Enabling Comparability and Data Mining with the Arelle® Open Source Unified Model

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

Research Objective

Two key points of the call for this conference are the role of XBRL in comparability of financial information and XBRL enabled tools for data mining. The authors participate in an open source platform for XBRL, known as Arelle, that provides initial capabilities in this area and wish that publication of the underlying model and API of the current project can lead to continued evolution of both open source tooling and the understanding of XBRL technology.

XBRL suppliers have necessarily focused on commercial objectives, somewhat making it difficult for researchers to participate in the direction of the technology and in unencumbered use of XBRL for research projects. The Arelle platform has been used for both comparability objectives (through implementation of XBRL Formula and Versioning) and data mining objectives (including a sample GUI for simple text and formula-based data mining of online XBRL filings).

Methodology

This project began with a goal to support XBRL extension features not widely available, beginning with Versioning, which provides a formal model of the differences between taxonomies, and evolving to newly emerging validation (IFRS Global Filer Manual) and full implementation of XBRL formula. The goal was a compact model-driven architecture that unified these extensions in a manner that was both a comprehensible and maintainable implementation (in terms of source code size), and easily usable by the research community (as well as being open sourced).

The XBRL International 2010 Brussels conference presented interim progress of the XBRL Strategic initiative Abstract Modeling Task Force, which is defining UML models for the XBRL Base Specification and Dimensions, and UML models from the GRC/XML project, which was modeled from financial reporting semantics. Both differ. The authors were encouraged to document the model currently in Arelle, to provide a basis of encouraging the contributing community to both refine the model and further engage it in research activities.

Overview of unified model approach

This paper describes the unified model using UML. An earlier paper (Fischer and Mueller, 2011), describes the origins and goals for the project.

Arelle's model integrates the objects of XBRL instances, inline instances, XBRL Discoverable Taxonomy Sets (DTSes), XBRL formula linkbases, XBRL versioning metamodels (models of DTS comparisons), and testcases. The integration of modeling of testcases provides a practical benefit in that its concrete realization allows for continual verification of tool performance as it is extended and adapted by its users.

A goal of the model was to form the basis of a minimalist initial implementation, 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 realization of the model, at the current stage of progress, is an API that is compact and implemented with a minimum of coding.

A goal of publishing the modeling is to further enhance providing a platform for XBRL training.

An MVC (model-view-controller) architecture has been selected. A top level UML view is shown in Figure 1.

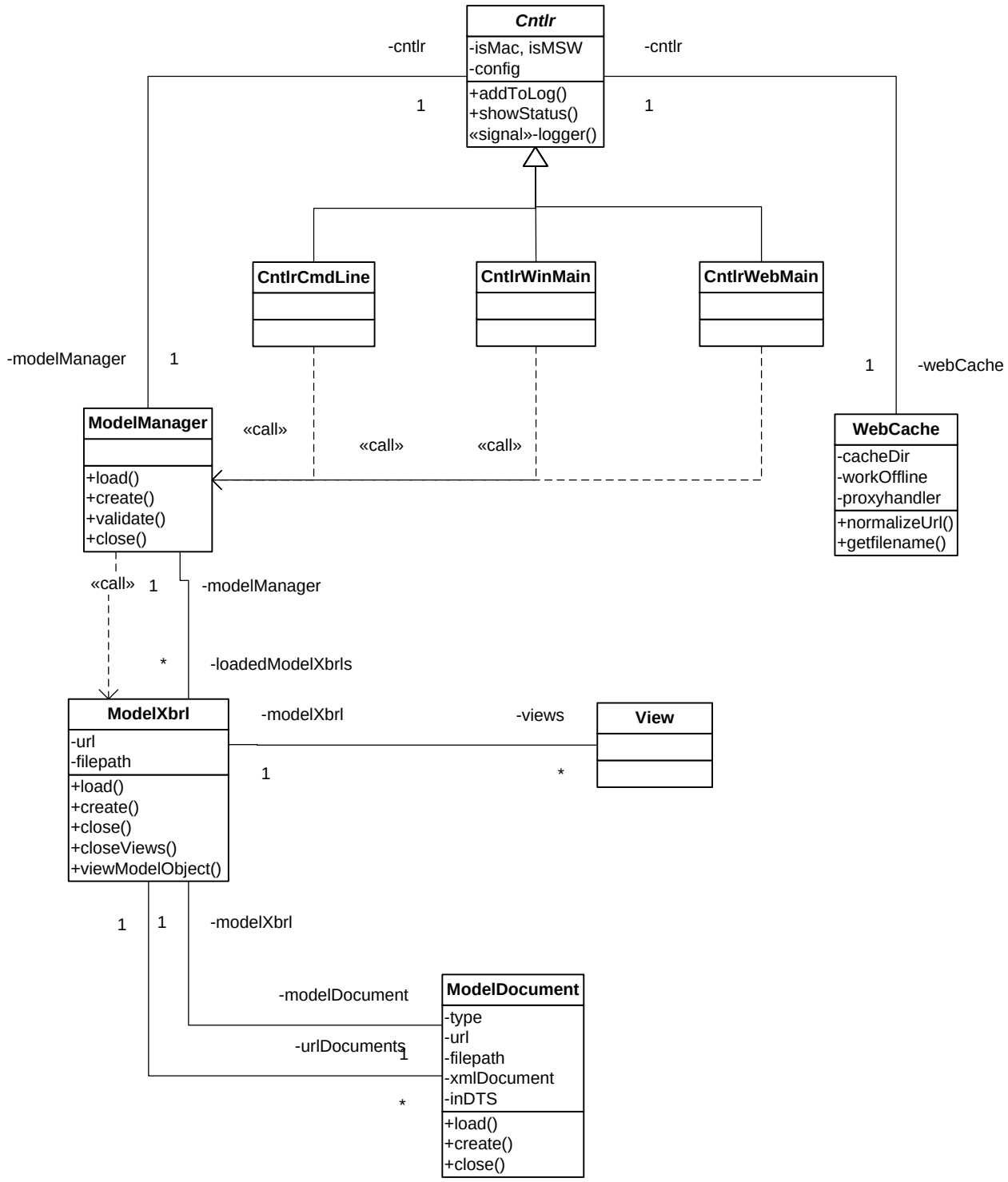


Figure 1. Model View Controller diagram

ModelXbrl 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 (Cntlr) represents interaction with external users and external programmatic control, such as by specialized controllers for GUI (CntlrWinMain), web (CntlrWebMain, to be developed), and

command line (CntlrCmdLine). The command line controller is a pattern for integration to custom projects for macroscopic XBRL operations (such as load/validate/export CSV, etc).

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

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**.**

Top level operation

A controller object is instantiated, CntlrWinMain for the GUI and CntlrCmdLine for command line batch operation. The controller superclass initialization sets up specifics such as directory paths, for its environment (Mac, Windows, or Unix), sets up a web file cache, and retrieves a configuration dictionary of prior user choices (such as window arrangement, validation choices, and proxy settings).

The controller most likely will load an XBRL related object, such as an XBRL instance, taxonomy, testcase file, versioning report, or RSS feed, by requesting the model manager to load and return a reference to its modelXbrl object. The modelXbrl object loads the entry modelDocument object(s), which in turn load documents they discover (for the case of instance, taxonomies, and versioning reports), but defer loading instances for test case and RSS feeds. The model manager may be requested to validate the modelXbrl object, or views may be requested as below. (Validating a testcase or RSS feed will validate the test case variations or RSS feed items, one by one.)

Examples are provided under the discussions of the controller, below.

Top level APIs are found in the package ModuleManager, that coordinates the Controller and Model objects:

ModelManager.py: ModelManager provides an interface between modelXbrl's and the controller. Model manager is a singleton object, one is created in initialization of a controller.	
Methods	
load(file, nextAction=None)	Returns a modelXbrl with a loaded an instance, inline XBRL, schema, or linkbase, and discovers the DTS, or loads a versioning report and its from/to DTSes, testcase index file, testcase variations file, or functions registry file. • file may be a FileSource object or string file name. • nextAction is a status line text string, if any, to show on completion. The modelXbrl that is loaded is 'stacked' so that any modelXbrl operations such as validate, and close, operate on the most recently loaded modelXbrl, and compareDTSes operates on the two most recently loaded modelXbrl's.
validate()	Validates the most recently loaded modelXbrl (according to validation properties)
compareDTSes(versReportFile)	Compare two most recently loaded DTSes, saving versioning report in to the file name provided.
close()	Closes the most recently loaded modelXbrl
Properties	

validateDisclosureSystem	True if disclosure system is to be validated (e.g., EFM)
disclosureSystem	Disclosure system object. To select the disclosure system, e.g., “gfm”, moduleManager.disclosureSystem.select(“gfm”).
validateCalcLB	True for calculation linkbase validation.
validateInferDecimals	True for calculation linkbase validation to infer decimals (instead of precision)
validateUTR	True for validation of unit type registry
defaultLang	The default language code for labels selection and views (e.g. “en-US”), set from the operating system defaults on startup.

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, referring to Figure 1, 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, versioning report, or RSS feed, 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, with SQL whenever its 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 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.

The next sections will examine the model abstracted at the class and relationships level (omitting operations), for DTS models, instance models, and DTS formula models.

Model Objects

Figure 2 shows the Model Objects, which represent the loaded and discovered XML document structure.

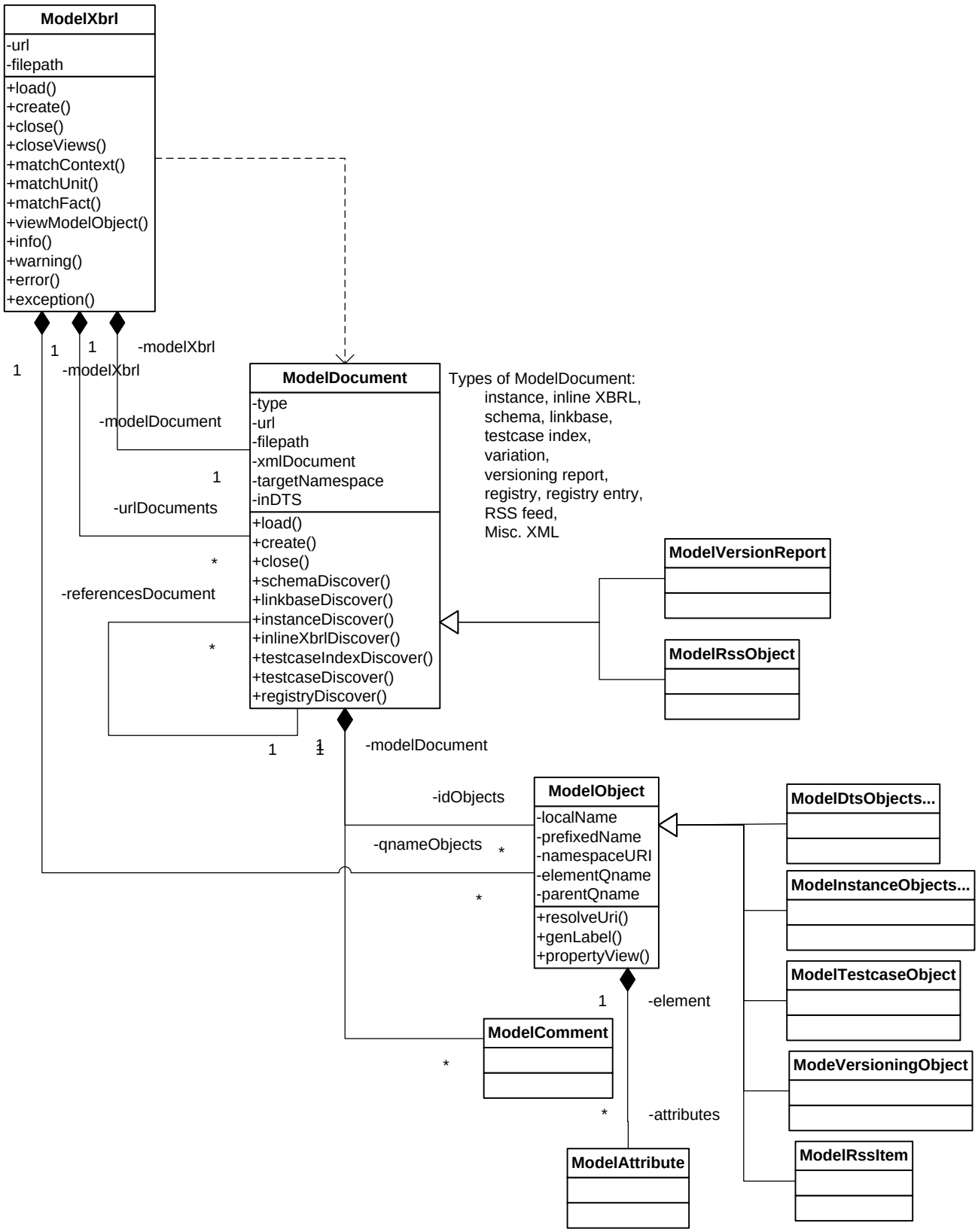


Figure 2. Model Objects

The modelXbrl class can instantiate a modelXbrl and its documents either by loading (and discovery) or by creating (such as an empty instance document, to receive new XBRL formula output facts or a new instance document). It can notify all active views to render or select a specific object for view. It can

match specific criteria to find a context, unit, or fact in an instance (such as to match a generated instance document against an expected testcase result intstance).

The modelDocument performs discovery and initialization when loading documents. For instances, schema and linkbase references are resolved, as well as non-DTS schema locations needed to ensure PSVI-validated XML elements in the instance document (for formula processing). For DTSes, schema includes and imports are resolved, linkbase references discovered, and concepts made accessible by qname by the modelXbrl and ID at the modelDocument scope. Testcase documents (and their indexing files) are loaded as modelDocument objects.

Specialized modelDocuments are the versioning report, which must discover from and to DTSes, and an RSS feed, which has a unique XML structure.

ModelObjects, representing the XML elements within a document, are implemented as custom lxml proxy objects. Each modelDocument has a parser with the parser objects in ModelObjectFactory.py, to determine the type of model object to correspond to a proxied lxml XML element. Both static assignment of class, by namespace and local name, and dynamic assignment, by dynamic resolution of element namespace and local name according to the dynamically loaded schemas, are used in the ModelObjectFactory.

ModelObjects are grouped into Python modules to ensure minimal inter-package references (which causes a performance impact). ModelDtsObjects collects DTS objects (schema and linkbase), ModelInstanceObjects collects instance objects (facts, contexts, dimensions, and units), ModelTestcaseObject collects testcase and variation objects, ModelVersioningObject has specialized objects representing versioning report contents, and ModelRssItem represents the item objects in an RSS feed.

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.

