A Three-Layered XML View Model: A Practical Approach

Rajugan, R.[1], Elizabeth Chang[2], Tharam S. Dillon[1], and Ling Feng[3]

[1] eXel Lab, Faculty of IT, University of Technology, Sydney, Australia
{rajugan, tharam}@it.uts.edu.au
[2] School of Information Systems, Curtin University of Technology, Australia
Elizabeth.Chang@cbs.cutin.edu.au
[3] Faculty of Computer Science, University of Twente, The Netherlands
ling@ewi.utwente.nl

Abstract. Since the early software models, abstraction and conceptual semantics have proven their importance in software engineering methodologies. For example, Object-Oriented conceptual modeling offers the power in describing and modeling real-world data semantics and their inter-relationships in a form that is precise and comprehensible to users. Conversely, XML is becoming the dominant standard for storing, describing and interchanging data among various Enterprises Information Systems and databases. With the increased reliance on such self-describing, schema-based, semi-structured data language/(s), there exists a requirement to model, design, and manipulate XML data and associated semantics at a higher level of abstraction than at the instance level. But, existing Object-Oriented conceptual modeling languages provide insufficient modeling constructs for utilizing XML schema like data descriptions and constraints, and most semi-structured schema languages lack the ability to provide higher levels of abstraction (such as conceptual models) that are easily understood by humans. To this end, it is interesting to investigate conceptual and schema formalisms as a means of providing higher level semantics in the context of XML-related data engineering. In this paper, we use XML view as a case in point and present a three-layered view model with illustrated examples taken from a real-world application domain. We focus on conceptual and schema view definitions, view constraints, and the conceptual query operators.

1 Introduction

In software engineering, many methodologies have been proposed to capture real-world problems into manageable segments, which can be communicated, modeled, and developed into robust maintainable software systems. Since the early software models, abstraction and conceptual semantics have proven their importance in software engineering methodologies. For example, in Object-Oriented (OO) conceptual models, they have the power in describing and modeling real-world data semantics and their inter-relationships in a form that is precise and comprehensible to users [17, 23]. With the emergence of semi-structured data, Semantic Web (SW) [44], web services [46], and ubiquitous systems, it is important to investigate new data models

for Enterprise Information Systems (EIS) that can correlate and co-exist with heterogeneous, multifunctional schemas under the context of Enterprise Content Management [5]. Any such data models should accommodate heterogeneous schemas and changing model and data requirements at a higher level of abstraction, yet adoptable to the current software engineering paradigm.

Conversely, since the introduction of eXtensible Markup Language (XML) [47], it is fast emerging as the dominant standard for storing, describing, and interchanging data among various EIS and heterogeneous databases. In combination with XML Schema [49], which provides rich facilities for constraining and defining XML content, XML provides an ideal platform and flexibility for capturing and representing complex EIS data formats. But, OO modeling languages (such as UML [34], Extended-ER[19]) provide insufficient modeling constructs for utilizing XML schema based data descriptions and constraints. Also, XML Schema lacks the ability to provide higher levels of abstraction (such as conceptual models) that are easily understood by humans [20, 21].

In data engineering, some forms of data "abstraction" and perspectives have been provided by the view formalisms and have been supported by data model specific query language. Since the introduction of relational data model [16, 19], motivation for views has changed over the last two decades. At present, views are widely used in, (i) user access and user access control applications [41]; (ii) defining user perspectives/profiles [10]; (iii) designing data perspectives; (iv) dimensional data modeling [32]; (v) providing improved performance and logical abstraction (materialized views) in data warehouse/OLAP and web-data cache environments [22, 24, 31, 40]; (vi) web portals & profiles; and (vii) Semantic Web, for sub-ontology or Ontology views [42, 51]. From this list, it is apparent that the uses and applications of views are realized more than their originally intended purpose proposed by Date et al. (i.e., the 2-Es - data Extraction and Elaboration [52]), with extensive research being carried out by both researchers and industry to improve their design, construction, and performance. Yet, in our view, the view concept still remains as a data language and model dependent low-level (instance) construct. In the OO paradigm, the modeling languages provide minimal or no semantics to capture abstract view formalisms at the conceptual level [35, 52] and the existing XML technology standards have no support for concrete view formalisms.

To tackle this issue, in our early work, we have proposed to extend the concept of semi-structured XML view with conceptual and schemata notions [35]. In this paper, we present in a systematic way, (i) a view formalism for XML with there levels of abstractions, namely, *concept*, *schema*, and *instance*; (ii) detailed modeling primitives for such a view formalism, including constraints and conceptual operators; and (iii) the transformation methodology across various abstraction levels. Please note that the intention of this paper is neither to propose a new view standard for XML nor extensions to XML query languages. Rather, we focus on providing XML view mechanism at the conceptual, schema, and document levels by means of OO conceptual modeling. To help illustrate our concepts, we conduct a real-world case study in a fictitious global logistic company called LWC & e-Solutions Inc., e-Sol in short. Also, our three-layered view model has been utilized in real-world, data intensive application scenarios [36, 39] such as; (i) design of XML document warehouses and distributed

FACT repositories, (ii) design of websites and web portals, (iii) design of User Access Control (UAC) and UAC middleware, and (4) design of (Ontology) views for Semantic Web.

The rest of this paper is organized as follows. In section 2, we review early work in view related domains, followed by the description of our case study example in section 3. This is followed by the formal introduction of our three-layered XML view model in section 4. We detail each of the three layers in section 5, 6, and 7, respectively. We conclude the paper and outline future work in section 8.

2 Related Work

Here we first briefly look at the history of the view mechanisms available today and some of the proposals for new view mechanisms supporting semi-structured data and constraint specification for views. Existing view models can be grouped into four categories, namely; (a) early (namely relational) view models, (b) OO view models, (c) semi-structured (namely XML) view models and (d) views for SW.

