Modeling Views in the Layered View Model for XML Using UML

R. RAJUGAN, THARAM S. DILLON
eXel Lab, 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, Perth, Australia
Email: Elizabeth.Chang@cbs.cutin.edu.au

LING FENG
Faculty of Computer Science, University of Twente, The Netherlands
Email: ling@ewi.utwente.nl

Received: January 28 2006; revised: March 26 2006

***Abstract*— In data engineering, view formalisms are used to provide flexibility to users and user applications by allowing them to extract and elaborate data from the stored data sources. Conversely, since the introduction of EXtensible Markup Language (XML), it is fast emerging as the dominant standard for storing, describing, and interchanging data among various web and heterogeneous data sources. In combination with XML Schema, XML provides rich facilities for defining and constraining user-defined data semantics and properties, a feature that is unique to XML. In this context, it is interesting to investigate traditional database features, such as view models and view design techniques for XML. However, traditional view formalisms are strongly coupled to the data language and its syntax, thus it proves to be a difficult task to support views in the case of semi-structured data models. Therefore, in this paper we propose a Layered View Model (LVM) for XML with conceptual and schemata extensions. Here our work is three-fold; first we propose an approach to separate the implementation and conceptual aspects of the views that provides a clear separation of concerns, thus, allowing analysis and design of views to be separated from their implementation. Secondly, we define representations to express and construct these views at the conceptual level. Thirdly, we define a view transformation methodology for XML views in the LVM, which carries out automated transformation to a view schema and a view query expression in an appropriate query language. Also, to validate and apply the LVM concepts, methods and transformations developed, we propose a view-driven application development framework with the flexibility to develop web and database applications for XML, at varying levels of abstraction.**

***Index Terms*— View formalisms, data modeling, XML, UML/OCL, XML views, OO conceptual models, conceptual views, schema views, instance views, Layered View Model (LVM)**

I. INTRODUCTION

Electronic data publication and dissemination is a complex and challenging task. Only in the past decade, such publications of data are made easier, mostly due to the success of Internet and the web technologies. Many people see published or distributed data as web pages, electronic documents, web (database) query results etc. Such web data, in comparison to processed structured data (such as in a relational database system) are usually unstructured, unorganized, schemaless and increasingly exponentially. In addition, there is a growing trend to produce and publish vast amount of scientific and business data by automatically generating volumes of data by combining one or more distributed data sources, stored dispersedly over the Internet, for the consumption of both human and machines. In this context, it is important to provide database like features [1], such as representing, efficient querying and retrieving techniques, efficient storage and versioning, to such large amount of data, to be of any use . Conversely, such large amount web data have generated enormous research interest and demand on traditional database systems and created new era of data models and technologies, such as (web) data warehouses, ontologies, medication systems, middleware, etc. [1], [2], [3], [4], [5].

In the new era of electronic data publication, meaningful information retrieval for a given context has proved to be a challenging task due the vast amount of unstructured and uncategorized data available to the users. The majority of this data accumulated from the World Wide Web (WWW) [6] and the shared data repositories is dispersed stored over the Internet [1]. The exponential growth of the web and its technologies, and the growing number of information sources gathered or stored at multiple locations, created the problem of standardization vs. customization, where anyone can represent everything according to one's own representation and annotations. This triggered a series of new research directions such as metadata description languages, search engines, and knowledge representation techniques for the web to organize, describe, manage and disperse information in an orderly manner using the web as the new medium. To this effect, many initiatives have been proposed, namely; (a) the formation of new standard bodies and consortium, to define and promote standards for the web, such as the WWW consortium [6], ebXML (www.ebxml.org), OASIS (http://www.oasis-open.org) etc., (b) new data models (HTML [7], XML [8], XML Schema [9], RDF [10], OWL [11]), etc., (c)transformation and query languages proposals (e.g. XQuery [12] , RDQL [13], XSLT [14]) and (d) new technological frameworks (e.g. Web Services [15]) and (e) new knowledge representation paradigms such as the Semantic Web [16].

A. XML

The eXtensible Markup Language (XML) is emerging as the core data representation (and storage) standard for the web, from annotated web pages to Semantic Web and ontology bases. Since its adoption by the W3C as a standard in 1996, which was derived from its predecessor SGML, XML was originally intend to be a simple, yet meaningful data description and encoding language for electronic data representation, publishing and dissemination over the web, readable for both human and machines. Its tag-based, self descriptive, semi-structured data model provides facilities for user-defined tags and description that enables authoring of XML documents easier and customizable, yet allows external readers (human, agents and applications) to interpret or integrate them correctly. Due to this, the core data model for majority of the new and emerging web technologies, such as XML Schema, XQueryX, Web Databases, Web Services, Semantic Web, web protocols (e.g. SOAP, XAML) are based on XML.

XML, from a data engineering point of view, it can be considered as semi-structured. From a database point of view, it can be considered as a document than an inter-connected collection of data elements. Since it is self-descriptive, tag descriptions embedded within the document, thus enabling the reader (human or machine) to interpret the content without a need for managed metadata repositories or metadata schema. But, XML document by itself cannot provide validity to a document (i.e. does the document is lexically correct as the author intended?). For this purpose, the W3C proposed Document-Type-Definition (DTD) and its successor XML Schema (XSD).

DTD (and XSD) contain user-defined element type definitions and attribute descriptions to describe an XML document, its structure and the meaning of it content. The replacement of DTD is the XML Schema, which is richer in vocabularies and considerable extends the capabilities of DTD for defining and constraining the content of XML documents [17] [18]. The XSD, which itself is represented in XML, provides a set of advanced constructs, such as XML document usage, meaning, relationships and a rich set of constraints, that can be grouped into three groups of twelve components, to define and constraint XML documents. One of the main features of XML/XML Schema is the separation of document content from its presentation and/or structure.

B. Modeling XML Data

In the context of data engineering, though XML is rich in data modeling constructs, XML focus on syntactic structure than data semantics and their relationships [17] [18]. Due to this nature, modeling and representing real-world objects using XML/XML Schema proves to be a challenging task. Since Object-Oriented (OO) conceptual modeling [19] [20] is capable of modeling real-world objects, relationships, constraints and constructs, focusing on semantic, structural and dynamic aspects at varying level of abstraction, which is precise and comprehensible to the users, it is interesting to study OO modeling approaches for XML data/document modeling.

In practice, an OO model provides the blue print of a system or problem being considered. From a data engineering point of view, OO has been successfully applied to solve complex problems such as real-time databases to data warehousing and OLAP queries. The success of the OO model is mainly due to its ability capture, model and communicate a problem in question with needed level of abstraction using industry standard notations and techniques.

Similarly, for structured data (noticeably the relational data), the database technologies have proven their success in many areas, such as models, efficient storage and retrieval techniques, performance improvement techniques, model automation and transformation tool etc. Unlike the semi-structured data (i.e. XML) models, for the structured data, data storage, retrieval and manipulation have evolved to provide performance and ROI that is acceptable to industry and enterprise standards. Thus, it is interesting to investigate OO approaches to traditional database concepts, theory and practice, in the context of web and XML. Here, this paper investigates a view model for XML and its applications in designing XML e-solutions.

C. Databases, Views and XML

In data engineering, the notion of views is essential in databases, as it allows various users to see data from different viewpoints. This is made possible by data ”abstraction” and data ”partitioning” provided by the view mechanism in a language specific, well-defined data model. Since the introduction of the successful view mechanism for the relational data model [21], [22], motivation for views has changed over the last two decades. At present, views are widely used in:

1) user access and user access control applications
2) defining user perspectives/profiles [23]
3) designing data perspectives
4) dimensional data modeling [24], [25], [26]
5) providing improved performance and logical abstraction (materialized views) in data warehouse/OLAP and web-data cache environments [24], [27]
6) web portals & profiles
7) extraction of document structures or sub graphs in semi-structured documents [5]
8) Semantic Web, for sub-ontology or Ontology views [28], [29]

From this list, it is apparent that the uses and applications of views are realized more than their initially envisioned by Codd et al. [30](i.e., data elaboration vs. the 2-Es; data Extraction and Elaboration), with extensive research being carried out by both researchers and industry to improve view design, construction, and performance. But, the view specification and definition still remains as a data language and model dependent low-level (instance) construct. Conversely, in conceptual modeling paradigms such as OO or E-R/DFD, the modeling notation and/or languages provide no semantics to capture view formalisms at the conceptual (or logical) level.

Therefore, given the shortcomings of the existing view mechanisms and the growing popularity of XML in traditional structured data domains, it becomes essential to have a view mechanism that allows database designers to support and maintain existing applications, while designing new applications for emerging data standards and languages to utilize XML data/documents. Thus, in adapting or extending database concepts to XML (e.g. views) domain, there exist few challenges that need to be considered. They are:

1) Dual property: in XML, for a given (view) query expression, it should be able to construct result sets from stored both data-centric and document-centric XML documents without modifications.
2) Heterogeneity: in contrast to relational data, where all tuples in relations are of the same structure, XML provides varying and (non-atomic) complex structures. Therefore, in a stored XML document, depending upon such structures and embedded constraints (reference, choice, all, sequence etc), sub-elements of XML hierarchy may be missing and/or repeated. At present, many existing XML view definition languages need complex data definition and specification formalisms to perform such tasks.
3) Uniqueness: each tuple in a relation can be uniquely accessed due to key-base constraints (primary key, foreign key) defined for them. In the case of XML, its semi-structured nature prevents similar constraints from being placed. But in comparisons with relational models, defining and validating such a key mechanism for XML is a computationally expensive and complex task, as unlike relational, the XML structure is hierarchical. Also, unlike relational, XML is not dependent on keys and normalization but on node hierarchy and ordering. But there exists some work on indexing, ”normalization” for XML and defining key-based constraints to improve query processing for XML [31].
4) Ambiguity: an XML document, due to its self-descriptiveness, can be vague in structural definition. Therefore, one document may have multiple schemas defined. Therefore, querying and defining views for such documents is a difficult task, and thus view definition may vary from one language to another.
5) Ordering: data extraction and elaboration in XML (and in many semi-structured data) documents should keep the original order of structures. Thus, query language (and views) should be aware of such constructs and process them accordingly. Again, at present, in the context of view definition and querying, due to non-defined standards, tasks such as specifying and processing ordering vary from one view definition language to another.
6) Logical or schema constraints: in many XML documents, constraints such as element repetition, ordering etc. are only visible and constrained at the schema level than being embedded within the document elements. Thus, in such circumstances, the view definition (and the query) language should be able to access and process such logical level constructs to keep the XML document semantics intact.
7) Customization: most existing view definition languages provide view customization only at the instance (or result) level. They do not allow customization of XML document structures at the higher level (e.g. schema) that may be required for processing some queries and/or document constraints (e.g. model integration, abstract query processing, etc.).
8) Language independence: most existing XML view mechanisms do not support view definition that is independent of view specification languages, thereby lacking the flexibility required in extracting hierarchical data structures without loss of syntactical XML document semantics that is required for XML (and in general for semi-structured data) and
9) View validity: Another challenging task of XML view is its validity. Using valid XML document as stored/base document does not necessarily guarantee a valid XML view/(s) [32]. This is because in relational or OO database systems, there is a fixed model and a catalogue of defined data types (domain) which can validate views. In XML there is no fixed data model for XML documents.

Therefore it is apparent that the XML differ from traditional data models on (a) data format, (b) data structure and (c) data organization. Thus, in summary, it is clear from the aforementioned that a view mechanism for XML may be benefit more if:

- it is independent of the view specification language,
- it provides view specification (and possibly definition) at a higher level (e.g. schema) than at the instance level,
- the view specification and definition are native to XML document structure and semantics,
- it adheres to generic definition that is deployable in a varity of present and emerging XML technologies and storage models, as XML (and semi-structured data) standards are an on going process,
- it supports XML data extraction and (preferable) elaboration at varying levels of abstraction, such as instance, query and schema levels.

The next section outlines some ideal characteristics for an XML view mechanism based on the challenges.

D. Properties of an Ideal View Mechanism for XML

A structured data item, e.g. a person’s first name conform to one system wide data type (or schema type) that defines its data format, domain and possible values. These structured data items are usually singular (or simple) or singled valued. Also, such data values can be accessed via query language, which first determined the data type (or schema) from the global schema and execute the query against the data sets (or tuples). But advancements in OO and database technologies have enabled one to have complex data items (or Abstract Data Types (ADT)) and provide techniques to access such complex data structures to be accessed via query languages [19]. The popularity and the efficiency of relational model have made the structured data as the de-facto standard for many Information Systems. The success of relational model evident from two facts; (a) though successful in modeling and programming, unpopularity native OO databases and (b) successful adaptation of OO (logical) data models using core relational database engines.

