

Overview

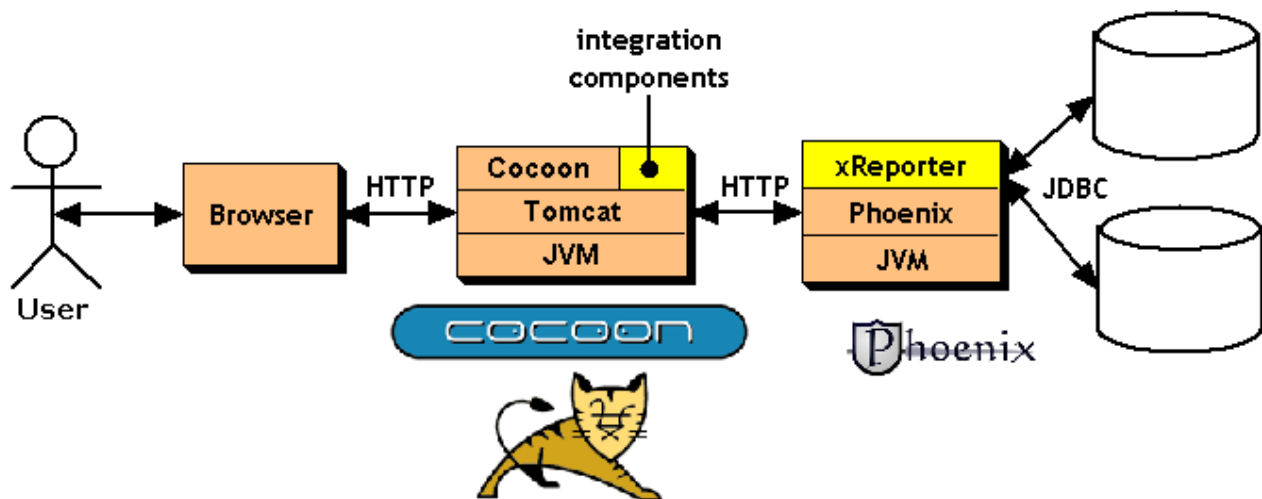
This document provides a quick overview of the main facts about xReporter.

Purpose

The goal of xReporter is web-based, read-only consultation of databases. The user can choose from a set of predefined reports. Reports can be created without any programming (though you'll need SQL knowledge). An xReporter report is not static: the user can be asked for a number of parameters, and can tweak the output using filtering and sorting, as well as change the column layout.

Technical Architecture

xReporter consists of two main parts: the xReporter server (running inside Apache Avalon Phoenix) and the publishing layer (based on [Apache Cocoon](http://cocoon.apache.org/)¹). The xReporter server contains all the core functionality but always produces XML, so it is up to the publishing layer to generate the HTML interface that the user sees in his browser.



The xReporter server contains the core functionality: it interprets the report definitions, manages the database connections, handles the flow, and so on. The publishing layer sits between the user (web browser) and the xReporter server. It passes incoming requests on to the xReporter server and styles its XML response.

Splitting the publishing layer from the reporting server has a number of advantages:

- the publishing layer can be rather easily replaced by another technology (it consists mainly of some stylesheets and does not contain any application logic).
- the reporting server has a well-defined, HTTP/XML-based interface. This interface can also be used from other applications. It would, for example, be possible to use a scripting language to automate the generation of some reports and send them out using email. In fact, this is the way we do automated testing of the reporting server. The HTTP/XML interface is fully [documented](#)².