2.1 Early View Models

The relational view formalism [18, 19, 24, 26, 27] has been discussed extensively in many industrial and research forums since its proposal by Date [16]. The relational (classical) definition of a view is based on ANSI/SPAC three-schema architecture (Tsichritzis & Klug 1978) [16, 19, 27], where a view is treated as a *virtual relation*, constructed by a query which is executed on one or more stored relations [16, 19]. Later the concept of view was extended to support complex queries and/or aggregate/summary queries. Generally, a relational view definition is persistent which contains the list of attributes or elements that are incorporated within the view, together with the declarations on how to extract those elements from the underlying stored relations or classes, or from another view [16, 19]. A relational view can be queried, joined with another relations or classes, and be included in another view definition.

During the OO revolution, the relational view definitions were extended to OO data models by Won Kim et al. [26, 27], Abiteboul et al. [2], and Chang [13]. Here the views were defined in a synonymous manner to the relational model and/or extending the relational definition [26] when needed (supporting the 2-Es; data Extraction, Elaboration and some research directions towards data Extension). 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* [13]. 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.

2.2 Views for Semi-Structured Data

Since the emergence XML, the need for semi-structured data models, which have to be independent of the fixed data models and data access, violates fundamental properties of persistent data models. Many researchers attempted to solve these issues by using graph based [55] and/or semi-structured data models [3, 28]. 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 [1] and later more formally by Sophie Cluet et al. [15]. 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 [1]. 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. He also argues that, an XML view specification should rely on a data model (like ODMG [8] model) and a query language. In the paper [15], they discuss in detail on how abstract paths/DTDs are mapped to concrete paths/DTDs. These concepts, which are implemented in the Xyleme project [29], 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. Other view models for XML include; (a) the MIX (Mediation of Information using XML) view system [30] and (b) an intuitive view model for XML using Object-Relationship-Attribute model for Semi-Structured data (ORA-SS) [14]. This is one of the first view model that supports some of abstraction above the data language level.

In related work in Semantic Web (SW) [45] paradigm, some work has been done in views for SW [42], where the authors proposed a view formalism for RDF document with support for Resource Description Framework (RDF) [43] schema (using a RDF schema supported query language called RQL). This is one of the early works focused purely on RDF/SW paradigm and has sufficient support for logical modeling of RDF views. But RDF is an object-attribute-value triple, where it implies object has an attribute with a value [21]. But RDF makes only intentional semantics and not data modeling semantics. Therefore, unlike views for XML, views for such RDF (both logical and concrete) have no tangible scope outside its domain. In related area of research, the authors of the work propose a logical view formalism for ontology [51] with limited support for conceptual extensions, where materialized ontology views are derived from conceptual/abstract view extensions.

2.3 View Constraints

In data modeling, specifications often involve constraints. In the case of views, it is usually specified by the data language in which they are defined in. For example, in relational model, views are defined using SQL and a limited set of constraints can be defined using SQL[16, 19], namely, (1) presentation specific (such as display headings, column width, pattern order etc), (2) range and string patterns for aggregate fields, (3) input formats for updatable views, and (4) other DBMS specific (such view materialization, table block, size, caching options etc).

In Object-Relational and OO models, views had similar constraints but they are more extensive and explicit due to the data model. The views here are constructed and specified by DBMS specific (such as OQL [8]) and/or external languages (such as C++, Java or O_2C [2]). It is a similar situation in views for semi-structured data paradigm, where rich set of view constrains are defined using languages such as OQL based LOREL [4]. But the work by authors of [14] provides some form of higher-level view constraints (under ORA-SS model) for XML views, while the work in [42] provides some form of logical level view constraints to be defined in views for in SW/RDF paradigm. Here, for our view formalism, we look into using UML/OCL as our view constraint specification language. Also, our work should not be confused with work such as [7], where authors use OCL to "model" (not to specify) relational views, which utilizes OCL from a data engineering point of view than a for constraint specification.

3 An Illustrative Case Study

The e-Sol Inc. aims to provide logistics, warehouse, and cold storage space for its global customers and collaborative partners. The e-Sol solution includes a standalone and distributed Warehouse Management System (WMS/e-WMS), and a Logistics Management System (LMS/e-LMS) on an integrated e-Business framework called e-Hub [11] for all inter-connected services for customers, business customers, collaborative partner companies, and LWC staff (for e-commerce B2B and B2C). Some real-world applications of such company, its operations and IT infrastructure can be found in [11, 12, 25].

In WMS (Fig. 3 & 4), customers book/reserve warehouse and cold storage space for their goods. They send in a request to warehouse staff via fax, email, or phone, and depending on warehouse capacity and customers' grade (individual, company or collaborative partner), they get a booking confirmation and a price quote. In addition, customers can also request additional services such as logistics, packing, packaging etc. When the goods physically arrive at the warehouse, they are stamped, sorted, assigned lots numbers and entered into the warehouse database (in Lots-Master). From that day onwards, customers get regular invoices for payments. In addition, customers can ask the warehouse to handle partial sales of their goods to other warehouse customers (updates Lots-Movement and Goods-Transfer), sales to overseas (handled by LMS) or take out the goods in full or in partial (Lots-Movement). Also customer can check, monitor their lots, buy/sell lots and pay orders via an e-

Commerce system called e-WMS. In LMS, customers use/request logistics services (warehouse or third-party logistics providers) provided by the warehouse chains. This service can be regional or global including multi-national shipping companies. Like e-WMS, e-LMS provide customers and warehouses an e-Commerce based system to do business. In e-Hub, all warehouse services are integrated to provide one-stop warehouse services (warehouse, logistics, auction, goods tracking, payment etc) to customers, third-party collaborators and potential customers. A context diagram of the system is given in Fig. 1, followed by some detailed models in Fig. 3-4.

In e-Sol, due to the business process, data have to be in different formats to support multiple systems, customers, warehouses and logistics providers. Also, data have to be duplicated at various points in time, in multiple databases, to support collaborative business needs. In addition, since new customers/providers to join the system (or leave), the data formats has to be dynamic and should be efficiently duplicated without loss of semantics. This presents an opportunity to investigate how to use our XML conceptual, schema and instance views to design e-Sol at a higher level of abstractions to support changing business, environments, and data formats.

