An Approach for Application Ontology Building and Integration Enactment

H. Guergour, R. Driouche and Z. Boufaïda.

***Abstract*— The problem dealing with heterogeneity, even semantic has been deeply investigated in the field of ontology. We reflect upon the suitability of the ontology as a candidate for solving the problem of heterogeneity and ensure greater interoperability between applications.**

As a consequence, we proposed an ontological approach for application ontology building, which can be profitably exploited for integrating applications.

We describe in this paper how ontologies may be used to model heterogeneous applications. We first look at what aspects need to be described for the purpose of application model design in the context of enterprise integration. Then we show how these aspects are related to each others.

***Index Terms*— Application ontology, EAI, interoperability, Semantic heterogeneity.**

I. Introduction

A new technology called EAI (Enterprise Application Integration) has emerged as a field of enterprise integration. In essence, EAI provides tools to interconnect multiple and heterogeneous enterprise application systems such as CRM (Customer Relationship Management), SCM (Supply Chain Management), ERP (Enterprise Resource Planning) and legacy systems. The most difficulty of this interconnection is due to the fact that the integrated systems were never designed to work together [12], [14].

Collaboration of heterogeneous partners leads to the interoperability issue [2], which represents a major barrier in the business sector. Obstacles to heterogeneity arise from the fact that partners do not share the same semantics for the terminology of their business process models. Moreover, they use various collaboration scenarios with different organizational constraints. In addition, the growing heterogeneity of standards for information interchange implies that no partner has enough power to impose their standard. So, semantic heterogeneity occurs because there is a disagreement about the meaning, i.e. inconsistent interpretation. In the semantic Web, ontologies are often seen as new solutions providing semantically enriched information exchange facilities [13]. They provide a common terminology that captures key distinctions in business domain.

In our context, integration is the process of linking heterogeneous applications, to make a unit complete and confers to it properties related to the interoperability and the coherence of applications. Many attempts have been made to integrate different applications. In most approaches, the remaining problems are still twofold. They are developed for specific business sectors and they do not cope with the challenge of incorporating semantics into applications [6], [7]. An architecture based on ontology is often seen as new solution providing exchange facilities of semantically enriched information. It supports mapping process for integrating local ontologies related to heterogeneous and distributed applications. For us, ontologies should be used for two main reasons: first, for modeling the application's structure and behaviour in a precise and rigorous way and second, for representing vocabularies and providing semantic rules of mapping in order to integrate enterprise applications [6].

The rest of the paper is organized as follows: Section 2 outlines some important related work. The section 3 shows more details on our architecture for application integration, gives a description of the two levels: applicative and collaborative. Section 4, describes our application ontology building process based on Methontology [18]. Finally, Section 5 discusses conclusion and sketches future work.

II. Related work

EAI is the process of adapting a system to make distributed and heterogeneous applications work together to carry out a common objective [12]. In companies, the essential requirement for heterogeneous and distributed applications is to be able to exchange information and services with other ones in a semantically rich and sound way. Thus, semantics should be captured, verified and used to validate reliable information exchange. This is commonly referred to as the problem of interoperability. Ontology is an appropriate way to enable interoperability. It includes an explicit description of both a domain structure and the related terms describing this

Manuscript received October 23, 2006. This work was supported by the LIRE Laboratory of Computer Science Department, Mentouri University of Constantine.

H. Guergour is with the LIRE Laboratory, Department of Computer Science, Mentouri University of Constantine, 25000, Algeria, (phone: 213-3181-8817; fax: 213-3181-8817; e-mail: habib_guergour@ yahoo.fr).

R. Driouche is with the LIRE Laboratory, Department of Computer Science, Mentouri University of Constantine, 25000, Algeria, (phone: 213-3181-8817; fax: 213-3181-8817; e-mail: driouchera@ yahoo.fr).

Z. Boufaïda, is with the LIRE Laboratory, Department of Computer Science, Mentouri University of Constantine, 25000, Algeria (e-mail: boufriche@hotmail.com).

domain. It allows applications to agree on the terms, they use when communicating.

However, an EAI model provides the language used to specify an explicit definition of an enterprise. It must have the expressiveness to capture the sets of applications, its activities that they perform and the resources required by these activities. One of the basic concepts, which enable us to capture the integration, is the structure, the behaviour and the domain of the application.

The focus of this paper is on the application ontology building process based on Methontology [18].