Given these facts, it is essential to investigate the core properties of a native view mechanism for XML. Here, we outline some of these envisaged properties of an ideal view mechanism for XML that is being considered in this paper. They are:

- A view mechanism for XML should be independent of a fixed data model, and rely on dynamic XML Schema/DTD like mechanism for data description and elaboration. This is a contradistinction to some of early work on XML views, such as [33], where authors argue that an XML view model should depend on ODMG [34] like structured data model. The argument for this is that, XML is well adopted and accepted due to its data model independence and the rich data description (semantics) provided by its self-descriptive tags. Adapting an ODMG like data model for XML and XML views may lose its purpose and introduce complex schema (or logical wrapper) processing and management, resulting in loss of data semantics, meaning and performance in various applications.
- A view mechanism for XML should be independent of one particular query language (e.g. SQL) or propriety data languages, and should be able to support multiple query languages as, one core purpose of XML is interoperability, where one source XML data segment/document may need to be instantiated at multiple target IS sites, by multiple query/data languages, generating similar data sets. Also, to accommodate fast emerging, robust and efficient query and data (or domain) languages (e.g. LOREL [35], RDQL [13], SPARQL[36], SQL '03 [37] etc.), the view model (and the view instantiation) should be independent of one specific query/data language.
- Typically, an XML view may be a base for defining other views (view of a view) and may have to stratify queries issued against it. Thus, a view mechanism for XML should provide validation [32] in addition to well-formed view instances. That is, a XML view instance should be a "self-contained" or "mini" XML document with its own validation mechanism (such as view DTD or Schema), as unlike in structured data views, there exists no global schema to constraint and validate XML view instances.
- The XML view instance should not only provide individual data elements, but also the sub-tree hierarchy (without loss of data semantics and integrity) from the original stored document/(s) structure.
- XML data items inherently carry many item specific constraints such as simple/complex content, the notion of ordering and choice and item specific constraints such as "facet".
- A view mechanism for XML should support easy integration into existing and new XML applications and support construction of architectural constructs such as data cubes, dimensional data models and XML (data) interfaces.

The intension of the view mechanism proposed here is to:

1) provide a generic view mechanism for XML that can be utilized in the present and upcoming XML technology standards environments (e.g. XML databases, web services, ontologies),
2) support specification and definition of views at a higher level of abstraction than at the document or instance level,
3) provide view validity without using external validates and/or applications. That is, the view mechanism should utilise XML Schema like mechanism to validate constructed views and
4) support MDA like initiatives (i.e. platform independent view mechanism) for XML solutions.

Also, it is NOT the intention of this paper to propose a new view standard for XML or extensions to XML query languages. Rather, the focus is to provide a view mechanism for XML at the conceptual, schema, and document levels by means of OO conceptual modeling. To help illustrate the concepts, a real-world case study of a fictitious global logistic company called LWC & e-Solutions Inc., e-Sol in short, is used and described in detail in section III below.

In this section, we presented issues such as, data models, model requirements and challenges that are important in the context of XML. The rest of this paper is organized as follows. In section II, we review some of the early works in view related domain, followed by section III, which describes an example case study used to illustrate our concepts and notions in this paper. This is followed by the introduction and the semantics of our Layered View Model (LVM) for XML in Section IV. Section V presents modeling issues related to the views in the LVM, followed by a detailed discussion on modeling views in the LVM using UML/OCL in section IV. Section VII describes the schemata transformation methodology adopted in the LVM. Section VIII presents some potential real-world application scenarios that utilizes our LVM, followed by section IX, which concludes the paper and outline our future research direction.

II. Related Work

In this section, a discussion on view formalisms that are available today is presented, including some of the proposals for new semi-structured data view formalisms and constraint specification for such views. Existing view mechanisms can be grouped into four categories, namely; (a) early (namely relational) view formalisms, (b) view formalisms for Object-Oriented (OO) paradigm, (c) semi-structured (namely XML) view formalisms and (d) views for SW. A detailed discussion on these view models can be found in our work in [39], [38]. Here we only look at views for semi-structured data.

A. *Views for Semi-Structured Data*

Since the emergence of semi-structured data models, namely XML [8], the need for semi-structured data models to be independent of the fixed data structure and description go against the fundamental properties of the classical structured data models. However, since its introduction, much work has been directed towards providing traditional database-like features such as data access, query language, views, etc. to XML in an ongoing process.

XML documents which are tag-based and self-describing data documents represent a hierarchical tree structure. At the conceptual level, they can be visualised as hierarchical trees or graphs. An XML document is usually associated with a Document Type Definition (DTD) [40] or XML Schema [9] which is used to define and constrain the syntax and structure of a document. Though XML is usually considered as semi-structured data, there exist some differences between the two.

Many researchers have attempted to resolve view related issues in the semi-structured paradigm by using graph based [1], [41], [42] and/or OO based data models [5], [43], [44]. But, in all the above cases, as in the case of relational and OO, the actual view specification and definitions are available only at the lower levels of abstraction (implementation level) and not at the conceptual and/or logical level. This is one of the main issues of concern for views in the semi-structured (and XML) data models.

Semi-structured data (from XML to Semantic Web), unlike structured data do not follow a rigid, well-defined structure (as opposed to relational and OO data) and usually constrained and described by schemas. For example, the popularity of XML is due to its ability to support heterogeneity of data and structure for a given concept. That is, at the schema level, XML may define and constrain an XML document, while allowing XML document to have incomplete information. In such situations, defining and specifying views using the classical approach, that is, using a data manipulation or query language, may have allow for flexibility of dealing with incomplete or missing information. This is a complex task in comparison to writing simple view definitions.

Conversely, at a higher level of abstraction than the data language level, XML (and semi-structured) data can be easily visualized, defined, cataloged, queried and customized to suit one's need [1], [17], [18], [28], [45]. Thus, providing a view formalism for semi-structured data should consider specifying views at a higher level of abstraction than relational or OO views.

B. *Declarative XML Views*

One of the early discussions on XML views was by Serge Abiteboul [33] and later more formally by Sophie Cluet et al. [46]. They proposed a declarative notion of XML views. Abiteboul points out that:

1) a view for XML, unlike relational or OO views, should do more than just providing different presentation of underlying data [33]. 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;
2) XML views should use OO views as the foundation, should follow analogous to (his [47]) OO view concepts, such as virtual values, virtual classes and imaginary classes;
3) an XML view specification should rely on a data model (like ODMG [34] model) and a query language. Query language, he stats as the central issue for view definition, as query languages for XML are still at their initial stages of development or still evolving, and no view creation facilities are provided. W3C XPath [48] and XQuery [49] are such examples, together with the SQL 2003 standard;
4) an XML view specification should be more than relational and OO views as XML is deployed over web. Thus, web specific characteristics, such as replication, change notification, etc. should be part of the view specification;
5) an XML view should allow for dealing with incomplete information.

The above points in Abiteboul's work identify some of the core issues that are concerns for view formalism for XML. However, his proposition to couple XML to a structured data model (ODMG) and to query languages, in our opinion may restrict the capabilities of XML and may be considers as an extended OO model for XML than a native XML view formalism. Also, the problems that are associated with the mixed OO class and view hierarchy is still an issue here in these proposed XML views.

C. *View Mechanism in Xyleme*

In the works [44], [46], authors extend some of the concepts proposed in [33], and proposed a development of a view formalism for XML to support web scale XML data warehouse environment: the Xyleme [50], [51] project. In regards to the XML views, due to its scale, the authors propose a semi-automatic approach to view generation and they discuss in detail on how abstract XML paths/DTDs are mapped to concrete paths/DTDs. More formally an XML view definition in Xyleme is stated as [46]:

"*....A view defined by a set of pairs $< p, p' >$, called mappings, where p is a path in the abstract DTD and p' a path in some concrete DTD..*"

These concepts, which are implemented in the Xyleme project provides 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 modelling, these view concepts provide no support. The view model 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. Characteristics of Xyleme views include [44], [46], [50]:

(i) due to the scale of Xyleme data set, XML views enables the user to query a single structure that summarizes a domain of interest (as opposed to multiple schemas);
(ii) Xyleme views operates on web scale clusters (as opposed to relational or OO data), XML data gathered by web crawlers that are automatically classified by external tools;
(iii) a view query in Xyleme is defined by a union of many queries over many clusters. Usually, here, the views are constructed automatically;
(iv) the query language for XML views in Xyleme is a variant (extension) of the OQL [34] and also satisfy some of the XQuery features;
(v) the concepts related the view domain is considered as concrete and the concepts related to the XML view is considered abstract (e.g. DTD, values, etc.);
(vi) in regards to the properties of the XML view in Xyleme:
    - View definition is a OQL like
    - View is defined by a set if path pairs (or mappings) with a path in the abstract DTD and path in the concrete DTD
    - View schema is provided by an abstract DTD

As stated earlier, XML views in Xyleme provide one of the most comprehensive implementation architectures for constructing web scale XML views. It is scalable and based on core OO data model principles. However, given the scale of the Xyleme project, it is desirable to have language-neutral, high-level, view specification and definition formalism. This will arguably provide a better transformation and mapping formalism to the existing domain specific XML document structures and language-specific view definitions. Also, a high-level view specification formalism will provide view visibility to end users rather than semi-auto generated view specification code. Another desirable feature for Xyeleme is the utilization of XML Schema as opposed to DTD as XML Schema is better suited for such web scale task (added deceptions and vocabularies, available data descriptors, support for multi-schema documents, etc.).

D. Views in the ORA-SS Model

Object-Relationship-Attribute Model for Semi-Structured Data (ORA-SS) is one of the early works that looked at view specification at a high-level abstraction (as opposed to query language) using visual constructs was proposed in [32], [52]. This is also one of the first works that looks at using XML Schema and XQuery in specifying XML views. Here, XQuery is used as the view/query language over the views. The view definition includes an additional view declaration clause before XQuery expression.

The ORA-SS data model is a visual notation with schema, instance, and functional dependency and inheritance diagrams. It is comparable to the Extended-ER model with three basic concepts: object classes, relationship types and attributes. Here, an object class corresponds to an entity type an element in XML documents. A relationship type describes a relationship among object classes. Attributes are properties, and may belong to an object class or a relationship type. ORA-SS data model has four diagrams: the schema diagram, the instance diagram, the functional dependency diagram and the inheritance diagram. This is one of the first view models that support some of abstraction above the data language level.

The XML view in ORA-SS is done in the following steps:

(i) transformation of the stored XML documents into the ORA-SS model (diagram);
(ii) additional semantics that are not feasible to specify at the instance XML document level, but provided in the ORA-SS model are added, including: participation constraints of object classes and distinction between attributes of object classes and relationship types. Also, with end-user participation (manually) the following sub-tasks are performed:
    - identification of key attributes of each object class;
    - identification attributes that belong to object classes;
    - identification of relationship types among object classes;
    - identification of attributes of relationship types.
(iii) The development of a set of rules to guide the design of valid XML views. These include, model specific four visual transformation operations for creating XML views, namely, selection, projection, join and swap operation;
(iv) Algorithm driven validation for the XML views constructed.

This is one of the early works that elaborates on the conceptual model like semantics to define XML views. However, the drawback this approach includes:

(i) ORA-SS model is constructed from instance level semantics; that is, it represents and XML document as opposed to representing conceptual or logical semantics that may have lost or not present in the XML document. Thus, this work may be analogous to Query-by-Example tools in relational or work such as [53], [54], [55] where XML document semantics are used to create visual representation of the XML document semantics and used to construct XML views,
(ii) Though the view formalism provides semantic enrichment with user participation and explicit diagrammatic representation, it does not consider the addition of conceptual semantics or important constraints that are useful for the XML domain, such as ordered composition, exclusive disjunction, etc.,
(iii) View definitions are possible only if the source data is accurate and complete. No provision is given in the ORA-SS model to deal with missing or incomplete XML node, element or attribute values,
(iv) Validation requires external algorithms.