ModelXbrl.py: ModelXbrl objects represent loaded instances and inline XBRL instances and their DTSes, DTSes (without instances), versioning reports, testcase indexes, testcase variation documents, and other document-centric loadable objects.	
Static methods	
load(modelManager, url, nextaction=None, base=None, useFileSource=None)	Returns a new modelXbrl, performing DTS discovery for instance, inline XBRL, schema, linkbase, and versioning report entry urls. • url may be a filename or FileSource object • nextaction is a string to use as status line prompt on conclusion of loading and discovery • base is the base URL if any (such as a versioning report's URL when loading to/from DTS modelXbrl). • useFileSource is for internal use (when an entry point is in a FileSource archive and discovered files expected to also be in the entry point's archive.
create(modelManager, newDocumentType=None, url=None, schemaRefs=None, createModelDocument=True, isEntry=False)	Returns a new modelXbrl, with entry point XML document pre-populated with header and top level elements. For example, can be used by formula processing to create an instance document that is being generated internally and may (or not) be actually saved to a file later. • newDocumentType is a value of ModelDocument.type (e.g., INSTANCE) • url is a real or fake URL (so the document can have a base URL if needed). • schemaRefs is a list of URLs when creating an empty

	INSTANCE, to use to discover (load) the needed DTS modelDocument objects. • isEntry is True when creating an entry (e.g., instance)
Methods	
close()	Closes any views, formula output instances, modelDocument(s), and dereferences all memory used
closeViews()	Close views associated with this modelXbrl
reload(nextaction, reloadCache=False)	Reload a modelXbrl's source files, preserving any open views (that get reloaded), nextaction as per load() above, reloadCache True specifies that any cached files (from web) will be reloaded (if working online)
relationshipSet(arcrole, linkrole=None, linkqname=None, arcqname=None, includeProhibits=False)	Returns a relationship set matching arcrole, linkrole (wild if None), linkqname (wild if None), arcqname (wild if None), or special collective arcroles 'XBRL-dimensions', 'XBRL-formula', and 'Table-rendering'. If relationship set was previously resolved, returns from a cache.
matchSubstitutionGroup(elementQname, subsGrpMatchTable)	Resolve a subsitutionGroup for the elementQname from the match table { (substitutee, returnedObject), …} , used to determine xml proxy object class for xml elements and substitution group membership
isInSubstitutionGroup(elementQname, substitutionQnames)	True if element is in substitution group for qnames (used by formula and generics validation)
matchContext(scheme, identifier, periodType, start, end, dims, segOCCs, scenOCCs)	Finds matching context, by aspects, as in formula usage, if any
matchUnit(multiplyBy, divideBy)	Finds matching unit, by aspects, as in formula usage, if any
matchFact(otherFact)	Finds matching fact, by context qname, v-equality, lang, decimals, and precision, if any, used in verifying an formula standard output instance fact has a match in a testcase expected results file.
modelObject(objectId)	Finds a model object by an ordinal ID which may be buried in a tkinter view id string (e.g., 'somedesignation_ordinalnumber').
viewModelObject(objectId)	Finds model object, if any, and synchronizes any views displaying it to bring the model object into scrollable view region and highlight it
info(code, message, named arguments)	Logs a message as info, by code, logging-system message text (using %(name)s named arguments to compose string by locale language), resolving model object references (such as qname), to prevent non-dereferencable memory usage. Supports logging system parameters, and special parameters modelObject, modelXbrl, or modelDocument, to provide trace information to the file, source line, and href (XPath element scheme pointer). Supports the logging exc_info argument.
warning(code, message, named arguments)	" but as warning
error(code, message, named arguments)	" but as error, and also saves error code and assertion results (in 'errors' list) for validation
exception(code, message, named arguments)	" but as exception
Properties	
urlDocs	Dict, by URL, of loaded modelDocuments
errors	List of error codes and assertion results, which were sent to logger, via error() method above, used for validation and post-processing
logErrorCount, logWarningCoutn, logInfoCount	Counts of respective error levels processed by modelXbrl logger

arcroleTypes	Dict by arcrole of defining modelObjects
roleTypes	Dict by role of defining modelObjects
qnameConcepts	Dict by qname (ModelValue.QName) of all top level schema elements, regardless of whether discovered or not discoverable (not in DTS)
qnameAttributes	Dict by qname of all top level schema attributes
qnameAttributeGroups	Dict by qname of all top level schema attribute groups
qnameTypes	Dict by qname of all top level and anonymous types
baseSets	Dict of base sets by (arcrole, linkrole, arc qname, link qname), (arcrole, linkrole, *, *), (arcrole, *, *, *), and in addition, collectively for dimensions, formula, and rendering, as arcroles 'XBRL-dimensions', 'XBRL-formula', and 'Table-rendering'.
relationshipSets	Dict of effective relationship sets indexed same as baseSets (including collective indices), but lazily resolved when requested.
qnameDimensionDefaults	Dict of dimension defaults by qname of dimension
facts	List of top level facts (not nested in tuples), document order
factsInInstance	List of all facts in instance (including nested in tuples), document order
contexts	Dict of contexts by id
units	Dict of units by id
modelObjects	Model objects in loaded order, allowing object access by ordinal index (for situations, such as tkinter, where a reference to an object would create a memory freeing difficulty).
qnameParameters	Dict of formula parameters by their qname
modelVariableSets	Set of variableSets in formula linkbases
modelCustomFunction Signatures	Dict of custom function signatures by qname
modelCustomFunction Implementations	Dict of custom function implementations by qname
views	List of view objects
langs	Set of langs in use by modelXbrl
labelRoles	Set of label roles in use by modelXbrl's linkbases
hasXDT	True if dimensions discovered
hasTableRendering	True if table rendering discovered
hasFormulae	True if formulae discovered
formulaOutputInstance	Standard output instance if formulae produce one.
Log	Logger for modelXbrl

ModelDocument.py: ModelDocument objects represent each loaded document from an XML source.	
Static methods	
load(modelXbrl, uri, base=None, referringElement=None, isEntry=False, isDiscovered=False, isIncluded=None, namespace=None, reloadCache=False)	Returns a new modelDocument, performing DTS discovery for instance, inline XBRL, schema, linkbase, and versioning report entry urls. • uri may be a filename or FileSource object • referringElement is the source element causing discovery or loading of this document, such as an import or xlink:href • isEntry is True for an entry document • isDiscovered is True if this document is discovered by XBRL rules, otherwise False (such as when schemaLocation and xmlns were the cause of loading the schema) • isIncluced is True if this document is the target of an xs:include • namespace is the schema namespace of this document, if known and applicable

	• reloadCache is True if desired to reload the web cache for any web-referenced files.
create(modelXbrl, type, uri, schemaRefs=None, isEntry=False)	Returns a new modelDocument, created from scratch, with any necessary header elements (such as the schema, instance, or RSS feed top level elements) • type is the type value (see below) • schemaRefs is a list of URLs when creating an empty INSTANCE, to use to discover (load) the needed DTS modelDocument objects. • isEntry is True when creating an entry (e.g., instance)
Type (subclass)	
SCHEMA, LINKBASE, INSTANCE, INLINEXBRL, DTSENTRIES, VERSIONINGREPORT, TESTCASESINDEX, TESTCASE, REGISTRY, REGISTRYTESTCASE, RSSFEED	Enumerated type representing modelDocument type
Methods (of modelDocument objects)	
objectId(refId="")	Returns a string surrogate representing the object index of the model document, prepended by the refId string.
relativeUri(uri)	Returns a uri relative, if possible, to document base
close()	Closes modelDocument, including referenced documents, releases lxml parser internal objects and frees lxml memory.
gettype()	String value of enumerated type name
schemaDiscover()	Initiates XBRL 2.1 discovery within schema documents
baseForElement(element)	Determines xml base of element, from any xml:base on element or ancestors, and document base
linkbasesDiscover()	Initiates discovery linkbases (e.g., within schema file)
linkbaseDiscover()	Initiates discovery of a linkbase (in schema or linkbase file)
discoverHref(element, nonDTS=False)	Processes a href, loading modelDocument if not already loaded, and if nonDTS is False, causing XBRL 2.1 discovery of document. If nonDTS is False and document is already loaded but not previously discovered (its inDTS is False), then its 2.1 discovery is performed.
instanceDiscover()	Initiates discovery of an instance.
inlineXbrlDiscover()	Initiates discovery of an inline XBRL instance.
testcaseIndexDiscover()	Initiates discovery of a testcase index document (which discovers referenced testcase variations)
testcaseDiscover()	Initiates discovery of a testcase variations document
registryDiscover()	Initiates discovery of a registry (such as formula registry, and its referenced testcase variations, in a different format from testcases).
Properties	
modelDocument	Self (provided for consistency with modelObjects)
modelXbrl	The owning modelXbrl
type	The enumerated document type
uri	Uri as discovered
filepath	File path as loaded (e.g., from web cache on local drive)
basename	Python basename (last segment of file path)
xmlDocument	The lxml tree model of xml proxies
targetNamespace	Target namespace (if a schema)
objectIndex	Position in lxml objects table, for use as a surrogate
referencesDocument	Dict of referenced documents, key is the modelDocument, value is why

	loaded (import, include, href)
idObjects	Dict by id of modelObjects in document
modelObjects	List of modelObjects discovered in document in document order
hrefObjects	List of (modelObject, modelDocument, id) for each xlink:href
schemaLocationElements	Set of modelObject elements that have xsi:schemaLocations
referencedNamespaces	Set of referenced namespaces (by import, discovery, etc)
inDTS	Qualifies as a discovered schema per XBRL 2.1

ModelVersReport.py: ModelVersReport is a specialization of ModelDocument for Versioning Reports.	
Static methods	
create(modelXbrlFromDTS, modelXbrlToDTS)	Returns a new modelXbrl representing a Version Report object, by creation of its modelXbrl, its ModelVersReport (modelDocument), and diffing the from and to DTSes • modelXbrlFromDTS is the modelXbrl of fromDTS • modelXbrlToDTS is the modelXbrl of toDTS
Methods	
close()	Closes any views, formula output instances, modelDocument(s), and dereferences all memory used
versioningReportDiscover()	Initiates discovery of versioning report
diffDTSes(versReportFile, fromDTS, toDTS, assignment="technical", schemaDir=None)	Initiates diffing of fromDTS and toDTS, populating the ModelVersReport object, and saving the versioning report file). • versReporFile is the file name to save the versioning report • fromDTS, toDTS are the modelXbrl's to be diffed • assignment is "technical", "business", etc. for the assignment clause • schemaDir is a directory for determination of relative path for versioning xsd files (versioning-base.xsd, etc).
Properties	
fromDTS	From DTS (modelXbrl object)
toDTS	To DTS (modelXbrl object)
assignments	Dict by id of ModelAssignment objects
actions	Dict by id of ModelAction objects
namespaceRenameFrom	Dict by fromURI of ModelNamespaceRename objects
namespaceRenameTo	Dict by toURI of ModelNamespaceRename objects
roleChanges	Dict by uri of ModelRoleChange objects
conceptBasicChanges	List of ModelConceptBasicChange objects
conceptExtendedChanges	List of ModelConceptExtendedChange objects
equivalentConcepts	Dict by qname of equivalent qname
relatedConcepts	DefaultDict by qname of list of related concept qnames
relationshipSetChanges	List of ModelRelationshipSet objects
instanceAspectChanges	List of ModelInstanceAspectChange objects
typedDomainsCorrespond	Dict by (fromDimConcept,toDimConcept) of bool that is True if corresponding

ModelRssObject.py: ModelRssObject is a specialization of ModelDocument for RSS Feeds.	
Methods	
rssFeedDiscover()	Initiates discovery of RSS feed

ModelObject.py:

ModelObject is a custom lxml proxy object, implemented as a specialization of etree.ElementBase, and used as the superclass of discovered and created objects in XML-based objects in Arelle. ModelObject is also used as a phantom proxy object, for non-XML objects that are resolved from modelDocument objects, such as the ModelRelationship object.
ModelObjects persistent with their owning ModelDocument, due to reference by modelObject list in modelDocument object.
ModelConcept and ModelProcessingInstruction are custom proxy objects for etree.CommentBase and PIBase.
ModelAttribute stores PSVI attribute values for each object.

ModelObject Methods (in addition to lxml ElementBase methods, such as getparent())	
prefixedNameQname(prefixedName)	Returns ModelValue.QName of prefixedName using element and ancestors xmlns.
resolveUri(hrefObject=None, uri=None, dtsModelXbrl=None)	Returns modelObject within modelDocment that resolves a URI based on arguments: • hrefObject (hrefElement, modelDocument, id), or • uri (element scheme pointer), and • dtsModelXbrl (default is the element's own modelXbrl)
ModelObject Properties (in addition to lxml CommentBase properties, such as .text)	
modelDocument	Owning ModelDocument object
modelXbrl	modelDocument's owning ModelXbrl object
objectId(refId="")	Returns a string surrogate representing the object index of the model document, prepended by the refId string.
localName	W3C DOM localName
prefixedName	Prefix by ancestor xmlns and localName of element
namespaceURI	W3C DOM namespaceURI (overridden for schema elements)
elementNamespaceURI	W3C DOM namespaceURI (not overridden by subclasses)
qname	ModelValue.QName of element (overridden for schema elements)
elementQname	ModelValue.QName of element (not overridden by subclasses)
parentQname	ModelValue.QName of parent element
id	Id attribute or None
elementAttributesTuple	Python tuple of (tag, value) of specified attributes of element, where tag is in Clark notation
elementAttributesStr	String of tag=value[,tag=value…] of specified attributes of element
xValid	XmlValidation.py validation state enumeration
xValue	PSVI value (for formula processing)
sValue	s-equals value (for s-equality)
xAttributes	Dict by attrTag of ModelAttribute objects (see below) of specified and default attributes of this element.
ModelComment (custom etree.CommentBase) Properties	
modelDocument	Owning ModelDocument object
ModelProcessingInstruction Properties (in addition to lxml PIBase properties)	
modelDocument	Owning ModelDocument object
ModelAttribute (slot-based, like java struct, to store PSVI attribute values on ModelObject)	
modelElement	Owning ModelObject element lxml proxy object
attrTag	Attribute tag (Clark notation)
xValid	XmlValidation.py validation state enumeration
xValue	PSVI value (for formula processing)
sValue	s-equals value (for s-equality)

Model DTS Objects

Figure 3 shows the Model DTS Objects, which represent the loaded and discovered XML document structure except for relationship sets, which are constructed on demand (by validation or view operations).

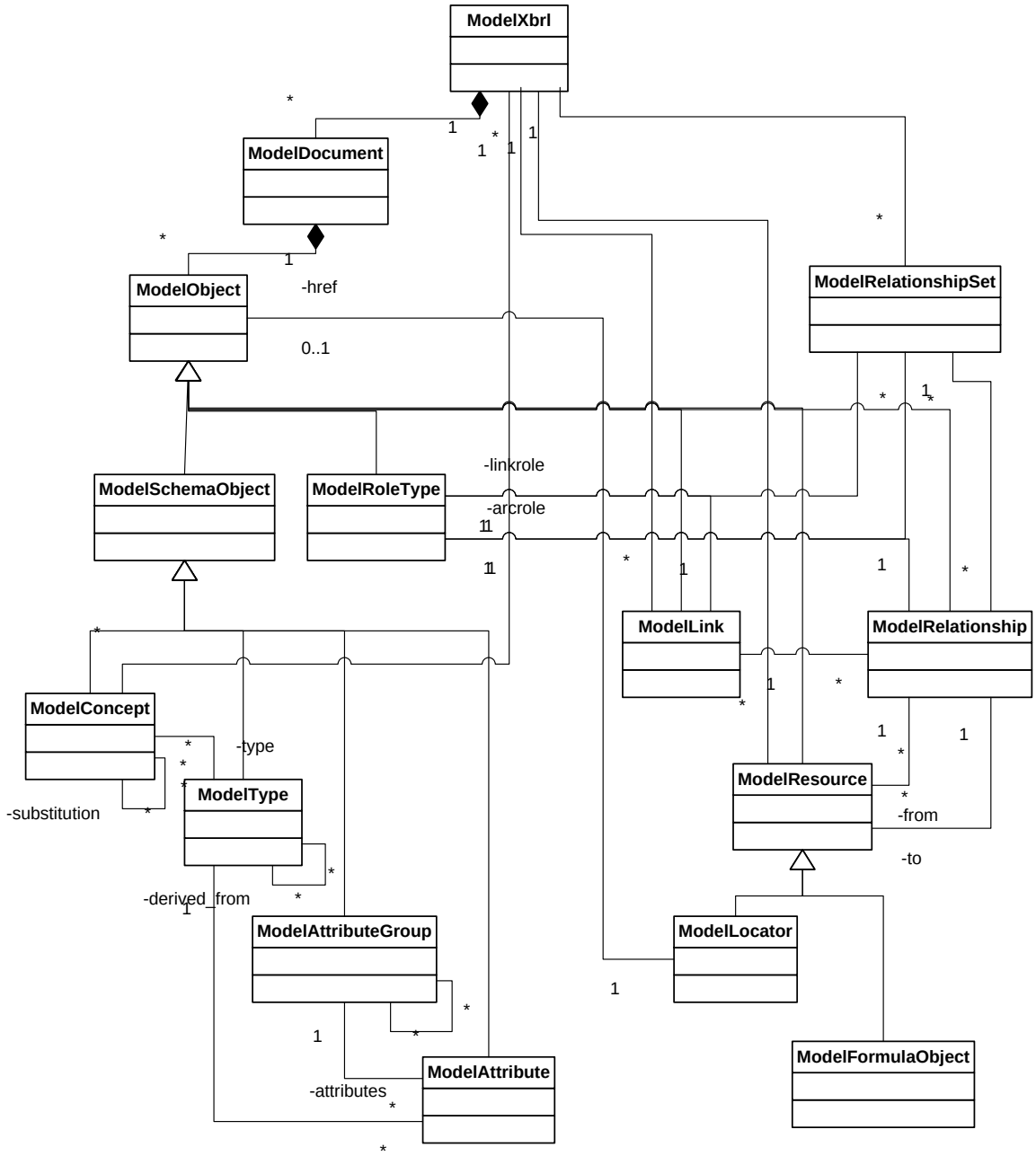


Figure 3. Model DTS Objects

XBRL processing requires element-level access to schema elements. Traditional XML processors, such as lxml (based on libxml), and Xerces (not available in the Python environment), provide opaque schema models that cannot be used by an XML processor. Arelle implements its own elment, attribute, and type processing, in order to provide PSVI-validated element and attribute contents, and in order to access XBRL features that would otherwise be inaccessible in the XML library opaque schema models.

