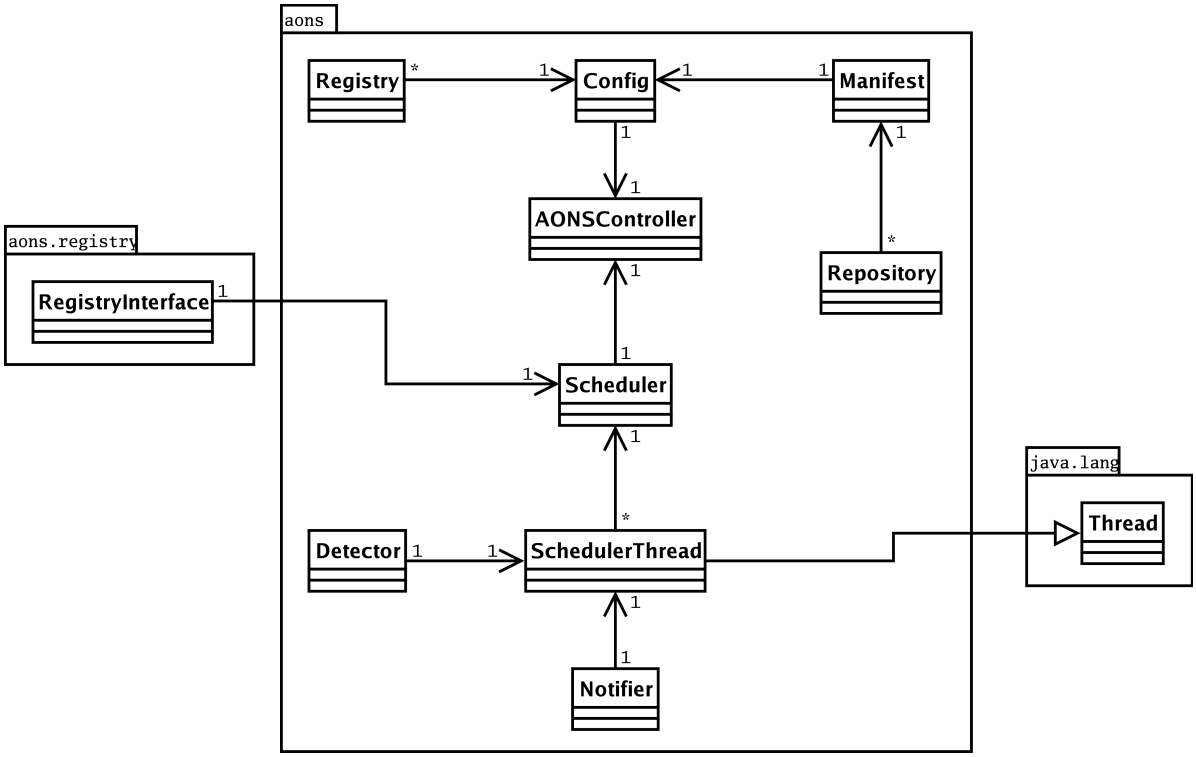

Illustration 6: aons package class diagram

AONS Core

Execution

The AONS Core package contains the main class called `AONSController` This is called first when AONS is invoked.

When AONS is invoked the command line arguments are parsed. The configuration file is then parsed and the `config` object populated with the information therein.

The `Scheduler` object is then created and started. The operation of the `Scheduler` is described in detail in the JavaDoc comments in the `aons/Scheduler.java` file. Essentially the `Scheduler` spawns a new `SchedulerThread` for each repository in the configuration file and then waits for the next scheduled registry update. When a registry update is scheduled it wakes up, updates the registry and then sleeps again until the next scheduled update, indefinitely. The `SchedulerThread` objects are the ones which actually schedule and initialise the repository A `SchedulerThread` object sleeps until the repository obsolescence detection is due, then wakes up, performs the obsolescence detection, sends the notification to the repository manager (if necessary) then goes back to sleep.