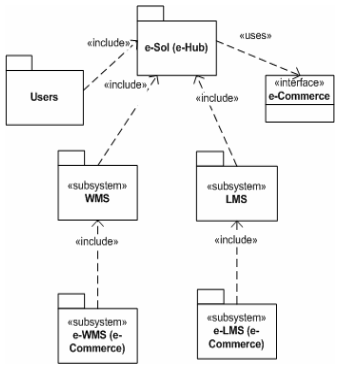

Fig. 1. e-Sol, context diagram

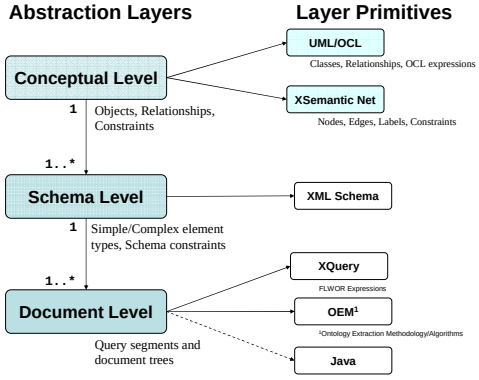

Fig. 2. Three-layered view model for XML

4 A Three-Layered XML View Model

Our XML view study is based on the following two postulates about the real world.

Postulate 1: The term ***context*** refers to the domain that interests an organization as a whole. It is more than a measure, and implies a meaningful collection of objects, relationships among these objects, as well as some constraints associated with the objects and their relationships, which are relevant to its applications. For example, "`people`", "`order`", and "`customer`" can be examples of context in the e-Sol system.

Postulate 2: The term ***view*** refers to a certain perspective of the context that makes sense to one or more stakeholders of the organization or an organization unit at a given point in time. For example, "`processed-order`" and "`overdue-order`" are two contrasting views in the "`order`" context of the e-Sol system.

Fig. 2 outlines our XML view model, which is comprised of three different levels, namely, *conceptual level*, *schema level*, and *instance level*.

(i) The top conceptual level describes the structure and semantics of XML views in a way which is more comprehensible to human users. It hides the details of view implementation and concentrates on describing objects, relationships among the objects, as well as the associated constraints upon the objects and relationships. This level can be modeled using some well-established modeling language such as UML [34], or our developed XML-specific XSemantic Net [20, 39], etc. Thus, the modeling primitives include object, attribute, relationship, and constraint. The output of this level is a well-defined *valid* conceptual model in UML, XSemantic Net, or even OMG's MOF (Meta-Object-Factory), which can be either visual (such as UML class diagrams) or textual (in the case of UML/XMI models).

(ii) The middle scheme level describes the schema of XML views for the view implementation, using the XML Schema language. Views at the conceptual level are mapped into the views at the schema level via the transformation mechanism developed in our previous work [20, 21]. The output of this level will be in either textual (such as XML Schema language) or some visual notations that comply from the schema language (such as graph).

(iii) The bottom instance level implies a fragment of instantiated XML data, which conforms to the corresponding view schema defined at the upper level.

For simplicity, we also call XML view at conceptual level *XML conceptual view*, XML view at schema level *XML schema view*, and XML view at instance level *XML instance view*. We elaborate each of the three levels in the following sections.

5 Conceptual Views

Context is presented in UML using modeling primitives like object, attribute, relationship and constraint in this study. To enable the construction of a valid XML conceptual view from a context, we introduce the notion of *conceptual operator* whose details can be found in our work [37]. These conceptual level operators are comparable to relational operators in the relational model, but they operate on conceptual level objects and relationships. They are grouped into set operators, namely union, difference, intersection, Cartesian product and unary operators namely projection, rename, restructure, selection and joins, and can facilitate systematic construction of conceptual views from context. These conceptual operators can be easily transformed into query segments, user-defined functions and/or procedures for implementation. By doing so, they help the modeler to capture view construct at the abstract level without knowing or worrying about query/language syntax.

***Definition 1*: An XML conceptual view V^c** is a 4-ary tuple $V^c = (V^c_{name}, V^c_{obj}, V^c_{rel}, V^c_{constraint})$, where V^c_{name} is the name of the XML conceptual view V^c, V^c_{obj} is a set of objects in V^c, V^c_{rel} is a set of object relationships in V^c, and $V^c_{constraint}$ is a set of constraints associated with V^c_{obj} and V^c_{rel} in V^c.

***Definition 2*:** Let $C = (C_{name}, C_{obj}, C_{rel}, C_{constraint})$ denote a context which consists of a context name C_{name}, a set of objects C_{obj}, a set of object relationships C_{rel}, and a set

of constraints associated with its objects and relationships $C_{constraint}$. Let $\hat{\lambda}$ be a set of conceptual operators. $\boldsymbol{V^c} = (V^c_{name}, V^c_{obj}, V^c_{rel}, V^c_{constraint})$ is called a **valid XML conceptual view of the context *C***, if and only if the following conditions satisfy:

(i) For any object $\forall o \in V^c_{obj}$, there exist objects $\exists o_1, \ldots, o_n \in C_{obj}$, such that $o = \lambda_{1\ldots}\lambda_m (o_{1,\ldots,} o_n)$ where $\lambda_{1\ldots}\lambda_m \in \hat{\lambda}$. That is, o is a newly derived object from existing objects $o_1, \ldots, o_n$ in the context via a series of conceptual operators $\lambda_{1,\ldots}\lambda_m$ like select, join, etc.

(ii) For any constraint $\forall c \in V^c_{constraint}$, there exists a constraint $\exists c' \in C_{constraint}$ or a new constraint c'' constraints associated with V^c_{obj} or V_{rel}.

(iii) For any hierarchical relationship $\forall r^h \in V^c_{rel}$, there *does not exist* a relationship between one or more and V^c_{obj} and C_{obj}.

(iv) For any association relationship/dependency relationships $\forall r^a \in V^c_{rel}$, there *may exist a relationship between one or more* V^c_{obj} *and* C_{obj}.