1. <http://cocoon.apache.org/>

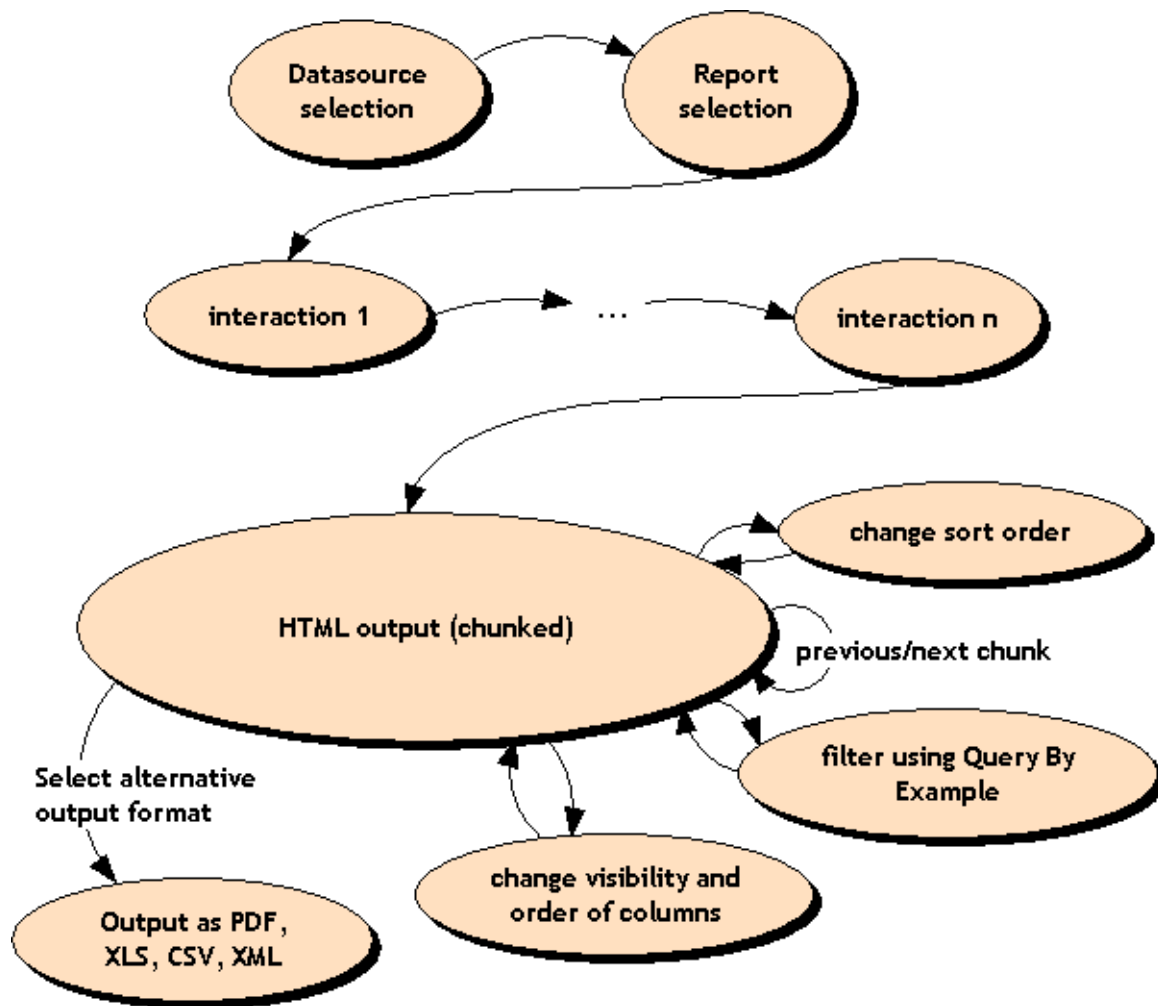
- the publishing layer can be scaled independently from the reporting server. E.g. multiple Cocoon instances can talk to the same reporting server. This can be important when the publishing is a heavy process (e.g. generating large PDF reports, charts, ...).
- the reporting server itself has a quick deploy/restart cycle, which is very useful during development. The publishing layer (Cocoon) on the other hand is mainly configured by stylesheets, which are dynamically reloaded when changed.

xReporter can connect with any JDBC-datasource. The full power of SQL is available in the reports.

Functionality from the user's point of view

Let's have a look at how xReporter works from an end-user point of view (see also the diagram below):

- first the user selects a data source. Only data sources for which the user has access rights are shown.
- the user selects a report from the report catalog. The report catalog only contains those reports for which the user has access rights, and which are compatible with the selected data source. When the user selects a report, xReporter will make an instance of this report.
- (optional) during one or more interaction steps, the user can be asked for parameters.
- (optional and invisible for the user) some SQL-statements can be executed to create temporary database tables. In case of more complex needs, custom Java classes can be executed. xReporter will automatically cleanup the temporary tables when the report instance expires.
- finally, the output of the report (which results from some select-query) is shown to the user. The output is by default chunked in sets of 50 records.
- the user can further customise the report by changing the sort order, drop columns or change their location, and filter the output using a "Query By Example" grid.
- if desired the user can download the report in a variety of formats such as PDF, Excel, CSV and XML.



Data Sources

The first thing to do is define the data sources, i.e. the different databases on which you'd like to run reports. For optimal speed, database connection pooling is supported. xReporter has its own connection pooling implementation, which has the unique feature that all connection pools are managed by one thread. As a consequence, xReporter can easily handle hundreds of data sources, without requiring hundreds of threads to manage these pools.

xReporter allows to assign types to data sources. These types identify the kind of data that the data source contains (thus its schema, or what tables it contains). This makes it possible to check the compatibility between a data source and a report: a report requires a certain data source type, and can hence only be executed against data sources implementing that type.

Flexible access control to data sources is also available: you can limit the data sources to which a user has access.

Report definitions

Creating a new report definition is as simple as editing a single XML file. In the following sections we describe some of the features of reports.

Flow steps: interaction and temporary tables

A report can contain (optionally) a number of interaction steps during which parameters can be asked from the user. Each parameter has an associated data type. A data type defines what type of data it is (a string, number, date, ...), the prompt to show to the user, a default value, a selection list, validation rules, etc. All this is configured with some XML tags, and xReporter will take care of asking the user these parameters and validating them.