The actual obsolescence detection happens in the `Detector` class. It is invoked through the `detect()` method after being initialised with the `Config` object and the `Repository` object. belonging to the particular repository it is servicing. The `detect()` method obtains the format summary from the repository and creates a new XML document to store the obsolescence report in. It then calls the `detectFormat()` method and passes the format summary to it. The `detectFormat()` method is where the detection takes place. It was originally planned to have a `detect*()` method for each of the `Format`, `Software` and `Recommendation` registries; however, it was deemed unnecessary. The whole process of using a monolithic function like this for detecting different kinds of obsolescence isn't easily extensible and it is suggested that in a future version this be changed to a better method (see Appendix 2 for more details).

The `detect()` method returns an XML document containing the obsolescence information. This XML document is then transformed with an XSLT stylesheet to produce the XHTML notification report. A `Notification` object is created and the repository it is notifying for is passed to the constructor. The `notify()` method is then called and passed the notification report. The `notify()` method simply sends and email to the repository manager, it is possible to extend the `Notifier` class and override the `notify()` method for future implementation of alternate notifying methods.

Transformations

Illustration 7 shows the data transformations which take place within the AONS core. The diagram uses the same key as the previous transformation diagram. The summary file from the repository is used by the `Detector` class along with the information in the database to produce the obsolescence information XML document. This document is then transformed by `results-to-summary.xsl` into the AONS notification report.

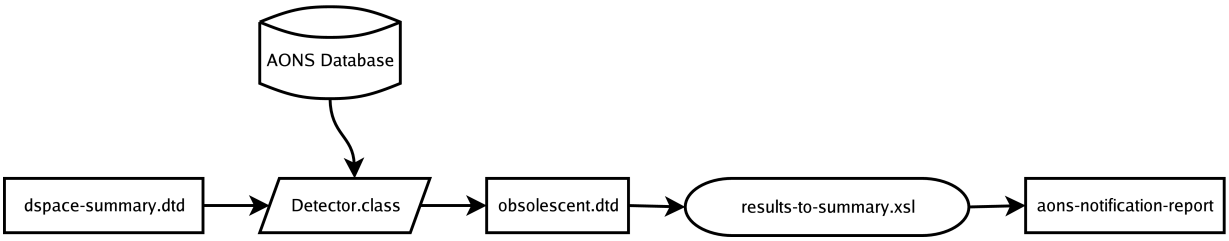

Illustration 7: aons core data transformations

Registry Interface

Execution

The registry interface is responsible for retrieving the format, software and recommendation information from the PRONOM and LCSDF registries. It accesses this data via HTTP from their respective sources and munges it into a nice XML form which can then be converted into SQL for injection into the MySQL database. An instance of the `RegistryInterface` class is invoked by the Scheduler when it needs to update the registry database. Updating involves retrieving the raw format, software and recommendation data from each of the registries. To increase performance this data is retrieved simultaneously using three different threads.

When the `RegistryInterface.update()` method is called an inline class called `Search` is declared which extends the Java class `Thread` This class takes a `PRONOMQuery` or an `LCQuery` object as a parameter to its constructor and when run populates a result `Document` with the raw XML data retrieved from the respective registry. The `update()` method creates three instances of this inline class which it then executes simultaneously. A conceptual diagram of the concurrently executing threads is contained in Illustration 9. Each horizontal section of this diagram represents a different thread of execution with the arrows indicating a thread spawning or joining and the boxes indicating tasks being performed.

After the raw format, software and recommendation data has been downloaded from its web sources a mapping is created between the format data retrieved from PRONOM and that retrieved from LCSDF. This is done so that the various recommendations contained in the LCSDF data can be associated with the format and software data from PRONOM. This is not such an easy task since they don't share any well defined common attributes which can be used for this mapping. An attempt is made to associate formats by name; however, this can be problematic as there are no strict conventions for naming formats (a format may be named “JFIF” in PRONOM and “JPEG File Interchange Format” in LCSDF). The class `PRONOMLCSDFMapper` is used to create the mapping. It uses a fairly simple method, it checks every PRONOM format against every LCSDF format and