A range of methods and techniques have been reported in the literature regarding ontology building methodologies [15]. Mike Uschold's methodology [16], Michael Grüninger and Mark Fox's methodology [17] and Methontology [18] are the most representative. Grüninger methodology is only limited to ontologies using first-order-logic languages. Uschold's and Methontology have a common that they start from the identification of the ontology purpose and the need for domain knowledge acquisition. Uschold proposes a codification of knowledge in a formal language. In Methontology, a set of intermediate representations independent of the formal language to be used is expressed. Thus, Methontology enables experts and ontology designers who are unfamiliar with implementation environments to build ontologies from scratch.

For the ontology evaluation, Ushold's methodology includes this activity but does not state how to carry it out. Grüninger and Fox propose the identifying a set of competency questions. Evaluation in Methontology occurs throughout the ontology development.

For our purpose, we have chosen the Methontology for the application ontology building. It enables the construction of ontologies at the knowledge level. It includes the identification of the ontology development process, a life cycle based on evolving prototypes and particular techniques to carry out each activity.

III. Integration Architecture

In the system, we identify several types of legacy, client/server and Web applications, developed using different programming languages. They work on different operating system platforms and use various format for the exchange of data. By using application ontologies, we enhance communication between applications, for the benefit of integration. Hence, ontologies serve as stable basis for understanding the requirements for the user applications [10].

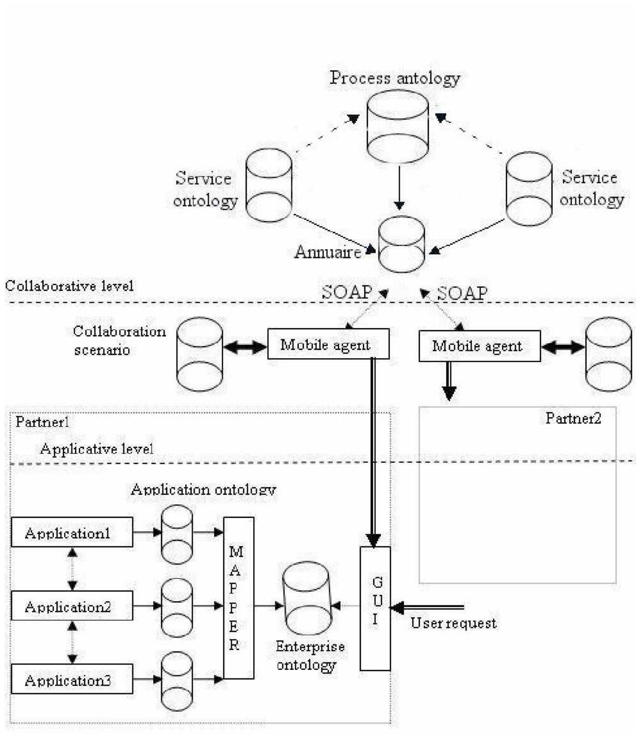

Fig.1. Integration system architecture

The integration system we propose aims at offering a support for integrating heterogeneous and distributed applications, accessing multiple ontologies (Fig.1). It provides a communication framework as a central service. It permits an appropriate exchange of information between applications ontologies and generates the global one. The introduced framework tries to enhance the ontology mapping, which enables the reuse of mapping information for managing heterogeneity. The integration process is based on the semantic bridges to indicate the semantic equivalence of ontology entities for assembling them. These applications are linked seamlessly to partners, vendors and suppliers through a common interface.

Furthermore, we give an overview about the two-level approach for application integration, the applicative level and the collaborative one:

- Applicative level consists of heterogeneous and distributed applications. Each application has its own local ontology. Our important direction is the development of a communication framework for ontology mapping. In our architecture, we aim to overcome the gap between local ontologies application, according to the semantic relations. A special component, named mapper, is invoked to perform its tasks for building the global ontology. The latter can be seen as enterprise ontology and permits the resolution of semantic conflicts in both concepts and attributes [10].

- Collaborative level takes place in the business process collaboration with partners. Each company has a mobile agent that is responsible for requesting and providing the services and the negotiation for selecting the best partner basing itself on criteria (e.g., price limits, product configurations or delivery deadlines). It uses the collaboration scenario for achieving business process. The mobile agent permits to perform the integration tasks according to process ontology and using optimized itinerary. The latter improves the quality of the system and reduces the response time. The EbXML [20] extended scenario is based on the integration of agent paradigm to guarantee a saving of search time, to negotiate business parameters and to offer a great performance especially in the presence of the characteristic of mobility which solves problems related to the networks while decreasing consumption in resources networks [22].

IV. Building application ontology

In this section, we will build an application ontology which concerns EAI domain. For this purpose, the ontology consists of a classification of relevant characteristics of applications. We have some pertinent information about the application, such as application-behaviour, application-domain, application-structure …etc. These concepts are inspired from EAI domain [10], [12], Web services [4], [21] and middleware technologies [1]. Let us start the building of application ontology using Methontology [18], we have affected two modifications there, the first on the specification phase and the second on the conceptualization one.