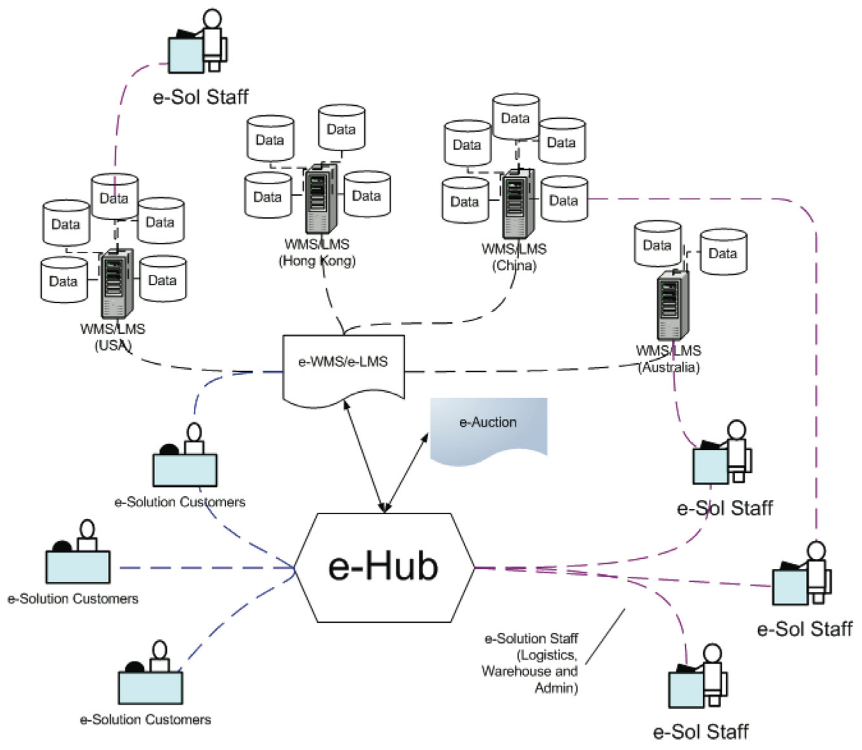

Fig. 1. e-Sol context diagram.

E. Technology Specific View Formalisms

In this section, we look at some of the XML view proposals that are the by-product of another concept or technological solution. These include: Active XML [4] view system and the MIX view [56], [57] system. Though not comprehensive enough to be considered as an XML view concept, they do address some of the issues associated with XML views. The following sections look at this research in some detail.

F. Active XML Views

As described above in Active View systems, the Active XML [4], [43] views are the result of using XML as the data model in the Active Views system. The Active Views system is built on top of Ardent Software XML repository based on the O_2 OODBMS system [58]. In the Active Views system, since view is presented as an object, it allows for both properties and methods. The Active Views system uses the OQL as a view definition language and the Lorel [35], [59] language as the query language over the views.

G. Views in the MIX System

The MIX View system [56], [57] is a by-product of developing web scale mediator systems. It is based on supporting mediator architecture to provide the user with an integrated view of the underlying heterogeneous information/data sources. The MIX system employs XML as the data exchange and integration medium between mediator components and the XML DTD to provide structural descriptions of the data.

The Query language of MIX system is Xmas [60] (XML Matching and Structuring Language) which is claimed to be a high-level, declarative query language. The Xmas provides: (a) object fusion and pattern matching on the input XML data and (b) grouping and order constructs for generating new integrated XML objects.

The interface to MIX is provided by the graphical user interface BBQ (Blended Browsing and Querying) [61] which support Query-by-Example operations. The output of a BBQ operation is a Xmas query. Though MIX system provides support for XML views, it is not a XML View by nature. It is a by-product to support data mediation for web-based information systems. Though powerful, the drawbacks include:(a) no standalone framework to support XML views and non-standard language/(s) used to query/manipulate data, (b)it does not utilize XML Schema, a semantically rich replacement for XML DTD and (c) the resulting views are not systematically validated.

H. Views for Semantic Web

In related work in Semantic Web (SW) [62] paradigm, some view mechanisms have been proposed in [28], [63], [64], where the authors present a RDF views with support for RDF [10] 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. The extension of this work (and other related projects) can be found at [65]. RDF is an object-attribute-value triple, where it implies object has an attribute with a value [18]. It only makes intentional semantics and not data modeling semantics. Therefore, unlike generic view mechanisms, views for such RDF (both logical and concrete) have no tangible scope outside its domain. In related

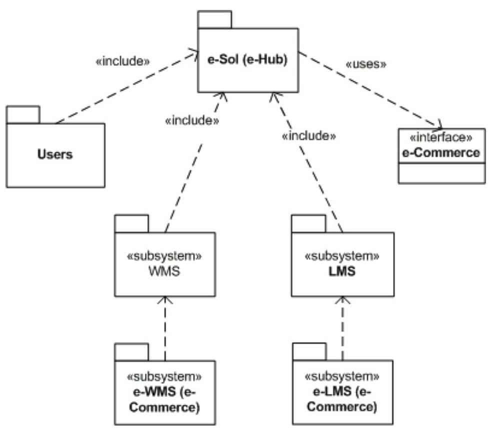

Fig. 2. e-Sol system overview.

area of research, the authors of the work proposed a logical view formalism for ontology [29], [66] with limited support for conceptual extensions, where materialized ontology views are derived from conceptual/abstract view extensions. Another area that is currently under development is the view formalism for SW meta languages such as OWL. Here, we only highlight one of view formalism that is under development for OWL, namely views for OWL in the "User Oriented Hybrid Ontology Development Environments" [67] project.

III. AN ILLUSTRATIVE CASE-STUDY EXAMPLE

The e-Sol Inc. aims to provide logistics, warehouse, and cold storage space for its global customers and collaborative partners, as illustrated in the context diagrams in Figs. 1 and 2. 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 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 . For ease of understanding, a simplified use-case diagram of the system is provided in Fig. 3.

In WMS (Figs. 6 and 7), 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.

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 utilize the XML conceptual, schema and instance views to design e-Sol at a higher level of abstractions to support changing business, environments, and data formats.

In this paper, we gradually build our concepts and notions using the example described above. A context diagram of the system is given in Fig. 2, followed by detailed WMS model in Fig. 6 - 7.

IV. THE LAYERED VIEW MODEL (LVM)

The Layered View Model (LVM) for XML is comprised of three different levels of abstraction, namely, conceptual level, logical or schema level, and document or instance level. Fig. 4 outlines this view model. The layered view model 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 [70], [71], 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 `bounded-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.

A. Conceptual Level

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 [72], or our developed XML-specific XSemantic net [17], 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

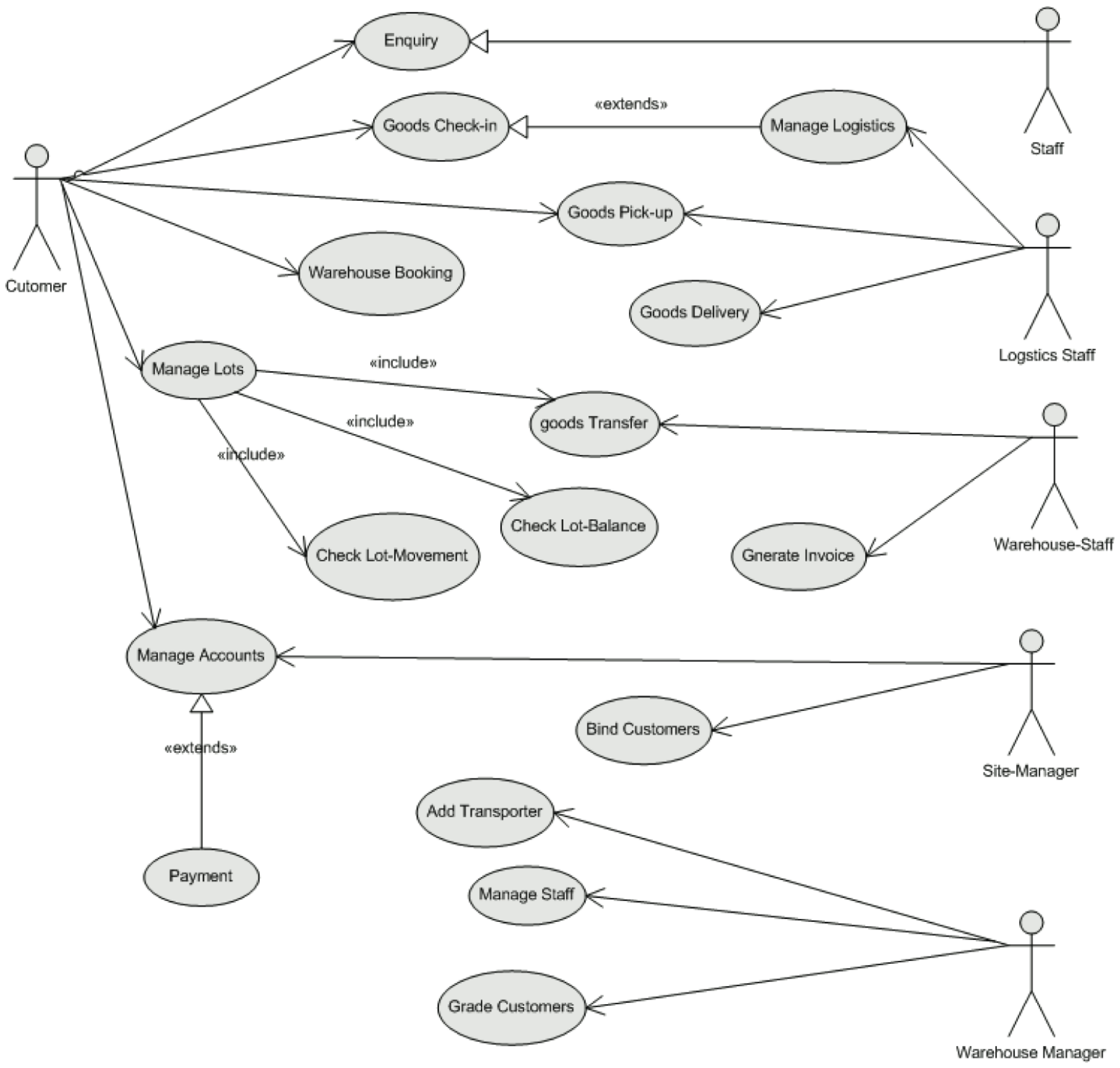

Fig. 3. A simplified e-Sol use-case diagram.

Meta-Object-Factory (MOF) [72], which can be either visual (such as UML class diagrams) or textual (in the case of UML/XMI models).

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.

Context is presented in UML using modeling primitives like object, attribute, relationship and constraint in this study. To enable the construction of a valid conceptual view from a context, we introduce the notion of *conceptual operator* λ . These conceptual level operators are comparable to relational operator in the relational model, but they operate on conceptual level objects and relationships.

Conceptual operators 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. The set of binary and unary operators provided here is a complete or basic set; i.e. other operators, such as division operator and compression operators can be derived from these basic set of operators.

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 λ be a set of conceptual operators. V^c =

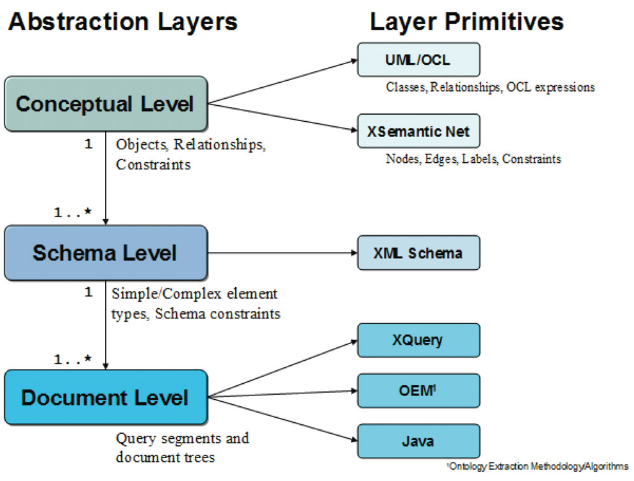
Abstraction Layers
Layer Primitives
UML/OCL
Classes, Relationships, OCL expressions
Conceptual Level
XSemantic Net
Nodes, Edges, Labels, Constraints
1
Objects, Relationships, Constraints
1..*
Schema Level
XML Schema
1
Simple/Complex element types, Schema constraints
1..*
XQuery
OEM¹
Document Level
Java
Query segments and document trees
¹Ontology Extraction Methodology/Algorithms

Fig. 4. The Layered View Model for XML (context diagram).

(V^c_{name}, V^c_{obj}, V^c_{rel}, $V^c_{constraint}$) is called a valid conceptual view of the context C, if and only if the following conditions satisfy:

(i) For any object ***o*** $\in$ V^c_{obj}, there exist objects $\exists o_1, o_2, ...o_n \in C_{obj}$, such that ***o*** = $\lambda_1\lambda_2....\lambda_m$ (o_1, o_2,....o_n) where λ_1 λ_2....λ_m $\in \lambda$. That is, ***o*** is a newly derived object from existing objects o_1, o_2,....o_n on in the context via a series of conceptual operators $\lambda_1\lambda_2....\lambda_m$ like select, join, etc.

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

(iii) For any hierarchical relationship ***rh*** $\in$ 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 ***ra*** $\in$ V^c_{rel}, there may exist a relationship between one or more V^c_{obj} and C_{obj}.

B. Logical Level

The middle scheme level Fig. 4 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 previous works such as; (i) UML to XML Schema [18] and (ii) XSemantic nets to XML Schema [17]. 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).