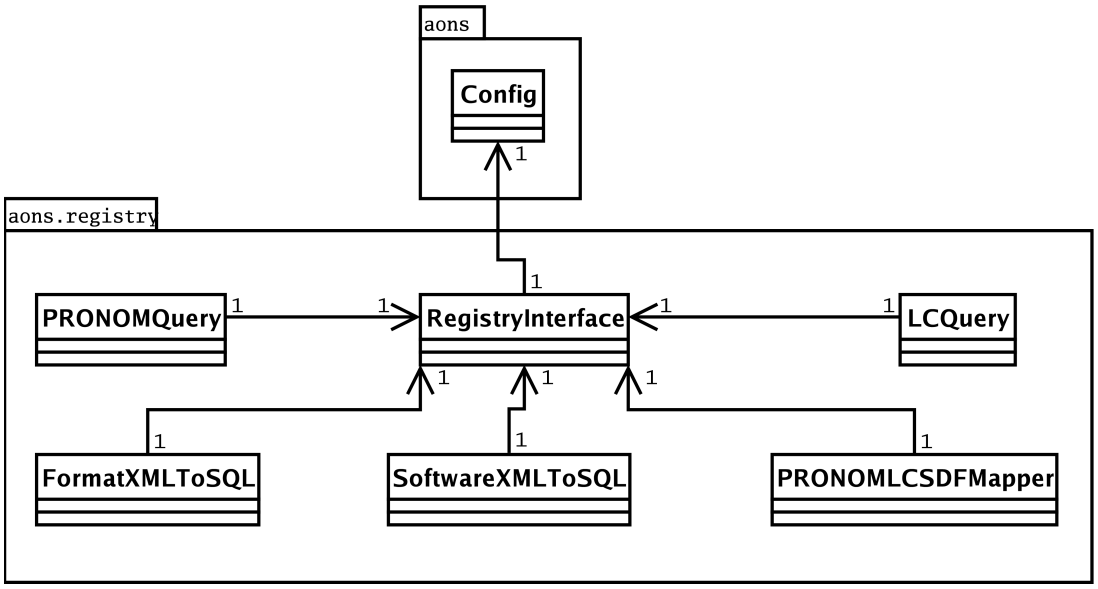

Illustration 8: `aons.registry` package class diagram

runs a comparison on the names of the formats. If the formats are deemed to be the same then an entry is created in the XML mapping file. This mapping file is later used in an XSLT transformation. A better comparison algorithm would certainly improve obsolescence detection results; however, this is out of the scope of this prototype project.

Transformations

After the mapping is complete a series of XSLT transformations is performed on the raw XML data from the registries. A diagram describing this series of transformations is depicted in Illustration 10. This diagram uses the same key as the previous two data transformation diagrams.

The raw format data is first transformed with `format-reg-simplify.xsl` to simplify the data and strip out all the unnecessary elements. This simplification is then processed with the `FormatXMLToSQL` class to convert the XML data to SQL which is then inserted into the `format_registry` table of the MySQL database.

A very similar process is used for the raw software data except is is processed using `software-reg-simplify.xsl` and the `SoftwareXMLToSQL` class.

The transformation process for the raw XML data from LCSDF is different due to the need to create a mapping between the LCSDF formats and the PRONOM formats. Firstly the `PRONOMLCSDFMapper` class uses the raw PRONOM format data and the raw LCSDF data to create an XML mapping file. Those mappings are then fed into the `recommendation-gen.xsl` stylesheet along with the raw LCSDF data which produces an XML document containing the correctly mapped recommendations. These recommendations are then transformed with the `recommendations-to-sql.xsl` stylesheet which converts them into SQL which is then inserted into the MySQL database. This transformation process uses a stylesheet to convert the XML to SQL instead of a Java class (like the formats and software transformation processes). This is because the formats and software transformation processes were completed first and it was not realised until after this that it would be more appropriate to perform this transformation using a stylesheet instead of a Java class. The two XML to SQL transformation classes were not changed as they worked and it was not a priority.

