# XML VIEWS, PART III
## *An UML Based Design Methodology for XML Views*

Rajugan R., Tharam S. Dillon
*Faculty of Information Technology, University of Technology, Sydney, Australia*
*Email:{rajugan, tharam}@it.uts.edu.au*

Elizabeth Chang
*School of Information Systems, Curtin University of Technology, Australia*
*Email: Elizabeth.Chang@cbs.curtin.edu.au*

Ling Feng
*Faculty of Computer Science, University of Twente, The Netherlands*
*Email: ling@ewi.utwente.nl*

Keywords: OO conceptual models, XML views, Conceptual views, XML, UML, XML Schema

Abstract: Object-Oriented (OO) conceptual models have the power in describing and modelling real-world data semantics and their inter-relationships in a form that is precise and comprehensible to users. Today UML has established itself as the language of choice for modelling complex enterprises information systems (EIS) using OO techniques. Conversely, the eXtensible Markup Language (XML) is fast emerging as the dominant standard for storing, describing and interchanging data among various enterprises systems and databases. With the introduction of XML Schema, which provides rich facilities for constraining and defining XML content, XML provides the ideal platform and the flexibility for capturing and representing complex enterprise data formats. Yet, UML provides insufficient modelling constructs for utilising XML schema based data description and constraints, while XML Schema lacks the ability to provide higher levels of abstraction (such as conceptual models) that are easily understood by humans. Therefore to enable efficient business application development of large-scale enterprise systems, we need UML like models with rich XML schema like semantics. To address such issue, in this paper, we proposed a generic, semantically rich view mechanism to conceptually model and design (using UML) XML domains to support data modelling of complex domains such as data warehousing and e-commerce systems. Our approach is based on UML and UML stereotypes to design and transform XML views.

## 1 INTRODUCTION

In software engineering, many methodologies have been proposed to capture real-world problems into manageable segments, which can be communicated, modelled and developed into error-free maintainable software modules/systems. Similarly, in the case of data models, the main objective of conceptual data models is to define real-world objects and their relationships in such a way that they represent meaningful units of information with respect to the semantics of the domain in question (Jorge H. Doorn, C. Rivero, & (eds), 2002). These models span from early data centred models (e.g. ER/DFD) to the modern Object-Oriented (OO) models, where a software system is modelled at varying levels of abstractions, namely conceptual, logical and physical levels. Here the conceptual level being the highest level of abstraction (being close to the real-world), while the physical level being the data/programming modules (being close to the actual system and implementation specific).

Therefore, in building a well defined blue-print of a software system, it is essential that, for a given set of data objects, we capture all feasible contexts for the data as possible. This is because all software systems, during their lifetime provides not just one, but many perspectives of the data that they transact or store. Thus it is imperative that we cater for such demand in early stages of the system development, such as the conceptual model. In all conceptual

models, there exist one or more constructs to capture data objects and their inter-relationships such as; (1) in ER, entities and relationships, (2) EER entities and enhanced relationship and (3) in OO classes, structural and role relationships (UML), but no mechanisms to capture transitive and/or dynamic data perspectives (views).

In database systems, views are persistent (Jacek Blazewicz, Wieslaw Kubiak, Tadeusz Morzy, Rusinkiewicz, & (eds), 2003); that is to say, view definitions are stored unless they are changed or deleted. In classical data oriented systems, views are initially used to provide access control to the underlying stored data. Later views, in addition to user-access control, are used as a short-hard for complex queries or frequently used user queries and to complement various Application Programming Interfaces (APIs) in the relational model. Thus they form the external schema of the three-schema architecture (or ANSI/SPARC architecture [Tsichritzis & Klug 1978]); (1) the conceptual schema, (2) the storage/internal schema, and (3) the external schema of the relational models (Kim & Kelly, 1995). Due to these implications, relational view definitions are visible only at the lower levels of the system development lifecycle and/or at the operational phase of the systems. Thus, design of views are normally left to database programmers and done without consideration for other system aspects such as flexibility, change and/or re-use. But with new realizations for views in complex domains (such as web, data warehousing/OLAP, ERP and e-Commerce), coupled with new data models/standards available (such as OO-DBMS (Abiteboul & Bonner, 1991; Dillon & Tan, 1993; Kim & Kelly, 1995)), the demand for a well defined and maintained view mechanism has increased. Also, with the introduction of XML (W3C-XML, 2004) (semi-structured data), the requirement for a view mechanism has changed, as an XML view mechanism has to deal with both its structure and data (unlike in structured data, where data is independent of its structure and depended on its data model) (Abiteboul, 1999; Rajugan R., Chang, Dillon, & Ling, 2003).

Since its introduction in 1996, XML has become an increasingly important data format for both data-centric and document-centric applications. This includes semi-structured data (web applications) and traditional structured data (legacy, database applications) intended for dissemination, manipulation and publication among both homogenous and heterogeneous systems. An XML document contains a non-scalar, set-based hierarchical document tree with interconnected nodes (branches) hosting special instructions (such as entities, relationship, constraints etc) called tags (defined by users), which enclose identifiable parts of the document (Renguo Xiaou, Tharam S. Dillon, Elizabeth Chang, & Ling Feng, 2001a). Thus XML is said to be self-describing and since it separates data from presentation (unlike HTML) it is reusable. With the Introduction of DTD and later its replacement XML Schema (W3C-XSD, 2004), XML provides a flexible yet powerful data model.

