# Achieving Transparent Integration of Information, Documents and Processes

Jingzhi Guo
*Faculty of Science and Technology, University of Macau,*
*Av. Padre Tomás, Pereira, S.J., Taipa, Macau, Tel: +853-397 4890*
*Email: jzguo@umac.mo*

## Abstract

*Business interoperation is important especially in electronic business. It requires the integration of business information, business documents and business processes. Nevertheless, while progress is made in these individual integration aspects, the issue of how to integrate the integrated results of business information, documents and processes requires to be resolved, that is, the vertical integration of the already-integrated results. This paper proposes a novel TRANSCODE approach to resolving this issue. This approach first describes business information, business documents and business processes in three TRANSCODE Structures, and then implements, conceptualizes and reifies them in a three-layer TRANSCODE Model, which is implemented in three XML specifications that demonstrates the concept reusability and model flexibility of TRANSCODE approach. Finally, the paper discusses its theoretic base and compares it with ebXML.*

## 1. Introduction

Business interoperation is an important research topic in electronic business [8], and has been studied in the integration fields of heterogeneous business information [4], heterogeneous business documents [9], and heterogeneous business processes [13]. These researches all concern a non-trivial issue, that is, two business entities are difficult to interoperate with each other to fulfill their shared task due to the autonomous and heterogeneous computing environments. Traditionally, solutions to this problem are individually resolved in different levels. For example, product information integration aims to make heterogeneous product information interoperable [1]. Business document integration supports the alignment of heterogeneous business documents in a same document management system (e.g. www.UDEF.org). Business process integration targets the coordination of the inconsistent processes from different companies [13]. While all these solutions contribute to their individual research realms, an interesting question is asked: how the integrated business information could be effectively used in business documents and how the interoperable business documents should be effectively utilized in business processes?

An integrated solution to integrating business information, documents and processes is important [9]. It can increase the reuse of existing business integration results and save labor costs in business reengineering.

This paper aims to propose a novel *transparent coding* (TRANSCODE) approach to vertically integrate business information, business documents and business processes. It is *transparent* because it allows the integrated business information to be openly used in business document integration, which further to be openly utilized in business process integration. Through this approach, the reuse of integrated results is available.

Contributions of the paper are: (1) TRANSCODE representation, which represents information, documents and processes of businesses in three independent domains but could be transparently referenced; (2) three-layer TRANSCODE model, in which each domain could be autonomously designed in its own way; and (3) three XML specifications, which implement TRANSCODE model for feasibility demonstration.

In the rest of this paper, TRANSCODE representations are provided in Section 2. Section 3 presents a three-layer TRANSCODE model for vertical integration solution. In Section 4, the TRANSCODE model is implemented in three XML specifications. Section 5 discusses and compares the approach with some related work. The final section summaries the paper and provides the future work.

## 2. TRANSCODE Representation

This section introduces the TRANSCODE representation to represent correlated integration domains of information, documents and processes of businesses.

### 2.1. Business Information

*Business information* is the fundamental information of a company such as products, assets, people and organization. It specifies the *basic knowledge* of a company. It has the following characteristics:

*Unit of concept*. A piece of information can be represented as a unit of *concept*, which is a semantic unit having a syntactic structure and semantic *denotation* [4], e.g. given *c*(*an*) = "shoes", the *c*(*an*)= "" is a structure while "shoes" is a semantic denotation.

*Hierarchically divisible*. A concept is a node of a *vector concept tree* [3] that may be *connoted by* many lower level concepts (i.e. connotation) [4], e.g. electronics(refrigerator(price(currency, value, piece), color)).

*Uniquely identifiable*. Any concept is the result of a given *context* [3], thus it is unique and can be uniquely identified, e.g. refrigerator → C.52.14.15.1.

*Strongly grouped*. A concept belongs to a concept group, e.g. refrigerator ∈ products, currency ∈ scalar type, dozen ∈ unit type, 18.8 ∈ value type and "white" ∈ constant type. The information group strongly affects the way of how a concept to be reified in a specific context.

*Possibly reified*. A piece of concept could be reified as a specific value, e.g. color→ white and cashier→David.

*Strongly typed reification*. A *reified value* of a concept is always strongly typed, e.g. the value "David" of concept "cashier" is strongly typed by "string". A reification may take a value of number, constant, scalar, or unit.

*Numeric value scalar*. A numeric value always has a scalar for measuring the value, e.g. the "USD" in USD2/pair or "person" in 10 persons per trip. An *orphan numeric value* (e.g. 3 or 0.33) is meaningless in business.

*Numeric value unit*. A numeric value always at least has one unit to refer to scaled value, e.g. "dozen" and "pair" in USD10/dozen pair. If more units involved, they can be converted to atomic valued units such as 24 piece = 12 (a dozen) × 2 (a pair).

*Conversion functions for scalar and unit*. A scalar or a unit may associates with a conversion function, e.g. Currency function for converting "USD" to other currency. Conversion result affects the associated numeric value.

To be more operational, business information can be represented in the following definition.