ModelConcept represents a schema element, regardless whether an XBRL item or tuple, or non-concept schema element. The common XBRL and schema element attributes are provided by Python properties, cached when needed for efficiency, somewhat isolating from the XML level implementation.

There is thought that a future SQL-based implementation may be able to utilize ModelObject proxy objects to interface to SQL-obtained data.

ModelType represents an anonymous or explicit element type. It includes methods that determine the base XBRL type (such as monetaryItemType), the base XML type (such as decimal), substitution group chains, facits, and attributes.

ModelAttributeGroup and ModelAttribute provide sufficient mechanism to identify element attributes, their types, and their default or fixed values.

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), based on the semantics of XLink arcs.

PSVI-validated instance data are determined during loading for instance documents, and on demand for any other objects (such as when formula operations may access linkbase contents and need PSVI-validated contents of some linkbase elements). These validated items are added to the ModelObject lxml custom proxy objects.

Linkbase objects include modelLink, representing extended link objects, modelResource, representing resource objects, and modelRelationship, which is not a lxml proxy object, but represents a resolved and effective arc in a relationship set.

ModelRelationshipSets are populated on demand according to specific or general characteristics. A relationship set can be a fully-specified base set, including arcrole, linkrole, link element qname, and arc element qname. However by not specifying linkrole, link, or arc, a composite relationship set can be produced for an arcrole accumulating relationships across all extended link linkroles that have contributing arcs, which may be needed in building indexing or graphical topology top levels.

Relationship sets for dimensional arcroles will honor and traverse targetrole attributes across linkroles. There is a pseudo-arcrole for dimensions that allows accumulating all dimensional relationships regardless of arcrole, which is useful for constructing certain graphic tree views.

Relationship sets for table linkbases likewise have a pseudo-arcrole to accumulate all table relationships regardless of arcrole, for the same purpose.

Relationship sets can identify ineffective arcroles, which is a requirement for SEC and GFM validation.

ModelDtsObject.py:

This module contains DTS-specialized ModelObject classes: ModelRoleType (role and arcrole types), ModelSchemaObject (parent class for top-level named schema element, attribute, attribute groups, etc), ModelConcept (xs:elements that may be concepts, typed dimension elements, or just plain XML definitions), ModelAttribute (xs:attribute), ModelAttributeGroup, ModelType (both top level named and anonymous simple and complex types), ModelEnumeration, ModelLink (xlink link elements), ModelResource (xlink resource elements), ModelLocator (subclass of ModelResource for xlink locators), and ModelRelationship (not an lxml proxy object, but a resolved relationship that reflects an effective arc between one source and one target).

Each of these classes inherits ModelObject and lxml etree.ElementBase methods and properties.

ModelRoleType Properties	
isArcrole	True for arcroleType, False for roleType
roleURI	Value of roleURI attribute
arcroleURI	Value of arcroleURI attribute
cyclesAllowed	Value of cyclesAllowed attribute
Definition	Text of definition element, trimmed of whitespace
definitionNotStripped	Definition, but not trimmed of whitespace
usedOns	Set of ModelValue.QNames of used on element content.

ModelSchemaObject Properties (superclass for xs:element, xs:attribute, xs:*Types)	
Name	Value of name attribute
Qname	ModelValue.QName formed of name and the schema targetNamespace (overrides ModelObject qname)
ModelConcept Properties	
abstract	Value of abstract attribute
isAbstract	True if abstract equals True
periodType	Value of periodType attribute
balance	Value of balance attribute
typeQname	ModelValue.QName of named type, if any, or anonymous type (element name plus anonymous type suffix), or substitution group's type.
niceType	Provides a type name suited for user interfaces: hypercubes as *Table*, dimensions as *Axis*, types ending in *ItemType* have *ItemType* removed and first letter capitalized (e.g., *stringItemType* as *String*). Otherwise returns the type's localName portion.
baseXsdType	Attempts to return the base xsd type localName that this concept's type is derived from. If not determinable *anyType* is returned. E.g., for *monetaryItemType*, *decimal* is returned.
facets	Returns self.type.facets or None (if type indeterminate)
baseXbrliType	Attempts to return the base xbrli type localName that this concept's type is derived from, or None if not determinable.
isNumeric	True for a numeric xsd base type (not including xbrl fractions)
isFraction	True if the baseXbrliType is fractionItemType
isMonetary	True if the baseXbrliType is monetaryItemType
isShares	True if the baseXbrliType is sharesItemType
isTextBlock	Element's type.isTextBlock
type	Element's modelType object (if any)
substitutionGroup	modelConcept object for substitution group (or None)
substitutionGroupQnames	Ordered list of ModelValue.QNames of substitution groups (recursively)
subGroupHeadQname	Qname of the head of the substitution groups
isQualifiedForm	True if element has form attribute qualified or its document default
nillable	Contents of the nillable attribute or its default
isNillable	True if nillable
block	Contents of the block attribute
default	Contents of the default attribute
fixed	Contents of the fixed attribute or None
final	Contents of the final attribute or None
isRoot	True if element definition is at root (not nested)
isItem	True if xbrli:item is at head of substitution group (but False if element is xbrli:item itself)
isTuple	" if xbrli:tuple "
isLinkPart	" if link:part "
isPrimaryItem	True for an item that is not a hypercube or dimension
isDomainMember	"
isHypercubeItem	True if substitutes for hypercubeItem
isDimensionItem	" dimensionItem
isTypedDimension	" dimensionItem with typedDomainRef specified
isExplicitDimension	" dimensionItem without typedDomainRef
typedDomainElement	modelConcept object of the typedDomainRef or None
ModelConcept Methods	
instanceOfType(typeQname)	Returns True if element's type is derived from typeQname.

label(preferredLabel=None, fallbackToQname=True, lang=None, strip=False)	Returns effective label for concept, using preferredLabel role (or standard label if None), absent label falls back to element qname (prefixed name) if specified, lang falls back to tool-config language if none, leading/trailing whitespace stripped (trimmed) if specified. Does not look for generic labels (use superclass genLabel for generic label).
relationshipToResource(resourceObject, arcrole)	For specified object and resource (all link roles), returns first modelRelationshipObject that relates from this element to specified resourceObject.
substitutesForQname(subsQname)	True if element substitutes for specified qname
ModelAttribute Properties	
typeQname	QName of type of attribute
type	Attribute's modelType object (if any)
baseXsdType	Attempts to return the base xsd type localName that this attribute's type is derived from. If not determinable *anyType* is returned.
facets	Returns self.type.facets or None (if type indeterminate)
isNumeric	True for a numeric xsd base type (not including xbrl fractions)
isQualifiedForm	True if attribute has form attribute qualified or its document default
isRequired	True if use is required
default	Contents of the default attribute
fixed	Contents of the fixed attribute or None
dereference	If element is a ref (instead of name), provides referenced modelAttribute object, else self
ModelAttribute Properties	
isQualifiedForm	True (for compatibility with other schema objects)
Attributes	Dict of modelAttribute objects by qname
Dereference	If element is a ref (instead of name), provides referenced modelAttributeGroup object, else self
ModelType Properties	
name	The type's name (localName), if specified, otherwise the nearest ancestor element with a name + *@anonymousType*
isQualifiedForm	True (for compatibility with other schema objects)
qnameDerivedFrom	ModelValue.QName of the type that this type is derived from
typeDerivedFrom	ModelType object of the type that this type is derived from
baseXsdType	The xsd type localName that this type is derived from or: *noContent* for an element that may not have text nodes, anyType for an element that may have text nodes but their type is not specified, or one of several union types for schema validation purposes: *XBRLI_DATEUNION, XBRLI_DECIMALSUNION, XBRLI_PRECISIONUNION, XBRLI_NONZERODECIMAL.*
baseXbrliType	The localName of the parent type in the xbrli namespace, if any, otherwise the localName of the parent in the xsd namespace.
isTextBlock	True if type is, or is derived from, us-types:textBlockItemType or dtr-types:escapedItemType
isDomainItemType	True if type is, or is derived from, domainItemType in either a us-types or a dtr-types namespace.
elements	List of element QNames that are descendants (content elements)
facets	Dict of facets by their facet name, all are strings except enumeration, which is a set of enumeration values.
requiredAttributeQnames	Set of attribute QNames which have use=required.
defaultAttributeQnames	Set of attribute QNames which have a default specified
ModelType Methods	
isDerivedFrom(typeqname)	True if type is derived from type specified by QName

attributes	Dict of ModelAttribute attribute declarations by attribute QName
fixedOrDefaultAttrValue(attrName)	Descendant attribute declaration value if fixed or default, argument is attribute name (string), e.g., “precision”.
ModelEnumeration Properties (of an xsd:enumeration declaration element)	
value	Value attribute
ModelLink Properties (of an link:extended element)	
typeQname	QName of type of attribute
labeledResources	Dict of child labeled xlink:resources by their xlink:label
ModelResource Properties (of an xlink:resource element), superclass of ModelLocator and formula resource objects	
role	xlink:role attribute value or None if absent
xlinkLabel	xlink:label “
xmlLang	xml:lang “
viewText	Text of contained (inner) text nodes except for any whose localName starts with *URI*, for label and reference parts displaying purposes.
dereference	Self (provided for compatibility with ModelLocator (loc) objects
ModelLocator Properties (of an xlink:locator element) Subclass of ModelResource	
dereference	ModelObject that xlink:href resolves to.
ModelRelationship Properties (a ModelObject that does not proxy an lxml object)	
arcElement	ModelObject arc element of the effective relationship
fromModelObject	ModelObject of the xlink:from (dereferenced if via xlink:locator)
toModelObject	“ xlink:to “
get(), localName, namespaceURI, prefixedName, sourceline, tag, elementQname, qname, itersiblings()	Properties (and methods) proxies for the arc element of the effective relationship so that the non-proxy ModelRelationship can be used in terms as if it were the arc’s ModelObject.
fromLabel	Value of xlink:from attribute
toLabel	“ xlink:to “
arcrole	“ xlink:arcrole “
order	Float value of order attribute, or 1.0 if absent
priority	Int value of priority attribute or 0 if absent
weight	Float value of weight or None if absent
use	Value of use attribute
isProhibited	True if prohibited
preferredLabel	Value of preferredLabel attribute
variableName	Value (as string) of formula relationship name attribute
variableQname	Value as QName “, considering no prefix does not take xmlns default.
linkQname	QName of containing extended link element
contextElement	Value of xbrldt:contextElement attribute if present, else None
targetRole	Value of xbrldt:targetRole attribute if present, else None
Usable	Value of xbrldt:usable attribute if present, else *true*
isUsable	True for xbrldt:usable attribute value true or absent
closed	Value of xbrldt:closed attribute if present, else *false*
isClosed	True if closed=*true*
isComplemented	True for formula arc specifying complemented
isCovered	True for formula arc specifying covered
tableAxis	Table linkbase axis type
ModelRelationship Methods	
isIdenticalTo(otherRelationship)	True if modelRelationships are identical (same arc element, resolves to same from and to elements)

Model Instance Objects

Model Instance Objects are shown in figure 4.

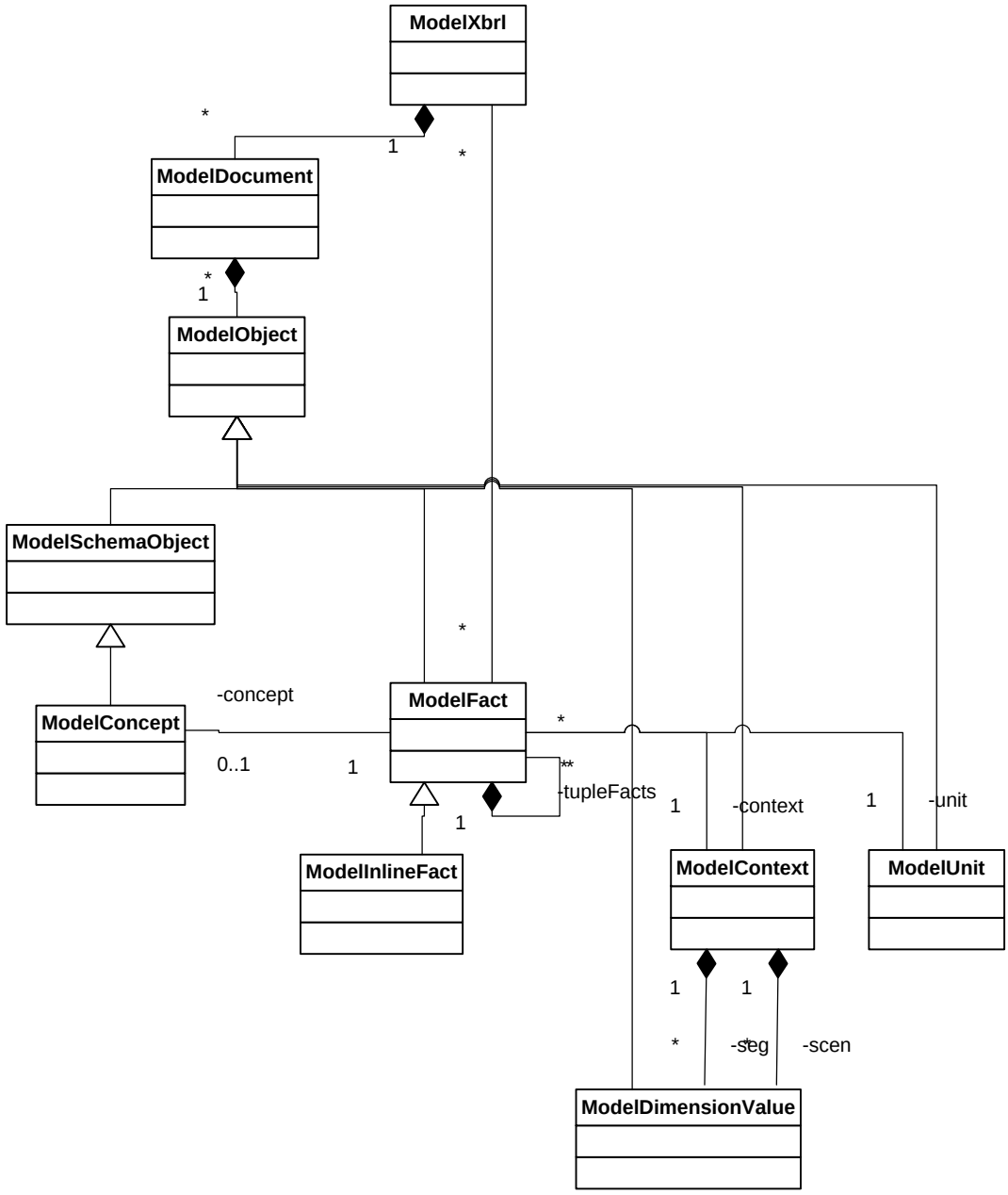


Figure 4. Model Instance Objects

Model facts represent XBRL instance facts (that are elements in the instance document). Model inline facts represent facts in a source xhtml document, but may accumulate text across multiple mixed-content elements in the instance document, according to the rendering transform in effect. All inline facts are lxml proxy objects for the inline fact and have a cached value representing the transformed value content. PSVI values for the inline fact's value and attributes are on the model inline fact object (not necessarily the element that held the mixed-content text).