Formally, let 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:A schema view V^s is a 4-ary tuple V^s = (V^s_{name}, $V^s_{simpleType}$, $V^s_{complexType}$, $V^s_{constraint}$), where V^s_{name} is the name of the schema view V^s, $V^s_{simpleType}$, $V^s_{complexType}$ 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, $V^s_{simpleType}$, $V^s_{complexType}$ 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 a 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 schema view of V^c, if and only if V^s is transformed from V^c by $\aleph^c_s$. That is, $\aleph^c_s$: $V^c \rightarrow V^s$.

C. Document Level

In Fig. 4, the bottom instance level implies a fragment of instantiated XML data, which conforms to the corresponding view schema defined at the upper level. Here, the conceptual operators are mapped to query expressions (e.g. XQuery [12]), which are syntax specific. 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 documents are constructed/behave, we classify XML instance views into three categories [39], namely, derived instance views, constructed instance views, and triggered instance views.

The document/instance views can be constructed via a native XML query language (e.g. XQuery, SQL 2003) or some specific algorithms such as Ontology Extraction Methodology (OEM) (the MOVE [66] system), which can be achieved by the transformation of conceptual operators λ defined at the conceptual level [73] into a language-specific query fragment λ_{query}, say FLWOR expressions in XQuery etc.

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

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

D. XML Views in the LVM

Thus, we can show that a view in the layered view model as a view that composes of three sub-view components, namely conceptual, logical and instance views. Thus we can define a an XML view in the LVM for XML (V^{XML}) as;

Definition 6:An XML view in the LVM V^{XML}, is a triple V^{XML} = (V^c, V^s, V^d), where V^c is the conceptual view of the XML view in V^{XML}, V^s is the schema (or logical) view of the XML view in V^{XML}, V^d is the instance (or document) view of the XML view in V^{XML}.

Here V^c provides the (conceptual) view definition, V^s provides the schema and V^d provides the query expression for the layered view.

V. Modeling Conceptual Views

The conceptual views are views that are captured at the conceptual level with conceptual semantics and constraints. To the best of authors' knowledge, there exists no notation to specify or define such high level views or view semantics using exiting OO modelling languages. Even in the ORA-SS view model [32], the views are only represented (in contrast to defining) using a diagrammatic notation. Thus, in the layered view model, to define and specify views at the conceptual level (conceptual views), new set of notations needed to be developed. These include; (a) a notation to represent conceptual views, visually, at the conceptual level, (b) a set of notations to represent conceptual view properties, (c) a notation to represent view constructs and (d) a notation to represent conceptual view constraints.

A. Class Vs. Types

For the purpose of this work, it is important to clarify the notion of types and classes. In the OO, there are some few distinctions between classes and types.

A type can be described as [19]; (a) it corresponds to a notion of Abstract Data Type (ADT) and in programming languages variables are declared or defined using such types, (b) some examples of typed based languages include C++ (Stroustrup et al. '86), O2 (Atkinson et al. '89), SIMULA 67 (Dahl '68) etc. In these languages, the type definition of the variables indicates to the system, the structure of the type and the operations that are allowed on the values of the variables at run time. Also, variable type checking is done at the compile to time to ensure program correctness and do not change during the execution (or runtime) of the program and (c) types are referred during the compile time and not during runtime, thereby ensuring program safety and runtime efficiency, but reducing program flexibility and user-friendliness.

Conversely, a class can be described as [19]; (a) a class consists of data structure and operations that are applicable to the class describing the states and behaviour that may be associated with its objects, (b) a template for creating new instance objects and (c) some examples of class based languages include SMALLTALK (Goldberg '83), Java (Sun Microsystems '96) Gemstone (Maier et al. '86). Unlike type based languages, a class is refereed during runtime than during compile time, thus providing programming flexibility, uniformity and user-friendliness at the cost of runtime efficiency.

It is apparent that, in the context of programming, there are differences between a class-based language and a type-based language. But in the context of modelling [19] [20] (such as in our work), there are more similarities than distinctions between the two. In OO modelling, both classes and types are notationally identical (in almost all OO modelling languages) and are capable of being a template for creating, manipulating and grouping objects. Though semi-structured data and their associated schemas (e.g. XML) exhibit strong characteristics of type-based systems in the context of programming, in the context of modelling, such difference do not matter as, at a higher level of abstraction, semi-structured data models exhibits strong similarities to both types and classes. Therefore, in our work with conceptual views, we do not distinguish types and classes.

B. Conceptual Views and Conceptual Constructs

To model conceptual views at the conceptual level, we adopt a specialized notation, which is similar to the notion of a class in OO modelling languages. A (conceptual) view class is a class with attributes, and optional methods [75] and constraints associated with its attributes and methods.

The modeling of conceptual views can be done using UML, plus a set of stereotypes [39] and newly introduced conceptual operators. In addition, to make view constraints more explicit and visible, we use OMG's Object Constraint Language (OCL) [76].

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 constraint specification. In the case of XSemantic nets, constraints are provided as part of the model elements [17].

Example 1: for example, `staff`, `order` and `customer` can be some of the context examples in the e-Sol system.

Example 2: `processed-order` and `overdue-order` are two contrasting conceptual views in the context of `order` of the e-Sol system.

C. Specifying Conceptual 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 the relational model, views are defined using SQL and a limited set of constraints that can be defined using SQL [21], [22], namely, (a) presentation specific (such as display headings, column width, pattern order etc), (b) range and string patterns for aggregate fields, (c) input formats for updatable views, and (d) 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 [34]) and/or external languages (such as C++, Java or O2C [47]). 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 [35]. But the work by authors of [32] provides some form of higher-level view constraints (under ORA-SS model) for XML views, while the work in [28] 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 [77], where authors use OCL to "specify" relational views, which utilizes OCL as a declarative view specification language than a for view constraint specification.

Constraint specification for conceptual views can be explicit and extensive in comparison to relational or OO view constraints as, the conceptual views are designed at a higher-level.

Here, the constraint specifications are restricted only by the modeling language semantics and not by the view definition language as in the case of relational (e.g. SQL) and/or OO (e.g. OQL) views. In addition, further constraints can be defined for conceptual views that includes; (a) unique constraints, (b) referential constraints, (c) ordered composition, (d) explicit homogenous composition/heterogeneous compositions, (e) adhesion and/or dependencies, (f) exclusive disjunction, (g) cardinality constraints, (h) domain constraints (range of values, min, max, pattern etc) and (i) constructional contents (set, sequence, bag, ordered-set).

Therefore, conceptual view constraints $V^c_{constraint}$ may be described as a collection of various types of constraints and shown as a set, such that;

$$V^c_{constraint} = \{ \texttt{unique ()}, \texttt{reference ()}, \texttt{order ()}, \texttt{homogenous ()}, \texttt{exclusive ()}, \texttt{<adhesion>}, \texttt{<domain-constraints>}, \texttt{<cardinality-constraint>}, \texttt{constructional ()} \}$$

1) Unique Constraint: 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 [19] [78]. 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 a DBMS environment, physical OID may contain physical object address while a logical OID points to a logical ID which may get mapped to the object's physical object address using some algorithm (B+ tree, hash table etc).

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 conceptual views, there is a need to explicitly state OIDs, and available to be visualized at the conceptual level. This is mainly due to the nature of key-independent XML trees. Here, OIDs provide a means of using them for the purpose of IDs, similar to that of primary/foreign key constraints available in the 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. For example, a unique constraint for an attribute may be specified as;

$$V^c_{constraint} = \{ \texttt{unique (<attribute-name>)} \}$$

Example 3: In the e-Sol, `staffID`, `customerID` are unique, which can be described as;

$$V^c_{constraint} = \{ \texttt{unique (staffID)} \}$$

$$V^c_{constraint} = \{ \texttt{unique (customerID)} \}$$

2) Referential Constraint: The referential constraint requires that, to have a valid referential constraint, there exists a (view) class or (view) attribute content that is equal or same in value to another (view) class or (view) attribute. Referential constraints help to maintain referential integrity of a domain.

$$V^c_{constraint} = \{ \texttt{<A> reference <B>} \}$$

A = V^c_1 , B = V^c_2 , where (V^c_1, V^c_2) $\in V^c$ and $V^c_1 \neq V^c_2$

3) Ordered Composition: In a real-world scenario, in an aggregation relationship (i.e. in conceptual view hierarchy), there may exists a "whole" object that is made of "part" or composite objects that occur in a predefined order. This signifies an important OO concept, *ordered composition*. For example in XML Schema construct such as with `<xs:sequence>`, we regularly observe that the tag `<xs:sequence>` signifies that the embedded elements are not only a simple assortment of components but these have a specific ordering. For example, ordering of a finite set of "part-of" conceptual views V^c_i, ($V^c_1 \in V^c$) can be shown as;

$$V^c_{constraint} = \texttt{order} (V^c_1, V^c_2, V^c_3, ..., V^c_n)$$

Example 4: As shown in Fig. 7, The composition relationship between `Goods-Transaction` and (`Lot-Master`, `Goods-Items`) is ordered. This can be described as;

$$V^c_{constraint} = \texttt{order (Lot-Master, Goods-Items)}$$

4) Homogenous Composition: In a real-world scenario, in an aggregation relationship (i.e. in conceptual view hierarchy), there may exists a "whole" object that is made of "part" or composite objects that are of the same type as the "whole" object. This kind of whole-part relationship is referred to as homogenous composition. From a modeling point of view, explicitly stating homogenous compositions in modeling conceptual views is beneficial in designing better schema or programming structures. For example, homogenous composition of a finite set of "part-of" conceptual views V^c_i, ($V^c_1 \in V^c$) is described as;

$$V^c_{constraint} = \texttt{homogenous} (V^c_1, V^c_{11}, V^c_{12}, ..., V^c_{111}, ...)$$

Example 5: For example, (Fig. 7) there exists homogenous composition between `Goods-Type` and `Goods-Sub-Type`, which can be described as;

$$V^c_{constraint} = \texttt{homogenous (Goods-Type, Goods-Sub-type)}$$

5) Exclusive Disjunction: Exclusive disjunction constraint may be applied to both aggregation and association type relationships between conceptual views. It implies that only one conceptual view class exclusively connected in a relationship. For example, exclusive disjunction of a finite set of conceptual views V^c_i, ($V^c_1 \in V^c$) is described as;

$$V^c_{constraint} = \texttt{exclusive} (V^c_1, V^c_2, V^c_3, ..., V^c_n)$$

Example 6: For example, (Fig. 7) an exclusive disjunction between `Internal-Lot-Movement` and `External-Lot-Movement` is shown below;

$$V^c_{constraint} = \texttt{exclusive (Internal-Lot-Movement, External-Lot-Movement)}$$

Example 7: Another example (Fig. 7) is the exclusive disjunction between `LMS` (sub-system) and the `Customer-Logistics`, which can be shown as;

$$V^c_{constraint} = \texttt{exclusive (LMS, Customer-Logistics)}$$

6) Cardinality Constraint: The cardinality constraint shows the number of instances of one view class that may relate to single instance of another. This is similar to that of relational and OO data models. Cardinality constraints include one-to-one (1..1), one-to-many (1..*), zero-or-one (0..1), zero-or-more (0..*), and many-to-many (*..*). This can be shown as;

$V^c_{constraint}$ = { `<cardinality-constraint>` }

where, `<cardinality-constraint>` := [1..1] | [1..N] | [0..1] | [0..N] | [N..M]

7) Dependency / Adhesion Constraint: Adhesion (or dependency) constraint indicates if participants of a relationship (aggregation, association, dependency, instance etc.) should coexist and hold fast to each other. Also, in the case of class-attribute relationships, this constraint may be used to indicate optional or compulsory attributes, while in most semantic relationships (aggregation, association) the adhesion relationships is always of type strong adhesion. Adhesion relationship is described as;

$V^c_{constraint}$ = { `<adhesion>` }

where, `<adhesion>` := `strong-adhesion()` | `weak-adhesion()`

Example 8: For example, in the e-Sol, there exists a strong adhesion between `Warehouse-Manager` and `Warehouse-Staff` (Fig. 6), as a warehouse staff entry should be there to have a warehouse manager. This can be described as;

$V^c_{constraint}$ = `strong-adhesion(`
`Warehouse-Manager`, `Warehouse-Staff)`

Example 9: In the e-Sol, there exists a strong adhesion between `Income` and `Salary` and `Benefit` packages (Fig. 6), which can be described as;

$V^c_{constraint}$ = `strong-adhesion((` `Income`, `Salary)`
AND `(Income`, `Benefit`))

Example 10: Another example, adhesion relationship between `External-Lot-Moment` and `Customer-Logistics` as such entry is optional (Fig. 7). This can be described as;

$V^c_{constraint}$ = `weak-adhesion(`
`External-Lot-Moment`, `Customer-Logistics)`

8) Domain Constraints: The notion of domain constraints include a wide range of constraint properties that can applied to the data values (and types) in a given domain. For example, the domain constraints for conceptual views include; (a) restriction of values, range of values and lengths of data types, (b) enumerated and/or pre-specified (default) values, (c) predefined patterns (e.g. date formats) etc. These constraints can be shown as;

$V^c_{constraint}$ = `<domain-constraints>`

where, `<domain-constraints>` := [`Val v`] | [`MinVal v`] | [`MaxVal v`] | [`Len v`] | [`MinLen n`] | [`MaxLen n`] | [`Val s`] | [`Pattern s`] and `v` is a numerical value and `s` is a string.