**Definition 1** (*Business Information Domain*): *BID*

Given a *business information domain BID*, then *BID* is a tuple *BID* = (*C*, *V*, *AN*, *IID*, *G*, *CO*, *VAL*, *DT*, *CVT*), where:

- *C* is set of concept structure symbols.
- *V* is a set of meaningful concept vocabularies such that *V* takes *C* as its form (i.e. *C* is syntactic structure) and *V* is the meaning conveyed in *C*. The *V* consists of the vocabularies of product information *R*, business documents *D*, business processes *P*, organization resources *O*, and other vocabularies $V_1$, $V_2$, …, $V_n$ such that $V = \{R, D, P, O, V_1, V_2, \ldots, V_n\}$.
- $AN \subset C$ is the symbol of annotation (i.e. denotation).
- $IID \subset C$ is the symbol of unique concept identifier such that $AN \xrightarrow{\text{determine}} IID$.
- $G \subset C$ is the symbol of concept group.
- $CO \subset C$ is the symbol of connotation.
- $VAL \subset C$ is the symbol of concept value structure paired with *C* such that *C* takes *VAL*, notated as $C \rightarrow VAL$.
- $DT \subset VAL$ is the symbol of data types of values.
- $CVT \subset VAL$ is the symbol of *conversion* functions for scalars, units and numeric values.
- *C* is said to *be implemented* to convey a specific concept vocabulary $V_i \in V$ iff *C* is a tuple $C = (AN, IID, G, CO, VAL, DT, CVT)$ such that $V_i \xrightarrow{\text{is assigned to}} C$, notated as $V_i = C(AN, IID, G, CO) \rightarrow VAL(DT, CVT)$, where:
  - $C(AN, IID, G, CO)$ is called an *implemented concept structure* on *V*, simply notated as *C*.
  - $VAL(DT, CVT)$ is called an *implemented concept value structure* for $C(AN, IID, G, CO)$, simply notated as *VAL*.
- An instance of an implemented concept structure $c \in C$ is a *conceptualization* of *C* iff all $an, iid, g, co \subseteq c$ respectively take their particular values such that $an \rightarrow value$, $iid \rightarrow value$, $g \rightarrow value$, $co \rightarrow value$ such that $c(value(an), value(iid), value(g), value(co))$, where $iid \in IID$, $an \in AN$, $g \in G$ and $co \in CO$. A conceptualization *c* of *C* is called as a *concept*, which is a concept in a vocabulary $V_i$ such that $c \in V_i$.
  - When all $c \in V_i$ is classified through their *iid*s on the vector concept tree (1, *i*, …, *i*) [3], we say $V_i$ is a *classified vocabulary*.
  - Recursively, if all $V_i \in V$ is classified through $V_i(value(iid), value(an), value(g), value(co))$ on the vector concept tree (1, *i*, …, *i*), we say *V* is a resource tree in the *BID* domain.
- A concept $c = c(value(an), value(iid), value(g), value(co))$ is *reified* iff its paired implemented concept value structure $val \in VAL$ is *instantiated* as $val(value(dt), value(cvt))$, and the *val* takes a particular value *value* such that $c \rightarrow value(val) \rightarrow value$.

For example, a piece of specifically conceptualized and reified business information (i.e. a reified concept) can be in the form of *c*(r.52.14.15.1.3.1, currency of price, scalarType, 0)→*val*(string, *Currency*)→ USD.

## 2.2. Business Documents

A *business document* is a composite concept of many business concepts, such as purchasing order. It specifies the *composite knowledge* of a company such that how a document concept is a composed from multiple vocabularies. It has the following characteristics:

*Unit of concept composition*. A business document is a *composite concept* consisted of a collection of document elements where each element is a concept, e.g. invoice.

*Uniquely identifiable document elements*. Each document element is an identified concept, e.g. vendor→2.8.

*Hierarchically arranged document elements*. All document elements in a document is hierarchically arranged through the vector concept tree [3], e.g. PurchasingOrder(Address(BillTo, ShipTo), ProductItems(item (name, specification, price, quantity))).

*External concept referenced*. A document element concept can reference to an external concept, e.g. the ad-

dress element can be referenced by the address concept in an organization vocabulary.

*Computing function*. A document element value can be a computing result of multiple values of other element values, e.g. the document element value of "total" concept can be the sum of the product items values.

*Document concept vocabulary*. All names of business documents are a type of business information, which can be classified in a document concept vocabulary through the vector concept tree.

More formally, a business document can be defined in the following representation.

**Definition 2** (*Business Document Domain*): *BDD*

Given a business document domain *BDD*, then *BDD* is a tuple *BDD* = (*DOC*, *D*, *T*, *E*, *EV*, *IID*, *AN*, *G*, *CO*, *RID*, *VAL*, *DT*, *FN*), where:

- *DOC* is document structure symbol.
- *D* is a document vocabulary such that for any particular document name $d \in D \in V \in BID$, $doc \in DOC$ as structure conveys the meaning of $d \in D$.
- *T* is document type symbol for specifying that the document is either conceptualized or reified.
- $E \subset DOC$ is element concept structure symbol.
- *EV* is a set of meaningful element vocabularies such that *EV* takes *E* as its syntactic structure and *EV* is the meaning conveyed in *E*.
- $IID, AN, CO, G \subset E$ are the symbols of document element concept identifier, annotation, connotation and concept group.
- $RID \subset E$ is external concept identifier symbol referenced to the external concepts such that $RID \rightarrow IID$.
- $VAL \subset E$ is the concept value structure symbol of document element.
- $DT, FN \subset VAL$ are symbols of data types and computing functions.
- *E* is said to be *implemented* to convey a particular document element vocabulary $EV_i \subseteq EV$ iff *E* is a tuple *E* = (*IID*, *AN*, *CO*, *G*, *RID*, *VAL*, *DT*, *FN*) such that $EV_i \xrightarrow{is\ assigned\ to} E$, notated as $E(IID, AN, CO, G, RID) \rightarrow VAL(DT, FN)$, where:
  - $E(IID, AN, CO, G, RID)$ is called the *implemented element concept* structure on $EV_i$, simply notated as *E*.
  - $VAL(DT, FN)$ is called the implemented element value structure for $E(IID, AN, CO, G, RID)$, simply notated as *VAL*.
- *DOC* is said to be *implemented* to convey a particular document *D* iff *DOC* is a tuple *DOC* = (*IID*, *AN*, *T*, *E*), notated as *DOC*(*IID*, *AN*, *T*, *E*), where *IID* is the symbol of document concept identifier $IID \subset D$ and *AN* is denotation of document.
- An implemented element structure $e \in E$ is a *conceptualization* of *E* iff $e(value(iid), value(an), value(co), value(g), value(rid))$, where $iid \in IID$, $an \in AN$, $co \in CO$, $g \in G$, $rid \in RID$. This *e* is called as *document element concept*.
- An implemented document structure $doc \in DOC$ is *conceptualized* iff $\forall e \subseteq E$ is conceptualized, and *IID* is instantiated to a particular $iid \in d \in D$, and *T* take a particular value "template" such that $doc = doc(value(iid), value(an), \text{"template"}, \{e\})$ This *doc* is called a *document template*.
- An *e* is *reified* iff $e(value(iid), value(an), value(co), value(g), value(rid)) \rightarrow val(dt, fn)$, where $val \in VAL$, $dt \in DT$ and $fn \in FN$.
- A document template *doc* is *reified* iff $\forall e \subseteq doc$ are reified and *T* takes the value "instance" such that $doc = doc(value(iid), value(an), \text{"instance"}, \{e \rightarrow val\})$.

For example, a simple conceptualized PurchaseOrder document template can be:

*doc(iid="d.1.2" an="purchase order" t= "template")(*
*e(iid="e.1", an="ShipTo", co="many", g="address",*
*rid = "addr345"),*
*e(iid="e.2", an="items", co="many", g="product",*
*rid = "prod23"))*

where document term *d* is (*iid*: d.1.2 *an*: purchase order) and *rid = addr345* and *rid = prod23* point to the address concept and product concepts defined in BID domain for users to reify the document in reification time.

## 2.3. Business Processes

A *business process* is a sequence of conditional operations on a set of business documents. It dynamically specifies the intra- and inter-activities of organizations as *activity pattern knowledge* [6]. Given a set of documents, a conditional operation on one document in different context may produce different resulting documents and trigger different conditional operations on them. These triggering conditions constitute different activity patterns between heterogeneous semantic communities [10] and are the issue of *business process interoperation*. For example, an operation SendQuote may send QutationSheet to receivers, where some may trigger operation ReceiveQuote if they understand the incoming SendQuote on QuotationSheet and some may simply ignore it if not.

This subsection devises the *document-based* business process in a *business process domain* (BPD).

**Definition 3** (*Business Process Domain*): *BPD*

Given a business process domain *BPD*, the *BPD* is a tuple *BPD* = (*PROC*, *P*, *O*, *IID*, *AN*, *VIS*, *DID*, *SND*, *RCV*, *S*, *LID*, *LOGIC*, *COND*, *DV*), where:

- *PROC* is process structure symbol.
- *P* is a process vocabulary such that for any particular process name $p \in P \in V \in BID$, $proc \in PROC$ as structure conveys the meaning of *p*.
- $O \subset PROC$ is process operation structure symbol.

- *IID*, *AN*, *VIS*, *DID*, *SND*, *RCV*, *S*, *LID* $\subset$ *O* are the symbols of identifier (*IID*), annotation (*AN*), and operation visibility (*VIS*) of the operation *O*, document identifier in processing (*DID* = *IID* of *DOC* $\in$ *BDD*), sender's address of (*SND*), receiver's address (*RCV*), process operation status (*S*), and proposed document processing logic identifier (*LID*), where visibility *VIS* has the status such as "public", "private" and "partner" to restrict the nature of the operation *O*, and process operation status *S* has status of "arrived", "acknowledged", "processed" and "sent".
- *LOGIC* is the symbol of a document processing logic identified by *LID*, in the form of a computing logic.
- *COND* $\subset$ *LOGIC* is the symbol of document processing conditional result.
- *DV* is the symbol of conditional value of *COND*.
- An process operation *O* is said to be *implemented* iff *O* is a tuple *O* = (*IID*, *AN*, *VIS*, *DID*, *SND*, *RCV*, *S*, *LID*), notated as *O*(*IID*, *AN*, *VIS*, *DID*, *SND*, *RCV*, *LID*) where *DID* identifies the incoming document and *LID* identifies processing logic *LOGIC*.
- *LOGIC* is said to be *implemented* iff *LOGIC* is a tuple *LOGIC* = (*LID*, *DID*, *COND*, *DV*, *O*) such that *LOGIC*(*LID*, *DID*, *COND*(*DV*))$\rightarrow$*O*, where *O* is the outgoing process operation.
- *PROC* is said to be *implemented* iff *PROC* is a tuple *PROC* = (*IID*, *AN*, *O*) such that for (*IID*, *AN*) $\in$ *P*, *PROC*(*IID*, *AN*, *O*).
- An implemented process operation *o* $\in$ *O* is said to be *conceptualized* iff all its elements are conceptualized such that *o*(*value*(*iid*), *value*(*an*), *value*(*vis*), *value*(*did*), *value*(*snd*) = EMPTY, *value*(*rcv*) = EMPTY, *value*(*s*) = EMPTY, *value*(*lid*)). This *o* is called as *process operation concept*.
- An implemented business process *proc* $\in$ *PROC* is said to be *conceptualized* iff all *o* $\in$ *O* of *proc* is conceptualized. This *proc* is called as *process template*.
- A conceptualized process is said to be *reified* iff one of its operation is triggered to process an incoming document and accordingly changes its status *S*.