The rest of this paper is organized as follows. Section 1.1 briefly looks at some early view models, definitions, conceptual data models and XML while, section 1.2 outlines our own work done in this areas followed by a motivating case study description (Section 1.3). Section 2 and 3 provide a detailed discussion on our XML view concepts, definitions and modeling issues, while Section 4 provides a discussion on XML view hierarchy. In section 5, we highlight some real-world application scenarios that use the XML view methodology. This is followed by section 6 which concludes this paper with some discussion on future research directions.

## 1.1 Related Work

Today motivation for views include; (1) user access (Elmasri & Navathe, 2000), (2) user perspectives/profiles (E. Chang & Dillon, 1994; E. J. Chang, 1996; Rajugan R. et al., 2003), (3) data perspectives (Abiteboul, Goldman, McHugh, Vassalos, & Zhuge, 1997; Elmasri & Navathe, 2000), (4) performance (materialised views in Data Warehouse/OLAP, web-data cache), (5) web portals & profiles, (6) dimensional data modelling (Lucie-Xyleme, 2001; Vicky Nassis, Rajugan, Dillon, & Rahayu, 2004; Vicky Nassis, Rajugan R., Dillon, & Rahayu, 2005) and (7) sub-ontology or ontology views (Volz, Oberle, & Studer, 2003a, 2003b). Though the usefulness of views are realized more than their originally intended use (user access control), and extensive research have been carried out by both researchers and industry to improve their design, construction and performance, the view concept is still a data language and model dependent, lower level construct (implementation). Here we first briefly look at history of the view mechanisms available today and some of the proposals for an XML view mechanism.

The relational (classical) definition of a view is based on ANSI/SPAC three-schema architecture, where a view is treated as a virtual relation, constructed by a query which is executed on one or more stored relations (Elmasri & Navathe, 2000). Later the concept of view was extended to support complex queries and/or aggregate/summary queries. During the OO revolution, the relational view definitions were extended to OO data models by

Won Kim et al. (Kim, 1990; Kim & Kelly, 1995), Abiteboul et al. (Abiteboul & Bonner, 1991), and Chang (E. J. Chang, 1996). Here the views were defined in a synonymous manner to the relational model and/or extending the relational definition (Kim, 1990), when a needed. They included the idea of the virtual class. Both relational and OO view concepts make two implicit assumptions; that the underlying data is structured and there exists a fixed data model and a data access/query language. But only Chang et al. allows some form of abstraction at a higher level, a view definition in the form of Abstract views (E. J. Chang, 1996). All other view definitions are defined at the data manipulation language level. This we argue is not enough to provide a real-world scenario and/or abstraction to complex domain. We argue that, providing view formalism at the conceptual level will improve the resulting view implementation, similar to a conceptual model of a software system.

Since the emergence of the Internet and XML, the need for semi-structured data models, which have to be independent of fixed data models and data access, violates fundamental properties of persistent data models. Many researchers attempted to solve these issues by using graph based (Zhuge & Garcia-Molina, 1998) and/or semi-structured data models (Abiteboul et al., 1997; Liefke & Davidson, 2000). Again, the actual view definitions are only available at the lower level of the implementation and not at the conceptual level. One of the early discussions on XML view was by Serge Abiteboul (Abiteboul, 1999) and later more formally by Sophie Cluet et al. (Cluet, Veltri, & Vodislav, 2001). They proposed a declarative notion of XML views.

Abiteboul et al. pointed out that, a view for XML, unlike classical views, should do more than just providing different presentation of underlying data (Abiteboul, 1999). This, he argues, arises mainly due to the nature (semi-structured) and the usage (primarily as common data model for heterogeneous data on the web) of XML. Also he argues that, an XML view specification should rely on a data model (like ODMG model) and a query language. In the paper (Cluet et al., 2001), they discuss in detail on how abstract paths/DTDs are mapped to concrete paths/DTDs. These concepts, which are implemented in the Xyleme project (Lucie-Xyleme, 2001; Xyleme, 2001), provide one of the most comprehensive mechanisms to construct an XML view to-date. The Xyleme project uses an extension of ODMG Object Query Language (OQL) to implement such an XML view. But, in relation to conceptual modeling, these view concepts provide no support. The view formalism is derived from the instantiated XML documents (instant level) and is associated with DTD in comparison to flexible XML Schema. Also, the Xyleme view concept is mainly focused on web based XML data.

To our knowledge, other than our work, there exists no research direction that explores the possibility of utilizing the view concept for conceptual modeling. Though we mainly focus on native XML data/document, our notation of conceptual views and XML views (Rajugan R. et al., 2003; Rajugan R., Chang, Feng, & Dillon, 2004) can be mapped to any existing data models that provide XML support, since it is defined at the higher level of abstraction, the conceptual level. Since the view definitions are not available at a higher level, it is a time consuming effort to reflect an errors and/or changes at the schema level of the domain to the resulting view definitions as all resulting view definitions have to re-written.

