

Open Source & XBRL: the Arelle® Project

Authors: Herm Fischer, Mark V Systems Limited and Diane Mueller, XBRLSpy Research Inc.

The Problem

Currently, there are a number of proprietary XBRL processors on the market each with varying degrees of conformity to the XBRL 2.1 specification and differing levels of implementation, integration, and support for XBRL extension modules, Filer Manual tests, and test suites. A quick look at <http://edgardashboard.xbrlcloud.com/edgar-index.html> on XBRLCloud, will give insight into usage and results with the wide variety of tools available for creating XBRL filings. Each has a slightly different approach to document creation, varying levels of validation leading to a wide range of errors and warnings, and different transparency in its implementation and degree of conformance. However, this should not be seen as a US centric problem, as the same issues arise in each jurisdiction where XBRL has been adopted. XBRL software developers struggle through the XBRL 2.1 specification, the companion dimensions specification, the Edgar and Global Filer Manuals, and make valiant attempts at implementing conformant processors. It is hard to know which may be more successful than others, because the users don't usually have integrated access to the test suites to check conformance, or the various extension modules such as versioning, rendering and formula. This situation sparks frequent XBRL online and blog discussion of variation in levels of conformity in XBRL instances and tools that consume and validate them.

We believe there is a need to supply software developers with a cohesive, easy-to-use programming model for building XBRL-compliant applications and curb the proliferation of non-conformant XBRL processors. Developers new to XBRL are forced to learn and interpret the entire XBRL 2.1 specification just to build a simple application. Advanced XBRL developers are forced to write tedious plumbing code; and tools authors are limited in what they can do to simplify the experience due to the underlying complexity.

XII itself is dependent on member vendors to implement and test changes to the specification or new modules, constraining XII to vendor member's resource availability and interest levels in specific aspects of the new modules – which are not always in line with the market needs and requirements to drive adoption of XBRL.

Our Approach

In order to advance the adoption and resolve the interoperability and conformance issues, we have launched an open source software effort to facilitate the ongoing development of XBRL-related technology resources that will be freely available to the public obtain, use, redistribute, and modify.

The XML standard, the underpinning technology of XBRL, is an open source project, with a strong, stable open source community supporting its care and maintenance. "Open source" is defined as: freely available technology licensed under terms compatible with Version 1.9 (or later) of the Open Source Definition, as established by the Open Source Initiative (see <http://www.opensource.org/>).

Using Python, an open source dynamic language known for high productivity, we have developed a highly portable software library for parsing and validating XBRL documents, and released it under the Apache 2 license. This will remove significant barriers to entry into the XBRL market; enable the development of reliable, stable and conformant financial software applications that would promote the adoption of XBRL technology across the globe.

The Arelle Project

ArELLE is a project to provide an easy to use open source facility for XBRL. The intent began to meet needs that are not commercially viable, such as to support under-development extension modules and test suite facilities, in a compact framework, and to support academic training and projects.

Support for XBRL versioning was an initial goal, to provide both a validation tool for versioning reports and a production tool to generate the basics of a versioning report that can be inferred by diffing two DTSes.

As the project evolved, Edgar and Global Filer Manual validation, Base Specification, Dimensions, Generic linkbase, Formula, and Eurofiling rendering linkbase, were added. Additional features were added to allow formula experimentation with existing real filings, by an RSS Watch facility.

ArELLE fully integrates test cases with the object models for XBRL instances and DTSes. This allows continual verification of tool performance as it is extended and adapted by its users.

Users can explore the functionality and features from either an interactive GUI or command line interface, and can develop their own controller interfaces as needed.

ArELLE is supported on the website <http://arelle.org>

Background

ArELLE's origin was a need to provide tooling focusing on early support for technical development of standards extension modules. These stages require a way to error check and validate prototype test suites facilities before commercial tools become available.

For this reason test case operation has been fully integrated to the object model and tool interfaces.