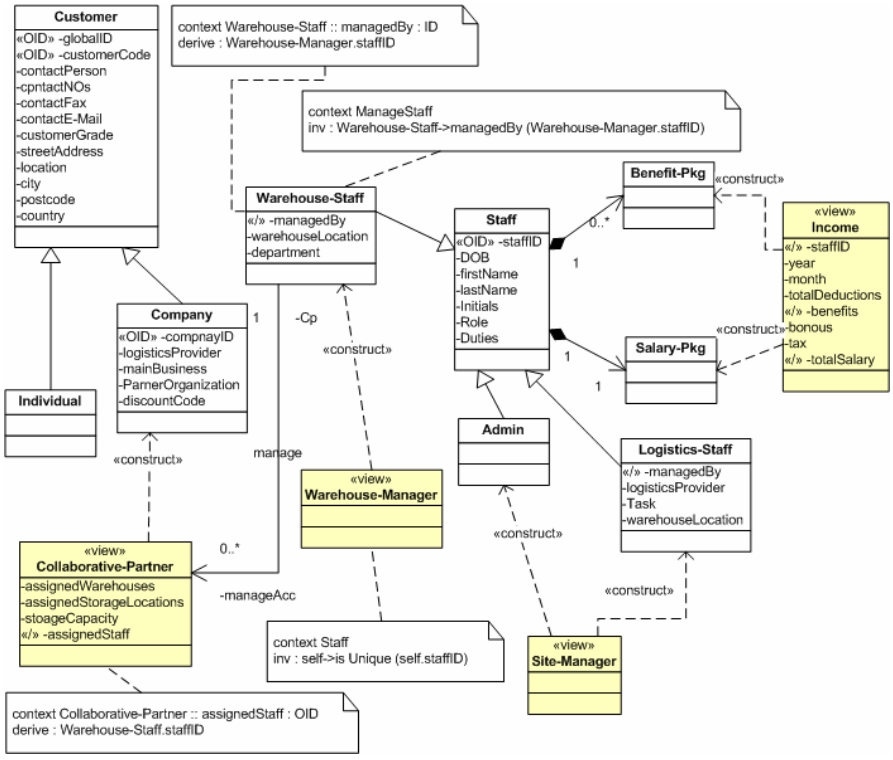

Fig. 3. e-Sol, core user model, relationships and constraints

Example 1: In Fig. 3, "`Warehouse-Manager`" is a valid XML conceptual view, named in the context of "`Staff`". Its objects are taken from the e-Sol "`Core-Users`" UML objects via the conceptual select [37] operator $\sigma_{warehouse\text{-}Staff.Role=\text{“}manager\text{”}}$ (*Core-Users*). Similarly, the conceptual view of name "`Site-Manager`" (Fig. 3) in the given context "`Staff`" is constructed via both SELECT and PROJECT operators.

Example 2: Conceptual views (Fig. 4), "`Customer-History`", "`Lot-Master-Charge-History`" and "`Rent-Warehouse-Space-History`" are perspectives / views in the context of "`Warehouse-History`" of the e-Sol system.

Example 3: Conceptual view (Fig. 3), "`Collaborative-Partner`" is a perspectives / view in the context of "`Customer`" in e-Sol.

Example 4: In the case of conceptual view "`Income`" (shown in Fig. 3), the conceptual construct is a conceptual JOIN operator with join conditions, where x = `Staff`, y = `Salary-Pkg` and z = `Benefit-Pkg`:

$$(x \rightarrow_{(x.staffID=y.staffID)} y) \text{ AND } (x \rightarrow_{(x.staffID=z.staffID)} z).$$

Example 5: In Fig.4 illustrates another valid conceptual view "`Lot-Master-Charge-History`" in the given context "`Lot-Management`". Here, at the conceptual level, it is stated as a *materialized* conceptual view, implying that it is a persistence view during the life time of the system. This characteristic is also stated in the OCL statement (Fig. 4).

It should be noted that, in our model, when referring to conceptual view OCL statements, it is provided only as a complement (such stating derived attributes, uniqueness etc.) to the *visual* UML model. Our aim is to keep the view definitions as *visual* as possible (in direct contrast to the work [7]) and avoid ambiguous and imprecise textual specifications of the view definitions.

5.1 Modeling Conceptual Views

The modeling of XML conceptual views can be done using UML, plus a set of stereotypes [38] and newly introduced conceptual operators [37]. In addition, to make view constraints more explicit and visible, we use OMG's Object Constraint Language (OCL) [33].

As our conceptual view mechanism is defined at a higher-level of abstraction, we can provide an explicit view constraint specification model, as most high-level OOCM languages (such as UML, XSemantic nets, E-ER) provide some form of constraint specification. In the case of XSemantic nets, constraints are provided as part of the model elements [20, 39]. In the next section, we will look at specifying constraints using UML/OCL.

5.2 Modeling Conceptual View Constraints

In UML, OCL, which is now a part of the UML 2.0 standard [34], can support unambiguous constraints specifications for UML models. In our conceptual view model, we incorporate OCL (in addition to built-in UML constraints features) as our view constraint specification language to explicitly state view constraints. It should be noted that, we do not use OCL to *define* views, rather state additional constraints using OCL. OCL supports defining *derived* classes [33, 50], which is close to a view concept [7]. It is of the form of;

```
context: <derived-class-name>::<new-attribute-name>: Type
derive: <source-stored-class>.
        [some expression representing the derivation rule]
          / [<source-stored-class-attribute>
```

To define our conceptual views, we show view classes visually, with the `<<view>>` stereotypes and the relationship between the stored class and the view as `<<construct>>` stereotype. Therefore, we do not require non-visual OCL view specification as shown above, but can be used to show some of the derivations rule for the attributes and/or operations to make the view definition more explicit and precise. It also supports specifying derived values and attributes in already existing views (and stored classes) and specified in the form of;

```
context Typename::assocRoleName: Type
derive: -- some expression representing the derivation rule
```