Example 11: In the e-Sol,

- Maximum length of the `customerID` field is 10;
  $V^c_{constraint}$ = `customerID (MaxLen = 10)`
- `Lot-Movement` date should include week no, in addition to the standard date format;
  $V^c_{constraint}$ = `lotMovement-Date (MaxLen = 10, Pattern dd-mm-yyyy, nn)`
- Staff (and customer) access code/password should have minium 6 characters;
  $V^c_{constraint}$ = `password (MinLen = 6)`

9) Constructional Constraints : Constructional constraints are constraints associated with constructional data types and values such as set, list, bag and union. Conceptual views can be defined based on one (or more) stored type and constructs set, list, bag etc. of that type.

- **List**: A list is a sequence of atomic types composed to represent one type. Further more, additional restrictions may be specified as part of the list constructional constraints, namely; (a) total length of the list, (b) minimum length allowed, (c) maximum length allowed, and (d) enumeration.

  Example 12: In the e-Sol, the `Goods-Sub-Type` (Fig. 6) has an attribute which is `Categories` that may contain 1 to 5 `Sub-Categories`.
  $V^c_{constraint}$ = `Categories := list (Sub-Categories)`
- **Union**: A union constructional constraint allows one type to be constructed using union of one or more atomic and list types [17].

  Example 13: In the e-Sol, the `Internal-Lot-Movement` (Fig. 7) date may be specified using typical date format or using week numbers.
  $V^c_{constraint}$ = `Lot-Movement-Date := union (date, number (99))`

A detailed discussion on constraint specification for (stored) XML domains in UML can be found in the work [18], while detailed discussion on XSemantic net constraints in [17]. Specifying these constraints in UML/OCL or XSemantic nets for conceptual views is similar to that of stored domain object constraints.

VI. Modeling Conceptual Views Using UML/OCL

In UML, the Object Constraint Language (OCL) [78], which is now a part of the UML 2.0 standard [73], can support unambiguous constraints specifications for UML models including specification of static and dynamic (e.g. messages constraints) model elements . Here, in our conceptual view model, we incorporate OCL (in addition to built-in UML constraint features such as cardinality constraints [18], [19]) as our view constraint specification language to explicitly state view constraints. It should be noted that, we do not use OCL to define or specify views, rather state additional constraints using OCL. OCL supports defining derived classes [78], [81], which is close to a view concept [79].

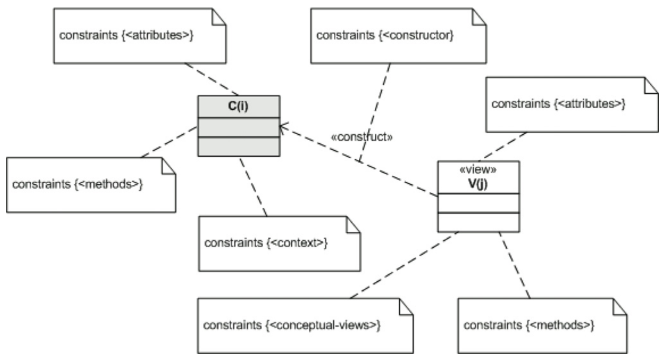
constraints {<attributes>}
constraints {<constructor}
C(i)
constraints {<attributes>}
«construct»
constraints {<methods>}
«view»
V(j)
constraints {<context>}
constraints {<conceptual-views>}
constraints {<methods>}

Fig. 5. Modeling conceptual views using UML/OCL.

OCL 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
```

A. Conceptual Views

Conceptual views are modelled in UML/OCL using the `<<view>>` stereotype. A view stereotype (Figs. 5–13) is a stereotyped class with (conceptual) view class definition, a set of view attributes (from one or more stored classes or derived via query constructs), a view constructor specification, optional view methods (e.g. derived attribute definitions etc.), and optional view constraints.

B. Conceptual Constructs

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. 5). This is to avoid confusion with the built-in UML dependency relationship and other stereotypes.

To model 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 may be used to show some of the derivations rule for the attributes and/or operations to make the view definition more explicit and precise. For example, as shown in Fig. 5, 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).

Example 14: In Fig. 6 - 7, `staff`, `warehouse`, and `customer` can be some of the context examples in the e-Sol system.

Example 15: Conceptual views, for example, `Warehouse-Manager` and `Warehouse-Staff` are two conceptual views in the context of `staff` of the e-Sol system.

Example 16: In Fig. 6, `Warehouse-Manager` is a valid conceptual view in the LVM, named in the context of `Staff`. Its constructed using the conceptual SELECT operator, which can be shown as;

$$\sigma_{warehouse-Staff.Role="manager"}(warehouse - staff)$$

Example 17: If a new domain requirement exists to add new conceptual view `Management-Memo` send to all `Warehouse-Manager` (Fig. 6), we can do that using Cartesian Product conceptual operator, where x = `Warehouse-Manager` and y = `Management-Memo`;

$$\chi_{(x,y)} = x\chi y$$

Example 18: In the case of conceptual view `Income` (shown in Fig. 7), the conceptual construct is a conceptual JOIN operator with join conditions, where x = `Staff`, y = `Salary-Pkg` and z = `Benefit-Pkg`:

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

C. Conceptual View Constraints

1) Unique Constraint: To visually model OID in UML class diagram, we define a stereotype OID, shown in Figs. 6–7 as an attribute. Together with attribute name and optional type definition, stereotype `<<OID>>` can be used in UML to indicate that the attribute that is a unique OID.

Example 19: In the case of conceptual view `Warehouse-Manager` (Fig. 6), we indicate the unique `staffID` by the following OCL expression;

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

2) Exclusive Disjunction: To visually show exclusive disjunction,

Example 20: In the case of `Lot-Movement`, 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. 7).

Example 21: In the case of `LMS` (sub-system) and `Customer-Logistics` exclusive disjunction can be shown the OCL statement "OR" between the relationships (as shown in 7).

Example 22: In the case of conceptual views `Warehouse-Manager` and `Warehouse-Staff`, in the context of `Staff` (Fig. 6), 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
```

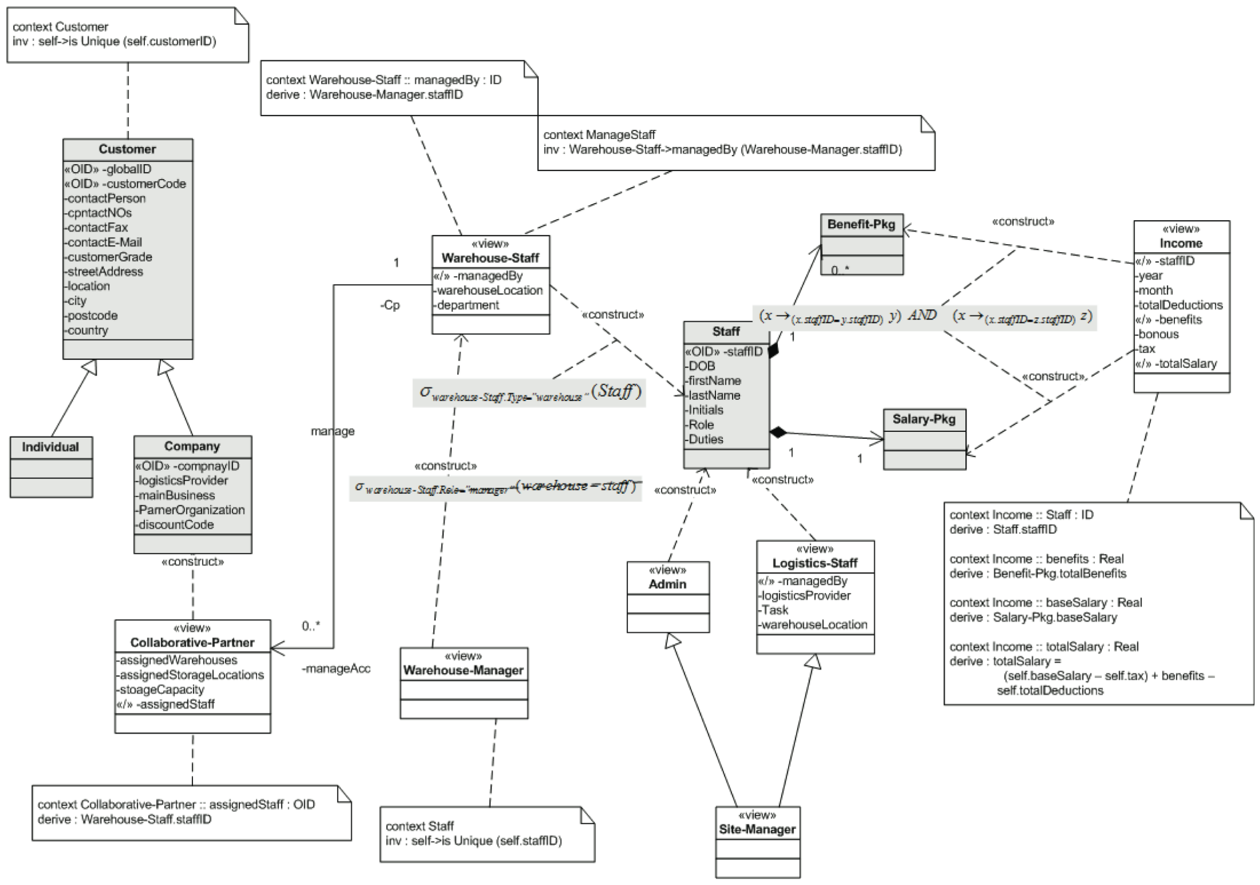
context Customer
inv : self->is Unique (self.customerID)
Customer
«OID» -globalID
«OID» -customerCode
-contactPerson
-cpntactNOs
-contactFax
-contactE-Mail
-customerGrade
-streetAddress
-location
-city
-postcode
-country
Individual
Company
«OID» -compnayID
-logisticsProvider
-mainBusiness
-ParnerOrganization
-discountCode
«construct»
«view» Collaborative-Partner
-assignedWarehouses
-assignedStorageLocations
-stoageCapacity
«/» -assignedStaff
context Collaborative-Partner :: assignedStaff : OID
derive : Warehouse-Staff.staffID
context Warehouse-Staff :: managedBy : ID
derive : Warehouse-Manager.staffID
context ManageStaff
inv : Warehouse-Staff->managedBy (Warehouse-Manager.staffID)
«view» Warehouse-Staff
«/» -managedBy
-warehouseLocation
-department
1
-Cp
manage
0..*
-manageAcc
«construct»
$\sigma_{warehouse\text{-}Staff.Type=\text{"}warehouse\text{"}}(Staff)$
«construct»
$\sigma_{warehouse\text{-}Staff.Role=\text{"}manager\text{"}}(warehouse = staff)$
«view» Warehouse-Manager
context Staff
inv : self->is Unique (self.staffID)
Benefit-Pkg
0..*
«construct»
$(x \rightarrow_{(x.staffID=y.staffID)} y)$ AND $(x \rightarrow_{(x.staffID=z.staffID)} z)$
Staff
«OID» -staffID
-DOB
-firstName
-lastName
-Initials
-Role
-Duties
1
1
1
Salary-Pkg
«construct»
«construct»
«construct»
«view» Admin
«view» Logistics-Staff
«/» -managedBy
-logisticsProvider
-Task
-warehouseLocation
«view» Site-Manager
«view» Income
«/» -staffID
-year
-month
-totalDeductions
«/» -benefits
-bonous
-tax
«/» -totalSalary
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

Fig. 6. e-Sol domain model (users) in UML/OCL.

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

Example 23: In the case of conceptual view `Income` (Fig. 7), 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
```

3) Constructional Constraints:

- **List**
  Example 24: As described in example 12, the `Goods-Sub-Type` attribute `Categories` may be stated using UML/OCL (Fig. 7) as follows;

```
context Goods-Sub-Type
inv: self->Categories : listOf
(self.Sub-Categories)
```

```
context Goods-Sub-Type
inv: Categories->length () := 5
```

- **Union** A union constructional constraint allows one type to be constructed using union of one or more atomic and list types [17].

  Example 25: In the e-Sol, the `Internal-Lot-Movement` date may be specified using typical date format or using week numbers.