Although there has been a long and steady interest in open source facilities, the importance of this need was highlighted by the XBRL Standards Board survey project, which found two relevant and consistent feedback points, (1) the request for open source facilities, and (2) the request for an independent API.

An advantage of open source is that fresh bright minds will, from time to time, come to look at the architecture and structure and see things that those, who are too deep in the forest to see the trees, may have overlooked. It is expected that the architecture will evolve over time and lead to continual product evolution and improvement.

ArELLE was designed to be a minimalist facility, as a counter-response to experience with other APIs that are large, hard to learn, and not directly supportive of the XBRL extension modules. In this case, the API is compact and implemented with a bare minimum of coding. ArELLE is platform independent, implemented in Python, completely from scratch.

To prove the viability of the initial API, it was suggested to assure that SEC Edgar Validation and the Versioning module were implementable, particularly as these have test suites. It was considered lesser priority to replicate functions widely available in other products, such as XML validation (this will be provided later on).

About the name: the sound of pronouncing XBRL makes a pseudonym, Ecksbee Arelle. Arelle is found on baby name websites as a valid girl's name.

The implementation is in Python 3.1, and is intended for Windows (any recent), Mac OS-X 10.3 to 10.6, or any Unix or Linux. Memory required is about twice of comparable commercial products, e.g., a US-GAAP filing might need 30-60MB (x32 ok), comparing two 2011 us-gaaps for versioning report generation might take 4.5G (x64 needed). Loading speed about half of commercial products.

Architecture

An MVC (model-view-controller) architecture has been selected.

Model represents the objects of XBRL: instances, inline-instances, DTS schemas and linkbases, individual test cases, test suites, formula, and versioning reports. The model has a modelManager, which manages the set of models loaded at a time.

The controller represents interaction with external users and external programmatic control, such as by GUI, web, and command line.

A view represents pre-defined API interactions with the model, to present object views for GUI, web, and textual use (e.g., CGI files).

A number of utility functions are included to make the code easier to read and more compact. These include XML utilities, URI utilities, and a customized Python web cache.

Validation operations are factored out to separate classes, as they are quite large to include with the objects that they validate for. Validation operations have been integrated to prevent redundant passes through object models.

Model

The intent of the model is to provide independence of the eventual serialization of XBRL, which for now is XML. The XSB Strategic Initiatives project has a task to develop an SQL model, which may for a basis for an alternate serialization to be consumed by Arelle.

From the top down, there is the necessity to process multiple instances (DTSes) of XBRL concurrently. A ModelManager coordinates them for the Controller, and is the interface to utility functions (such as the Python web cache), and application specific formalisms (such as the SEC restrictions on referencable base taxonomies).

Each loaded instance, DTS, testcase, testsuite, or versioning report is represented by an instance of a ModelXbrl object. The ModelXbrl object has a collection of ModelDocument objects, each representing an XML document (for now, alternate serialization whenever that time comes). One of the modelDocuments of the ModelXbrl is the entry point (of discovery or of the test suite).

Each modelDocument represents a set of modelObjects, which are specialized as follows according to the type of document. There is also one specialization of modelDocument, which is a modelVersReport, as the versioning report has different objects and methods than from any other XBRL modelDocument.

There is also an inherently different model, modelRelationshipSet, which represents an individual base or dimensional-relationship set, or a collection of them (such as labels independent of extended link role).

The model objects are, from general to more specific, models representing a superclass XBRL object, and models representing XBRL role/arcrole types, schema objects, concepts, attributes, and types; for Xlink, links, resources, and locators. Specialized resources represent formula linkbase objects, rendering linkbase objects, and versioning objects. Model relationships represent arcs and are suitable both for

effective arc base and relationship sets, and for determining of ineffective arcs (for Filer submission validation). Instance objects include facts (instance and inline XBRL), contexts, dimensions and units. Testcase objects represent fully integrated testcase variation objects.