Model context objects are the lxml proxy object of the context XML element, but cache and interface context semantics that may either be internal to the context, or inferred from the DTS (such as default dimension values). PSVI values for elements internal to the context, including segment and scenario

elements, are on the individual model object lxml custom proxy elements. For fast comparison of dimensions and segment/scenario, hash values are retained for each comparable item.

Model dimension objects not only represent proxy objects for the XML elements, but have resolved model DTS concepts of the dimension and member, and access to the typed member contents.

Model unit objects represent algebraically usable set objects for the numerator and denominator measure sets.

ModelInstanceObject.py: This module contains Instance-specialized ModelObject classes: ModelFact (xbrli:item and xbrli:tuple elements of an instance document), ModelInlineFact specializes ModelFact when in an inline XBRL document, ModelContext (xblrli:context element), ModelDimensionValue (xbrldi:explicitMember and xbrli:typedMember elements), and ModelUnit (xbrli:unit elements). Each of these classes inherits ModelObject and lxml etree.ElementBase methods and properties.	
ModelFact Properties	
concept	ModelConcept object of the fact.
contextID	Value of contextRef attribute, None if absent
context	ModelContext object of the fact, None if absent.
unitID	Value of unitRef attribute, None if absent
unit	ModelUnit object of the fact, None if absent
conceptContextUnitLangHash	Hash value of fact's concept QName, dimensions-aware context hash, unit hash, and xml:lang hash, useful for fast comparison of facts for EFM 6.5.12
isItem	concept.isItem
isTuple	concept.isTuple
isNumeric	concept.isNumeric (note this is *false* for fractions)
isFraction	concept.isFraction
parentElement	ModelObject of parent element (tuple or xbrli:xbrl)
ancestorQnames	Set of QNames of ancestor elements (tuple and xbrli:xbrl)
decimals	Value of decimals attribute (as string)
precision	Value of precision attribute (as string)
xmlLang	Value of xml:lang attribute
xsiNil	Value of xsi:nil attribute if present, else *false*
isNil	True if xsiNil is *true*
value	Text value of fact or default or fixed if any, otherwise None
fractionValue	(text value of numerator, text value of denominator)
effectiveValue	Effective value for views, *(nil)* if isNil, None if no value, locale-formatted decimal value (if decimals specified) , otherwise string value
vEqValue	v-equal value, float if numeric, otherwise string value
ModelFact Methods	
isVEqualTo(other)	v-equality of two facts
ModelInlineFact Properties specializes ModelFact for inline XBRL documents (as xhtml elements)	
qname	QName of concept from the name attribute, overrides and corresponds to the qname property of a ModelFact (inherited from ModelObject)
sign	Value of sign attribute
tupleID	Value of tupleID attribute
tupleRef	Value of tupleRef attribute
footnoteRefs	Value of footnoteRefs attribute
format	Value of format attribute

scale	Value of scale attribute
value	Overrides and corresponds to value property of ModelFact, for relevant inner text nodes aggregated and transformed as needed.
ModelContext Properties	
isStartEndPeriod	True for startDate/endDate period
isInstantPeriod	True for instant period
isForeverPeriod	True for forever period
startDatetime	Datetime value of start
endDatetime	Datetime value of end or instant, with adjustment to next day midnight as needed
instantDatetime	Datetime value of instant, with adjustment to next day midnight as needed
period	ModelObject of period element
periodHash	Hash for comparing periods (start, end, instant, forever)
entity	ModelObject of entity element
entityIdentifierElement	ModelObject of entity identifier element
entityIdentifier	(scheme value, identifier value)
entityIdentifierHash	Hash of entityIdentifier
hasSegment	True if a xbrli:segment element is present
segment	ModelObject of segment element
hasScenario	True if a xbrli:scenario element is present
scenario	ModelObject of scenario element
dimAspects	For formula and instance aspects processing, set of all dimensions reported or defaulted.
dimsHash	A hash of the set of reported dimension values.
segmentHash	Hash of the segment, based on s-equality values
scenarioHash	Hash of the scenario, based on s-equality values
nonDimHash	Hash, of s-equality values, of non-XDT segment and scenario objects
contextDimAwareHash	Hash of period, entityIdentifier, dim, and nonDims
contextNonDimAwareHash	Hash of period, entityIdentifier, segment, and scenario (s-equal based)
ModelContext Methods	
dimValues(contextElement)	ContextElement is *segment* or *scenario*, returns dict of ModelDimension objects indexed by ModelConcept dimension object.
hasDimension(dimQname)	True if dimension concept qname is reported by context (in either context element), not including defaulted dimensions.
dimValue(dimQname)	Returns ModelDimension object if dimension is reported (in either context element), or QName of dimension default if there is a default, otherwise None
dimMemberQname(dimQname, includeDefaults=False)	Returns QName of explicit dimension if reported (or defaulted if includeDefaults is True), else None
nonDimValues(contextElement)	ContextElement is either string or Aspect code for segment or scenario, returns nonXDT ModelObject children of context element.
isPeriodEqualTo(context2)	True if periods are datetime equal (based on 2.1 date offsets)
isEntityIdentifierEqualTo(context2)	True if entityIdentifier values are equal (scheme and text value)
isEqualTo(context2, dimensionalAspectModel= None)	If dimensionalAspectModel is absent, True is assumed. False means comparing based on s-equality of segment, scenario, while True means based on dimensional values and nonDimensional values separately.
ModelDimension Properties	
dimensionQname	QName of the dimension
Dimension	ModelContext object of the dimension
isExplicit	True if explicitMember element

isTyped	True if typedMember element
typedMember	Child ModelObject that is the dimension member element
memberQname	explicitDimension member QName
Member	ModelConcept object of the dimension member or None if typed
contextElement	Parent element localName (*segment* or *scenario*)
ModelDimension Methods	
isEqualTo(otherDimensionObject, equalMode	True if explicit member QNames equal or typed member nodes correspond, given equalMode (s-equal, s-equal2, or xpath-equal for formula)
ModelUnit Properties	
measures	Returns a tuple of multiply measures list and divide members list (empty if not a divide element)
hash	Hash of measures in both multiply and divide lists.
isDivide	True if unit has a divide element
isSingleMeasure	True for a single multiply and no divide measures
isEqualTo(otherUnit)	True of measures are equal
value	String value for view purposes, space separated list of string qnames of multiply measures, and if any divide, a “/” character and list of string qnames of divide measure qnames.

Model Formula Objects

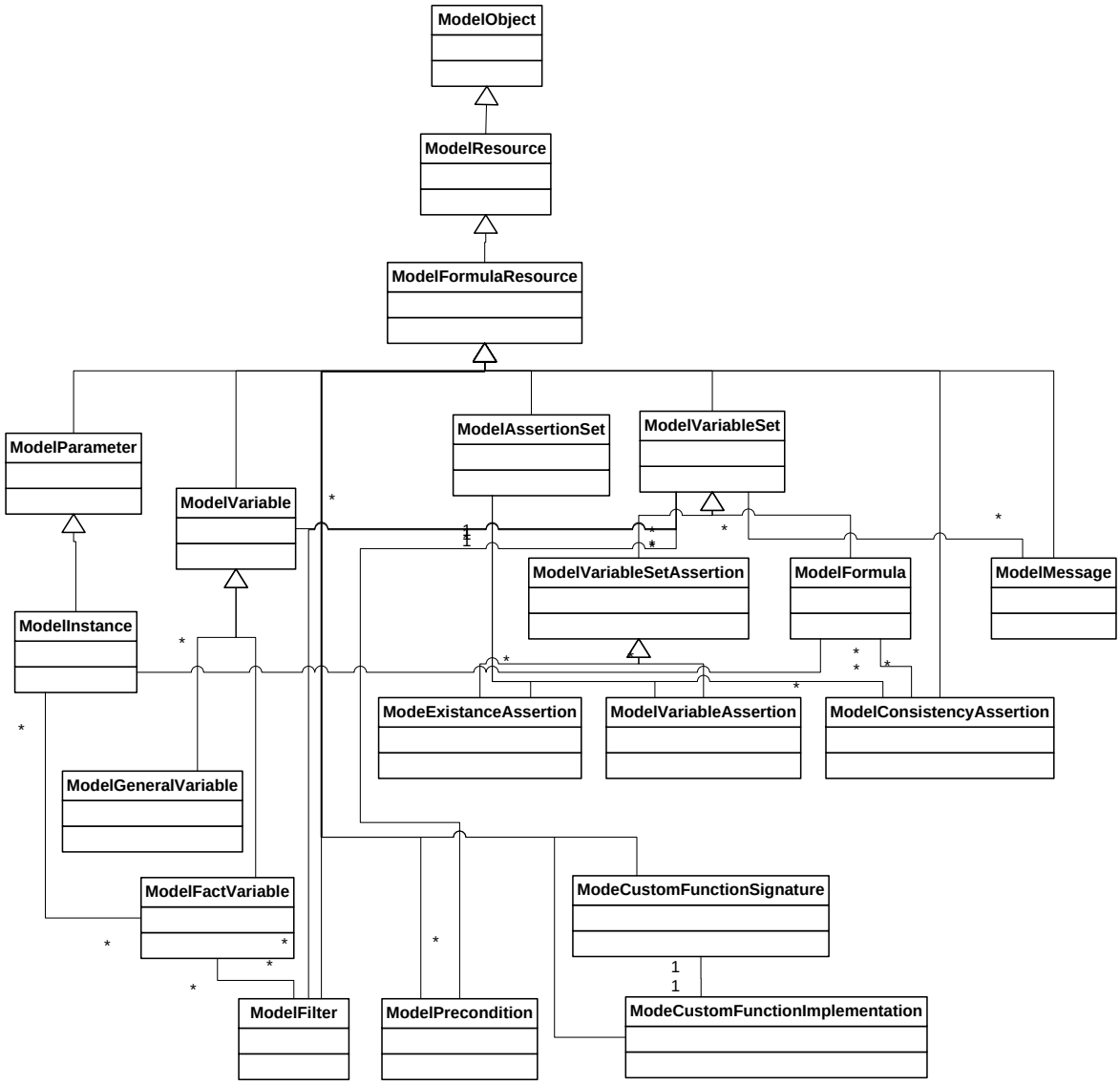


Figure 5. Model Formula Objects

The model of objects required for formula processing is represented by the lxml custom proxy objects for the formula resources. Each object that may have XPath 2 expressions has a set of methods that allow 'crawling' the formula graph to compile these expressions in advance of execution. Each filter has a set of methods for determining the coverable aspects, and to execute the filter against a set of candidate fact values. This significantly simplifies adding future aspects and aspect models, and expanding the set of defined filters.

The execution of a set of formulas is provided as part of validation, or as an independent set of one-by-one calls (such as required for function registry test cases)