```
context Internal-Lot-Movement
inv: self.Lot-Movement-Date: union
(date, integer)
```

VII. Transformation of Conceptual Views To Logical Views

In his section we look as some of the transformation of conceptual views to logical views. Here we only outline the constructs that require unique transformation formalism to address the view properties. A detail transformation of common mappings between UML and XML Schema (XSD) can be found in [18].

4) Unique Constraint: In order to transform unique constraints (namely OID) to view schema, there are 3 options, namely (a) our own transformation, (b) generic transformation methodologies proposed in such works as to enforce unique constraints and (c) the new W3C recommendation of `<xml:ID>` [82].

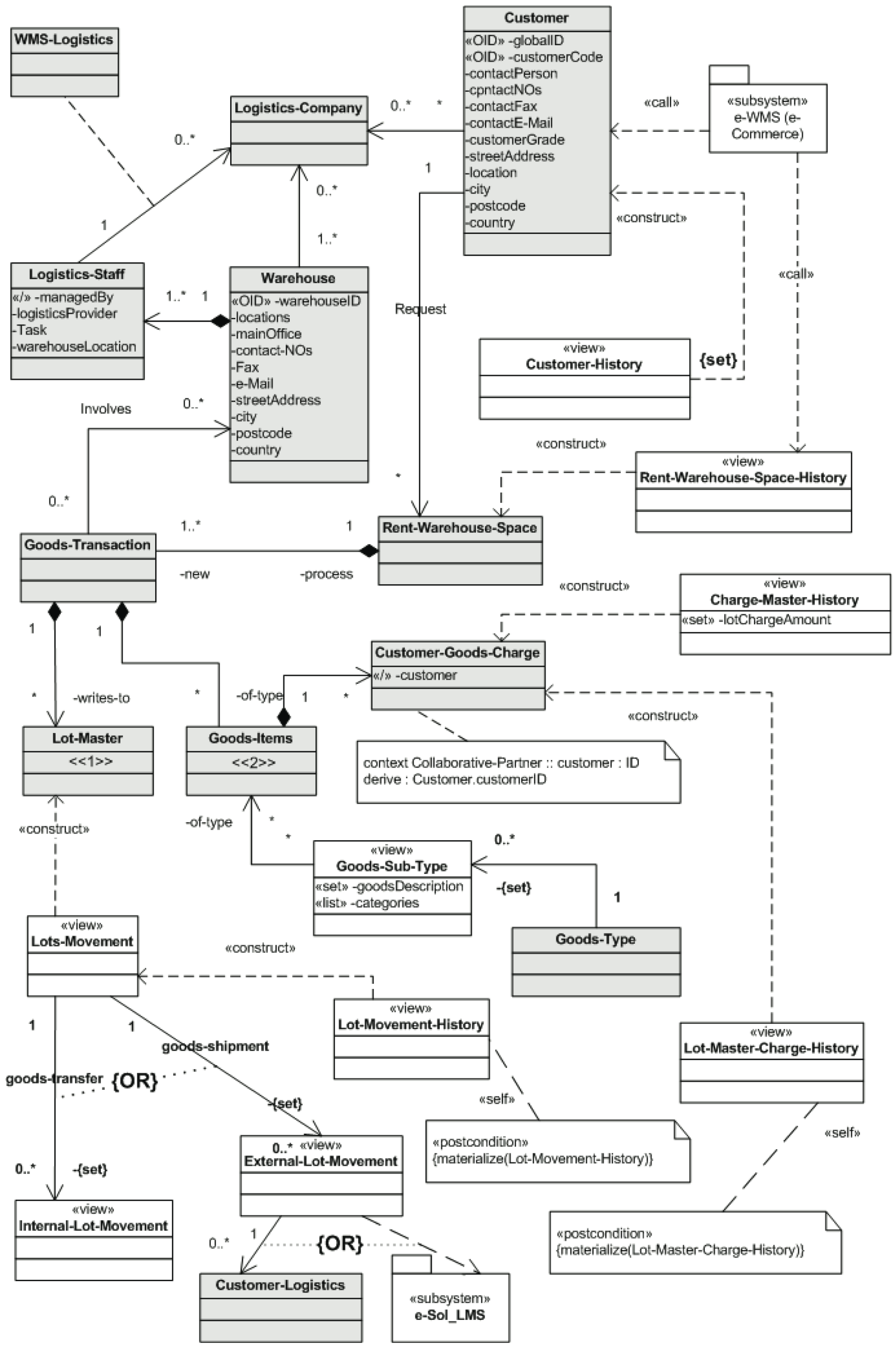

Fig. 7. e-Sol domain model (WMS) in UML/OCL.

1) Our first proposal to map OID constant to XML schema is to use XML Schema “unique” element in combination with the XSD “key” construct. The XSD unique element only assured the uniqueness when a key or data *exists*. This is in different from the XSD “key” constructs were an entry must always exist. Thus we use both constructs to map the conceptual view OIDs. For example, in XSD unique and key constructs are declared, as shown below;

```
<unique name= "<unique-constraint-name>">
    <selector xpath="<element-name" />
    <field xpath="<attribute-or-element-
name>" />
</unique>
```

```
<key name= "<key-name>">
    <selector xpath="<element-name" />
    <field xpath="<attribute-or-element-
name>" />
</key>
```

2) Another option to map OID like unique constraints to XML Schema was proposed by Pardede et al. [83], where they proposed to map unique constraints from UML like modelling languages to XML Schema. A detailed discussion on their work can be found in [83].
3) A detailed discussion on the new W3C recommendation to deal with IDs and unique constraints in XML schema can be found in [82]. This is one of the new approaches to transform conceptual view OIDs to logical view elements.

5) Referential Constraints: In transforming conceptual view referential constraints to logical view, two alternative solutions exists. They are (a) using XSD ID/IDREF constructs and (b) XSD KEY/KEYREF constructs. One of the major advantages of using KEY/KEYREF is that, it enable one to specify scope within which the uniqueness applies and it allows to create KEY (and KEYREF) from combination of any (simple, attribute or complex) content. The XSD declaration of KEY/KEYREF is similar to that of the unique constraints.

```
<key name= "key-name">
    <selector xpath="element-name-1" />
    <field xpath="attribute-or-element-name-
1" />
</key>


<keyref name= "keyref-name" refer="key-name" >
    <selector xpath="element-name-2" />
    <field xpath="attribute-or-element-name-
1" />
</key>
```

Example 26: The unique constraint shown in Figs. 6–7, (e.g. the `<<OID>>` in `Staff`) can be mapped to the view schema as shown in the code fragment (Code listing 1).

6) Ordered Composition: The ordered composition is mapped to logical view using the XSD **`sequence`** construct. The following XSD code fragment demonstrates this.

```
<xs:complexType name = "whole-complextype-
name" >
    <xs:sequence>
        <!- Additional nesting ->
        <xs:element name="part-
name1" type="part-type1" />
        <xs:element name="part-
name2" type="part-type2" />
        .........
    </xs:sequence>
</xs:complexType>
```

Example 27: The ordered/unordered compositions (e.g. in Fig. 7, `Lot-Master` and `Goods-Items`), shown using the stereotype (<<1>>`, `<<2>>`,...`) are mapped using the `<xs:sequence>` construct. The code fragment in Code listing 2 demonstrates this.

7) Exclusive Disjunction: The conceptual view exclusive disjunction is mapped to logical views using the XSD **`choice`** constructor. For example the following code fragment demonstrates this.

```
<xs:complexType name = "complextype-name" >
    <xs:choice>
        <!- Additional nesting ->
        ...........
    </xs:choice>
</xs:complexType>
```

Example 28: 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 3.

8) Constructional Constraints: The transformation of the constructional constraints to logical view schema is straight forward in the case of list and union types, while transformation of others such as set and bag needs to improvise as at the moment, XML Schema does not provide native support for these types.

- **List:**
  XML Schema has a built-in list types (IDREFS, NMTOKENS etc.) which is a sequence of atomic types. Also, one also can map derived (or user-defined) list types to derived list types by using the XSD **`list`** constructs. In addition, XSD offer few facet mechanism to refine the list constructs, namely [17] [18]; (a) length, (b) minLength, (c) maxLength and (d) enumeration. A simple XSD list declaration is shown below.
  ```
  <xs:simpleType name="list-name" >
      <xs:list itemType = "XDS-base-
  types" />
  </xs:simpleType>
  ```
  Example 29: In the e-Sol, the `Goods-Sub-Type` has an attribute which is Categories that may contain 1 to 5 `Sub-Categories`. Code listing 4 demonstrates this.
- **Union:**
  XML Schema provides support for union constructional constraint using the **`union`** construct.
  ```
  <xs:simpleType name="union-data-name" >
      <xs:union memberTypes= "xs-base-
  type" />
  </xs:simpleType>
  ```
  Example 30: In the e-Sol, the `Internal-Lot-Movement` date may be specified using typical date format or using the week number of the year. The code fragments in Code listing 5 demonstrates this.

VIII. Potential Applications of the View Mode

Designing software constructs using views are an inexpensive way of defining architectural frameworks in traditional data engineering approach. Some popular approaches include: application wrappers, data models, data description wrappers, application programming interfaces etc. [25], [26].

```
<!-- 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 -->
```

Fig. 8. Code Listing 1.

```
<!-- additional nesting --> <xs:complexType
name="Goods-TransactionType">
    <xs:complexContent>
        <xs:extension base="Goods-TransactionType">
            <!-- additional nesting -->
            <xs:sequence>
                <xs:element name="Lots-Master" type="Lots-MasterType"
minOccurs="1" maxOccurs="unbounded"/>
                <xs:element name="Goods-Items" type="Goods-ItemsType"
minOccurs="1" maxOccurs="unbounded"/>
            </xs:sequence>
            <!-- additional nesting -->
        </xs:extension>
    </xs:complexContent>
</xs:complexType>

<!-- additional nesting --> <xs:complexType name="Lots-MasterType">
    <!-- additional nesting -->
</xs:complexType>

<!-- additional nesting --> <xs:complexType name="Goods-ItemsType">
    <!-- additional nesting -->
</xs:complexType> <!-- additional nesting -->
```

Fig. 9. Code Listing 2.

Since the view definitions are not expensive from a storage point of view and easier (and flexible) to maintain (and in reflecting schema and data changes) than externally maintained application modules, it is well suited for defining virtual data architectures and/or framework. Also, the conceptual and logical extensions provided in the LVM provide layered

```
<!-- 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 -->
```

Fig. 10. Code Listing 3.

```
<!-- additional nesting -->
    <xs:simpleType name="Sub-Categories">
        <xs:list itemType="xs:string"/>
    </xs:simpleType>
    <!-- additional nesting -->
    <!-- additional nesting -->
    <xs:simpleType name="Categories">
        <xs:restriction base="Sub-Categories">
            <xs:length value="5"/>
        </xs:restriction>
    </xs:simpleType>
<!-- additional nesting -->
```

Fig. 11. Code Listing 4.