For example, a conceptualized business offer process may include the following four process operations:

*proc(iid = "p.3", an = "offer processing")*
*o(iid = "p.3_1", an="RequestOffer", vis = "public",*
*did="InquirySheet", snd, rcv, s, lid = "processInquiry"),*
*o(iid = "p.3_2", an="ProcessOffer", vis = "private",*
*did="ReceivedInquiry", snd, rcv, s, lid = "processOffer"),*
*o(iid = "p.3_3", an="ProveOffer", vis = "private",*
*did="UnprovedOffer", snd, rcv, s, lid = "proveOffer"),*
*o(iid = "p.3_4", an="MakeOffer", vis = "public",*
*did="ProvedOffer", snd, rcv, s, lid = "makeOffer").*

where each operation has an operation logic identified by *lid* to process the incoming business document identified by *did*. The processing triggers a forward operation in the sequence and produces an outgoing document.

In next section, we will describe the sharing relationship between the domains of BID, BDD and BPD in a tree-layer TRANSCODE model.

## 3. Three-Layer TRANSCODE Model

The three-layer TRANSCODE Model shown in Fig. 1 describes the knowledge sharing relationship, and states how the integrated business information can be shared in business document integration and how the integrated business documents can be shared in business process integration. The key to understanding the Model is *structure*, *concept*, and the *relationship* between structure and concept [4].

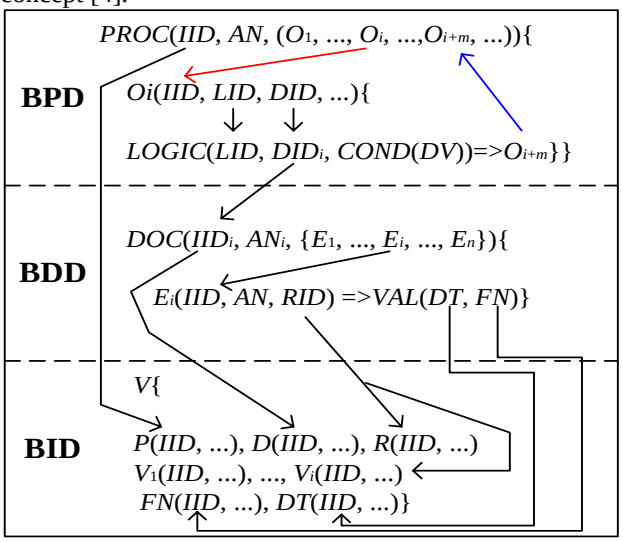


**Fig. 1: A three layer TRANSCODE model**

The Model consists of three layers. The bottom layer is the layer of *business information domain* (BID), where basic knowledge of business information is designed in a *vocabulary tree V*, which consists of *concept vocabularies* (*R*, *D*, *P*, $V_1$, …, $V_i$, …, $V_n$) of products, documents, processes and others. Each specific vocabulary $V_i$ is a set of *concepts* $\forall c \in C$ such that *c* has a unique identifier *iid* $\in$ *IID* that uniquely identifies the meaning of the concept *c* conveyed in a *concept structure* *c*(*an*, *iid*, *g*, *co*) and semantically conceptualized as *c*(*value*(*an*), *value*(*iid*), *value*(*g*), *value*(*co*)). In this layer, the basic data types (either primitives or compounds) are designed as a special *data type vocabulary DT*, which is used to reify vocabularies. A set of conversion functions for converting *scalars* (e.g. USD $\rightarrow$ GBP, AUD or liter $\rightarrow$ gallon) and *units* (e.g. dozen $\rightarrow$ piece) is designed as a special *conversion function vocabulary CVT* for being used in heterogeneous business information transformation.

The middle layer is the layer of *business document domain* (BDD), where the composite knowledge of business documents is designed in a set of *business document templates* such that $\forall doc \in DOC \subseteq$ BDD. Each template

*doc* is identified by a document concept identifier *iid* ∈ *d* ∈ *D* ∈*BID*, and consists of a set of document elements {*e*} ⊆ *E*. Any *e* has an element identifier *iid* ∈ *e* and an external concept identifier *rid* ∈ ($r \cup o \cup v_i$) ($r \in R$, $o \in O$ and $v_i \in V_i$ in BID) that semantically defines the meaning of element identifier such that *rid* → *iid*. Thus, both documents and document elements reuse the already-defined concepts of vocabularies. However, these reuses are independent of document templates *doc*(*value*(*iid*), …)(*e*$_1$(*value*(*iid*), *value*(*rid*), …), …, *e*$_i$(*value*(*iid*), *value*(*rid*), …)), where each of them is conceptualized from the given document structure *doc*(*iid*, …)(*e*$_1$(*iid*, *rid*, …), …, *e*$_i$(*iid*, *rid*), …) ∈ *DOC* in BDD.