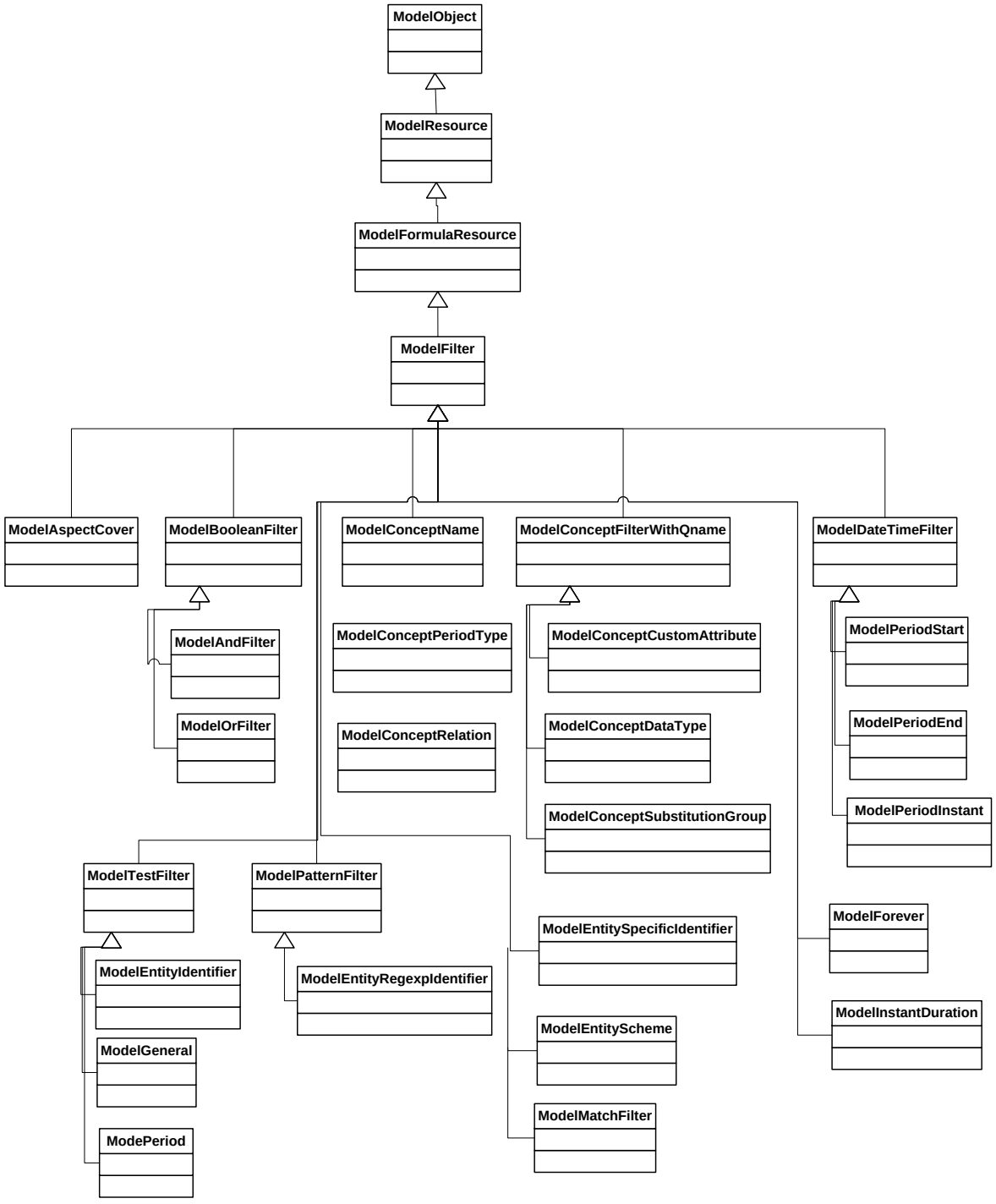


Figure 6. Model Formula Filter Objects, part 1

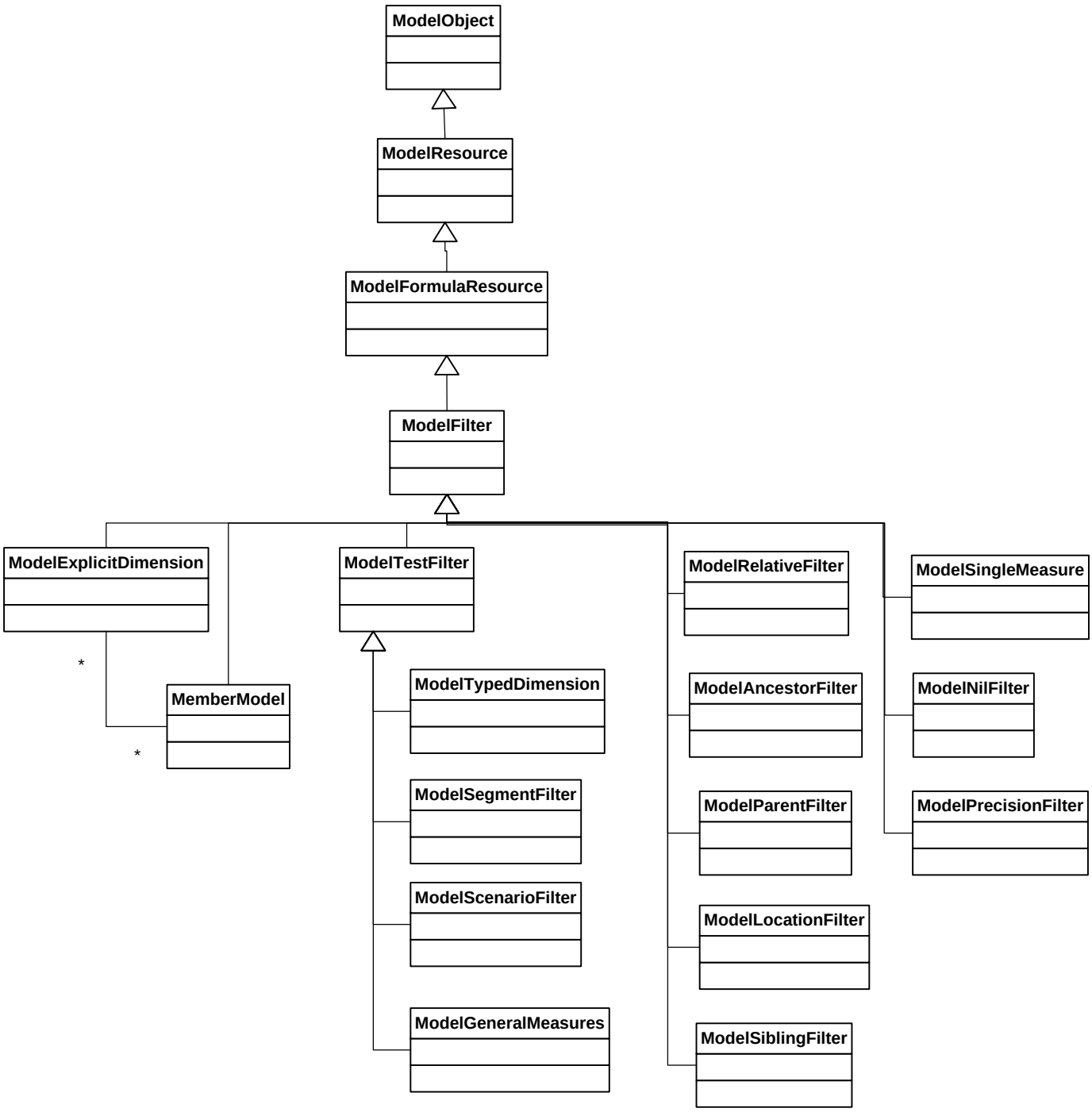


Figure 7. Model Formula Filter Objects, part 2

Validation objects

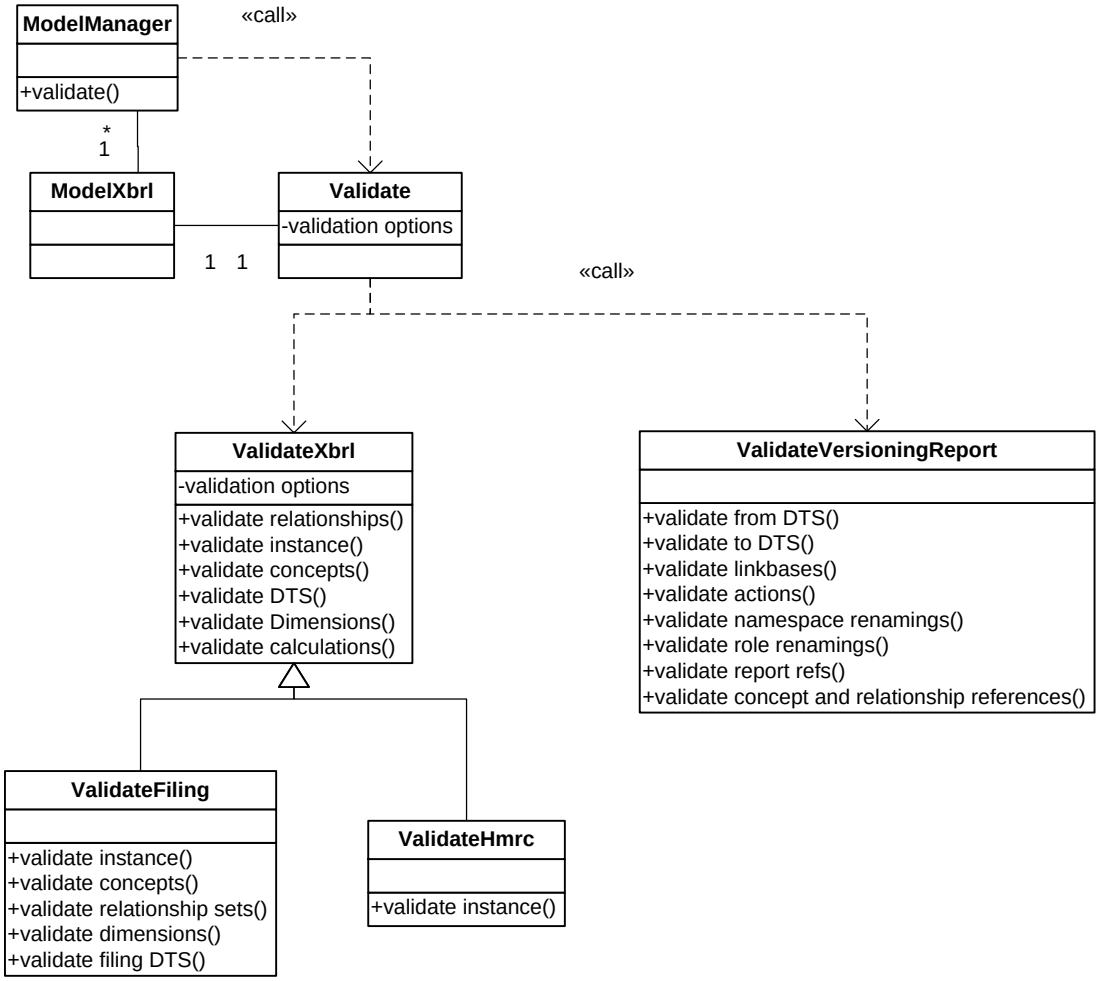
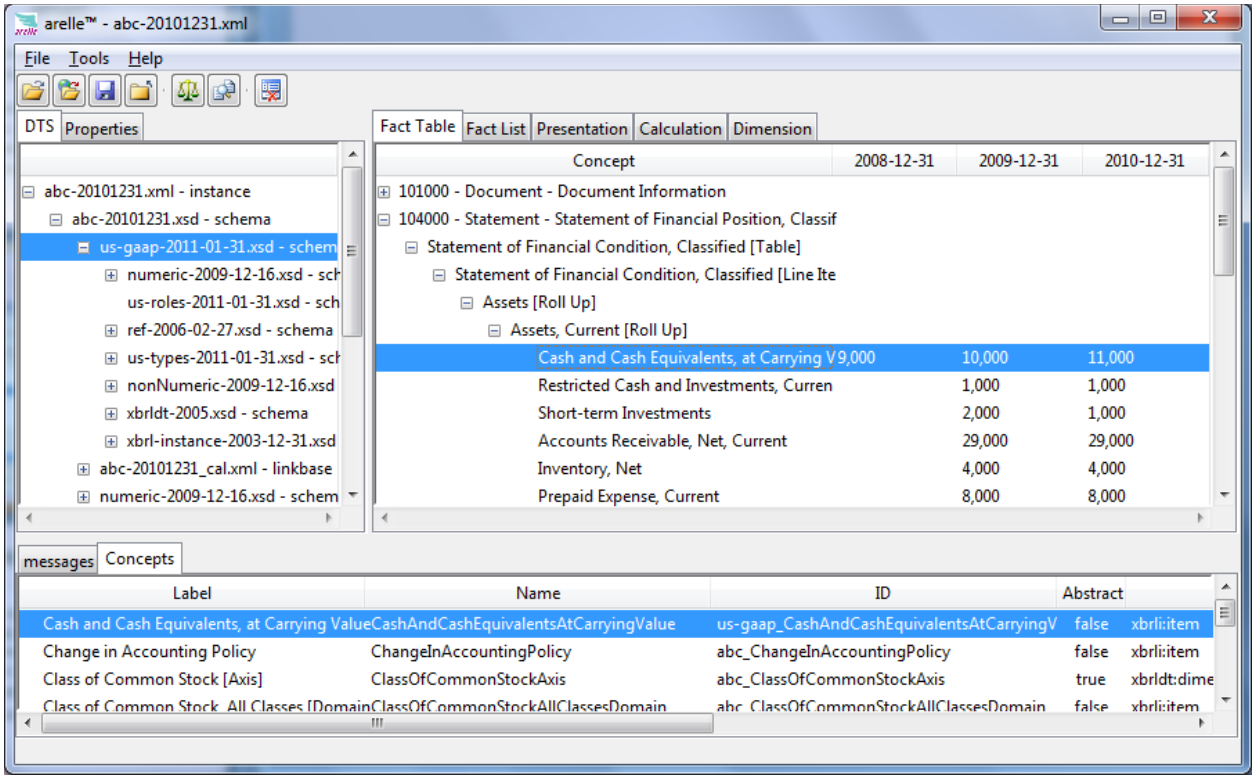


Figure 8. Validation class objects

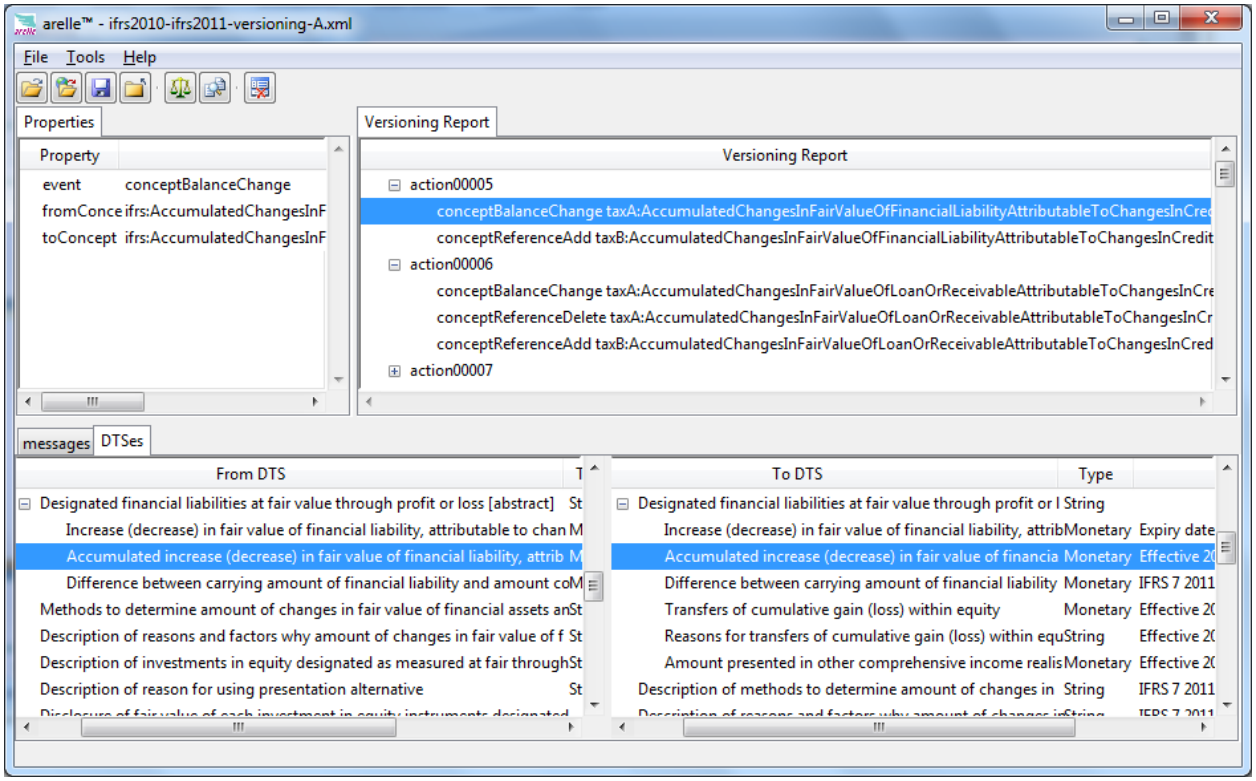
View

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 when time permits.