## 1.2 Our Work

Our work described in this paper includes; (1) enable modeling *conceptual views* using UML (at the conceptual level) and (2) map conceptual views to *XML view* schemas (at the logical level). First we propose an abstract notion of conceptual views based on UML. Secondly we map these conceptual views (in UML) to XML Views (XML Schema, at the logical model/schema level). Due to its abstract nature, XML conceptual views can be captured using any high level modeling languages such as Dillon & Tan notation (Dillon & Tan, 1993), UML (OMG-UML™, 2003), XMSemantic Nets (Rajugan R. et al., 2004) or Enhanced-ER (Enhanced or Extended Entity-Relationship Model (EER)) (Elmasri & Navathe, 2000) models.

In the paper (Feng, Chang, & Dillon, 2002), authors demonstrate how OO concepts can be captured in semantic network based modeling language in regards to XML domains and we have extended that work in (Rajugan R. et al., 2004) to model XML conceptual views. In this paper, we adopt OMG's UML as UML has established itself as the de-facto modelling langue of choice. UML provides a well defined rich collection of tools to model a given domain into needed level of abstraction. It can be said that, UML helps to provide a well-defined blue print for a software system that is easily understood both by users and developers alike. UML also provides extensibility to the modelling language in the form of *stereotypes* which we utilise in defining our *conceptual views* (discussed in Section 2). Another reason we adopt UML is that, many authors (Conrad et al.) including authors of the papers (Feng, Chang, & Dillon, 2003; Xiaou et al., 2001a; Renguo Xiaou, Tharam S Dillon, Elizabeth Chang, & Ling Feng, 2001b) have

intuitively shown mapping UML models to XML Schema, which we adopt as the mapping formalism (with some extensions) between our XML conceptual views (in UML) to XML views (XML Schema).

## 1.3 An Example Case-Study

As a motivating example/ case study, we use in this paper is a simple Conference System (CS). A conference `Paper` consists of one or more (up to a maximum of 6) `Author`/(s). A `Paper` can be a `Short Paper`, `Long Paper` or an `Extended Abstract`. A `Journal Paper` is similar to a `Long Paper` except it contains very detailed discussion on a subject and may have special material/(s) associated with it. An `Abstract` is part of a `Paper`. A `Paper` is classified as `Short Paper`, `Long Paper`, `Journal Paper` or an `Extended Abstract` depending on number pages and the depth of subject material covered in it.


Figure 1: Case Study example (Level 0)

Generally speaking, an `Extended Abstract` can be of maximum 2 pages, a `Short Paper` between 5–12 pages, a `Long Paper` between 12–20 pages and a `Journal Paper` more than 25 pages. The page count for all papers includes all appendices, supplementary and special materials.

There can be exception and this is only approved by the Chairperson of each `Conference`. For each paper in the system, there must be a Conference