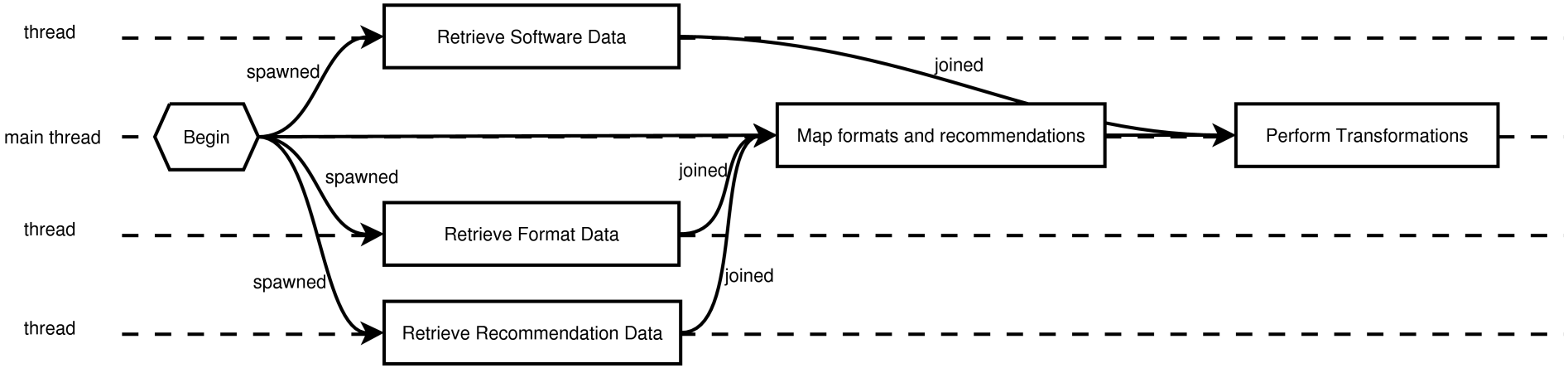

Illustration 9: Conceptual diagram of update() method concurrently executing threads

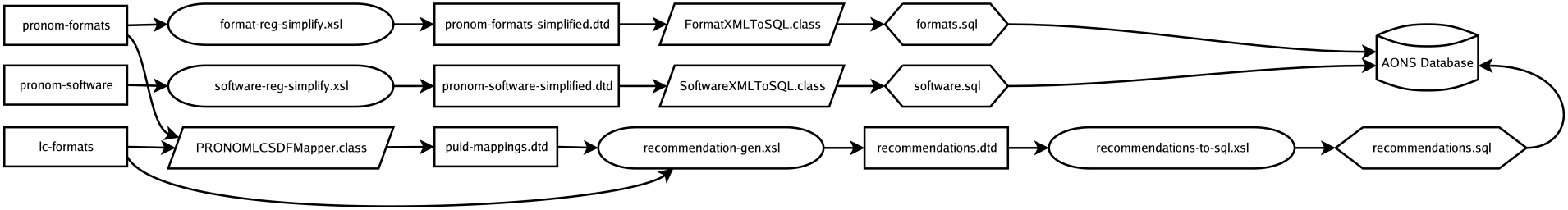

Illustration 10: aons.registry data transformations

Appendix 1: Database Structure

The MySQL database which contains the Format, Software and Recommendation data for AONS is comprised of three tables:

- format_registry
- software_registry
- recommendation_registry

These tables are governed by the relationships expressed in Illustration 11.