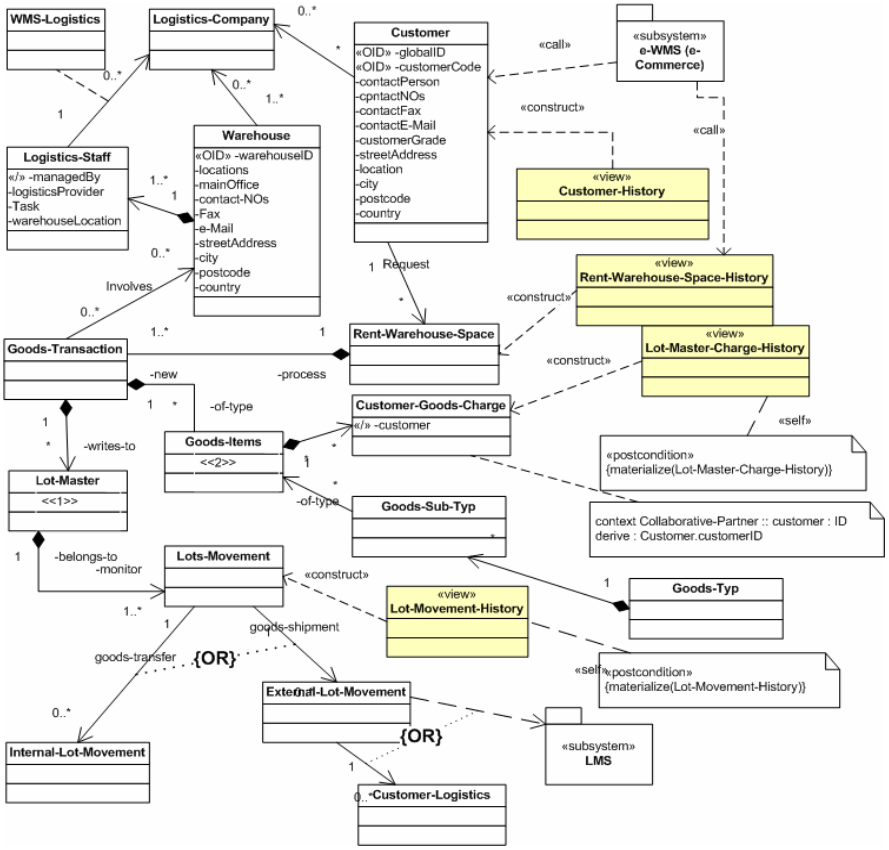

Fig. 4. e-Sol, core data stores, relationships and constraints

In addition, further constraints can be defined for conceptual views including; (1) domain constraints (range of values, min, max, pattern etc), (2) constructional contents (set, sequence, bag, ordered-set), (3) ordering (4) explicit homogenous composition/heterogeneous compositions, (5) adhesion and/or dependencies (6) exclusive

disjunction and many more. Specifying these constraints using OCL expression in conceptual views are similar to that of stored domain objects.

Example 6: In the case of conceptual view "`Income`" (Fig. 3), the following OCL statements hold true;

```
context Income :: Staff : ID
derive : Staff.staffID

context Income :: benefits : Real
derive : Benefit-Pkg.totalBenefits

context Income :: baseSalary : Real
derive : Salary-Pkg.baseSalary

context Income :: totalSalary : Real
derive : totalSalary =
            (self.baseSalary – self.tax)
         + benefits –
            self.totalDeductions
```

Example 7: In the case of conceptual view "`Warehouse-Manager`" (Fig. 3), we indicate the unique `staffID` by the following OCL expression;

```
context Staff
inv : self->isUnique(self.staffID)
```

Example 8: In the case of conceptual views "`Warehouse-Manager`" and "`Warehouse-Staff`", in the context of "`Staff`" (Fig. 3), we indicate the adhesion relationship between them using the following OCL statements given below.

```
context Warehouse-Staff :: managedBy : ID
derive : Warehouse-Manager.staffID

context Warehouse-Manager
inv: self.responsibleFor := Set(Warehouse-Staff.staffID)

context ManageStaff
inv : Warehouse-Staff->managedBy (Warehouse-Manager.staffID)
```

Example 9: In the case of conceptual views "`Lot-Movement`" (Fig. 4), the exclusive disjunction between `Internal-Lot-Movement` (stored goods change owners) and `External-Lot-Movement` (goods shipped outside the warehouse) can be show via the OCL statement "OR" between the relationships as shown in Fig. 4.

Example 10: Similarly, the exclusive disjunction between conceptual views `Customer-Logistics` (where customers provide/use their own logistics service provider to move goods out of the warehouses) and `LMS` (where customers utilize the warehouses' own logistics services), can be show via the OCL "OR" statement (Fig. 4).

6 XML Schema Views

An XML conceptual view can be transformed/mapped into a corresponding XML schema view (in XML Schema language) using the techniques we developed in [20, 21, 53, 54]. Table 1 provides a very brief overview of our transformation rule mechanism. Due to space limitation, we refer readers to [20, 21, 38, 39, 53, 54] for detailed descriptions of mapping Object-Oriented conceptual models to XML Schema.

Formally, let $\aleph_s^c$ denote the transformation mechanism which can translate a set of objects, their relationships and constraints into a set of simpleType/complextType definitions for XML elements/attributes and associated element/attribute constraints in the XML Schema language.

***Definition 3*: An XML schema view V^s** is a triple $V^s = (V^s_{name}, V^s_{simpleType}, V^s_{complexType}, V^s_{constraint})$, where V^s_{name} is the name of the XML schema view V^s, $V^c_{simpleType}$, $V^s_{complexType}$ are simple and complex type definitions for XML elements/attributes, and $V^s_{constraint}$ is a set of constraints upon the defined XML elements/attributes. Here, $V^c_{simpleType}$, $V^s_{complexType}$, and $V^s_{constraint}$ are expressed in the XML Schema Language, and V^s_{name} is also the name of the resulting XML schema file, i.e., a valid W3C XML document name.

***Definition 4*: Given an XML conceptual view** $V^c = (V^c_{name}, V^c_{obj}, V^c_{rel}, V^c_{constraint})$, $V^s = (V^s_{name}, V^s_{simpleType}, V^s_{complexType}, V^s_{constraint})$ is a **valid XML schema view of V^c**, if and only if V^s is transformed from V^c by $\aleph^c_s$. That is, $\aleph^c_s : V^c \rightarrow V^s$.

To get the flavor of such a transformation, in the following, we exemplify how the OCL constraints at the conceptual level are mapped to the XML schema level.