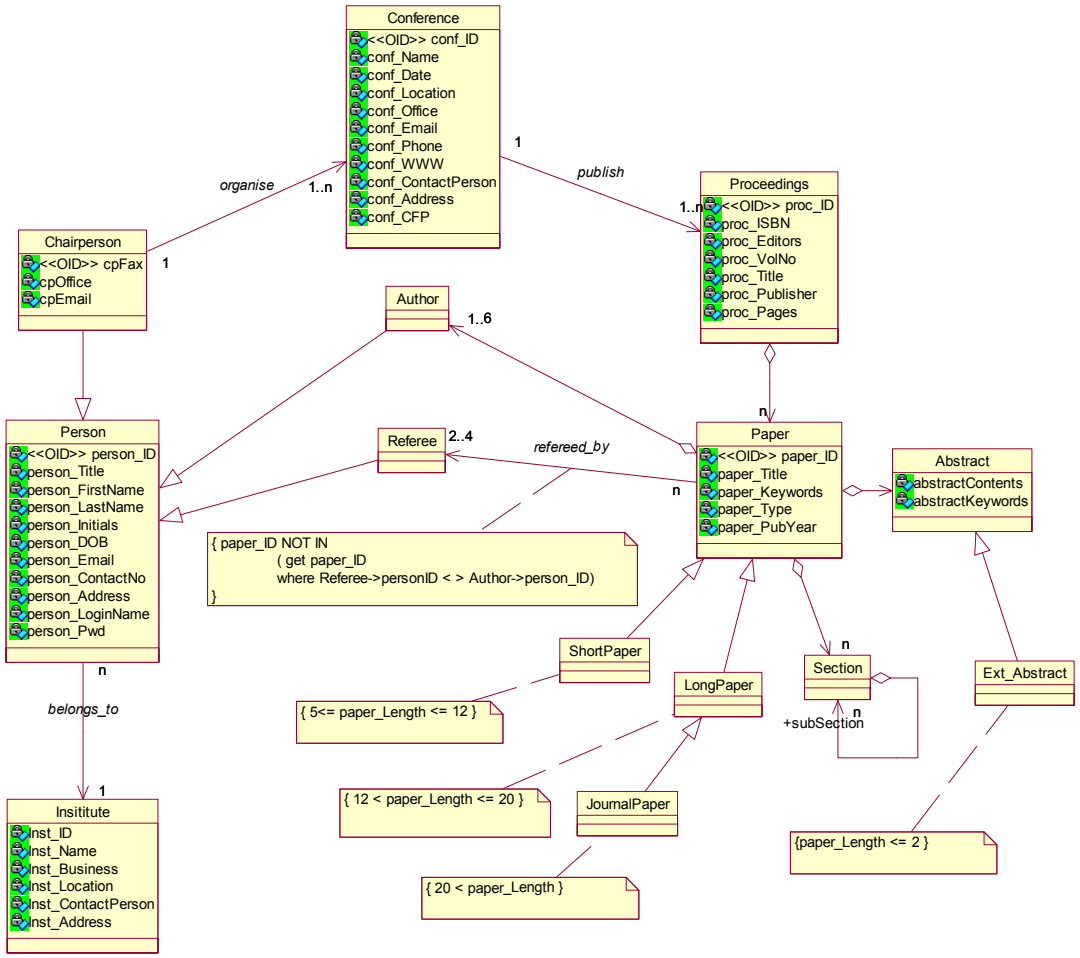


Figure 2: UML model of the example case study

associated with it together with two Referees (max of 4). All Referees in the system has to be approved by the conference chair with which s/he is associated with. An `Author` must belong to an `Institute` (academic education or industry). When a paper is submitted to a `Conference`, at least one `Author` of the paper has to register to attend the `Conference`. A UML representation of this domain model is given in Fig. 1 and 2.

## 2 CONCEPTUAL VIEWS

*A **conceptual view*** (Rajugan R. et al., 2003; Rajugan R. et al., 2004) *is the one which is defined at the conceptual level with higher level of abstraction.*

In simple terms, a *conceptual view* (shown in Fig. 3 - 6 ) describes how a collection of XML tags make sense to a domain user (here we use this term very generally to refer to all people who are working in a particular domain and not to task specific people) at the conceptual/abstract level. A typical XML domain may contain few XML documents to many thousands of semantically related, clusters of XML documents (and their related schemas) depending on the real world requirement. At a given instant, only a subset of these cluster of XML tags, its specification and their data values (information) may be of use or required by a domain user. This subset of XML tags, collectively form a conceptual view which is of interest to the domain user at a point in time.

In related literature, the notion of *conceptual views* is non-existent. From relational to semi-structured and XML, the view concept begins at the data manipulation language level.We argue that, providing view formalism at the conceptual level (*abstract views*) will improve the resulting view implementation similar to that of a conceptual model what does to a software system. An abstract view formalism will; (1) Provide data abstraction to view data set similar to a class (in OO) does to real-world objects, (2) Enable the software designers (not the programmers) to visualise, construct and validate constructed data sets (views) that are normally left to implementers, (3) Utilise as a tool to communicate better with the domain users (DU) and to improve domain user feedbacks (as DU usually used to visualise data as a constructed data sets (views) than a stored/modelled data class), (4) Be utilised in other areas, such as User Interface Engineering (UIE), where abstract constructs can be constructed at the conceptual level to capture Abstract User Interface (AUI) objects (E. J. Chang, 1996), where the user interface objects are identified based on what the user interface does and not how it is done and (5) Be utilised by system designers to add additional data semantics at a higher level of abstractions to data intensive domains (such as XML based domains),

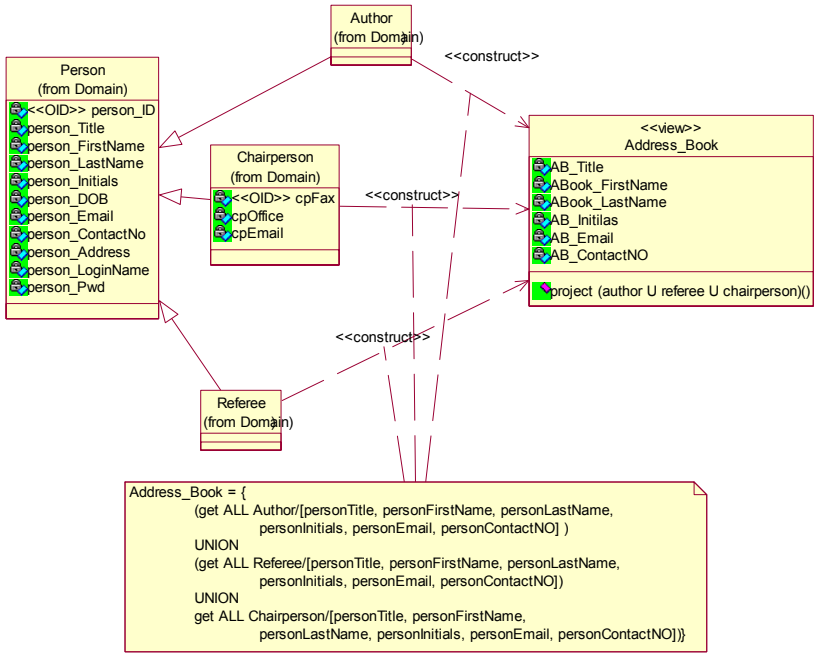


Figure 3: Conceptual view example (in UML)

where the meaning of data is important than the data itself. In doing so, the designers are motivated by the fact that, they only need to worry about what is need than how to do it.

**Note:** In this paper, we only consider static aspects of the view mechanism (both in abstract and concrete).

## 2.1 Conceptual View and UML™

To model conceptual views in UML, we introduce a set of stereotypes and conceptual operators. In addition, to make view constraints more explicit and visible, we use declarative view constraint specification language (discussed in Section 2.2), similar to OMG's Object Constraint Language (OCL). Though our future work will focus on OCL for view constraints, to keep the concepts presented in this paper simple and complex-free, we adopt the declarative view constraint approach. In Fig. 3, 5 and 6 shows some example conceptual views constructed for the example case study.