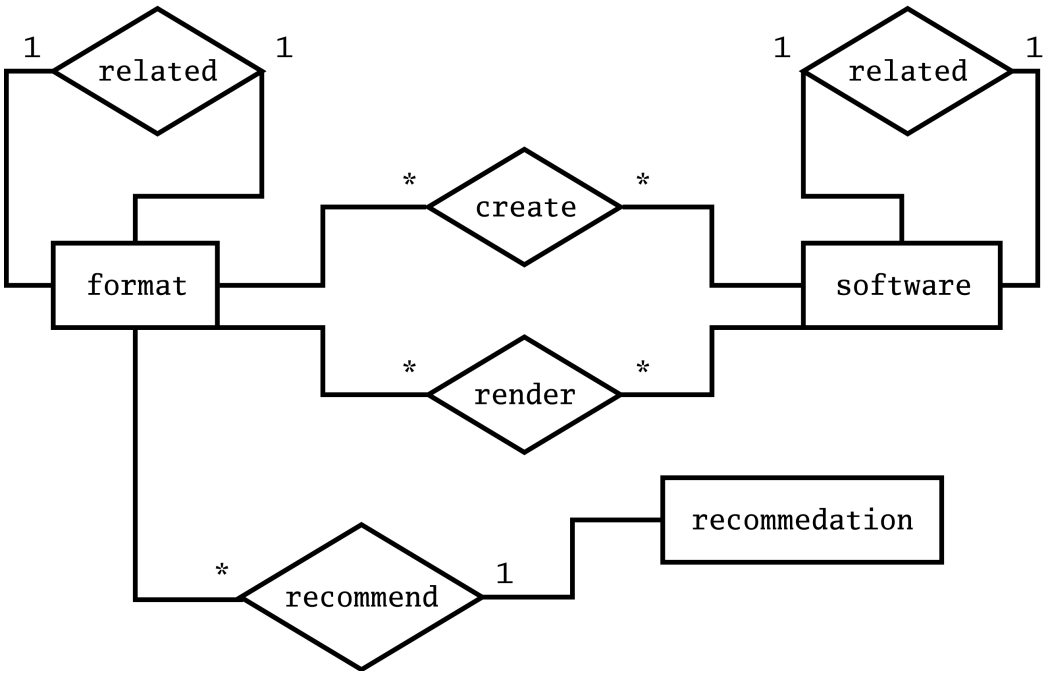

Illustration 11: AONS database Entity-Relationship diagram

The relationships described above are not explicitly defined in the database table structure but are enforced in the code which interfaces with the database.

These tables have the following structure:

format_registry

Field	*Type*	*Description*
id	INT	Unique identifier for formats in this database
name	VARCHAR (64)	Format name
puid	VARCHAR(16)	PRONOM Unique Identifier
mime	VARCHAR(32)	MIME type
version	VARCHAR(16)	Format version

* Primary Key

Field	*Type*	*Description*
disclosure	VARCHAR(16)	Disclosure of the specifications of this format, full, none, etc...
sub	INT	This format is a subsequent version of the format contained in this field
prev	INT	This format is a previous version of the format contained in this field

software_registry

Field	*Type*	*Description*
id*	INT	Unique identifier for software in this database
name	VARCHAR(64)	Software name
version	VARCHAR(64)	The software version
formats_create	VARCHAR(2048)	Comma-separated list of formats which this software can create
formats_render	VARCHAR(2048)	Comma-separated list of formats which this software can render
released	DATE	The date on which this software was released
withdrawn	DATE	The date on which this software was withdrawn (i.e. not supported any more)
sub	INT	This format is a subsequent version of the format contained in this field
prev	INT	This format is a previous version of the format contained in this field

recommendation_registry

Field	*Type*	*Description*
id*	INT	Unique identifier for this recommendation in this database
puid	VARCHAR(16)	PRONOM Unique Identifier
lcid	VARCHAR(16)	The Library of Congress (LCSDF) ID
format_name	VARCHAR(1024)	The name of the format (in the format_registry table) which this recommendation references
recommendation	VARCHAR(4096)	A textual representation of the recommendation
authority	VARCHAR(128)	The authority behind this recommendation
authority_url	VARCHAR(256)	URL for the recommendation authority

* Primary Key

Field	*Type*	*Description*
date	DATE	Date of the recommendation

Appendix 2: Future Improvements

Some suggested future improvements for AONS. In no particular order:

1. Make AONS understand the "check back in a few hours" message the repository interface provides if a summary is in generation. AONS could take note of the time and re-schedule then.
2. Optimise the memory usage and time in the preservation metadata determination process.
3. Implement obsolescence detection checks in a nicer and more extensible manner. Perhaps some object oriented approach using different classes for different obsolescence checks.
4. Improve the algorithm for mapping between PRONOM and LCSDF formats. Ideally they would all use a common identifier.
5. Automatically generate software version relationships in the database. This is really to fix a (easily fixable on their end) deficiency in PRONOM so maybe not so important.

Appendix 3: JavaDocs

Embedded in code.