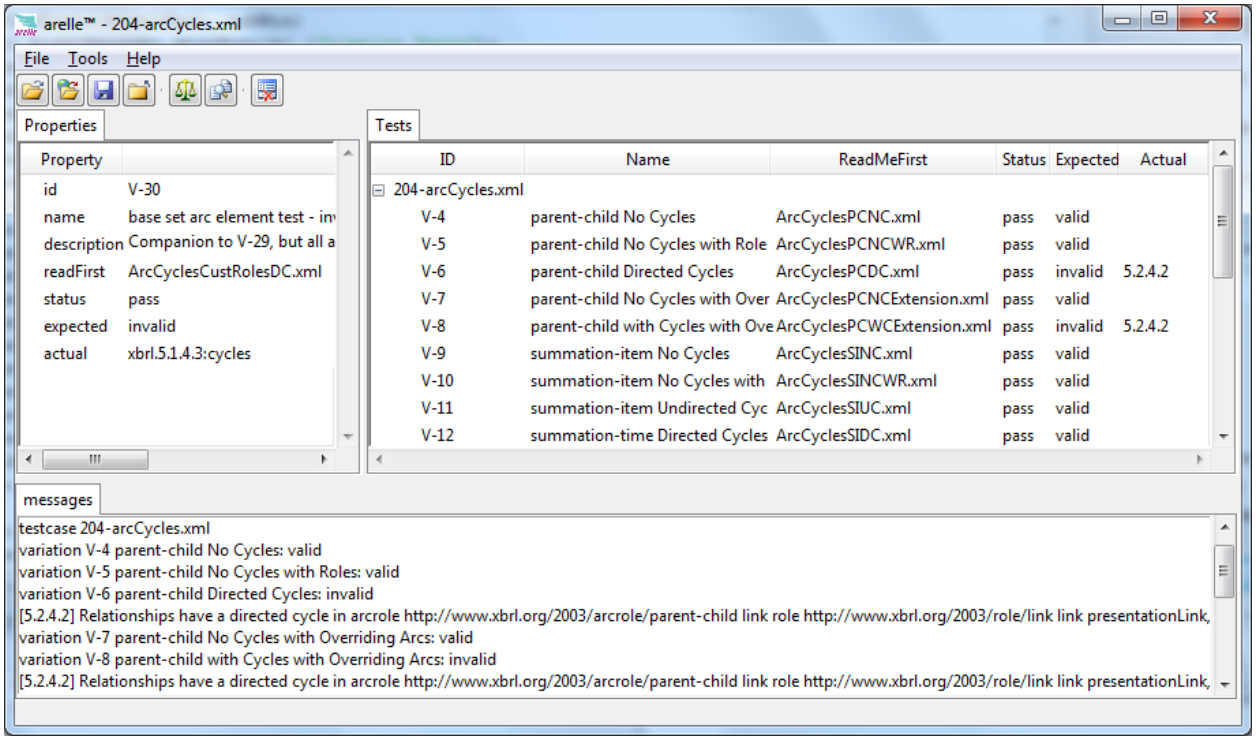
The typical instance and DTS views are synchronized for fact, relationship, concept, and other views of the DTS. Selection events of any one view synchronize others that present the same object:


arelle™ - abc-20101231.xml
File Tools Help
DTS Properties
abc-20101231.xml - instance
abc-20101231.xsd - schema
us-gaap-2011-01-31.xsd - schem
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 Fact List Presentation Calculation Dimension
Concept 2008-12-31 2009-12-31 2010-12-31
101000 - Document - Document Information
104000 - Statement - Statement of Financial Position, Classif
Statement of Financial Condition, Classified [Table]
Statement of Financial Condition, Classified [Line Ite
Assets [Roll Up]
Assets, Current [Roll Up]
Cash and Cash Equivalents, at Carrying V 9,000 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
messages Concepts
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

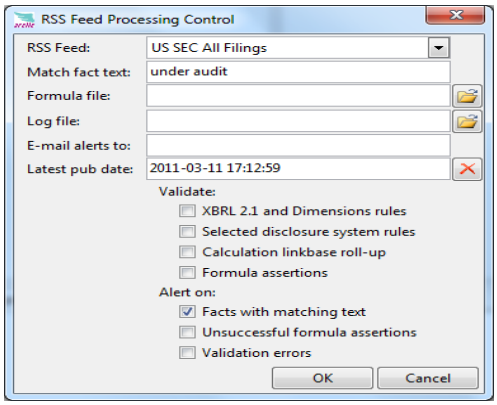
Versioning reports are entering use with the 2011 IFRS release. Here is a view of the 2010 and 2011 IFRS taxonomies with the actions of the versioning report interlinked to IFRS-style ("ITI") displays of the from and to DTSes. Full validation of the versioning report is available, and was helpful in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

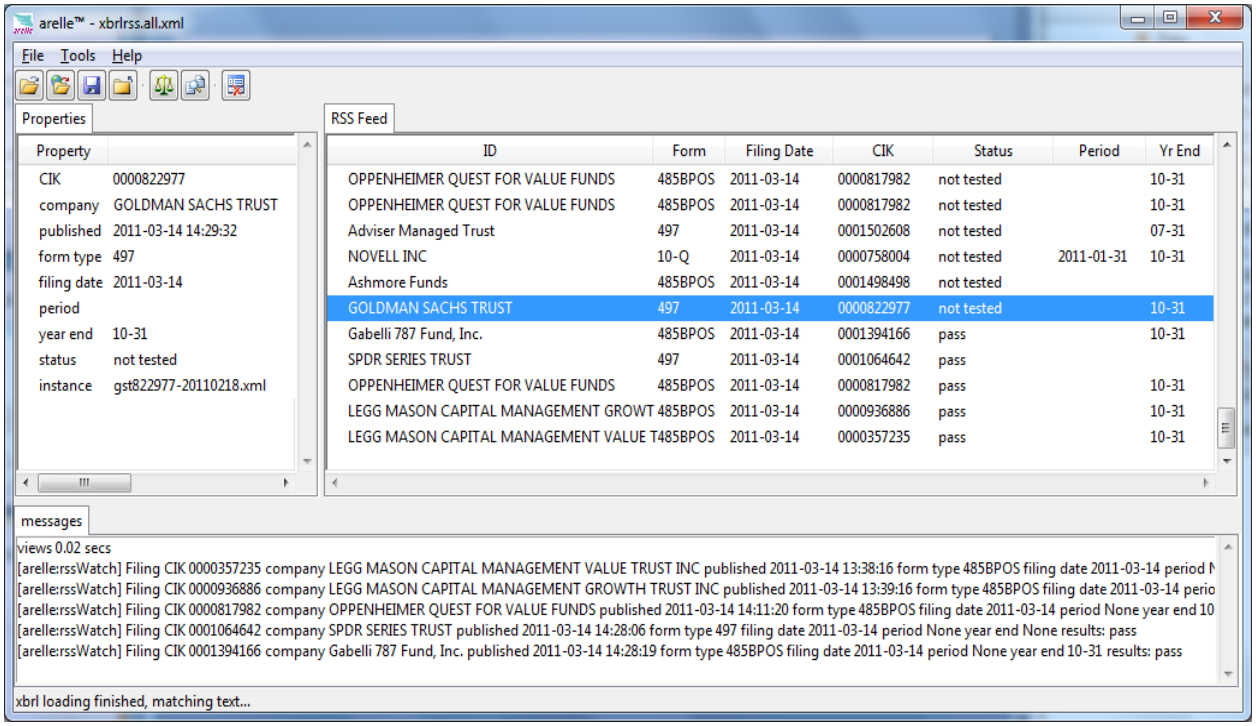


Several view features extend the test suite concept to training and exploration of XBRL formula. An RSS Watch facility allows the user to specify a regular expression or formula (as a set of assertions) to run against the SEC live RSS feeds. This can be started, resumed, or left to run in background.



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 lastest 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.

Arelle 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.

Explanation of the use of the command line and GUI interfaces is provided on the product web site and in contributed user manuals, and is deemed out of scope for this paper.

Cntlr.py: Cntlr is a superclass for a singleton conroller object that represents a running Arelle instance.
Methods

__init__(logFileName=None, logFileMode=None, logFileEncoding=None, logFormat=None)	Initialize new controller object. Initialize directories, system functions (such as clipboard, if provided), internal directory names, web cache, model manager, default logger (if logFileName provided).
addToLog(message, messageCode="", file="", sourceLine="")	Add a text message to log, with optional message code, reference to file and sourceLine.
showStatus(message, clearAfter=None)	For GUI controllers, a message to be shown in status line, to keep user aware of progress. If clearAfter (in milliseconds) is provided, remove message after time elapses.
close(saveConfig=False)	Close the controller, saving config changes if indicated.
Properties	
isMac	True if Macintosh (False for Microsoft Windows and linux)
isMSW	True if Microsoft Windows (False for Mac and linux)
config	A dictionary of user preferences and interactions to be 'pickled' (per python), such as screen positions, pane locations, validation preference settings, formula parameter settings.
userAppDir	Local directory for application (/libarary/Applications/Arelle on Mac, Users AppInfo/Arelle on Windows)
webCache	Web cache object for proxy and caching of web-accessed files
modelManager	Model manager object for interface to model objects
Logger	Default logger, if logFileName provided to initialization.

Comand-line controller use

The command-line controller provides a simple interface that could be used in a batch environment to extract facts from instances for processing by non-XBRL aware tools.

The example script scripts/exportCsvFromXbrlInstance.bat uses the Arelle module CntlrCmdLine to extract selected fields from an instance. The example which could be nested in a shell script for loop or equivalently called within the API, specifies:

```
%ARELLE% --file "%INSTANCEFILE%" --csvFactCols "Label unitRef Dec Value EntityScheme
EntityIdentifier Period Dimensions" --csvFacts "%OUTPUTCSVFILE%" 1>  "%OUTPUTLOGFILE%" 2>&1
```

The output here is in csv format, consisting of these possible fields (example noted as the string –csvFactCols):

- Label: The standard label for the fact's concept
- Name: The prefixed name of the fact's concept
- contextRef: The id of the fact's context, if any
- unitRef: The id of the fact's unit, if any
- Dec: The decimals attribute value, if provided for the fact
- Prec: The precision attribute value, if provided for the fact
- Lang: The xml:lang attribute value, if provided for the fact
- Value: The fact's value (text content of fact element)
- EntityScheme: The fact's context entity identifier (if an item)
- EntityIdentifier: The fact's context scheme (if an item)
- Period: The fact's period (instant date, or star and end dates, if an item)
- Dimensions: All non-default dimension and explicit member prefixed names.

The implementation of this function provides an API pattern for coding something more customized.

Custom controller API examples

The source code example, examples/ExampleLoadValidate.py, is a controller which initializes a default logger, loads an instance document (from which the DTS is discovered), and validates (with options to include calculation linkbase validation and do that by inferringDecimals). Here is a walk-through its source code:

```
# this is the controller class
class CntlrValidateExample(Cntlr.Cntlr):

    # init sets up the default controller for logging to a file (instead of
to standard output via terminal window)
    def __init__(self):
        # initialize superclass with default file logger
        super().__init__(logFileName="c:\\temp\\test-log.txt")

    def run(self):
        # create the modelXbrl by load instance and discover DTS
        modelXbrl = self.modelManager.load("c:\\temp\\test.xbrl")

        # select validation of calculation linkbase using infer decimals
option
        self.modelManager.validateInferDecimals = True
        self.modelManager.validateCalcLB = True

        # perfrom XBRL 2.1, dimensions, calculation
        self.modelManager.validate()

        # close the loaded instance
        self.modelManager.close()

        # close controller and application
        self.close()

# if python is initiated as a main program, start the controller
if __name__ == "__main__":
    # create the controller and start it running
    CntlrValidateExample().run()
```

The next source code example, examples/ExampleLoadEFMValidate.py, is a controller which extends the preceding example by initiating SEC Edgar Filer Manual (EFM) validation. (Other validation patterns, such as GFM, may also be selected, as with the GUI main program) The comments reflect only what differs from the preceding example:

```
class CntlrEfmValidateExample(Cntlr.Cntlr):

    def __init__(self):
        super().__init__(logFileName="c:\\temp\\test-log.txt)

    def run(self):
        # select SEC Edgar Filer Manual validation before validation (causes
file name and contents checking
        self.modelManager.validateDisclosureSystem = True
        self.modelManager.disclosureSystem.select("efm")

        modelXbrl = self.modelManager.load("c:\\temp\\test.xbrl")

        self.modelManager.validateInferDecimals = True
        self.modelManager.validateCalcLB = True
```

```
        # perfrom XBRL 2.1, dimensions, calculation and SEC EFM validation
        self.modelManager.validate()

        self.modelManager.close()

        self.close()

if __name__ == "__main__":
    CntlrEfmValidateExample().run()
```

The next source code example, examples/ExampleLoadSavePreLbCsv.py, is a controller which modifies the previous example to use the view for CSV file output, to show the relationship set as a tree, titled "Presentation", for the parent-child arcrole (all link roles):

```
class CntlrCsvPreLbExample(Cntlr.Cntlr):

    def __init__(self):
        super().__init__(logFileName="c:\\temp\\test-log.txt")

    def run(self):
        modelXbrl = self.modelManager.load("c:\\temp\\test.xbrl")

        # output presentation linkbase tree as a csv file
        viewRelationshipSet(modelXbrl, "c:\\temp\\test-pre.csv",
"Presentation", "http://www.xbrl.org/2003/arcrole/parent-child")

        self.modelManager.close()

        self.close()

if __name__ == "__main__":
    CntlrCsvPreLbExample().run()
```

The next source code example, examples/CustomLogger.py, is a controller which provides a log handler that can be modified to process log entries by the message string, name-value pairs, or as otherwise appropriate. In this version the message text is sent to standard output (terminal window):

```
class CntlrCustomLoggingExample(Cntlr.Cntlr):

    def __init__(self):
        # no logFileName parameter to prevent default logger from starting
        super().__init__()

    def run(self):
        # start custom logger
        CustomLogHandler(self)

        modelXbrl = self.modelManager.load("c:\\temp\\test.xbrl")

        self.modelManager.validateInferDecimals = True
        self.modelManager.validateCalcLB = True

        self.modelManager.validate()

        self.modelManager.close()

        self.close()

import logging
class CustomLogHandler(logging.Handler):
    def __init__(self, cntlr):
        logger = logging.getLogger("arelle")
```

```
        self.level = logging.DEBUG
        self.setFormatter(logging.Formatter("[%(messageCode)s] %(message)s -
%(file)s %(sourceLine)s"))
        logger.addHandler(self)

    def emit(self, logRecord):
        # just print to standard output (e.g., terminal window)
        print(self.format(logRecord))

if __name__ == "__main__":
    CntlrCustomLoggingExample().run()
```

Data Mining

Data mining will be explored prior to Comparison, because comparison builds on the examples learned from data mining. It is introduced by a set of mining patterns and examples how each may be implemented in Arelle.

(In this paper mining is conducted on XBRL instances, without a separate process of Extraction, Transformation, Database loading and Business Intelligence operations. The authors plan a future paper on database and business intelligence integration.)

Mining objectives

We provide examples to demonstrate three goals

1. Determining validation or validity of some aspect of instances
2. Determining which instances have some text content or meet some criteria (ratio over threshold)
3. Extraction of data to load analytic system (XBRL to database or analysis tool)

Further development of serious heavy-duty applications are likely to be programmatically controlled and long running (batch programs), however Arelle has GUI functionalities to scan collections of data that are:

1. Accessed by an RSS feed (SEC). Great when uniform access to external files is needed. Each examined RSS item is treated the same as every other.
2. Accessed by XBRL testcase variation files. Useful for local files where there are parameters unique to each variation, or specialized tests to perform on each.

API-based programmatic control is most likely to be used on large sets of data. We will examine

1. Command scripts (use of shell commands to control execution), may also support RPC callability in integrated environments
2. API programmed (use of Python coding to control execution). Can also be extended to server environments

Examination agents that will be considered are:

1. Validation sets (XBRL, dimensions, calculation linkbase, Filer Manual test suites)
2. CSV output of selected aspects (item, concept, period, dimensions)
3. Regular expressions for text matching of fact contents
4. Formula assertions to produce Boolean or messaged results

5. Formula output instances to provide translated, normalized, or derived facts
6. API coding to process each instance

RSS watch examples

The RSS watch is a GUI-based controller which can be used to assess:

- Validity
- Formula assertion results

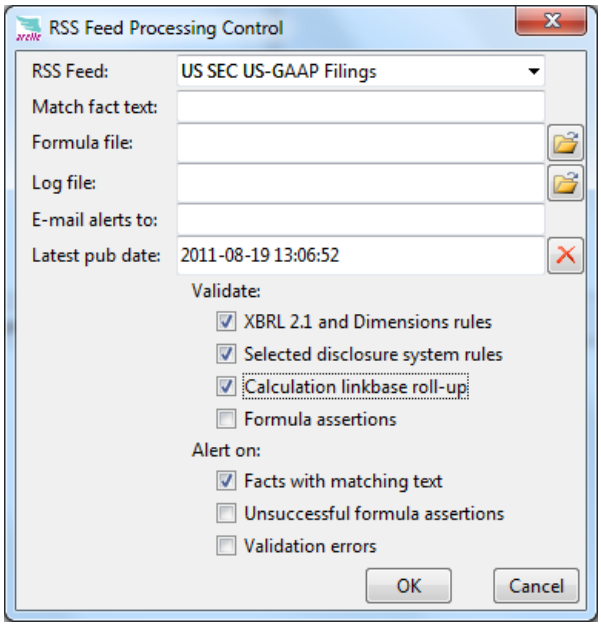
Because it is user interface driven, it is useful for demonstrations, but less likely to be used on enormous batches of data (those better for command line or API operation).

A user interface allows selection of options of what to watch for (validity, text matching, or formula assertions), and a scrolling window shows results of examining the feed items according to the options.