```
<!-- additional nesting -->
    <xs:simpleType name="Lot-Movement-DateType">
        <xs:union memberTypes="xs:date xs:integer"/>
    </xs:simpleType>
<!-- additional nesting -->
```

Fig. 12. Code Listing 5.

abstraction, data semantics, and constructs (Fig. 13) that can be utilized in a systematic manner, similar to the MDA initiatives.

Since the introduction of Model-Driven Architecture (MDA) [84] initiative by OMG, platform independent models play a vital role in system development and data engineering. Under the MDA initiative, first the model of a system is specified via an abstraction notation that is independent of the technical or deployment specifications (i.e. Platform Independent Model or PIM) and then the PIM is mapped or transformed into a deployment model (i.e. Platform Specific Model or PSM) by adding platform or deployment specific information. To support MDA initiatives in data engineering, data semantics, constraints and model requirements have to be specified precisely at a higher level of abstraction.

In the context of MDA solutions for XML domains, it is still a challenging task to produce PIMs despite the flexibility and the semantic richness of the semi-structured schema languages. This is mainly due to OO modelling languages such as OMG UML, E-ER etc. provide insufficient modelling constructs for utilizing XML schema like data descriptions and constraints, while XML Schema lacks the ability to provide higher levels of abstraction (such as conceptual models, visual constraints, etc.) that are easily understood by humans. This is due to the fact that models are often abstract representations which only keep so much of the detail as is relevant to the particular problem being considered [19], [20]. In this context, XML Schema generally is too low a representation to permit users to interact, visualize or understand it. To rectify this situation, many researchers in works such as [17], [18], [83], have applied intuitive techniques, notations and transformation methodologies to capture XML (and other semi-structured data) semantics at the conceptual level. This presents an

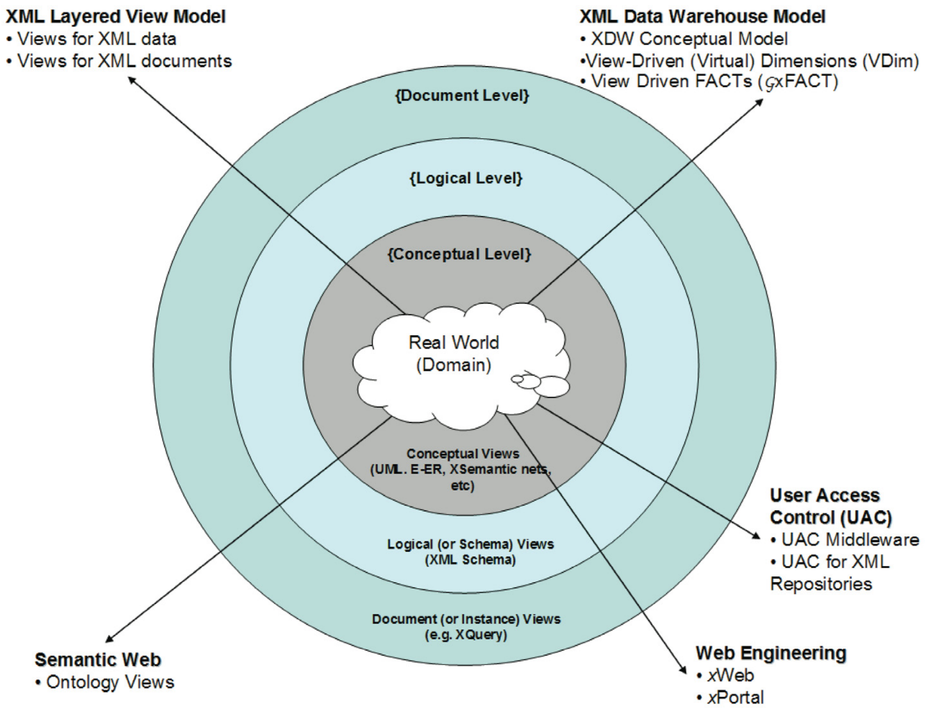
XML Layered View Model
• Views for XML data
• Views for XML documents
XML Data Warehouse Model
• XDW Conceptual Model
•View-Driven (Virtual) Dimensions (VDim)
• View Driven FACTs (ςxFACT)
{Document Level}
{Logical Level}
{Conceptual Level}
Real World (Domain)
Conceptual Views (UML. E-ER, XSemantic nets, etc)
Logical (or Schema) Views (XML Schema)
Document (or Instance) Views (e.g. XQuery)
User Access Control (UAC)
• UAC Middleware
• UAC for XML Repositories
Web Engineering
• xWeb
• xPortal
Semantic Web
• Ontology Views

Fig. 13. Application of LVM views in real-world scenarios.

opportunity to investigate data views as a means of providing data abstraction and semantics in PIMs for data intensive MDA solutions, such as XML document warehouses. Here, we briefly present some of the potential real-world applications of our LVM for XML in the context of EIS and e-solutions.

A. Views for XML Databases and Repositories

In relational DBMS systems, views are used in the context of access control, query refinement, performance enhancement and providing data perspectives for complex aggregate data. In the Enterprise Content Management (ECM) [84] framework, is a data intensive task, especially when handling large volumes of distributed heterogeneous data, such as data warehousing. Yet, with increasing heterogeneous database schemas (relational, XML and other formats) and contents challenge the traditional database and view techniques. Therefore for XML database and repository designers can develop their platform independent view formalism using higher-level modeling languages and use automated tools to implement such views in their multi-site, multi-platform DB/Repository systems. In doing so, the view definitions are not coupled to a specific query syntax and/or specification where they are exposed rapid to changes.

B. XML-View based UAC Middleware Design

The proposed growth for XML repositories in the ECM framework and their use in either to store data or as an interoperability layer for legacy applications provides the need to investigate user access control in such repositories [85]. The widespread use of XML highlights the need for high-level, flexible and expressive access control models for XML documents to protect sensitive and valuable information from unauthorized access (both by humans and machines/agents). Traditionally, views provided user access control mechanism in many DBMS. In the work [85], authors present an XML view-based access control model, which supports access control for both human (and agent) users. The design methodology proposed is based on the LVM views and support conceptual level design of UAC constraints and acts as a middleware layer for XML repositories and the databases alike.

C. XML Document Warehouse Design

As stated before, ECM is a data intensive task, especially when handling large volumes of distributed heterogeneous data, such as in data warehouse environments. To address such an issue, the authors of the work [86] proposed a view-driven design methodology for modeling and designing XML document warehouses (XDW), using our LVM for modeling dimensional (XML) data. Later they extended their work to accommodate warehouse user requirements in [87]. The proposed XDW model consists of: (a) user requirement level, (b) conceptual level, (c) logical level and (e) instance level.

D. Collaborative Web Engineering

The increase in enterprise web content in XML and storage of data in XML document format will provide greater semantic clarity and enable easier access and evaluation of the semantically rich web contents. For example, in an Industrial setting, a web solution may comprise of partner companies/franchise, where they have similar web content but varying user interface design and/or design. In order to keep the web content descriptive among business partners, yet discrete, where a particular user/staff may want to get an appropriate view of the data (for e.g. warehouse storage information, bookings, supplier information, storage capacity etc). One way to handle such a complex task is to model and build semantic-aware enterprise websites [88] and web portals [89], using LVM views, where the web content and their associated user interface definitions are captured at the conceptual level (using conceptual views) and mapped to logical view schemas and documents where, additional presentation constraints (such as local company web styles/formats) are locally applied depending on the requirements.

E. Views for Semantic Web (SW) Paradigm

Views for SW requires some form of abstraction, as SW documents and querying are done at the logical level or with logical syntaxes to handle heterogeneous schemas such as in multi-site ontology bases [38], [90]. Therefore, we argue that a view formalism for SW requires 2-Es (data Extraction and Elaboration) [90]. Though there exists some work in regards to a logical view formalism for SW [28], [64], most of them are tied to SW language specific schema/syntaxes that do not provide conceptual extensions. In related work [38], [90], our view formalism provide support for ontology extraction in the form of materialised ontology views in Ontology Extraction Methodology (OEM) [29], [66]. In the work, authors investigated how our LVM can be applied to sub-ontology design and extraction under the OEM framework.

IX. Conclusion and Future Work

In this paper, we presented a Layered View Model (LVM) for XML with conceptual and schemata extensions. First we presented semantics of the LVM followed by a visual view constraint specification model using UML/OCL. Later we highlighted some of the high level modeling issues associated with the LVM. Later we presented how conceptual views in the LVM are mapped to its schema and document level equivalent, with illustrated case study examples.

For future work, some issues deserve further investigation. First, the automation of the mapping between the conceptual operators to various XML query language expressions (such as XQuery or SQL/X). Second is the investigation into specifying dynamic perspectives of the conceptual views, which then can be applied to traditional data, Semantic Web and web service domains.

References

[1] Abiteboul, S., P. Buneman et al. (1999) *Data on the Web : from relations to semistructured data and XML*. London, UK, Morgan Kaufmann.

[2] Chan, S., T. S. Dillon, et al. (2002) Applying a mediator architecture employing XML to retailing Inventory Control. *The Journal of Systems and Software* **60**: 239-248.

[3] Do, H. H. and E. Rahm (2004) Flexible integration of molecular-biological annotation data: The genmapper approach. *Proceedings of the 9th International Conference on Extending Database Technology* (EDBT '04), Heraklion, Crete, Greece.

[4] Abiteboul, S., B. Amann et al. (1999) Active Views for Electronic Commerce. *Proceedings of the 25th International Conference on VLDB*, Edinburgh, Scotland.

[5] Abiteboul, S., R. Goldman et al. (1997) Views for Semistructured Data. *Workshop on Management of Semistructured Data*, USA.

[6] W3C-WWW (1997) World Wide Web (WWW), http://www.w3.org/, The World Wide Web Consortium (W3C).

[7] W3C-HTML (1997) HyperText Markup Language (HTML), Rel 4.01 (http://www.w3.org/MarkUp/), The World Wide Web Consortium (W3C).

[8] W3C-XML (2004) Extensible Markup Language (XML) 1.0, (http://www.w3.org/XML/), The World Wide Web Consortium (W3C).

[9] W3C-XSD (2001) XML Schema (http://www.w3.org/XML/Schema), W3C. 2004.

[10] W3C-RDF (2004) Resource Description Framework (RDF), (http://www.w3.org/RDF/), The World Wide Web Consortium (W3C).

[11] W3C-OWL (2004) OWL: Web Ontology Language 1.0 reference (http://www.w3.org/2004/OWL/), W3C.

[12] W3C-XQuery (2004) XQuery 1.0: An XML Query Language (http://www.w3.org/TR/xquery). XML Query Language (XQuery), The World Wide Web Consortium (W3C).

[13] W3C-RDQL (2004) RDQL – A Query Language for RDF, (http://www.w3.org/Submission/2004/SUBM-RDQL-20040109/), W3C.

[14] W3C-XSL (2003) Extensible Stylesheet Language (XSL) (http://www.w3.org/Style/XSL).

[15] W3C-WS (2002) Web Services Activity, (http://www.w3.org/2002/ws/), W3C.

[16] W3C-SW (2005) The Semantic Web (http://www.w3.org/2001/sw/), W3C.

[17] Feng, L., E. Chang et al. (2002) A Semantic Network-based Design Methodology for XML Documents. *ACM Transactions on Information Systems* (TOIS) **20(4)**: 390 - 421.

[18] Feng, L., E. Chang et al. (2003) Schemata Transformation of Object-Oriented Conceptual Models to XML. *International Journal of Computer Systems Science and Engineering* **18, No. 1(1)**: 45-60.

[19] Dillon, T. S. and P. L. Tan (1993) *Object-Oriented Conceptual Modeling*, Prentice Hall, Australia.

[20] Graham, I., A. C. Wills et al. (2001) *Object-oriented methods : principles and practice*. Harlow, Addison-Wesley.

[21] Date, C. J. (2003) *An introduction to database systems*. New York, Pearson/Addison Wesley.

[22] Elmasri, R. and S. Navathe (2004) *Fundamentals of database systems*. New York, Pearson/Addison Wesley.

[23] Chang, E. and T. S. Dillon (1994) Integration of User Interfaces with Application Software and Databases Through the Use of Perspectives. *1st International Conference on Object-Role Modeling* (ORM '94), Australia.

[24] Rafanelli, M. (Ed) (2003) *Multidimensional Databases: Problems and Solutions*, Idea Group Inc.

[25] Gopalkrishnan, V., Q. Li et al. (1999) Star/Snow-flake Schema Driven Object-Relational Data Warehouse Design and Query Processing Strategies. *1st First International Conference on Data Warehousing and Knowledge Discovery* (DaWaK '99), Florence Italy, Springer.

[26] Mohania, M. K., K. Karlapalem et al. (1999) Data Warehouse Design and Maintenance through View Normalization. *10th International Conference on Database and Expert Systems Applications* (DEXA '99), Florence, Italy, Springer.

[27] Gupta, A., I. S. Mumick et al. (1999) *Materialized views: techniques, implementations, and applications*, MIT Press.

[28] Volz, R., D. Oberle et al. (2003) Views for light-weight Web ontologies. *Proceedings of the ACM Symposium on Applied Computing* (SAC '03), USA, ACM Press New York, NY, USA.

[29] Wouters, C., T. S. Dillon et al. (2004) Ontologies on the MOVE. *9th International Conference on Database Systems for Advanced Applications* (DASFAA '04), Jeju Island, Korea, Springer-Verlag GmbH.

[30] Codd, E. F. (1990) *The Relational Model for Database Management: Version 2*, Addison Wesley Publishing Company.

[31] Blanken, H. M. (2003) *Intelligent search on XML data: applications, languages, models, implementations, and benchmarks*. Berlin ; London, Springer.

[32] Chen, Y. B., T. W. Ling et al. (2002) Designing Valid XML Views. *Proceedings of the 21st International Conference on Conceptual Modeling* (ER '02), Tampere, Finland, Springer-Verlag London, UK.