The top layer is the layer of *business process domain* (BPD), where the activity pattern knowledge of business processes is designed in a set of *business process templates* such that ∀*proc* ∈ *PROC*. Each template *proc* is identified by a process concept identifier *iid*∈*p*∈*P*⊆*BID*, and consists of a sequence of *process operations* such that ∀$O_i$ ∈ *O*. Any $O_i$ has an operation identifier *iid* ∈ *IID* and operates on a business document template identified by a document concept identifier *did* ∈ $O_i$. This *did* calls and reuses an existing document template *doc* through calling *logic*∈*LOGIC* via logic identifier *lid*∈*LID*∈*LOGIC* by mapping *did* onto *iid* ∈ *doc*∈*DOC*⊆BDD. After *doc* is called, the corresponding *logic* computes the logic value *dv* of *cond*∈*COND* to trigger subsequent operation $O_{i+m}$∈*proc* in operation sequence until termination operation. In this layer, both document and logic templates *doc* and *logic* are reused, but independent of business process template *proc*(*value*(*iid*), *value*(*an*), {$O_1$(*value*(*iid*), *value*(*did*), *value*(*lid*), …), …, $O_i$(*value*(*iid*), *value*(*did*), *value*(*lid*), …)}), which is conceptualized from the process structure *proc*(*iid*, *an*, {$O_1$(*iid*, *did*, *lid*, …), …, $O_i$(*iid*, *did*, *lid*, …)}) ∈ *PROC* in BPD.

The above Model utilizes concept identifiers generated on *vector concept tree* (1, *i*, …, *i*) [3] to realize the sharing and integration of business information, business documents and business processes. It provides the features of the high reuse of individual integration results in BID and BDD and the flexibility of independent design of business information, documents and processes.

# 4. XML Implementation of TRANSCODE

Since business information, business documents and business processes are independently structured, conceptualized, and reified, the TRANSCODE model can be independently implemented into three different XML specifications: *XML business information* (XBI), *XML business document* (XBD) and *XML business process* (XBP) to demonstrate the feasibility of the Model.

## 4.1. XML Business Information

Given a vocabulary structure *V* = *C*(*AN*, *IID*, *G*, *CO*) → *VAL*(*DT*, *CVT*), its XML DTD can be provided:

```
<!ELEMENT voc (c*)> <! -- voc.dtd -->
<!ATTLIST voc an CDATA #REQUIRED  iid ID #REQUIRED>

<!ELEMENT c (c*,val?)>
<!ATTLIST c  iid ID #REQUIRED   an CDATA #REQUIRED
          co CDATA #REQUIRED  g CDATA #REQUIRED>

<!ELEMENT val (#PCDATA)>
<!ATTLIST val  cvt CDATA #IMPLIED dt CDATA #REQUIRED>
```

With this DTD, any business information vocabulary e.g. product information *r* ∈ *V* can be *syntactically structured*, and further *semantically conceptualized* and *reified* as following:

```
<?xml version="1.0"?><!DOCTYPE voc SYSTEM "voc.dtd">
<r iid="r" an="product information">
    … …
  <c iid="r.1.1" an="electronic appliance" g="category" co="*">
     <c iid="r.1.1.1" an="refrigerators" g="product" co="*">
      <c iid="r.1.1.1.1" an="color" g="constant" co="0">
            <val dt="string">white</val></c>
      <c iid="r.1.1.1.2" an="price" g="attribute" co="3">
            <c iid="r.1.1.1.2.1" an="currency" g="scalar" co="0">
              <val dt="string" cvt="Currency">USD</val> </c>
            <c iid="r.1.1.1.2.2" an="value" g="value" co="0">
              <val dt="decimal" cvt="Value">700</val></c>
            <c iid="r.1.1.1.2.3" an="unit" g="unit" co="0">
              <val dt="string" cvt="Unit">piece</val></c></c></c></c>
    … …
</r>
```

In the example, if no element *val* is included, then the *r* is only a conceptualization (i.e. a set of concepts).

## 4.2. XML Business Document

Given a business document structure *DOC* = *DOC*(*IID*, *AN*, *T*, *E*(*IID*, *AN*, *CO*, *G*, *RID*) → *VAL*(*DT*, *FN*)), its XML DTD can be presented:

```
<!ELEMENT doc (e*)> <! -- doc.dtd -->
<!ATTLIST doc an CDATA #REQUIRED
        iid ID #REQUIRED t CDATA #IMPLIED>

<!ELEMENT e (e*,val?)>
<!ATTLIST e
        an CDATA #REQUIRED  co CDATA #IMPLIED
        g CDATA #IMPLIED  iid ID #REQUIRED
        rid CDATA #IMPLIED>

<!ELEMENT val (#PCDATA)>
<!ATTLIST val  dt CDATA #REQUIRED fn NMTOKEN #IMPLIED>
```