The RSS feed processing access filings by downloading the zipped submission, and the class FileSource can load xml without expanding the files, which provides reasonably fast operation despite the internet transfer. (It would be vastly slower to download the 'expanded' xml files, or to unzip and expand contents on the local file system after downloading.)

Determining RSS validity

The options dialog is accessed from the tools->RSS feed->options menu, or right click on an RSS items panel.

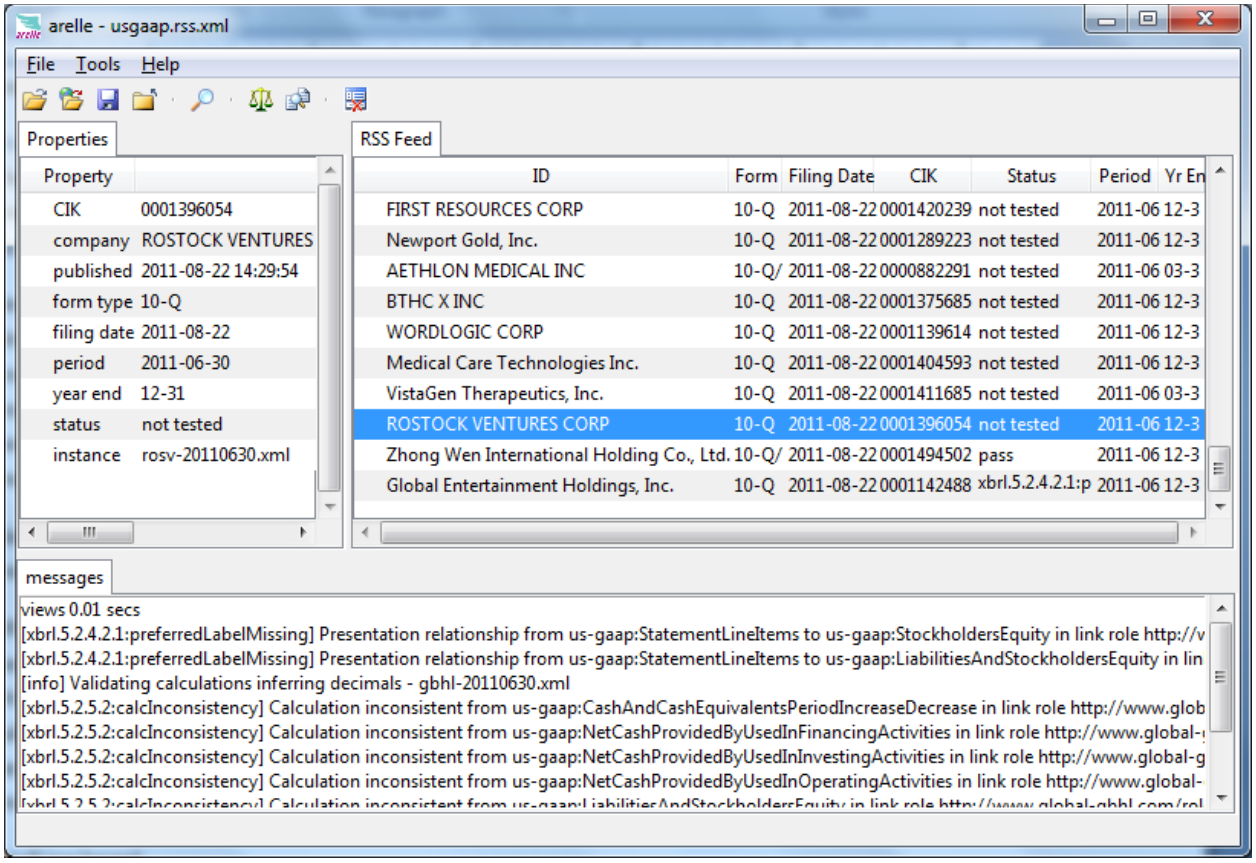


Here the options selected are XBRL 2.1, dimensions, disclosure system rules (SEC EFM was previously selected from the tool->validate menu), and calculation linkbase roll-up (prefer decimals was previously selected from the tools->validate menu).

Alert on facts matching text would generate an e-mail alert if one were specified.

When tools->RSS feed->start is selected, the RSS feed currently on the source website (the SEC in this case), is obtained, and any items whose publication date is newer than the entry in "Latest pub date" is validated/tested/examined. To cause all items to be reviewed regardless of last date entry, press the red X clear button on the "latest pub date" row.

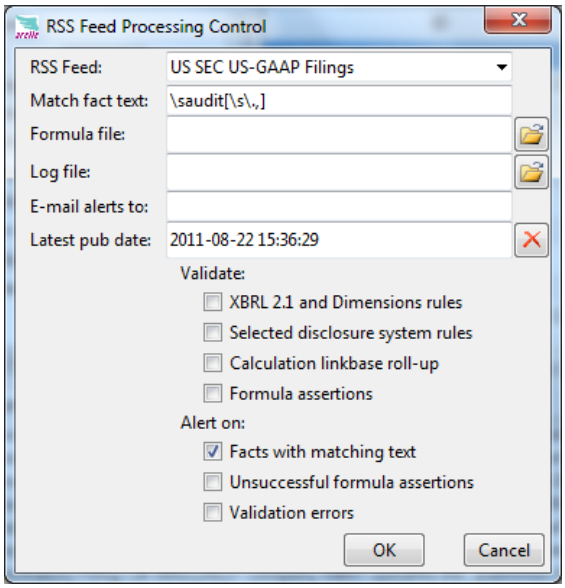
Here are the results of the above options (company filing and date visible, as well as validation issues noted):



The Global Entertainment Holdings has two preferred labels missing and calculation inconsistences, , and the Zhong Wen filing has no issues noted.

Capturing or saving the log file allows these items to be examined or later processed, whereas providing a custom log handler allows the message codes, file names, line numbers, XPath element-scheme identifier of the element, and error message parameters to be post processed.

Searching for text



In this next example, we search for a text string, "\saudit[\s\.,]", e.g., the word audit preceded by a whitespace and followed by whitespace, period, or comma. (The goal was to exclude matching unaudited or audited.) Each instance loaded has each fact value (text contents) checked, with the following API in the module WatchRSS.py.

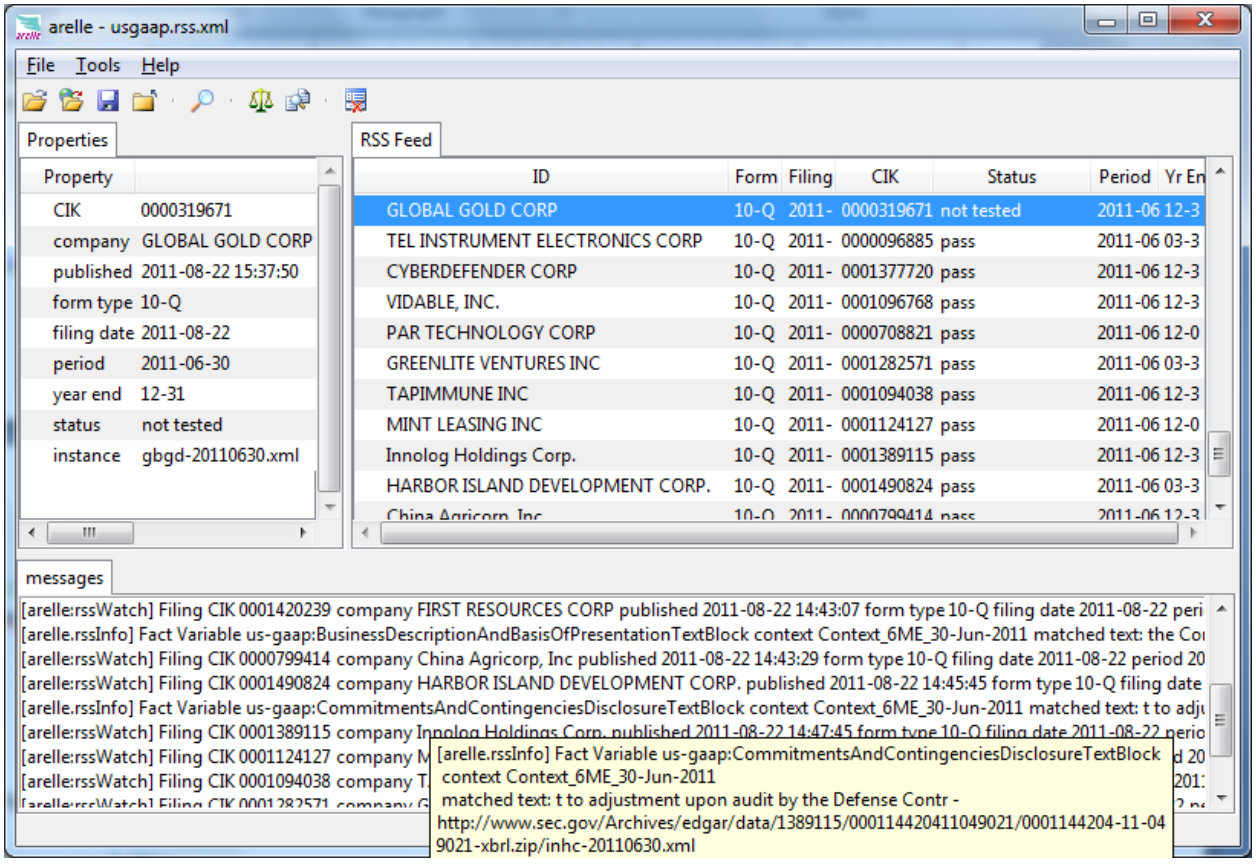
These API steps, for the loaded instance of "modelXbrl", check the value (text contents) of each fact item, and use the regular expression pattern search of Python. If there's a match, the from and to are used to display the characters which were matched to the regular expression:

```
for fact in  modelXbrl.factsInInstance:
    v = fact.value
    if v is not None:
        m = matchPattern.search(v)
        if m:
            fr, to = m.span()
```

```
            msg = _("Fact Variable {0}\n context {1}\n matched text:
{2}").format(
fact.qname, fact.contextID, v[max(0,fr-20):to+20])
            modelXbrl.info("arelle.rssInfo",
                           msg,
                           modelXbrl=modelXbrl)
```

(The same API could be used in any prior skeletal examples to accomplish the same search).

Here's a found match, note the tooltip at bottom of messages pane, where the mouse was hovering, for HARBOR ISLAND's 10-Q, where a contingencies disclosure noted adjustment upon audit by the Defense … was matched:

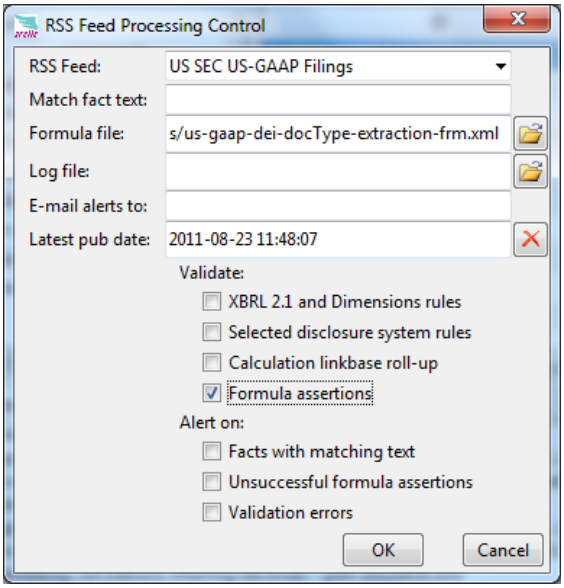


Searching by formula examples

Next we'll begin to develop formulas to match facts of interest.

Simple formula to extract form type used in each filing

Example formula linkbase examples\us-gaap-dei-docType-extraction-frm.xml simply provides an assertion message noting the form type of each filing. Here is the full formula linkbase:




```
<?xml version="1.0" encoding="UTF-8"?>
<!-- linkbase with prefix declarations used below,
including us-gaaps of 2008-2011 -->
<link:linkbase
  xmlns:link="http://www.xbrl.org/2003/linkbase"
  xmlns:xbrli="http://www.xbrl.org/2003/instance"
  xmlns:xlink="http://www.w3.org/1999/xlink"
  xmlns:generic="http://xbrl.org/2008/generic"
  xmlns:validation="http://xbrl.org/2008/validation"
  xmlns:va="http://xbrl.org/2008/assertion/value"
  xmlns:variable="http://xbrl.org/2008/variable"
  xmlns:cf="http://xbrl.org/2008/filter/concept"
  xmlns:gf="http://xbrl.org/2008/filter/general"
  xmlns:msg="http://xbrl.org/2010/message"

xmlns:xfi="http://www.xbrl.org/2008/function/instance"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:dei-08="http://xbrl.us/dei/2008-03-31"
  xmlns:dei-09="http://xbrl.us/dei/2009-01-31"
  xmlns:dei-11="http://xbrl.sec.gov/dei/2011-01-31"
  xmlns:xbrldt="http://xbrl.org/2005/xbrldt"
xsi:schemaLocation="
  http://www.xbrl.org/2003/linkbase http://www.xbrl.org/2003/xbrl-linkbase-2003-12-31.xsd
  http://xbrl.org/2008/generic http://www.xbrl.org/2008/generic-link.xsd
  http://xbrl.org/2008/assertion/value http://www.xbrl.org/2008/value-assertion.xsd
  http://xbrl.org/2008/variable http://www.xbrl.org/2008/variable.xsd
  http://xbrl.org/2008/filter/concept http://www.xbrl.org/2008/concept-filter.xsd
  http://xbrl.org/2008/filter/general http://www.xbrl.org/2008/general-filter.xsd
  http://xbrl.org/2010/message http://www.xbrl.org/2010/generic-message.xsd
  ">

  <!-- arcroleRef's for arcroles used in the example -->
  <link:arcroleRef arcroleURI="http://xbrl.org/arcrole/2008/variable-set"
xlink:href="http://www.xbrl.org/2008/variable.xsd#variable-set" xlink:type="simple"/>
  <link:arcroleRef arcroleURI="http://xbrl.org/arcrole/2008/variable-filter"
xlink:href="http://www.xbrl.org/2008/variable.xsd#variable-filter" xlink:type="simple"/>
  <link:arcroleRef arcroleURI="http://xbrl.org/arcrole/2010/assertion-satisfied-message"
xlink:href="http://www.xbrl.org/2010/validation-message.xsd#assertion-satisfied-message"
xlink:type="simple"/>
  <link:arcroleRef arcroleURI="http://xbrl.org/arcrole/2010/assertion-unsatisfied-message"
xlink:href="http://www.xbrl.org/2010/validation-message.xsd#assertion-unsatisfied-message"
xlink:type="simple"/>
  <link:roleRef roleURI="http://www.xbrl.org/2008/role/link"
xlink:href="http://www.xbrl.org/2008/generic-link.xsd#standard-link-role" xlink:type="simple"/>

  <generic:link xlink:type="extended"
    xlink:role="http://www.xbrl.org/2008/role/link">

    <!-- a value assertion that is always true, so that every filing will match
         and produce an assertion message with the DEI document type found -->
    <va:valueAssertion xlink:type="resource"
      xlink:label="extract-DEI-DocumentType"
      id="extract-Document-Type"
      test="true()"
      aspectModel="dimensional" implicitFiltering="true"/>

    <!-- the message, providing the DEI document type, for each filing that has one -->
    <msg:message xlink:type="resource" xlink:label="DEI-document-type-message"
```

```
      xlink:role="http://www.xbrl.org/2010/role/message"
      xml:lang="en">DEI document type is {
          $docType
      }</msg:message>

    <!-- arc from value assertion to the message -->
    <generic:arc xlink:type="arc" xlink:arcrole="http://xbrl.org/arcrole/2010/assertion-
satisfied-message"
      xlink:from="extract-DEI-DocumentType" xlink:to="DEI-document-type-message" order="1.0"/>

    <!-- arc from the value assertion to the variable which is the document type -->
    <variable:variableArc xlink:type="arc" xlink:arcrole="http://xbrl.org/arcrole/2008/variable-
set"
     name="docType" xlink:from="extract-DEI-DocumentType" xlink:to="vDocType"/>

    <!-- filter for document type, by concept name, and 'required context' (all dimensions
defaulted) -->
    <variable:variableFilterArc xlink:type="arc"
xlink:arcrole="http://xbrl.org/arcrole/2008/variable-filter"
        complement="false" cover="true" xlink:from="vDocType" xlink:to="fDocType"/>

    <variable:factVariable xlink:type="resource" bindAsSequence="false"
       xlink:label="vDocType"  />

    <!-- filter accepts 2009, 2009, or 2011 document type concept -->
    <cf:conceptName xlink:type="resource" xlink:label="fDocType">
       <cf:concept>
          <cf:qname>dei-08:DocumentType</cf:qname>
       </cf:concept>
       <cf:concept>
          <cf:qname>dei-09:DocumentType</cf:qname>
       </cf:concept>
       <cf:concept>
          <cf:qname>dei-11:DocumentType</cf:qname>
       </cf:concept>
    </cf:conceptName>

    <!-- Edgar Filer Manual's required context means all dimensions absent (defaulted, segment
must be empty -->
    <gf:general xlink:type="resource" xlink:label="fDocType"
     test="empty( xfi:segment(.) )" />

  </generic:link>
</link:linkbase>
```

What is interesting to note is that the document type concept may be of several taxonomy namespace years (2008, 9, and 11), and the concept name filter allows listing these alternatives. The general filter is a trick to assure that only the concept which has no dimensions (segment being empty) is to be considered. This is needed in most SEC filings to exclude matching facts where non-defaulted dimensions are provided (and use of the dimension filter for this is problematical, because it would require us to list all dimensions that might be used, but the list of dimensions varies from filer to filer).