Validation operations are separated from the objects that are validated, because the operations are complex, interwoven, and factored quite differently than the objects being validated. There are these validation modules at present: validation infrastructure, test suite and submission control, versioning report validation, XBRL base spec, dimensions, and formula linkbase validation, Edgar and Global Filer Manual validation.

View

View facilities are segregated by the means of rendering, to modules dealing (at present) with CSV result files (static views), GUI window panes (dynamic and interlinked views), and later Web Views (dynamic interlinked and deferred-delivery views). Web based views will be added after extension modules are completed.

The typical instance and DTS views are synchronized for fact, relationship, concept, and other views of the DTS. Selection events of any one view synchronize others that present the same object:

The screenshot shows the arelle™ software interface with the following components:

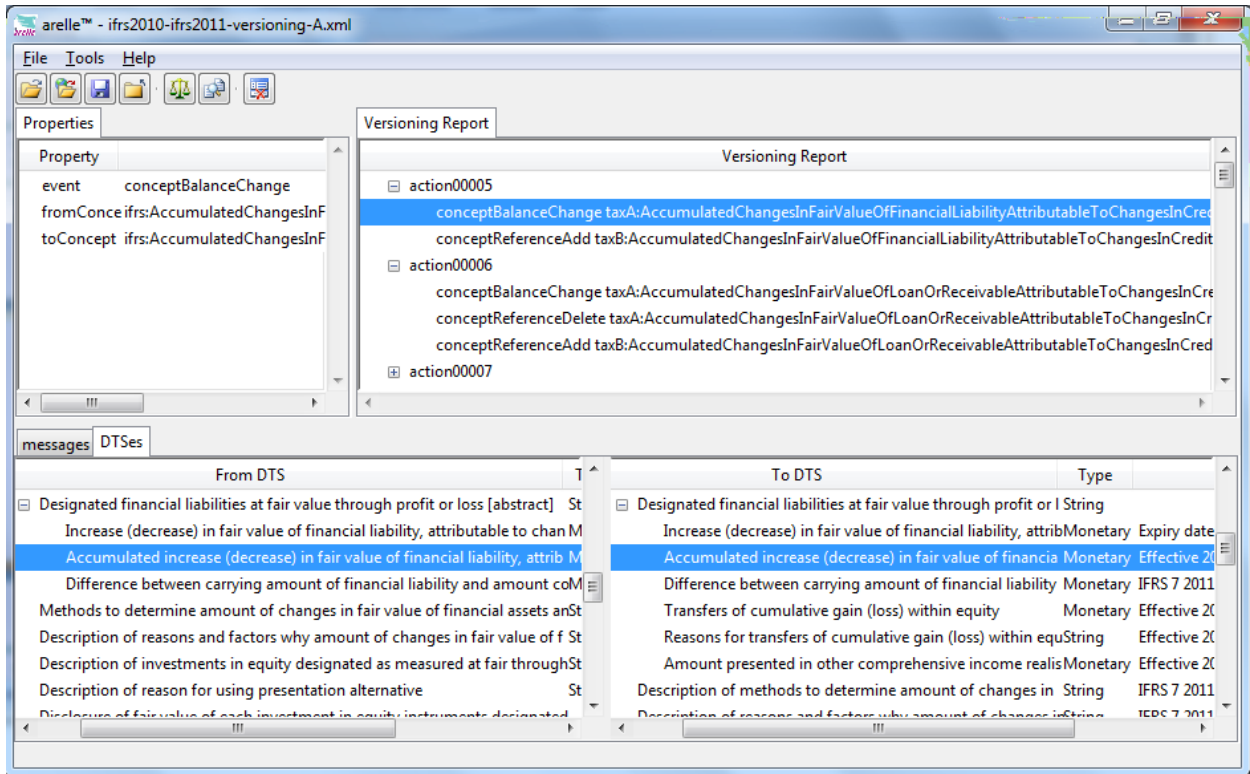
- File Explorer (Left):** A tree view showing the project structure:
 - abc-20101231.xml - instance
 - abc-20101231.xsd - schema
 - us-gaap-2011-01-31.xsd - schem (selected)
 - numeric-2009-12-16.xsd - sch
 - us-roles-2011-01-31.xsd - sch
 - ref-2006-02-27.xsd - schema
 - us-types-2011-01-31.xsd - sch
 - nonNumeric-2009-12-16.xsd
 - xbrldt-2005.xsd - schema
 - xbrl-instance-2003-12-31.xsd
 - abc-20101231_cal.xml - linkbase
 - numeric-2009-12-16.xsd - schem