The *conceptual operators* (Rajugan R. et al., 2004) enable systematic construction conceptual views. These operators can be easily transformed into query segments, user defined functions and/or procedures for implementation. By doing so help the modeller to capture view construct at the abstract level without knowing or worrying about query/language syntax. They are grouped into set operators, namely union, difference, intersection, Cartesian product and unary operators namely projection, rename, restructure, selection and joins.

## 2.2 Constraint Specification

The constraint specification we used here is declarative; that is, it is simple, OCL/SQL like and helps to explain our view model (at the conceptual level) more explicitly in UML. As shown in Fig. 4, where a view $V_j$ is constructed from a stored class $C_i$, the view constraints are shown over the `<<construct>>` relationship.

## 2.3 UML and Stereotypes

In UML, a stereotype is based on an existing base model element or on a variant of the base model element, to provide extensibility and model management for an existing, well-defined model. Here, we use UML stereotypes to provide conceptual semantics to view formalism, defined over a stored/domain data model such as shown in Fig. 2. The following sections discuss some of the main stereotypes used to capture conceptual views.

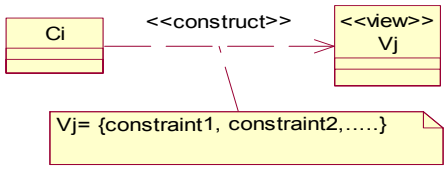


Figure 4: A conceptual view (in UML)

## 2.4 Constructor: <<construct>>

The show the relationship between a conceptual view and the stored class/(es) from which it is constructed, we use directed-dashed line with `<<construct>>` keyword shown above the line (Fig. 4). This is to avoid confusion with the built-in UML dependency relationship and other stereotypes. As shown in Fig. 4, where a view $V_j$ is constructed from a stored class $C_i$, the relationship is as `<<construct>>` relationship; the relationship that exists between a conceptual view and its stored class/(es). If a conceptual view is constructed over an existing conceptual view (view of a view), same relationship is used show the hierarchy (the base conceptual view and the new conceptual view).

## 2.5 Object Identifier: <<OID>>

In an OO system, an object has a unique system-wide identifier that is independent of the values of its attribute/(s), called *Object Identifier* or OID (Dillon & Tan, 1993; Jacek Blazewicz et al., 2003). When created, an object will be referred to, using its system assigned OID during its entire existence. In DBMS systems, OIDs can be either *logical* or *physical* depending on it nature.

In many OO conceptual models and diagrams, though the concept of OID is assumed to be an implicit concept (unlike primary keys in E/ER), in our work, with *conceptual views*, we have a need to explicitly state the OIDs and should be available to visualize at that highest level of abstraction. Therefore, here, we provide a means of using OIDs for the purpose of IDs, similar to that of primary/foreign key constraints available in E/ER models. We argue that, just utilizing OID (a unique concept to OO systems) in our conceptual model provides additional semantics, such as providing Id/keys, referential and integrity constraints that are visually lacking in many OO conceptual modelling technique.

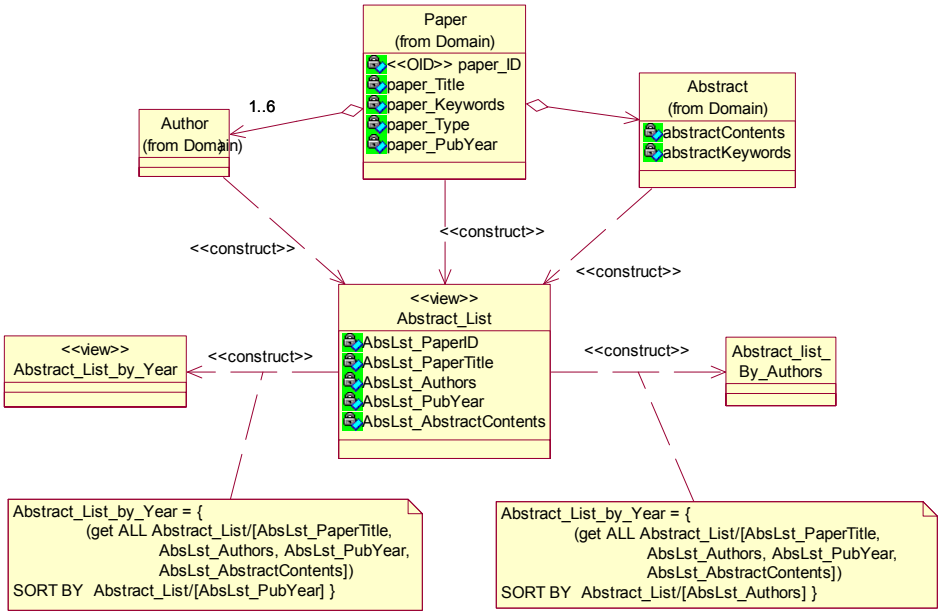


Figure 5: Another conceptual view example (the contents of the Abstract_List package)

To visually model OID in UML class diagram, we define a stereotype `<<OID>>`, shown in Fig. 2, 3, and 5 as an attribute type. Together with attribute name and optional type definition, OID stereotype `<<OID>>` can be used in UML to indicate that the attribute that is an OID. Later in the implementation of the system, these OID can be mapped to XML Schema specific ID/KEY and UNIQUE constraints.