The results of running this are a set of assertion messages (the extraction message appears above the rssWatch filing that notes the results):

```
[message:extract-Document-Type] DEI document type is 10-Q - examples/us-gaap-dei-docType-
extraction-frm.xml 45
 [arelle:rssWatch] Filing CIK 0001120096
 company Sino Clean Energy Inc
 published 2011-08-23 06:41:59
 form type 10-Q/A
 filing date 2011-08-23
 period 2011-06-30
 year end 12-26
 results: pass - http://www.sec.gov/Archives/edgar/usgaap.rss.xml
```

```
 [message:extract-Document-Type] DEI document type is 10-Q - examples/us-gaap-dei-docType-
extraction-frm.xml 45
[arelle:rssWatch] Filing CIK 0000869484
 company CALL NOW INC
 published 2011-08-23 09:02:11
 form type 10-Q/A
 filing date 2011-08-23
 period 2011-06-30
 year end 12-31
 results: pass - http://www.sec.gov/Archives/edgar/usgaap.rss.xml
```

The formula working group has suggested structured messages, in which case the form type could be output as a name-value pair to the python logging function, which could be captured by a custom logger at a later date.

Ratios formula to extract ratios from each 10-K/10-Q filing

The most thorough analysis of XBRL ratios for SEC filings is in (Debreceny, et al, 2010). The paper analyzes the likelihood that SEC filers have used well known concept names that can be used to identify concepts of ratios, or have provided extension concepts of their own choosing. In the case of home-made concepts, the calculation linkbase ancestry and descendancy can be often used to infer how the filer chose their own concept for the expected ratio term. Such detective work could be done in a formula, or in ordinary Python code, but we'll begin with simple looking for terms that high percentages of filers report.

The example in examples/us-gaap-ratio-cash-frm.xml looks for these ratios:

- Cash ratio, reported by 97% of filers
  - (Cash+Marketable Securities)/Current Liabilities
- Current ratio, " 98%
  - Current Assets/Current Liabilities

In the following extracts of this formula linkbase, the message construct provides the cash ration and the current ratio. In both cases, since the divisor, currentLiabilities may not be found (probably because it was reported by some extension or us-gaap concept not noted in the currentLiabilities filter), then instead of raising a divide-by-zero error, *not available* is shown.

The dei document type fact is used to establish the 'required context' (that with no dimensions, and only binds for a 10-K or 10-Q, due to this function in the general filter:

```
index-of(('10-K', '10-Q'), .)
```

If it is bound, then it's unit aspect is covered by the aspect cover filter, so that the rest of the variables, which are monetary, can be implicitly filtered:

```
<acf:aspectCover xlink:type="resource" xlink:label="fDocType">
  <acf:aspect>unit</acf:aspect>
</acf:aspectCover>
```

Each of the monetary terms is an instant, which must be matched to the end of the document type's duration context, thus the period filter to match the end boundary of document type's duration:

```
<pf:instantDuration xlink:type="resource" xlink:label="fCash"
  boundary="end"
  variable="docType" />
```

The assertion, fact variables, and filters of this example are:

```
<va:valueAssertion xlink:type="resource"
  xlink:label="cash-ratio"
```

```
      id="cash-ratio"
      test="true()"
      aspectModel="dimensional" implicitFiltering="true"/>

    <msg:message xlink:type="resource" xlink:label="cash-ratio-message"
      xlink:role="http://www.xbrl.org/2010/role/message"
      xml:lang="en">Cash ratio {
          if ($currentLiabilities ne 0) then
             (($cash + $marketableSecurities) div $currentLiabilities)
          else
             'not available'
      }, Current ratio {
          if ($currentLiabilities ne 0) then
             ($currentAssets div $currentLiabilities)
          else
             'not available'

      }</msg:message>

    <generic:arc xlink:type="arc" xlink:arcrole="http://xbrl.org/arcrole/2010/assertion-
satisfied-message"
      xlink:from="cash-ratio" xlink:to="cash-ratio-message" order="1.0"/>

    <!-- fact variable, arcs, and filters for document type, restricted to 10-K or 10-Q -->
    <variable:factVariable xlink:type="resource" bindAsSequence="false"
       xlink:label="vDocType"  />
    <variable:variableArc xlink:type="arc" xlink:arcrole="http://xbrl.org/arcrole/2008/variable-
set"
     name="docType" xlink:from="cash-ratio" xlink:to="vDocType"/>
    <variable:variableFilterArc xlink:type="arc"
xlink:arcrole="http://xbrl.org/arcrole/2008/variable-filter"
        complement="false" cover="true" xlink:from="vDocType" xlink:to="fDocType"/>
    <cf:conceptName xlink:type="resource" xlink:label="fDocType">
       <cf:concept>
          <cf:qname>dei-11:DocumentType</cf:qname>
       </cf:concept>
    </cf:conceptName>

    <!-- Edgar Filer Manual required context means all dimensions absent (defaulted)
         and only bind for a 10-K/10-Q filing (don't bind for other form types) -->
    <gf:general xlink:type="resource" xlink:label="fDocType"
     test="empty( xfi:segment(.) ) and index-of(('10-K', '10-Q'), .) " />

    <!-- must cover unit because this variable is a string type -->
    <acf:aspectCover xlink:type="resource" xlink:label="fDocType">
      <acf:aspect>unit</acf:aspect>
    </acf:aspectCover>

    <!-- cash -->
    <variable:factVariable xlink:type="resource" bindAsSequence="false"
       xlink:label="vCash"  />
    <variable:variableArc xlink:type="arc" xlink:arcrole="http://xbrl.org/arcrole/2008/variable-
set"
     name="cash" xlink:from="cash-ratio" xlink:to="vCash"/>
    <variable:variableFilterArc xlink:type="arc"
xlink:arcrole="http://xbrl.org/arcrole/2008/variable-filter"
        complement="false" cover="true" xlink:from="vCash" xlink:to="fCash"/>
    <cf:conceptName xlink:type="resource" xlink:label="fCash">
       <cf:concept>
          <cf:qname>us-gaap-11:Cash</cf:qname>
       </cf:concept>
       <cf:concept>
          <cf:qname>us-gaap-11:CashAndCashEquivalentsAtCarryingValue</cf:qname>
       </cf:concept>
    </cf:conceptName>
    <!--  Must match period-end of documentType period -->
    <pf:instantDuration xlink:type="resource" xlink:label="fCash" boundary="end"
variable="docType" />

    <!-- marketable securities -->
```

```
    <variable:factVariable xlink:type="resource" bindAsSequence="false" fallbackValue="0"
       xlink:label="vMarketableSecurities"  />
    <variable:variableArc xlink:type="arc" xlink:arcrole="http://xbrl.org/arcrole/2008/variable-
set"
     name="marketableSecurities" xlink:from="cash-ratio" xlink:to="vMarketableSecurities"/>
    <variable:variableFilterArc xlink:type="arc"
xlink:arcrole="http://xbrl.org/arcrole/2008/variable-filter"
        complement="false" cover="true" xlink:from="vMarketableSecurities"
xlink:to="fMarketableSecurities"/>
    <cf:conceptName xlink:type="resource" xlink:label="fMarketableSecurities">
       <cf:concept>
          <cf:qname>us-gaap-11:MarketableSecuritiesCurrent</cf:qname>
       </cf:concept>
    </cf:conceptName>
    <pf:instantDuration xlink:type="resource" xlink:label="fCash" boundary="end"
variable="docType" />

    <!-- current liabilities -->
    <variable:factVariable xlink:type="resource" bindAsSequence="false" fallbackValue="0"
       xlink:label="vCurrentLiabilities"  />
    <variable:variableArc xlink:type="arc" xlink:arcrole="http://xbrl.org/arcrole/2008/variable-
set"
     name="currentLiabilities" xlink:from="cash-ratio" xlink:to="vCurrentLiabilities"/>
    <variable:variableFilterArc xlink:type="arc"
xlink:arcrole="http://xbrl.org/arcrole/2008/variable-filter"
        complement="false" cover="true" xlink:from="vCurrentLiabilities"
xlink:to="fCurrentLiabilities"/>
    <cf:conceptName xlink:type="resource" xlink:label="fCurrentLiabilities">
       <cf:concept>
          <cf:qname>us-gaap-11:LiabilitiesCurrent</cf:qname>
       </cf:concept>
    </cf:conceptName>
    <pf:instantDuration xlink:type="resource" xlink:label="fCurrentLiabilities" boundary="end"
variable="docType" />

    <!-- current assets -->
    <variable:factVariable xlink:type="resource" bindAsSequence="false" fallbackValue="0"
       xlink:label="vCurrentAssets"  />
    <variable:variableArc xlink:type="arc" xlink:arcrole="http://xbrl.org/arcrole/2008/variable-
set"
     name="currentAssets" xlink:from="cash-ratio" xlink:to="vCurrentAssets"/>
    <variable:variableFilterArc xlink:type="arc"
xlink:arcrole="http://xbrl.org/arcrole/2008/variable-filter"
        complement="false" cover="true" xlink:from="vCurrentAssets" xlink:to="fCurrentAssets"/>
    <cf:conceptName xlink:type="resource" xlink:label="fCurrentAssets">
       <cf:concept>
          <cf:qname>us-gaap-11:AssetsCurrent</cf:qname>
       </cf:concept>
    </cf:conceptName>
    <pf:instantDuration xlink:type="resource" xlink:label="fCurrentAssets" boundary="end"
variable="docType" />
```

The results of running this assertion set are shown here for one filing

```
[message:cash-ratio] Cash ratio 0.66, Current ratio 1.87 - examples/us-gaap-dei-ratio-cash-
frm.xml 45
[arelle:rssWatch] Filing CIK 0001089063
 company DICKS SPORTING GOODS INC
 published 2011-08-24 13:49:40
 form type 10-Q
 filing date 2011-08-24
 period 2011-07-30
 year end None
```

Output instance approach of using formula linkbase

These formula linkbase examples are shown in the form of assertions that create output messages. The messages may be paramterized, structured, and processed by a logging system for capture.

An alternative use of formula linkbase is to produce an output instance document for each formula processed. Each of the extracted values and computed ratios could be entered to such an output instance designed to capture facts of multiple submissions. In such a case the output instance could have a compact taxonomy of just the extracted concepts and a concept for each ratio, allowing facts of many instance documents to be aggregated together in a simple output instance. The facts of each extraction could reflect the source in the filing entity and document type period, or have some other scheme. Arelle has the needed functionality to implement such an approach, either as a future addition to the RSS watch feature, or as an API based Controller module.

Finding the un-expected concepts for standard terms in filer extension taxonomies

In (Debreceney, et al. 2010, section "Alternative Elements"), a set of patterns is described to search for a semantic match, particularly due to filer's choices of concepts that may not represent the expected us-gaap concept, either due to choice of a more-specific us-gaap concept, or providing an extension taxonomy concept. These patterns suggest checking the parent and/or child calculation elements in attempt to identify the desired element.

Formula linkbase provides a concept relation filter and a set of corresponding functions that may be used to construct custom functions for these patterns. (There was not sufficient time in preparing this paper to develop examples.)

Arelle's python API can likewise be used for the same purpose, if the data mining engineer were to implement the task in the API instead of the formula linkbase. If a set of graph patterns were pre-determined for the alternative element search, Python's set-algebra features may provide very fast execution of such an algorithm.

Normalizing mined data for comparability

This section has used a stepwise approach to mining, one instance at a time, without regard for how the results were comparable to another instance. The goal of data mining may be to prepare for analytics that must relate the mined ratios and numbers to other instances, of possibly different reporting periods, and of certainly different filing taxonomies. The next section addresses initial ideas of comparability.

Comparability

As this paper is being written the XBRL Comparability Task Force is engaged in developing a framework for comparability, which will, when available, enhance the ideas presented here.

Direct comparison of small sets of instances

If a comparison operation is specific to a small number of instances, and the extracted and derived terms to be obtained are well known, it may be appropriate to build a formula linkbase for this purpose.

The multi-instance capability of formula linkbase allows a number of instance documents, each with their own and possibly independent DTS, to be processed together by assertions and formulas that generate output instances.

A variable and its filters may specify the source of its inputs as one or more instances. This allows either accepting multiple values for one variable from different instances, or coding multiple variables to each accept one term from one instance. If, for example, averaging a fact of a concept known to stably exist over a set of periods, with compatible dimensions (or none), one variable could obtain a single sequence from all input instances. On the other hand, if implicitly aligning periods, dimension values, and entities and units, were difficult, it may be more appropriate to use separate variables for each source instance.

Hand coding such a formula linkbase may be reasonable for a well known and small set of data (such as the referenced Debreceny 2010 ratios), but not possible for a large set of data (to provide general investor comparability of an entire financial report).

Automated comparison of large sets of instances

Each filing in a large set of comparison candidates will need to be evaluated, ranked in suitability for mining of the desired terms, and then populated to an analytical toolset (such as found in BW/BI).

Normalization may need to adjust concepts even in same filing (extensions, choice of concept), use of relationships to infer semantic meaning of names. The ancestor/descendant patterns previously referenced may be used to create metrics of normalizability for each comparison candidate.

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.

References

Fischer, H., and Mueller, D. 2011. Open Source & XBRL: the Arelle® Project, 2011 Kansas University XBRL Conference, April 29-30 2011, Overland Park, Kansas. Avalable from: http://web.ku.edu/~eycarat/myssi/_pdf/4-Fischer%20-Mueller-Open-source-ArelleProject.pdf

Debreceney, R., Alessandro, d'E., Felden, C., Farewell, S., and Piechocki M. 2011. Feeding the Information Value Chain: Deriving Analytical Ratios from XBRL filings to the SEC, 2011 Kansas University XBRL Conference, April 29-30 2011, Overland Park, Kansas. Avalable from: http://web.ku.edu/~eycarat/myssi/_pdf/2-Debreceny-XBRL%20Ratios%2020101213.pdf