This DTD has syntactically implemented the business document structure *DOC*, and can be semantically conceptualized e.g. a purchase order template as following:

```
<?xml version="1.0"?><!DOCTYPE doc SYSTEM "doc.dtd">
<doc iid="d.5" an="purchase order" t="template">
   <e iid="e.1" an="order date" co="0" g="date" rid="t.3"/>
   <e iid="e.2" an="address" co="2" g="org" rid="addr">
      <e iid="e.2.1" an="ship to" co="*" g="addr" rid="addr"/>
      <e iid="e.2.2" an="bill to" co="*" g="addr" rid="addr"/></e>
   <e iid="e.3" an="product items" co="*" g="product">
      <e iid="e.3.1" an="refrigerator" co="0" g="ref11" rid="ref11"/>
      <e iid="e.3.2" an="quantity" co="0" g="unit" rid="u.2.5"/>
```

```
    <e iid="e.3.3" an="price" co="0" g="value" rid=""/></e>
  <e iid="e.3.4" an="ship date" co="0" g="date" rid="t.3"/>
</doc>
```

In above document template, the value of "*co*" refers to the number of lower level connotation concepts (e.g. *co*="0" means no connotation), the value of "*g*" refers to group concept identifier (e.g. *g*="date" means date concept), and the value of "*rid*" refers to the concept reuse that is identified in group concept (e.g. *rid*="t.3" is a concept reuse in "date" group concept"). If *co*="0" and *value*(g)=*value*(rid), the *rid* is the reuse of group concept only (e.g. *rid*="ref11"). If *co*="*" and *value*(*g*)=*value*(*rid*), the *rid* reuses group concept and all its lower level connoted concepts during reification. For example:

```
<e iid="e.2.1" an="ship to" co="*" g="addr" rid="addr">
  <e iid="e.2.1.1" an="name" co="0">
    <val dt="string">Collins</val></e>
  <e iid="e.2.1.2" an="street">
    <val dt="string">7 Broadway</val></e>
  <e iid="e.2.1.3" an="city">
    <val dt="string">New York</val></e>
  <e iid="e.2.1.4" an="state">
    <val dt="string">NY</val></e>
  <e iid="e.2.1.5" an="country">
    <val dt="string">USA</val></e>
  <e iid="e.2.1.6" an="zip"><val dt="pint">10002</val></e></e>
```

The conceptualized and reified purchase order example illustrated above has demonstrated the flexible reuse of the external concepts in BID.

### 4.3. XML Business Process

Given a business process structure *PROC* = *PROC*(*IID*, *AN*, *O*(*IID*, *AN*, *VIS*, *DID*, *SND*, *RCV*, *LID*) and *LOGIC*(*LID*, *DID*, *COND*(*DV*))→*O*, its XML DTD implementation is as following:

```
<!ELEMENT proc (o*)> <!-- proc.dtd -->
<!ATTLIST proc   iid ID #REQUIRED   an CDATA #REQUIRED>
<!ELEMENT o (logic)>
<!ATTLIST o
    iid ID #REQUIRED an CDATA #REQUIRED
    did CDATA #REQUIRED  lid CDATA #REQUIRED
    rcv CDATA #REQUIRED  snd CDATA #REQUIRED
    vis NMTOKEN #REQUIRED>
<!ELEMENT logic (cond*)>
<!ATTLIST logic  lid ID #REQUIRED  did CDATA #REQUIRED>
<!ELEMENT cond (#PCDATA)>
<!ATTLIST cond dv CDATD #REQUIRED>
```

Based on this DTD, process designers can create business process templates, e.g. an offer process:

```
<?xml version="1.0"?><!DOCTYPE proc SYSTEM "proc.dtd">
<proc iid="p.3" an="offer process">
 <o iid="p.3_1" an="RequestOffer" vis="public" did="InquirySheet"
    snd="" rcv="" lid="processInquiry">
   <logic lid="p.3_1-lgc" did="InquirySheet">
     <cond dv="1">p.3_2</cond>
     <cond dv= "e">exception</cond></logic></o>
  <o iid="p.3_2" an="ProcessOffer" vis="private"
    did="ReceivedInquiry" snd="" rcv="" lid="processOffer">
   <logic lid="p.3_2-lgc" did="ReceivedInquiry">
     <cond dv="1">p.3_3</cond><cond dv="2">p.3_5</cond>
     <cond dv="e">exception</cond></logic></o>
  <o iid="p.3_3" an="ProveOffer" vis="private" did="UnprovedOffer"
    snd="" rcv="" lid="proveOffer">
   <logic lid="p.3_3-lgc" did="UnprovedOffer">
     <cond dv="1">p.3_4</cond>
     <cond dv="e">exception</cond></logic></o>
  <o iid="p.3_4" an="MakeOffer" vis="public" did="ProvedOffer"
    snd="" rcv="" lid="makeOffer">
   <logic lid="p.3_4-lgc" did="ProvedOffer">
    <cond dv="1">p.3_5</cond>
     <cond dv="e">exception</cond></logic></o>
  <o iid="p.3_5" an="MakeOffer" vis="public" did="ReceivedOffer2"
    snd="" rcv="" lid="makeOffer">
   <logic lid="p.3_5-lgc" did="ReceivedOffer2">
     <cond dv="1">p.3_6</cond>
     <cond dv="e">exception</cond></logic></o>
</proc>
```

In this offer processing process template, only process operations p.3_1, p.3_4 and p.3_5 are set as "public". Thus, the sender does not know how the receiver processes the offer internally. The *visibility* (vis) feature is very important because nearly any company does not allow other companies to know its internal business processing. This process template also allows the flexible triggering of subsequent operations in processing through conditional value (*cond*). For example, for operation *iid*="p.3.2" with logic *lid* = "p.3_2-lgc", if its conditional value *dv*=1, then the next followed process operation is p.3_3, which requires an approval for any outside quotation. But if the result is *dv*=2, then the processed offer has no requirement for approval and the next process operation is *iid*="p.3_5".