- Fact Table (Center):** A table with columns for Concept, 2008-12-31, 2009-12-31, and 2010-12-31. The selected row is:

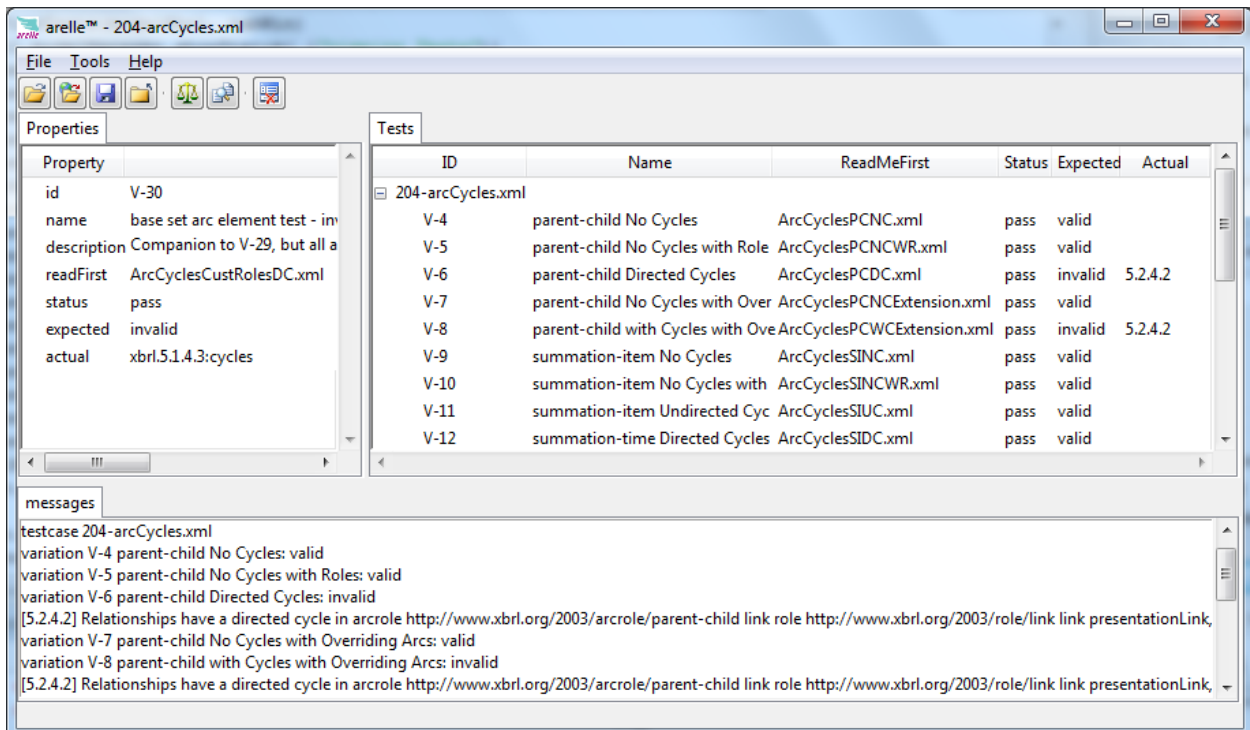
Concept	2008-12-31	2009-12-31	2010-12-31
Cash and Cash Equivalents, at Carrying Value	10,000	11,000	
Restricted Cash and Investments, Curren	1,000	1,000	
Short-term Investments	2,000	1,000	
Accounts Receivable, Net, Current	29,000	29,000	
Inventory, Net	4,000	4,000	
Prepaid Expense, Current	8,000	8,000	
- Concepts (Bottom):** A table with columns for Label, Name, ID, and Abstract. The selected row is:

Label	Name	ID	Abstract
Cash and Cash Equivalents, at Carrying Value	CashAndCashEquivalentsAtCarryingValue	us-gaap_CashAndCashEquivalentsAtCarryingV	false xbrli:item
Change in Accounting Policy	ChangeInAccountingPolicy	abc_ChangeInAccountingPolicy	false xbrli:item
Class of Common Stock [Axis]	ClassOfCommonStockAxis	abc_ClassOfCommonStockAxis	true xbrldt:dime
Class of Common Stock - All Classes IDomain	ClassOfCommonStockAllClassesDomain	abc_ClassOfCommonStockAllClassesDomain	false xbrli:item

Versioning reports are entering use with the 2011 IFRS release. Here is a view of the 2010 and 2011 IFRS taxonomies with the actions of the versioning report interlinked to IFRS-style (“ITI”) displays of the from and to DTSes. Full validation of the versioning report is available, and was helpful in preparing the IFRS 2010/2011 report.

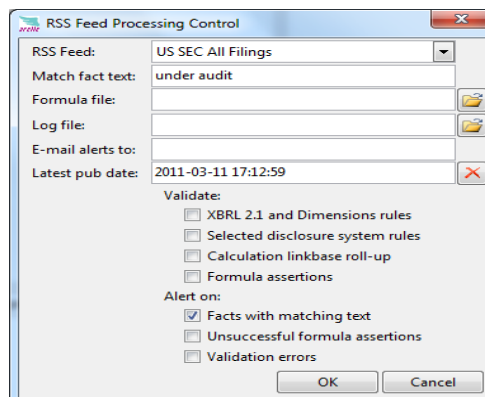


Synchronized viewing is provided for test suite operation. The intent is to encourage uniform access to test suites by all users. As a test is interactively executed the pass/fail status and log of errors can be viewed interactively. (There are scripts to run tests in batch mode too.)