[33] Abiteboul, S. (1999) On Views and XML. *Proceedings of the eighteenth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems* (PODS '99), Philadelphia, Pennsylvania, USA, ACM Press New York, NY, USA.

[34] Cattell, R. G. G., D. K. Barry et al. (Eds) (2000) *The Object Data Standard: ODMG 3.0*, Morgan Kaufmann.

[35] Abiteboul, S., J. Quass et al. (1997) The Lorel Query Language for Semistructured Data. *International Journal on Digital Libraries* **1(1)**: 68-88.

[36] SPARQL (2004) SPARQL Query Language for RDF, (http://www.w3.org/TR/2005/WD-rdf-sparql-query-20050217/), W3C.

[37] ANSI/ISO (2003) ANSI - SQL 2003. ISO-ANSI Working Draft. J. Melton, ANSI / ISO.

[38] Rajugan,R., E. Chang et al. (2005) Modeling Ontology Views: An Abstract View Model for Semantic Web. *1st International IFIP WG 12.5 Working Conference on Industrial Applications of Semantic Web* (IASW '05), Jyvaskyla, Finland, Springer IFIP Book Series.

[39] Rajugan,R., E. Chang et al. (2005) A Three-Layered XML View Model: A Practical Approach. *24th International Conference on Conceptual Modeling* (ER '05), Klagenfurt, Austria, Springer-Verlag.

[40] W3C-DTD (2001) XML Schema (http://www.w3.org/, W3C. 2005.

[41] Zhuge, Y. and H. Garcia-Molina (1998) Graph structured Views and Incremental Maintenance. *Proceeding of the 14th IEEE Conference on Data Engineering* (ICDE '98), USA, IEEE.

[42] Goldman, R., J. McHugh et al. (1999) From Semistructured Data to XML: Migrating the Lore Data Model and Query Language. *Proceedings of the 2nd International Workshop on the Web and Databases* (WebDB '99), Philadelphia, Pennsylvania.

[43] Abiteboul, S., O. Benjelloun et al. (2002) Active XML: A Data-Centric Perspective on Web Services, *BDA '02*.

[44] Aguilera, V., S. Cluet et al. (2002) Views in a Large-Scale XML Repository. *The International Journal on Very Large Data Bases* **11(3)**: 238-255.

[45] Wouters, C. (2006) *A Formalization and Application of Ontology Extraction*. School of Engineering and Mathematical Sciences Faculty of Sciences, Technology and Engineering. Melbourne, La Trobe University, Melbourne, Australia: 299.

[46] Cluet, S., P. Veltri et al. (2001) Views in a Large Scale XML Repository. *Proceedings of the 27th VLDB Conference* (VLDB '01), Roma, Italy.

[47] Abiteboul, S. and A. Bonner (1991) Objects and Views. ACM SIGMOD Record, *Proceedings of the International Conference on Management of Data* (ACM SIGMOD '91), ACM Press New York, NY, USA.

[48] W3C-XPath (1999) XML Path Language (XPath) Version 1.0 (http://www.w3.org/TR/xpath/). XML Path Language, The World Wide Web Consortium (W3C). November 1999.

[49] Chang, E., T. Dillon et al. (2003) A Virtual Logistics Network and an e-Hub as a Competitive Approach for Small to Medium Size Companies. *2nd International Human.Society@Internet Conference*, Seoul, Korea, Springer-Verlag.

[50] Xyleme: A Dynamic Warehouse for XML Data of the Web. *International Database Engineering and Applications Symposium* (IDEAS '01), Grenoble, France, IEEE Computer Society 2001.

[51] Xyleme (2001) Xyleme Project (http://www.xyleme.com/). Lucie-Xyleme (2001).

[52] Chen, Y. B., T. W. Ling et al. (2002) A Case Tool for Designing XML Views. *Second International Workshop on Data Integration over the Web* (DiWeb '02), Toronto, Canada, University of Toronto Press.

[53] Braga, D., A. Campi et al. (2005) XQBE (XQuery By Example): A visual interface to the standard XML query language. *ACM Transactions on Database Systems* (TODS) **30(2)**: 398-443.

[54] Braga, D. and A. Campi (2003) A Graphical Environment to Query XML Data with XQuery. *4th International Conference on Web Information Systems Engineering* (WISE '03), Rome, Italy, IEEE Computer Society.

[55] Augurusa, E., D. Braga et al. (2003) Design and Implementation of a Graphical Interface to XQuery. *ACM Symposium on Applied Computing* (SAC '03), Melbourne, USA, ACM.

[56] Ludaescher, B., Y. Papakonstantinou et al. (2000) Navigation-Driven Evaluation of Virtual Mediated Views. *Extending DataBase Technology* (EDBT '00), Springer.

[57] Ludaescher, B., Y. Papakonstantinou et al. (1999) View Definition and DTD Inference for XML. *Post-ICDT Workshop on Query Processing for Semistructured Data and Non-Standard Data Formats*.

[58] Benjelloun, O. (2004) *Active XML: A data centric perspective on Web services*. Paris XI University. Orsay, France, Paris XI University: 189.

[59] Abiteboul, S., J. Quass et al. (1997) Lore: A Database Management System for Semistructured Data. *SIGMOD Record*, ACM.

[60] Baru, C., B. Ludscher et al. (1998) Features and Requirements for an XML View Definition Language: Lessons from XML Information Mediation. *W3C's Query Language Workshop* (position paper).

[61] Munroe, K. and Y. Papakonstantinou (2000) BBQ: A Visual Interface for Integrated Browsing and Querying of XML. *Visual Database Systems* (VDB '00).

[62] W3C-SW (2005) The Semantic Web (http://www.w3.org/2001/sw/), W3C.

[63] Volz, R., D. Oberle et al. (2003) Implementing Views for Light-Weight Web Ontologies. *Seventh International Database Engineering and Applications Symposium* (IDEAS'03), Hong Kong, SAR, IEEE Computer Society.

[64] Uceda-Sosa, R., C. X. Chen et al. (2004) CLOVE: A Framework to Design Ontology Views. *23th International Conference on Conceptual Modeling* (ER '04), Shanghai, China, Springer-Verlag.

[65] KAON (2004) KAON Project (http://kaon.semanticweb.org/Members/rvo/-Folder.2002-08-22.1409/Module.2002-08-22.1426/view).

[66] Wouters, C., T. S. Dillon et al. (2004) A Practical Approach to the Derivation of a Materialized Ontology View. *Web Information Systems*. D. Taniar and W. Rahayu. USA, Idea Group Publishing.

[67] HyOntUse (2003) User Oriented Hybrid Ontology Development Environments,(http://www.cs.man.ac.uk/mig/projects/current/hyontuse/).

[68] Chang, E., W. Gardner et al. (2001) Virtual Collaborative Logistics and B2B e-Commerce. *e-Business Conference*, Duxon Wellington, NZ.

[69] ITEC (2002) iPower Logistics (http://www.logistics.cbs.curtin.edu.au/).

[70] Golfarelli, M., D. Maio et al. (1998) The Dimensional Fact Model: A Conceptual Model for Data Warehouses. *International Journal of Cooperative Information Systems* **7(2-3)**: 215-247.

[71] Trujillo, J., M. Palomar et al. (2001) Designing Data Warehouses with OO Conceptual Models. IEEE Computer Society, "*Computer*": 66-75.

[72] OMG (2003) UML 2.0 Final Adopted Specification (http://www.uml.org/UML2.0), OMG. 2005.

[73] Rajugan,R., E. Chang et al. (2005) A Layered View Model for XML Repositories and XML Data Warehouses. *The 5th International Conference on Computer and Information Technology* (CIT '05), Shanghai, China, IEEE CS Press.

[74] Rajugan, R. (2006) *A Layered View Model for XML with Conceptual and Logical Extension, and its Applications*. Faculty of Information Technology. Sydney, University of Technology, Sydney (UTS), Australia: 460.

[75] R.Rajugan, E. Chang et al. (2006) Modeling Dynamic Properties in the Layered View Model for XML Using XSemantic Nets. *International Workshop on XML Research and Applications* (XRA '06) to be held in conjunction with APWeb '06, Harbin, China, Springer-Verlag.

[76] OMG-OCL (2003) UML 2.0 OCL Final Adopted specification (http://www.omg.org/cgi-bin/doc?ptc/2003-10-14), OMG. 2005.

[77] Balsters, H. (2003) Modelling Database Views with Derived Classes in the UML/OCL-framework. *The Unified Modeling Language: Modeling Languages and Applications* (UML '03), USA, Springer.

[78] Doorn, J. H., L. C. Rivero et al. (2002) *Database Integrity: Challenges and Solutions*, Idea Group, Hershey, PA.

[79] Warmer, J. B. and A. G. Kleppe (2003) *The object constraint language : getting your models ready for MDA*. Boston, MA, Addison-Wesley.

[80] W3C-xml:ID (2005) xml:id Version 1.0 (http://www.w3.org/TR/2005/REC-xml-id-20050909/), The World Wide Web Consortium (W3C).

[81] Pardede, E., J. W. Rahayu et al. (2005) Preserving Conceptual Constraints During XML Updates. *International Journal of Web Information Systems* (IJWIS) **1(2)**: 65-82.

[82] OMG-MDA (2003) The Architecture of Choice for a Changing World, MDA Guide Version 1.0.1 (http://www.omg.org/mda/), OMG. 2005.

[83] Gaevic, D., D. Djuric et al. (2004) Converting UML to OWL Ontologies. *Proceedings of the 13 th International World Wide Web Conference*, NY, USA.

[84] AIIM (2005) The ECM Association (http://www.aiim.org/index.asp), AIIM.

[85] Steele, R., W. Gardner et al. (2005) Design of an XML View Based User Access Control (UAC) Middleware. *IEEE International Conference on e-Technology, e-Commerce and e-Service* (EEE-05), Hong Kong, IEEE.

[86] Nassis, V., R. Rajugan et al. (2005) Conceptual and Systematic Design Approach for XML Document Warehouses. *International Journal of Data Warehousing and Mining* **1(3)**: 63-87.

[87] Nassis, V., T. S. Dillon et al. (2006) An XML Document Warehouse Model. *The 11th International Conference on Database Systems for Advanced Applications* (DASFAA '06), Singapore, Springer.

[88] Rajugan,R., W. Gardner et al. (2005) Designing Websites with EXtensible Web (xWeb) Methodology. *International Journal of Web Information Systems* (IJWIS) **1(3)**: 179-191.

[89] Gardner, W., Rajugan, R. et al. (2004) xPortal: XML View Based Web Portal Design. *17th International Conference on Software and Systems Engineering and their Applications* (ICSSEA '04), Paris, France.

[90] Wouters, C., R. Rajugan et al. (2006) Ontology Extraction Using Views for Semantic Web. *Web Semantics and Ontology*. D. Taniar and W. Rahayu. USA, Idea Group: 01-40.

Rajugan Rajagopalapillai holds a bachelor's degree in Information Systems from La Trobe University, Australia. He has worked in the industry as chief application/database programmer in developing sports planing and sports fitness and injury management software and as database administrator. He was also involved in developing an e-Commerce solution for a global logistics (logistics, cold-storage and warehousing) company as a software engineer/architect. He is currently working as a research associate at the Faculty of Information Technology, University of Technology, Sydney (UTS) Australia. He has published research articles which have appeared in international refereed conference and journal proceedings. His research interests include Object-Oriented conceptual models, XML, Semantic Web, data warehousing and database systems. He is a member of IEEE, member of ACM and an Associate member of Australian Computer Society (AACS).

Professor Elizabeth Chang is the Director of Area Research Excellence for Frontier Technology and the Centre Extended Enterprise and Business Intelligence (CEEBI) and Professor of IT in School of Information Systems. She has over 100 scientific conference and journal papers in IT and numerous invited Keynote papers at International Conferences. All her research and development work are in the area of IT Applications, Software Engineering and Logistics Informatics. Her research interests include issues related to the process of producing an IT application, methodologies, Software and System Architecture, web services or Mediator based systems, Peer to Peer Communications, Trust Management, XML, XQL, RDF and Ontology, Mobile agents, Security and Privacy, Reliability and Fault Tolerance, Usability Metrics and application areas such as Logistic Informatics and Bio-informatics.

Professor Tharam S. Dillon is the Dean of the Faculty of Information Technology at University of Technology, Sydney (UTS) in Australia. His research interests include data mining, internet computing, e-commerce, hybrid neuro-symbolic systems, neural nets, software engineering, database systems and computer networks. He has also worked with industry and commerce in developing systems in telecommunications, health care systems, e-commerce, logistics, power systems, and banking and finance. He is editor-in-chief of the International Journal of Computer Systems Science and Engineering and the International Journal of Engineering Intelligent Systems, as well as co-editor of the Journal of Electric Power and Energy Systems. He is on the advisory editorial board of Applied Intelligence published by Kluwer in the US and Computer Communications published by Elsevier in the UK. He has published more than 400 papers in international and national journals and conferences and has written four books and edited five other books. He is a fellow of the IEEE, fellow of the Institution of Engineers (Australia), and fellow of the Australian Computer Society.

Dr Ling Feng is an associate professor at University of Twente in the Netherlands, and was an assistant professor at Tilburg University in the Netherlands (1999-2002), and a lecturer at Department of Computing in Hong Kong Polytechnic University in China (1997-1999). Her research interests are distributed object-oriented database management system, knowledge-based information systems, data mining and its applications, data warehousing, data/knowledge management issues in the Internet era, including the integration of database and web-based information technologies, XML databases, knowledge-based digital libraries, and mobile databases.