## 5. Discussion and Related Work

The TRANSCODE approach focuses on the vertical integration of business information, business documents and business processes between multiple organizations.

### 5.1. TRANSCODE on Product Map Theory

The underlying theory of TRANSCODE approach is Product Map [5], which represents a sign (i.e. representation) as a couple of *structure* and *concept*. Structure is meaningless if no context is imposed on. It becomes meaningful only after a concept (i.e. a contextual meaning) is conveyed. Metaphorically, a piece of paper is structure and understandable words on it are concepts. Since any concept has denotation that is again specified by connotation, the conveyed structure of the concept hence presents the feature of hierarchy (e.g. a vocabulary or a document hierarchy). Since concept denotation specifically defines concept in a particular position of a concept hierarchy, a concept can thus be uniquely identified by its hierarchical position (which produces IID). In a given context, a structure can be implemented as a certain form to generically convey meanings. However, an implemented structure does not necessarily lead to any concept if no one conveys meanings onto the structure. Thus, conceptualization of implemented structures (i.e. concepts) is needed to add

new concepts to vocabularies, libraries of document and process templates. After concepts are available, they can be used to describe the particular phenomena, which is a process of concept reification.

The thought of *structure* and *concept* makes us possible to vertically integrate information, documents and processes of businesses into a three-layer TRANSCODE model, where each domain is independent. In each domain, structure is *separated* from concepts, and concepts are *separated* from their reifications. The semantic linking between three separate domains (BID, BDD and BPD) is through the unique concept identifiers IID.

### 5.2. Comparing TRANSCODE with ebXML

The ebXML (www.ebxml.org) [11] is an important *de facto* industrial standard for global business data exchange. It allows contextually different companies to discover, register and reuse business information entities. Comparing with TRANSCODE developed in this paper, there are some points of differences.

*Business data representation*. The ebXML represents business data in monolithic Core Components (aggregated core component(basic core components(data type), associated core components)) while TRANSCODE represents business data (business information, documents and processes) in three separate aspects: structure, concept and reification. For ebXML, a business concept is defined as soon as the structure of a Core Component is created. The business definitions (i.e. concept) on Core Components are immediate and they are reflected on the *terms* (similar to IID of TRANSCODE) for Core Components themselves. For TRANSCODE, structure (e.g. DTD) is only syntax without any business semantics. It conveys business concepts (simply the pair values of IID and AN) only after business semantics is *collaboratively* designed at concept design time [6], and the reification of concepts is even postponed at use time.

The capability of separating structure from concept and reification implies that TRANSCODE provides not only the flexibility of the autonomous design of concepts but also the reuse of existing integration results. Business integration systems can be divided into three independent components: system design, concept design and concept use. In system design phase, systems design and maintain the integration structures (e.g. DTDs of XBI, XBD and XBP). In concept design phase, the concept designers collaboratively design the concepts. In reification phase, the concept users simply reify the already-defined concepts for routine business processing, without needing to know any integration tasks. This again implies that millions of non-experienced users can freely participate in integrated systems with lower cost and no technical obstacles.

*Understanding of business contextual semantics*. Both ebXML and TRANSCODE aim to resolve business data interoperation problem between different business contexts. Nevertheless, their approaches to the issue are different due to the understanding of business contextual semantics. The ebXML starts with relaxing the traditional business standards from static message definitions that have not enabled a sufficient degree of interoperability or flexibility. Thus, it adopts the solution of controlled vocabularies to create a relaxed business standard for interoperating large standards such as EDI and SWIFT. In such solution, users can register users' vocabularies, discover and reuse the already-registered vocabularies following the controlling rules of ebXML. Under this circumstance, users' contextual business semantics for documents and processes (i.e. Business Information Entities - BIE) base on the controlled semantics of Core Components through strict association. The result is that users have to understand what ebXML is in order to register and discover BIEs. This is not optimistic to SMEs (e.g. a 5-people company) in both financial and technical aspects.

TRANSCODE considers SMEs and has thus adopted the solution of collaborative concept mapping [6] as its external integration. That is, all SME are unique business contexts, which interoperate with each other through collaborative concept mapping via concept IID. With this solution, each SME maintains individual business context to achieve personalization. Specific to this paper, the vertical integration of business information, documents and processes becomes easy because the vertical integration happens within the individual context of a business organization. The implication is that companies are not necessary to tightly conform to controlled vocabularies like ebXML. What they require is to incrementally map their personalized vocabularies, document templates, and process templates onto those published in TRANSCODE business data providers through a simple given client program, whenever they need.

There are many other subtle differences between ebXML and TRANSCODE, which will not discuss here. In summary, TRANSCODE is a complement approach of ebXML that vertically integrates business data within an individual business context that is collaboratively mapped with others.