Table 1. Transformation rules from conceptual view to schema view

Conceptual view (in UML)	Schema view (in XML Schema)
View class and View class hierarchy (View-of-view)	Complex type construction (`xsd:complexType`) and complex type/element hierarchies
View class constraints (e.g. Ordering, homogenous composition etc.	Built-in XML Schema constraints.
View attributes	XML Schema simple types (integer, float, string, date, time etc.)
View attribute constraints (e.g. Object Identifier (OID)	XML Schema constructs using "facet" and XML Schema specific element/attribute constraints
Attribute grouping (constructional contents such as set, bag)	XML Schema list (`NMTOKENS`, `IDREFS`, & `ENTITIES`), or new list types using <`xsd:list`> construct and union by using <`xsd:union`> construct.
Structural Relationships (IS-A, composition, association)	<`xsd:sequence`>, <`xsd:choice`> & <`xsd:all`> with <`xsd:extension`> / <`xsd:restriction`> and `ID`, `IDREF`/`IDREFS` or `KEY` and `KEYREF` constructs
Domain constraints	"facets" mechanism to create new types, apply restriction etc.
Uniqueness Constraint	XML Schema <`xsd:unique`> construct
Cardinality Constraint	XML Schema `maxOccurs`/`minOccurs` construct
Referential Constraint	XML Schema ID, `IDREF`/`IDREFS` or `KEY` and `KEYREF` constructs.
Stereotype <<view>>	Complex type construction (`xsd:complexType`) and complex type/element hierarchies.
Stereotype <<OID>>	Combination of XML Schema simple types ID, `IDREF`/`IDREFS` or `KEY` and `KEYREF` constructs and XML Schema unique constraint.

Example 11: The ordered/unordered compositions (e.g. in Fig. 4, "`Lot-Master`" & "`Goods-Items`"), shown using the stereotype (`<<1>>`, `<<2>>`, …), are mapped using the <`xs:sequence`> construct; (a) at the conceptual view attribute level and (b) at the conceptual view class level. This is shown below in code listing 1.

Example 12: As shown in Example 9, the exclusive disjunction constraint (Fig. 4) between "`Internal-Lot-Movement`" and "`External-Lot-Movement`" can be mapped

XML Schema using the `<xs:choice>` schema construct, as shown below in code listing 2.

```
<!-- additional nesting -->
<xs:complexType  name="Composite_Object/attrbute">
    <xs:sequence>
        <xs:element name="Componant-A/attribute-a"
                    type="xs:OID"/>
            <!-- additional nesting -->
            <!-- additional nesting -->
    </xs:sequence>
</xs:complexType>
<!-- additional nesting -->
```

Code Listing 1: Code sample for example 11

```
<!-- additional nesting -->
<xs:element name="Lot-Movement">
    <xs:complexType>
        <xs:complexContent>
            <xs:extension base="LotMovementType">
            <!-- additional nesting -->
              <xs:choice>
                <xs:element name=
                    "Internal-Lot-Movement"
                        type="InternalLotMovementType"/>
                <!-- additional nesting -->
                <xs:element name=
                    "External-Lot-Movement"
                        type="ExternalLotMovementType"/>
                <!-- additional nesting -->
              </xs:choice>
            <!-- additional nesting -->
            </xs:extension>
        </xs:complexContent>
    </xs:complexType>
</xs:element>
<!-- additional nesting -->
```

Code Listing 2: Code sample for example 12

Example 13: The unique constraint (the `<<OID>>`) in `Staff` can be mapped to the view schema as shown in the code fragment below (Fig. 3 & 4).

```
<!-- additional nesting -->
  <xs:complexType name="staffType">
        <xs:sequence>
            <xs:element name="staffID" type="OIDType"/>
            <!-- additional nesting -->
            <xs:element name="managedBy">
                <xs:key name="manager">
                    <xs:selector/>
                    <xs:field xpath="staffID"/>
                </xs:key>
            </xs:element>
        </xs:sequence>
    </xs:complexType>
    <!-- additional nesting -->
    <!-- additional nesting -->
    <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>
  <!-- additional nesting -->
```

7 XML Instance Views

An XML instance view is an instantiated imaginary XML document which conforms to the XML schema view defined at the schema level. An instance view can be used for many purposes such as database views, materialised semantic Web-views, etc., and can be generated from simple projection of selected document tags to provide dynamic window into complex heterogenous documents. Based on how these docu-

ments are constructed/behave, we classify XML instance views into three categories [35], namely, *derived instance views*, *constructed instance views*, and *triggered instance views*.

XML instance views can be constructed via a native XML query language (e.g. XQuery [48], SQL 2003 [6]) or some specific algorithms such as Ontology Extraction Methodology (OEM) [52] (the MOVE [51] system), which can be achieved by the transformation of conceptual operators $\hat{\lambda}$ defined at the conceptual level [37] into a language-specific query fragment $\hat{\lambda}_{Query}$, say FLWOR expressions in XQuery etc. In a related work [52], the authors have shown how conceptual operators can be transformed and mapped to query algorithms in the MOVE system.

In this paper, we briefly discuss XQuery as the document view construct language. Here, we choose XQuery as document view constructor as it is gaining momentum as the language of choice for XML databases and repositories. It is a functional language [9] and its functionalities are comparable to early/mid SQL standards (in the relational model). In addition it is well-suited for our purpose better than other XML query languages (such as XPath or XSLT) as; (a) XQuery is easy to read and write in comparison XPath and XSLT, (b) in direct contrast to XQuery's powerful For-Let-Where-Order-Return (FLWOR) expression, XPath/XSLT are purely presentation oriented, (c) XQuery provides User-Defined Functions (UDF) and variable binding and (d) XQuery support XML Schema. Table 2 below shows a brief summary of such mappings (including some built-in XQuery operators).