In between each interaction step, a number of SQL-statements can be executed. These statements can create temporary tables, which is useful for more complex queries that cannot be expressed in a single select statement. If the temporary table cannot be created by a single SQL statement, there's also the possibility to plug in your own Java-class to create them. xReporter will automatically manage the naming and clean up of these temporary tables.

xReporter can store the values of parameters on per-user basis, so that if the user re-runs the same report at a later time, those parameters will be automatically pre-filled. This feature also works across reports: if you entered a parameter for one report and you need to supply the same parameter in another report, you can have this filled in automatically.

Report output

The actual report output is retrieved by a single select query, in which references can be made to the earlier user-provided parameters or temporary table names.

The report output can be a flat table, or it can be a report with groups and summaries (this only from xReporter versions > 1.0). In the last case, you can calculate things like the sum or average of a range of values. If there's any interest, we might also implement cross-tabulated output in future versions.

In the report definition you need to specify what columns you'd like to pull out of the resultset from the query, and for each column you define its data type (just like with the parameters in the interaction steps). In the data type you can e.g. specify the formatting pattern to be used when displaying the values. Data type definitions can be put in a data type catalog and be reused across multiple reports.

Internationalisation

All labels used in report definitions, such as the title of the report, the prompt of an input parameter or the title above a column are all defined in language-dependent resource bundles. Likewise, the formatting of dates and numbers is done according to the locale of the user.

Access control

Access to reports can be restricted based on user roles.

Creating report definitions

Currently no GUI is available for editing these report definitions, though the report format is [well described](#)³ and an XML Schema is available to help editing the report definitions using popular XML editors.

3. daisy:86-cd (Reports)

Various features

Stateless web application

xReporter follows the rules of the game for its web front-end: it is fully stateless, and does not require sessions to keep track of user parameters. This is because each time the user runs a report, a so-called "report instance" is created which has its own unique URL. This approach has some important advantages:

- It is possible to run multiple reports concurrently in different browser windows.
- Someone can create a report instance, copy the URL they see in their browser, and mail this to another user. Upon receiving the message, that user can then simply click the URL and see the report output (provided that the report instance has not yet expired, and that the other user has the necessary access rights on the data source and report).
- Back button behaviour as expected.

This style of designing web applications is also known as "[ReST⁴](#)" (Representational State Transfer).

Saving report instances

Once a user has created a report by entering any required parameters (and possibly changing other options like sort order), the report can be saved so that it can be quickly re-executed later on. Note that saving a report doesn't save its resultset data, only its parameters.

Customisable look

If you don't like the default look of xReporter, don't worry, it is fully customisable without touching the xReporter code. All rendering is done by XSLT-stylesheets. Maintaining your custom copy of these stylesheets can be a lot of work though, therefore we have structured the stylesheets in such way that it's easy to redefine the navigation and metadata borders, while reusing the core parts.

xReporter's report definitions contain no styling information at all, since this would depend on the output format (HTML, PDF, Excel, ...). To customise the display of reports without adjusting XSLT stylesheets, a mechanism using parameterisation files is available. For example, for HTML, you can specify column widths and column colors in such a file.

The structure of the presentation layer is [well documented⁵](#) so that it can be easily integrated in other Cocoon projects.

Logging

Apart from error logging, a number of important application events are [logged⁶](#) in a structured format, such as the creation of a new report or the retrieval of its output. This data can be used for usage analysis and billing.

4. <http://rest.blueoxen.net/>

5. [daisy:97-cd](#) (Cocoon Integration)

6. [daisy:89-cd](#) (Logging)

Platform independent

Being based on Java, xReporter is fully hardware and operating system independent and runs on both Unix variants and Windows. This makes it a safe bet for the future and allows deployment on free/open platforms like GNU/Linux.

HTTP data source

Usually the content of a report results from executing a query on a database. xReporter additionally offers the possibility to retrieve the content from a HTTP URL. This allows to integrate existing systems within xReporter. This requires that the content retrieved from the HTTP URL is well-formed XML.

Expression language

xReporter has a simple but powerful expression language used in various places throughout the program, such as for calculating default parameter values, implementing custom validation rules, and calculating summaries on reports with grouping. The expressions are similar to those in popular spreadsheets. The expression language can be extended with your own functions (implemented in Java). If you write functions that are generally applicable, please consider contributing them back to the xReporter project.

User properties

With each user, a set of custom properties can be associated. These properties can be fetched from expressions, and thus, for example, embedded as parameter in a SQL statement. The properties are also put in heading of the XML coming out of xReporter, and can thus be used in stylesheets to customise the display of the site.

Open Source

xReporter is available under a liberal, Apache/BSD style [license](#)⁷, making it possible to use, extend and redistribute xReporter, *even within a commercial context*.

Through the [xReporter mailing list](#)⁸, you have direct contact with other xReporter developers and users.

Commercial support and customisation contracts can be obtained from [Outerthought](#)⁹, the Open Source Java and XML Competence Center. Reduced fees are available for development which can be fed back into the open source project.

7. [daisy:65-cd \(License\)](#)

8. <http://lists.cocoonddev.org/mailman/listinfo/xreporter>

9. <http://outerthought.org/>