### 5.3. Other Related Work

An early vertical integration research about business data could be found in [9]. This work proposed to use an RDFT bridge to map heterogeneous concepts in each layer (i.e. product data for www.UNSPSC.org and www.eclass.de, business documents for www.cXML.org and www.xCBL.org and business processes). Nevertheless, other researches seldom cover vertical integration of business information, documents and processes. Most researches discuss business integration on individual level of product data integration [1], business document integration (www.UDEF.org), or business process integration [13]. In BID layer of TRANSCODE, business informa-

tion (BI) relates to ontology, which can be compared with BI vocabulary. However, existing ontology definitions or representations are diverse. A general comparison is difficult. Given Gruber's definition ("an explicit specification of a conceptualization") [2] or Uschold-Gruninger's definition ("a shared understanding of some domain of interest") [12], BI vocabulary resembles ontology in terms of explicitness and sharing understanding such that a BI vocabulary is an explicit collaboration result.

## 6. Conclusion

This paper has proposed a novel TRANSCODE approach to resolve the issue of the vertical integration of business information, business documents and business processes, where most existing integration solutions only focus on individual aspect of the above. This approach firstly represents business information, business documents and business processes in three independent structures, and then aligns these structures in a three-layer TRANSCODE Model. In this Model, business information domain provides basic knowledge of business organizations, business document domain provides composite knowledge of business document templates, and business process domain provides activity pattern knowledge of business process templates. The lower layer domain knowledge is reused and integrated into higher layer domain through unique concept identifiers. To test the feasibility of TRANSCODE Model, the Model has further been implemented in three XML specifications - XBI, XBD and XBP. The included examples of these specifications has demonstrated that the abstract TRANSCODE Model can be implemented, and the concept identifiers IID is an effective vehicle for semantically link lower layer concepts with higher layer concepts for reuse.

In this paper, an important methodology for business integration is implied, that is, the separation of structure from concepts, and the separation of concepts from reification. This methodology fully utilizes the characteristics of structure and concept developed in Product Map [5]. The separation enables the flexible design and use of business integration systems.

This paper is a mature part of the ongoing research project for globally integrating semantically heterogeneous business information, business documents and business processes. It has built a core representation limited to a set of homogeneous business organizations. Some future work include mapping the core representation with ad hoc semantically heterogeneous representations, the creation of conversion function library, and the contextual value translation between different natural languages.

## 7. Acknowledgements

The work reported in this paper has been partially supported by the University of Macau Research Grand.

## 8. References

[1] Fensel, D., Ding, Y., Omelayenko, B., Schulten, E., Botquin, G., Brown, M. and A. Flett, "Product Data Integration in B2B E-Commerce", *IEEE Intelligent Systems*, Vol. 16, No. 4, 2001, pp. 54-59.

[2] Gruber, T., "Toward Principles for the Design of Ontologies Used for Knowledge Sharing", Technical Report KSL-93-04, 1993.

[3] Guo, J. and C. Sun, "Context Representation, Transformation and Comparison for Ad Hoc Product Data Exchange", in: *Proc. of the 2003 ACM Symposium on Document Engineering*, ACM Press, 2003, pp. 121-130.

[4] Guo, J., Sun, C. and Chen, D., "Deconstruction and Reconstruction of Heterogeneous Electronic Product Catalogues for Semantic Interoperation", in*: Proc. of the 2004 IEEE Conf. on E-Commerce Technology* (*CEC'04*), IEEE Computer Society Press, 2004, pp. 333-336.

[5] Guo, J., *Integrating Ad Hoc Electronic Product Catalogues through Collaborative Maintenance of Semantic Consistency*, Chapter 4, PhD Thesis, Griffith University, 2005, http://www4.gu.edu.au:8080/ adt-root/ public/ adt-QGU20050824.125257.

[6] Guo, J, Sun, C. and D. Chen, "Transforming Heterogeneous Product Concepts through Mapping Structure", in: *Proc. of 2004 Int'l Conf. on Cyberworlds (CW'04)*, IEEE Computer Society Press, 2004, pp. 22-29.

[7] Manheim, M. and M. Fritz, "Information Technology Tools to Support Virtual Organization Management: A Cognitive Informatics Approach", *Organizational Virtualness*, Sieber, P. and J. Griese (Eds.), 1998, pp. 137-153.

[8] Medjahed, B., Benetallah, B., Bouguettaya, A., Ngu, A. and A. Elmagarmid, "Business-to-business Interactions: Issues and Enabling Technologies", *The VLDB Journal*, 12, 2003, pp.59-85.

[9] Omelayenko, B., Fensel, D. and Bussler, C., "Mapping Technology for Enterprise Integration", in: *Proc. of the 15th Int'l FLAIRS Conf.*, USA, 2002, pp. 419-424.

[10] Robinson, M. and L. Bannon, "Questioning Representations", in: *Proc. of ECSCW'91*, Amsterdam, Holland, 1991, pp. 219-233.

[11] UN/CEFACT, *ebXML Core Components Technical Specification Version 1.85 Draft*, United Nations Centre for Trade Facilitation and Electronic Business, Sept. 30 2002.

[12] Uschold, M. and M. Gruninger, "Ontologies: Principles, Methods and Applications", Knowledge Sharing and Review 11(2), 1996, pp.93-155.

[13] Wombacher, A., Fankhauser, P., Mahleko, B. and E. Neuhold, "Matchmaking for business processes based on conjunctive finite state automata", *Int. J. Business Process Integration and Management*, Vol. 1, No. 1, 2005, pp. 3-11.