Table 2. Transformation of conceptual operators to instance view query expressions

Conceptual Operator	XQuery Equivalent	Illustrative Example Code

Union	Built-in operator `(union` or `|)`	`let $a := document ("d1.xml")`
		`let $b := document ("d2.xml")`
		`return <union-result> {$a | $b}</union-result>`

Intersection	Built-in operator `intersect`	`let $a := document ("d1.xml")`
		`let $b := document ("d2.xml")`
		`return <intersect-result>`
		`{$a intersect $b}</intersect-result>`

Difference	Built-in operator `except`	`let $a := document ("d1.xml")`
		`let $b := document ("d2.xml")`
		`return <difference-result>`
		`{$a except $b}</difference-result>`

Cartesian Product	In any FLWOR expression, when more than one variable is bound to a `FOR` clause (no where clause)	`for $a in document("d1.xml"),`
		`$b in document("d2.xml")`
		`return <simple-CP-example> <a>{$a}</a>`
		`<b>{$b}</b>`
		`</simple-CP-example>`

Join		`for $a in document("d1.xml"),`
		`$b in document("d2.xml")`
		`where $a/ID = $b/ID`
		`return <simple-JOIN-example> <d1>{$a}</d1>`
		`<d2>{$b}</d2> </simple-JOIN-example>`

Project		Any valid FLWOR expression. Simplest PROJECT expression is `doc("d1.xml" )`
Select, Rename, Restruct(ure)	User defined functions and/or Built-in operates	Any valid FLWOR expression, with selective/conditional clause

Formally, we can have the following definition for XML instance views.

Definition 5*:* Given an XML schema view $\boldsymbol{V^s}$ and a set of XML source documents *S*, $\boldsymbol{V^i}$ is called a **valid XML instance view of $\boldsymbol{V^s}$ over *S*** if and only if $\boldsymbol{V^i}$ is a well-formed XML document, extracted from *S* by certain query operators in λ_{Query}, and conforming to the XML schema $\boldsymbol{V^s}$.

Example 14: As described in Examples 1, 7 and 8, the conceptual view operators of the view "`Warehouse-Manager`" can be mapped to the document view construct (XQuery expression) as shown below in the code segment.

```
for       $role in document ("staff.xml")//Role
where     $role = "Manager"
return <Warehouse-Manager> {$role} </Warehouse-Manager>
```

Example 15: If a new domain requirement exists to add new conceptual view Management-Memo send to all "`Warehouse-Manager`", we can do that using Cartesian Product conceptual operator. Here in document level, we can map that conceptual operator to the following XQuery expression;

```
for       $a in document("Warehouse-Manager.xml")//Role,
          $b in document("Management-Memo.xml")//Message
return <send-memo> <W-Mgr>{$a}</W-Mgr>
            <Mgr-Memo>{$b}</Mgr-Memo></send-memo>
```

8 Conclusion and Future Work

In this paper, we presented a view model with conceptual and schema extensions. We also presented a view constraint specification model (OCL based) and conceptual operators. We showed how conceptual views are mapped to its schema and document level equivalent, with illustrated case study examples.

For future work, some further issues deserve investigation. First, the investigation of a formal mapping approach to conceptual view constraints, to automate the view constraint model transformation between the conceptual and schema level constraints. Second, the automation of the mapping process between conceptual operators to various query language expressions (such as XQuery). Third, is the investigation into dynamic perspectives of the conceptual view formalism that can be applied to traditional data, Semantic Web and web service domains.

References

1. S. Abiteboul, "On Views and XML," Proc. of the 18th ACM PODS '99, USA, 1999.
2. S. Abiteboul and A. Bonner, "Objects and Views," Proc. of the Int. Conf. on Management of Data (ACM SIGMOD '91), 1991.
3. S. Abiteboul, R. Goldman, J. McHugh, V. Vassalos, and Y. Zhuge, "Views for Semistructured Data," Workshop on Management of Semistructured Data, USA, 1997.
4. S. Abiteboul, et al., "The Lorel Query Language for Semistructured Data," *Int. Journal on Digital Libraries*, vol. 1, pp. 68-88, 1997.
5. AIIM, "The ECM Association (http://www.aiim.org/index.asp)," AIIM, 2005.
6. ANSI and ISO, "ANSI - SQL 2003," ANSI / ISO 2003.

7. H. Balsters, "Modelling Database Views with Derived Classes in the UML/OCL-framework," The UML '03, USA, 2003.
8. R. G. G. Cattell, et al., "The Object Data Standard: ODMG 3.0," Morgan Kaufmann, 2000, pp. 300.
9. D. D. Chamberlin and H. Katz, *XQuery from the experts : a guide to the W3C XML query language*. Boston: Addison-Wesley, 2003.
10. E. Chang and T. S. Dillon, "Integration of User Interfaces with Application Software and Databases Through the Use of Perspectives," 1st Int. Conf. on ORM '94, Australia, 1994.
11. E. Chang, et al., "A Virtual Logistics Network and an e-Hub as a Competitive Approach for Small to Medium Size Companies," Int. Human.Society@Internet Conf., Korea, 2003.
12. E. Chang, et al., "Virtual Collaborative Logistics and B2B e-Commerce," e-Business Conf., Duxon Wellington, NZ, 2001.
13. E. J. Chang, "Object Oriented User Interface Design and Usability Evaluation," in *Department of CS-CE*: La Trobe University, Melbourne, Australia, 1996.
14. Y. B. Chen, T. W. Ling, and M. L. Lee, "Designing Valid XML Views," Proc. of the 21st Int. Conf. on Conceptual Modeling (ER '02), Tampere, Finland, 2002.
15. S. Cluet, P. Veltri, and D. Vodislav, "Views in a Large Scale XML Repository," Proc. of the 27th VLDB Conf. (VLDB '01), Roma, Italy, 2001.
16. C. J. Date, *An introduction to database systems*, 8th ed. New York: Pearson/Addison Wesley, 2003.
17. T. S. Dillon and P. L. Tan, *Object-Oriented Conceptual Modeling*: Prentice Hall, Australia, 1993.
18. J. H. Doorn, L. C. Rivero, and (eds), *Database Integrity: Challenges & Solutions*: Idea Group Publishing, Hershey, PA, 2002.
19. R. Elmasri and S. Navathe, *Fundamentals of database systems*, 4th ed. New York: Pearson/Addison Wesley, 2004.
20. L. Feng, E. Chang, and T. S. Dillon, "A Semantic Network-based Design Methodology for XML Documents," *ACM Trans. on IS (TOIS)*, vol. 20, No 4, pp. 390 - 421, 2002.
21. L. Feng, E. Chang, and T. S. Dillon, "Schemata Transformation of Object-Oriented Conceptual Models to XML," *Int. J. of Comp. Sys. Sci. & Eng.*, vol. 18(1), pp. 45-60, 2003.
22. V. Gopalkrishnan, Q. Li, and K. Karlapalem, "Star/Snow-flake Schema Driven Object-Relational Data Warehouse Design and Query Processing Strategies," 1st First Int. Conf. on Data Warehousing and Knowledge Discovery (DaWaK '99), Florence Italy, 1999.
23. I. Graham, A. C. Wills, and A. J. O'Callaghan, *Object-oriented methods : principles & practice*, 3rd ed. Harlow: Addison-Wesley, 2001.
24. A. Gupta, I. S. Mumick, and (eds), *Materialized views: techniques, implementations, and applications*: MIT Press, 1999.
25. ITEC, "iPower Logistics (http://www.logistics.cbs.curtin.edu.au/)," 2002.
26. W. Kim, "Research Directions in Object-Oriented Database Systems," Proc. of the 19th ACM Sym. on Principles of Database Systems, Nashville, Tennessee, USA, 1990.
27. W. Kim and W. Kelly, "Chapter 6: On View Support in Object-Oriented Database Systems," in *Modern Database Systems*: Addison-Wesley Publishing Company, 1995, pp. 108-129.
28. H. Liefke and S. Davidson, "View Maintenance for Hierarchical Semistructured," Proc. of the Second Int. Conf. on DaWak '00, London, UK, 2000.
29. Lucie-Xyleme, "Xyleme: A Dynamic Warehouse for XML Data of the Web," Int. Database Engineering & Applications Symposium (IDEAS '01), Grenoble, France, 2001.
30. B. Ludaescher, et al., "View Definition and DTD Inference for XML," Post-ICDT Workshop on Query Processing for Semistructured Data and Non-Standard Data Formats, 1999.
31. M. K. Mohania, K. Karlapalem, and Y. Kambayashi, "Data Warehouse Design and Maintenance through View Normalization," 10th Int. Conf. on DEXA '99, Italy, 1999.