## 2.6 Ordered Composition/Ordering

In real-world, composite objects being in an aggregation with one or more sub-objects, they also can be in a pre-defined order. For example in XML Schema construct such as with `<xsd:sequence>`, we regularly observe that the tag `<xsd:sequence>` signifies that the embedded elements are not only a simple assortment of components but these have a specific ordering. This signifies an important OO concept, *ordered composition*.

Simply said, to capture ordering, we add an UML stereotype that allows capturing of the ordered composition utilizing stereotypes to specify the objects' order of occurrence such as <<1>>, <<2>>, <<3>>, …. ,<<n>>. In related work (Vicky Nassis et al., 2004; Vicky Nassis, Rajugan R., Dillon et al., 2005), we have extensively discussed defining such ordered composition and mapping it to XML Schema. Due to page limitation we do not include that detailed discussion here.

# 3 XML VIEWS

*An **XML View** is an imaginary XML document which points to a collection of semantically related XML tags from an XML domain and satisfies a **Conceptual View** definition from the target XML conceptual domain* (Rajugan R. et al., 2003).

An imaginary XML document is said to be an XML View if and only if, it has a document name, a valid schema definition (which constrains and validates the document), a collection of semantically related tags and their namespaces (or domain), a set of new tags (if any) and their namespaces which are derived from others and a constructor that defines how the document will be materialized.

Since an XML View document may result in a few collections of XML tags to that of a whole semantically related cluster of XML tags, for the user, the resulting XML View document behaves as another XML document.

## 3.1 Mapping Conceptual views to XML views

An XML Schema is usually comprised of a set of schema components, such as type definitions and element declarations. There are 12 kinds of schema components in total, falling into three groups. The most used components include simple type and

complex type definitions, attribute declarations, and element declarations.

For example, in the example case study, conceptual view `Address_Book` will be mapped to XML view schema as a `complexType` in XML Schema, while conceptual view attributes such as `ABook_FirstName` and `ABook_LastName`, which will correspond XML Schema `simpleType` in XML view (Schema). Some of the conceptual view constraints are mapped to XML view schema in the form of XML Schema constraints such as `ID / IDREF`, `KEY / KEYREF`, `USE`, `minOccurs / maxOccurs`, `extension / restriction`, `order`, `sequence` etc. Similarly these constraints can also be used to map the OO relationships captured in UML to XML (view) Schema constructs. For example `extension`, can be used to map a IS-A relationship for extending the base class, while `ID` combined with `minOccurs/maxOccurs` can be used to map an association relationship between two nodes. A more detailed discussion on mapping OO generic concepts to XML Schema can be found in (Feng et al., 2003; Xiaou et al., 2001a; Xiaou et al., 2001b). In the following sections, we briefly show how some of the main view components (discussed in section 2) are mapped to XML views (schema).

## 3.2 Stereotype: <<OID>>

The `<<OID>>` stereotype is mapped to XML view (Schema) is shown below in the code listing.

```
<xs:complexType name="OIDType">
    <xs:sequence>
        <xs:element name="ID">
            <xs:unique name="OID">
                <xs:selector xpath="OIDType"/>
                <xs:field xpath="ID"/>
            </xs:unique>
        </xs:element>
    </xs:sequence>
</xs:complexType>
```

## 3.3 Stereotype: <<View>>

All conceptual views are initially mapped to a basic XML view type and extended to fit the new view definition. For example the basic <<view>> type is mapped into XML Schema as shown in the code listing below.

```
<xs:complexType name="viewType">
    <xs:sequence>
        <xs:element name="view_ID" type="OIDType"/>
        <xs:element name="view_Name"/>
        <xs:element name="view_Query" minOccurs="0"/>
    </xs:sequence>
</xs:complexType>
```

And other conceptual views (in our example the conceptual view "`Address_Book`") are derived from this basic view type. For example,

```
<xs:complexType>
    <xs:sequence>
        <xs:element name="Address_Book">
            <xs:complexType>
                <xs:complexContent>
                    <xs:extension base="viewType">
                        <xs:sequence>
                            <xs:element name="AB_Title"/>
                            <xs:element
name="ABook_FirstName"/>
                            <xs:element
name="ABook_LastName"/>
                            <!-- additional nesting -->
                            <!-- additional nesting -->
                            </xs:sequence>
                        </xs:extension>
                    </xs:complexContent>
                </xs:complexType>
            </xs:element>
    </xs:sequence>
</xs:complexType>
```

# 4 VIEW HIERARCHY

In related work (E. J. Chang, 1996; Kim, 1990; Kim & Kelly, 1995), we argued that, in OO systems, the view hierarchy and the stored class hierarchy should be kept separately from each other. In continuing the discussion of view hierarchy to XML domain, to avoid confusion, we need to clarify the issue of the relationship between *stored* XML documents and *view* documents (both conceptual and XML views). In our work (Rajugan R. et al., 2003), we argued that the view hierarchy in XML domain (both conceptual and XML) should be kept separately from the stored document hierarchy.