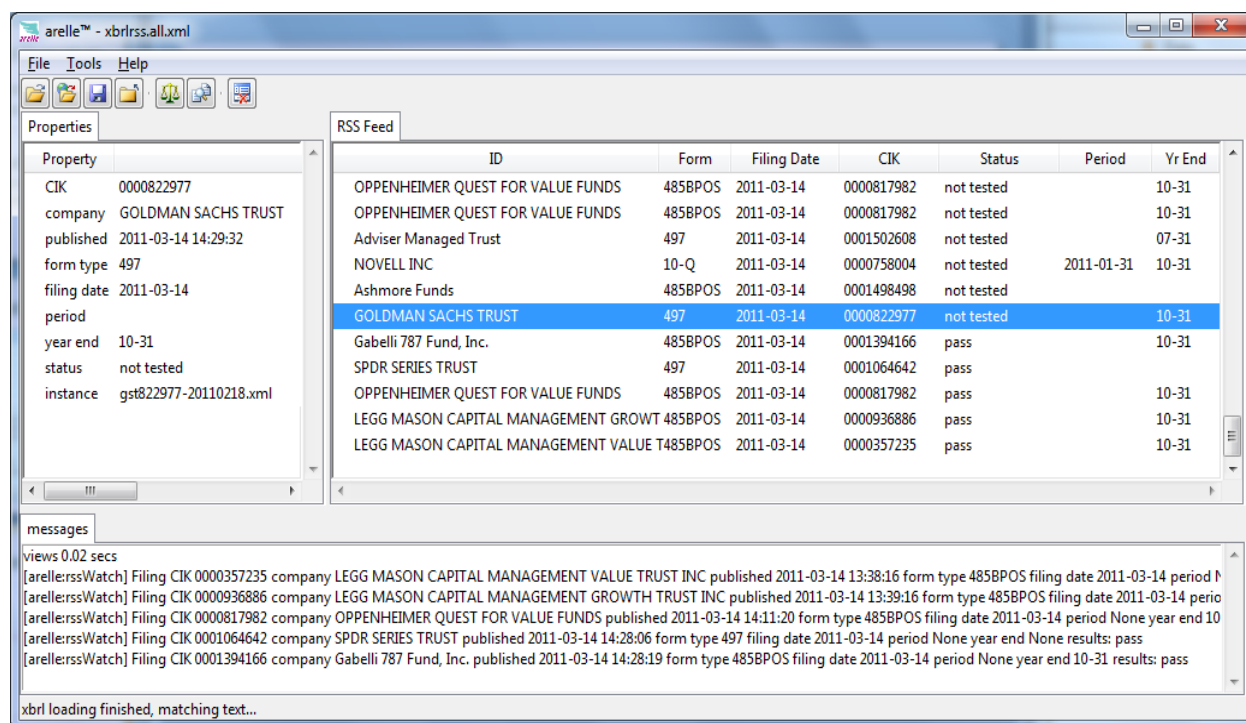


Several view features extend the test suite concept to training and exploration of XBRL formula. An RSS Watch facility allows the user to specify a regular expression or formula (as a set of assertions) to run against the SEC live RSS feeds. This can be started, resumed, or left to run in background.

There are configuration options, including which feed, where to e-mail alerts to, and whether to perform XBRL and Disclosure System validations



As this process runs, a view similar to the integrated test suite operation is shown, where one can see the latest published reports and the status of the testing:



Controller

In a standard definition, the controller receives input and initiates a response by making calls on model objects. A controller accepts input from the user and instructs the model and viewers to perform actions based on that input.

Arellé has these controllers: a superclass for shared common functionality, a command line based controller, suitable for batch file integration, and a GUI controller for mouse-and-menu operation. The GUI uses only graphic libraries distributed with the standard Python distribution (tkinter), so that it is fully compatible and consistent on all of the Python platforms. The batch controller is primarily used for scripted test operation and to perform formula and other tests on large collections of submissions.

Features for Academia

A goal of this project is a platform for XBRL training. This is supported by two key features distinguishing from other products, a compact code base, and a unified object model.

The size of XBRL platforms in their source code form is critical in the ability to support student exploration of features. When size exceeds what can be explored rapidly, the effort to dive into a

product becomes out of the scope of student project periods. Arelle owes its compactness to the Python features such as set operations and compact expressiveness, and its code appears to be smaller by the usual ratio claimed for Python (six-fold reduction of code size is often claimed for Python).

The internal architecture supports academic training by being based on a single set of integrated models, with extension features such as formula, versioning, and rendering, fully integrated.

Development Environment

There are several Python-based development environments, but most XBRL practitioners have spent their lives with Java. Eclipse can be configured for Python, and is compatible with Arelle.

The project is hosted at <http://arelle.org>, including source code, documentation, and a user forum.

About the authors:

Herm Fischer has contributed to the XBRL base specification, and its dimensions, formula, versioning, and rendering modules. He currently chairs the Formula Working Group and is vice chair of the Base Specification and Maintenance Working Group. He contributed to the 2007 XBRL US GAAP Project. Formerly with UBmatrix, Inc, he developed Taxonomy Designer, formula editors and processors, and XBRL processors. He participated in development of SEC validation and its test suite.

Diane Mueller is the founder/president of XBRLSpy Research Inc. She is an Open Source/Open Standards Evangelist, and a XBRL Implementation Strategist. Currently serves as the XBRL Canada representative to the XBRL International Steering Committee and Best Practices Board, and chairs the Technical Working Group on Rendering responsible for the Inline XBRL Specification. She is a frequent commentator and lecturer on Financial Compliance, XML Standards and Semantic Web technologies.