32. V. Nassis, et al., "Conceptual and Systematic Design Approach for XML Document Warehouses," *Int. J. of Data Warehousing and Mining*, vol. 1, No 3, 2005.
33. OMG-OCL, "UML 2.0 OCL Final Adopted specification (http://www.omg.org/cgi-bin/doc?ptc/2003-10-14)," OMG, 2003.
34. OMG-UML™, "UML 2.0 (http://www.uml.org/#UML2.0)," 2003.
35. R.Rajugan, E. Chang, T. S. Dillon, and L. Feng, "XML Views: Part 1," 14th Int. Conf. on Database and Expert Systems Applications (DEXA '03), Prague, Czech Republic, 2003.
36. R.Rajugan, et al., "Engineering XML Solutions Using Views," The 5th Int. Conf. on Computer and Information Technology (CIT '05), Shanghai, China, 2005.
37. R.Rajugan, E. Chang, T. S. Dillon, and L. Feng, "A Layered View Model for XML Repositories & XML Data Warehouses," The 5th Int. Conf. on CIT '05, China, 2005.
38. R.Rajugan, E. Chang, T. S. Dillon, and L. Feng, "XML Views, Part III: Modeling XML Conceptual Views Using UML," 7th Int. Conf. on Ent. IS (ICEIS '05), Miami, USA, 2005.
39. R.Rajugan, et al., "Semantic Modelling of e-Solutions Using a View Formalism with Conceptual & Logical Extensions," 3rd Int. IEEE Conf. on INDIN '05, Australia, 2005.
40. M. Rafanelli and (ed), "Multidimensional Databases: Problems and Solutions," Idea Group Inc., 2003, pp. 474.
41. R. Steele, W. Gardner, R.Rajugan, and T. S. Dillon, "A Design Methodology for User Access Control (UAC) Middleware," Proc. of the IEEE Int. Conf. on EEE '05, 2005.
42. R. Volz, D. Oberle, and R. Studer, "Views for light-weight Web ontologies," Proc. of the ACM Symposium on Applied Computing (SAC '03), USA, 2003.
43. W3C-RDF, "Resource Description Framework (RDF), (http://www.w3.org/RDF/)," 3 ed: The World Wide Web Consortium (W3C), 2004.
44. W3C-SW, "(http://www.w3.org/2001/sw/)," W3C, 2005.
45. W3C-SW, "Semantic Web, (http://www.w3.org/2001/sw/)," W3C, 2005.
46. W3C-WS, "Web Services Activity, (http://www.w3.org/2002/ws/)," W3C, 2002.
47. W3C-XML, "Extensible Markup Language (XML) 1.0, (http://www.w3.org/XML/)," 3 ed: The World Wide Web Consortium (W3C), 2004.
48. W3C-XQuery, "XQuery 1.0: An XML Query Language," in *XML Query Language (XQuery)*: The World Wide Web Consortium (W3C), 2004.
49. W3C-XSD, "XML Schema," vol. 2004, 2 ed: W3C, 2004.
50. J. B. Warmer and A. G. Kleppe, *The object constraint language : getting your models ready for MDA*, 2nd ed. Boston, MA: Addison-Wesley, 2003.
51. C. Wouters, T. S. Dillon, J. W. Rahayu, E. Chang, and R. Meersman, "Ontologies on the MOVE," 9th Int. Conf. on Db. Sys. for Adv. Apps. (DASFAA '04), Korea, 2004.
52. C. Wouters, Rajugan R., et al., "Ontology Extraction Using Views for Semantic Web," in *Web Semantics and Ontology*, USA: Idea Group Publishing, 2005.
53. R. Xiaou, et al., "Mapping Object Relationships into XML Schema," Proc. of OOPSLA Workshop on Objects, XML and Databases, 2001.
54. R. Xiaou, et al., "Modeling and Transformation of Object-Oriented Conceptual Models into XML Schema," 12th Int. Conf. on DEXA '01 2001, 2001.
55. Y. Zhuge and H. Garcia-Molina, "Graph structured Views and Incremental Maintenance," Proceeding of the 14th IEEE Conf. on Data Engineering (ICDE '98), USA, 1998.