This is because, as in relational and OO systems, modelling of XML documents share some relational and many OO features. Naturally, new *View documents* may form new document hierarchies (inheritance, aggregation, nested etc.), may extend the existing namespace of the stored XML namespace/(s) and may be used to provide dynamic windows to one or more stored heterogenous XML domains. Views in XML domain may also be used to provide imaginary schema changes (such new simple/complex tags, new document hierarchy, restructuring etc.). But, keeping in line with the arguments presented for OO views in (E. J. Chang, 1996; Kim, 1990; Kim & Kelly, 1995), we believe that the stored XML documents hierarchy and the *XML View* documents hierarchy should be kept separate. Many of the points made by Won Kim and Chang and Dillon et al. (Dillon & Tan, 1993) for OO views apply to XML views.

## 4.1 UML Package Diagram

As we stated earlier, the role of conceptual views is to provide different perspectives to a stored document class hierarchy. Also in the previous section we argued that, they can be grouped into logical groups/hierarchies. Here, if we look closely, each hierarchy/group that is very similar to that of a subject area (Dillon & Tan '93, Coad & Yourdon '90) (Dillon & Tan, 1993) (or class categories Booch 1991) in OO conceptual modeling techniques. When we allow logical grouping of conceptual views and their associated relationships to clarify a given perspective, we are giving the designer the abstraction needed to model a cluster of conceptual views, without worrying about external connectivity of the view cluster (Vicky Nassis et al., 2004; Vicky Nassis, Rajugan R., Dillon et al., 2005; Rajugan R. et al., 2003).

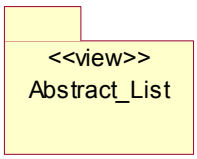


Figure 6: The UML model of a conceptual view using UML "package"

In order to capture this logical grouping in UML, we utilize the UML ***package*** construct. Intuitively, based on OMG's UML package specification (OMG-UML™, 2003), it describes our logical grouping of conceptual views into clusters (Vicky Nassis et al., 2004; Vicky Nassis, Rajugan R., Dillon et al., 2005). Given a conceptual view, in order to include additional semantics/refinements, we can construct additional new view-hierarchies. These hierarchies may form additional structural or dependency relationship with existing conceptual views or view hierarchies. To model such view hierarchy using UML packages, we introduced a `<<view>>` stereotype construct for packages as shown in Fig. 6. Some work has already been done in investigated packages for dimensional modelling (Sergio Lujan-Mora, Juan Trujillo, & Song, 2002a, 2002b). Here we use packages not just for grouping tool, but also as a structural construct as well as a new namespace model for the new view hierarchy.

# 5 REAL-WORLD APPLICATIONS OF THE XML-VIEWS

Since XML and XML driven solution frameworks are on the increase, it is important to provide models and techniques for XML, which is at a high enough level of abstraction but with rigorously defined standards that are to be more widely understood by both developers and non-technical users. To address some of these issues, here we proposed a generic XML view design formalism/methodology for XML domains to provide view-driven-architecture solutions for varied, yet complex enterprise systems.

For example, our work on XML views are utilized in; (1) XML Document Warehouse design (Vicky Nassis et al., 2004; Vicky Nassis, Rajugan R., Dillon et al., 2005; Vicky Nassis, Rajugan R., Rahayu, & Dillon, 2005); where the proposed conceptual design of a XML document warehouse model uses XML views as dimensions, (2) web engineering (Gardner, Rajugan R., Chang, & Dillon, 2004; Rajugan R., Gardner, Chang, & Dillon, 2005); where user-centred web portal and website are designed and implement using XML-view formalism and finally (3) User Access Control (UAC) design and implementation (Steele, Gardner, Rajugan R., & Dillon, 2005), where XML-view formalism is used as a middleware in providing UAC for XML repositories and databases.

# 6 CONCLUSION AND FUTURE WORK

Though very useful, existing view formalisms (for all data models including XML) lacks higher level modelling techniques and abstraction that is needed to describe, model, and communicate complex systems such as data warehouse and e-commerce systems. Therefore, in this paper, we presented a generic view design methodology for XML domains at three levels (conceptual, logical and document level) of abstraction. It is UML driven and semantically rich for designing enterprise solution and architectures.

For future work, a lot of issues deserve investigation. First, the application of OCL in specifying view constraints at the conceptual level and mapping between OCL and XML Schema. Second a well-formulated empirical study to focus on validating the view design methodology. Third is the investigation into dynamic perspectives of the XML view formalism. Another area that deserves investigation is the area of XML-view in the OMG proposal on querying MOF based models for MDA solutions.

# REFERENCES


Abiteboul, S. (1999). *On Views and XML.* 18th ACM SIGMOD-SIGACT-SIGART (PODS '99), USA.

Abiteboul, S., & Bonner, A. (1991). *Objects and Views.* ACM SIGMOD, ACM SIGMOD '91.

Abiteboul, S., Goldman, R., McHugh, J., Vassalos, V., & Zhuge, Y. (1997). *Views for Semistructured Data.* Work. on Mgmt. of Semistructured Data, USA.

Chang, E., & Dillon, T. S. (1994). *Integration of User Interfaces with Application Software and Databases Through the Use of Perspectives.* ORM '94, Australia.

Chang, E. J. (1996). *Object Oriented User Interface Design and Usability Evaluation.* Doctor of Philosophy (Ph.D), La Trobe University, Melbourne, Australia.

Cluet, S., Veltri, P., & Vodislav, D. (2001). *Views in a Large Scale XML Repository.* 27th VLDB '01, Italy.

Dillon, T. S., & Tan, P. L. (1993). *Object-Oriented Conceptual Modeling*: Prentice Hall, Australia.