To determine the purpose of ontology, we follow the specification phase describes in [19]. In this work, an RDF document is created to describe ontology to be built through its objective, its developers, its creation date, its scope, etc....

In the conceptualization phase, we fused the step of construction of concepts classification trees with the step of construction of a relations binary diagram in order to show all ontology concepts in single diagram and to have a clear understandable view of all ontology concepts.

A. Specification

We suggest starting the development of ontology, the definition of its domain and its scope. Thus, we need to answer at some fundamental questions:

• Which domain will cover ontology?

• In which purpose ontology will be used?

Ontology that we come to build concerns the domain of the enterprise application integration, we want to specify well the concepts relating to this domain and relations between them. These concepts must describe various types of applications, their models, their structures and the domain to which applications belong.

We summarize this phase in RDF document presented in figure 2.

```
<rdf:RDF>
…
<rdf: Description about=" URI of ontology" >
<Domain> Enterprise Application Integration </Domain >
<Date> September 30, 2006 </ Date >
<Developed-by>
  <rdf:Sequence>
       <rdf:_1  H. Guergour, LIRE laboratory University of Mentouri >
       <rdf:_2  R. Driouche, LIRE laboratory University of Mentouri>
       <rdf:_3  Z. Boufaïda, LIRE laboratory University of Mentouri >
  </rdf:Sequence>
</Developed -by>
  <Purpose> Ontology modeling the behavioral and structural properties of
  an application and properties of domain in which the application belongs.
  This ontology will facilitate the integration of different applications of the
  enterprise </ Purpose>
<Level_of_formality   formel />
<Terms>
   <rdf:Sequence>
       <rdf:_1  Application> <rdf:_2 Application-Behavior>
       <rdf:_3 Application-Structure > <rdf:_4 Application-Domain >
       <rdf:_5 Activity><rdf:_6 Atomic-activity >
       <rdf:_7 Composite-activity ><rdf:_8 Sequence-activity >
       <rdf:_9 Split-activity >    <rdf:_10 Choice-activity >
       <rdf:_11 Repeat-Until-activity > <rdf:_12 Application-model >
       <rdf:_13 Input > <rdf:_14 Parameter ><rdf:_15 Output >
       <rdf:_16 Precondition><rdf:_17 Effect >
       <rdf:_18 Computed-parameter> <rdf:_19 Computed-input >
       <rdf:_20 Computed-output ><rdf:_21 Computed-precondition >
       <rdf:_22 Computed-effect> <rdf:_23 IDL-description >
       <rdf:_24 XDR-description > <rdf:_25 WSDL-description >
       <rdf:_26 Creator ><rdf:_27 Date ><rdf:_28 Interface-structure >
       <rdf:_29 Product > <rdf:_30 Method-structure >
       <rdf:_31 Transportation > <rdf:_32 Input-structure >
       <rdf:_33 Output-structure > <rdf:_34 Level >
       <rdf:_35 Domain-description > <rdf:_36 Functional- description >
       <rdf:_37 Non- Functional-description> …
   </rdf:Sequence>
</Terms>
<Sources>
   <rdf:Sequence>
    <rdf:_1 " An ontology for semantic middleware: extending DAML-S
         beyond Web services">
    <rdf:_2 "Enterprise Application Integration". Edition Addison-Wesley,
         Boston et al. 2003.>
    <rdf:_3 Semantic Web Service Tutorial >
    <rdf:_4 rdf:resource= " kmi.open.ac.uk/projects/dip/resources/hicss-
         39/HICSS06-slides.ppt ">
   </rdf:Sequence>
</Sources>
</rdf:description>
</rdf:RDF>
```

Fig.2 RDF-document specification for Application ontology

B. Conceptualization

After the acquisition of the majority of knowledge in the first phase, we must organize and structure them by using semi-formal or intermediate representations which is easy to understand and independent of any implementation language. This phase contains several steps which are: Build the glossary of terms; Build the binary-relations and concepts classification diagram; Build the concepts Dictionary; Build the relations-tables; Build the attributes-tables; Build the logical-axioms table; Build the instances-table.

1) Build the glossary of terms: This glossary contains the definition of all terms relating to the domain (concepts, instances, attributes, relations) which will be represented in the application ontology, for example, in our case the terms Activity and E-Commerce are concepts but Set-of and Has-precondition represent relations. The table 1 provides a detailed list of the various terms used in ontology.