Elmasri, R., & Navathe, S. B. (2000). *Fundamentals of database systems* (3 ed.): Addison-Wesley.

Feng, L., Chang, E., & Dillon, T. S. (2002). A Semantic Network-based Design Methodology for XML Documents. *ACM Transactions on Information Systems (TOIS), 20, No 4*, 390 - 421.

Feng, L., Chang, E., & Dillon, T. S. (2003). Schemata Transformation of Object-Oriented Conceptual Models to XML. *International Journal of Computer Systems Science & Engineering, 18, No. 1*(1), 8845-8860.

Gardner, W., Rajugan R., Chang, E., & Dillon, T. S. (2004). *xPortal: XML View Based Web Portal Design.* 17th Int. Conf. ICSSEA '04, Paris, France.

Jacek Blazewicz, Wieslaw Kubiak, Tadeusz Morzy, Rusinkiewicz, M., & (eds). (2003). *Handbook on Data Mgmt. in IS*: Springer, Berlin ; New York.

Jorge H. Doorn, C. Rivero, L., & (eds). (2002). *Database Integrity: Challenges & Solutions*: Idea Group Publishing, Hershey, PA.

Kim, W. (1990). *Research Directions in Object-Oriented Database Systems.* Proc. of the 9th ACM SIGACT-SIGMOD-SIGART, USA.

Kim, W., & Kelly, W. (1995). Chapter 6: On View Support in Object-Oriented Database Systems. In *Modern Database Systems* (pp. 108-129): Addison-Wesley Publishing Company.

Liefke, H., & Davidson, S. (2000). *View Maintenance for Hierarchical Semistructured.* DaWak '00, UK.

Lucie-Xyleme. (2001). Lucie Xyleme: A dynamic warehouse for XML Data of the Web. *IEEE Data Engineering Bulletin, 24, No 2*, 40-47.

Nassis, V., Rajugan, R., Dillon, T. S., & Rahayu, W. (2004, September 1-3). *XML Document Warehouse Design.* Proc. of DaWaK '04, Zaragoza, Spain.

Nassis, V., Rajugan R., Dillon, T. S., & Rahayu, W. (2005). Conceptual and Systematic Design Approach for XML Document Warehouses. *Int. Journal of Data Warehousing and Mining, 1, No 3*.

Nassis, V., Rajugan R., Rahayu, W., & Dillon, T. S. (2005, 9-12 May 2005). *A Systematic Design Approach for XML-View Driven Web Document Warehouses.* Int. Work. on Ubiq. Web Sys. & Intel. (UWSI '05).

OMG-UML™. (2003). *Unified Modeling Language™ (UML) Version 1.5 Specification*: OMG.

Rajugan R., Chang, E., Dillon, T. S., & Ling, F. (2003). *XML Views: Part 1.* Proc. of DEXA '03.

Rajugan R., Chang, E., Feng, L., & Dillon, T. S. (2004). *XML Views, Part II: Modeling Conceptual Views Using XSemantic Nets.* Workshop & SSS on Industrial Informatics, The 30th IEEE IECON '04, S.Korea.

Rajugan R., Gardner, W., Chang, E., & Dillon, T. S. (2005, 9-12 May 2005). *xWeb: An XML View Based Web Engineering Methodology.* Int. Work. on Ubiq. Web Sys. & Intel. (UWSI '05).

Sergio Lujan-Mora, Juan Trujillo, & Song, I.-Y. (2002a). *Extending the UML for Multidimensional Modeling.* 5th Int. Conf. on UML & Apps. (UML '02), Germany.

Sergio Lujan-Mora, Juan Trujillo, & Song, I.-Y. (2002b). *Multidimensional Modeling with UML Package Diagrams.* Proc. of the 21st Int. Conf. on (ER '02).

Steele, R., Gardner, W., Rajugan R., & Dillon, T. S. (2005). *Design of an XML View Based User Access Control (UAC) Middleware.* IEEE Int. Conf. EEE '05, Hong Kong.

Volz, R., Oberle, D., & Studer, R. (2003a). *Implementing Views for Light-Weight Web Ontologies.* Proc. 7th IDEAS '03, Hong Kong.

Volz, R., Oberle, D., & Studer, R. (2003b). *Views for light-weight Web ontologies.* Proc. of the ACM Symposium on Applied Computing (SAC '03), USA.

W3C-XML. (2004, 4th February 2004). *Extensible Markup Language (XML) 1.0 (Third Edition)*. from http://www.w3.org/XML/

W3C-XSD. (2004). *XML Schema*, 2004, from http://www.w3.org/XML/Schema

Xiaou, R., Dillon, T. S., Chang, E., & Feng, L. (2001a). *Mapping Object Relationships into XML Schema.* Proc. of OOPSLA Work. on Objects, XML and DBs.

Xiaou, R., Dillon, T. S., Chang, E., & Feng, L. (2001b, September 3-5). *Modeling and Transformation of Object-Oriented Conceptual Models into XML Schema.* 12th Int. Conf. on DEXA '01.

Xyleme. (2001). *Xyleme Project*, from http://www.xyleme.com/

Zhuge, Y., & Garcia-Molina, H. (1998). *Graph structured Views and Incremental Maintenance.* Proc. of the 14th IEEE ICDE '98, USA.