TABLE I
TABLE OF TERMS GLOSSARY

Term name	Description
Application	An entity, a program or a set of activities having a certain behavior, a structure and a domain to which it belongs.
Application-Behavior	Sub-ontology modeling the behavioral properties of an application. Characterized by a set of activities and a model of their execution.
Application-Structure	Sub-ontology modeling the structural properties of an application. It specifies the interface between the application and the middleware.
Application-Domain	Sub-ontology modeling the properties of the domain to which the application belongs.
Activity	A function, a service, an entity or a work allowing to achieve such a spot.
Atomic-activity	An activity which we cannot divide it into sub-activities.
Composite-activity	An activity which we can extract from other activities
Input	An argument or a data which must be affected to an activity or program.
Output	Represents the result of the execution of an activity.
Precondition	Represents the conditions necessary to execute an activity.
Effect	Represents effects produced after the execution of an activity. An effect can prevent the execution of other activities.
IDL-structure	Affirms that the interfaces of an application are specified by IDL language, i.e. the application is connected to middleware CORBA.
WSDL-Structure	Affirms that the interfaces of an application are specified by WSDL language, i.e. application represents a Web Service.
Functional-description	This term allows to determine in which domain the activity or the application belongs.
Non-functional-description	This term allows to determine in which domain the parameter belongs.
(…)	(…)
Provides	Affirms that any application must have a certain behavior.
Has-domain	Affirms that any application must belong to a certain domain
Has-structure	Affirms that any application must be connected to a middleware.
Has-input	Indicates that an activity must have input data.
Has-output	Indicates that an activity must present results.
Description-type	Indicates that the domain can have functional description and non functional description. Exp. Travel-service is a functional description Person is non functional description
Maps-Method	Affirms that the method name in interface of a middleware must correspond to an activity.
Refers-to 1	Indicates that the functional description of term refers to an activity. Exp. Name_Instance1 is an instance of Hotel-reserv. Name_Activity1 is an instance of Activity. Name_instance1 refers to Name_activity1. Thus Name_activity1 is an activity of Hotel reservation. (Hotel-reserv sub-class of Functional-description)
Refers-to 2	Indicates that the non functional description of term refers to a parameter. Exp. Name_Instance1 is an instance of Hotel. Name_Parameter1 is an instance of Parameter. Name_instance1 refers to Name_Parameter1. Thus Name_Parameter is a parameter of Hotel. (Hotel sub-class of non-functional-description)
(…)	(…)
Name	Indicates the application name of an activity…
Type	Indicates the type of the parameter.
Code	Represents the product code.
(…)	(…)

The hierarchy of concepts classification shows the organization of the ontology concepts in a hierarchical order which expresses the relations sub-class – super-class.

We use the relation "Sub-Class-Of» between the classes to define their classification. C1 class is sub-class of C2 class if any instance of C1 class is an instance of C2 class. We follow a development process from top to bottom. We start with a definition of the general concepts of domain and then continue by the specialization of concepts. For example, we can start by creating classes for the general concepts: Application, Application-behavior, Application-structure, Application-domain, Application-Model, Domain-Description, Application-Model, Parameter, Computed-Parameter, Application-Structure, Application-Structure, Output-Structure, Date, Interface-Structure, Product, Method-Structure, Transportation, Input-Structure, Level and Creator.

2) Build the binary-relations and concepts classification diagram: In this phase, we will build our diagram in two principal steps; initially, we determine the organization of concepts, then we will connect the concepts by relations so necessary.

We represent the binary relations between classes by a diagram. In this diagram the classes are represented by rectangles and the relations by arrows (domain towards Co-domain) labeled by the name of the relation. We enrich this diagram by adding dotted arrows (sub-class towards class) to illustrate the organization between concepts

Figure 3 represents binary-relations and concepts classification diagram.

We are always in the conceptualization phase, for each tree of concepts classification obtained in the previous step; we build the following intermediate representations: concepts Dictionary, relations-tables, attributes-tables, logical-axioms table, instances-tables.

3) Build the concepts Dictionary: In this step, we will accord a semi-formal description of concepts which were presented in the classes hierarchy, this process corresponds to the creation of concepts dictionary accorded to Methontology. In this dictionary, we define for each concept: instances, attributes, relations which the source is this concept, synonyms and acronyms of this concept;

The table 2 represents a concepts dictionary for the domain « Application ».

4) Build the relations-tables: The binary relations are represented in the form of properties which attach a concept to another. For each relation whose source is in the tree of concepts classification, we define: its name, the name of the source concept, the name of the target concept, cardinality and the name of the inverse relation; For example, the table 3 represents a relations table for the domain « Application ».

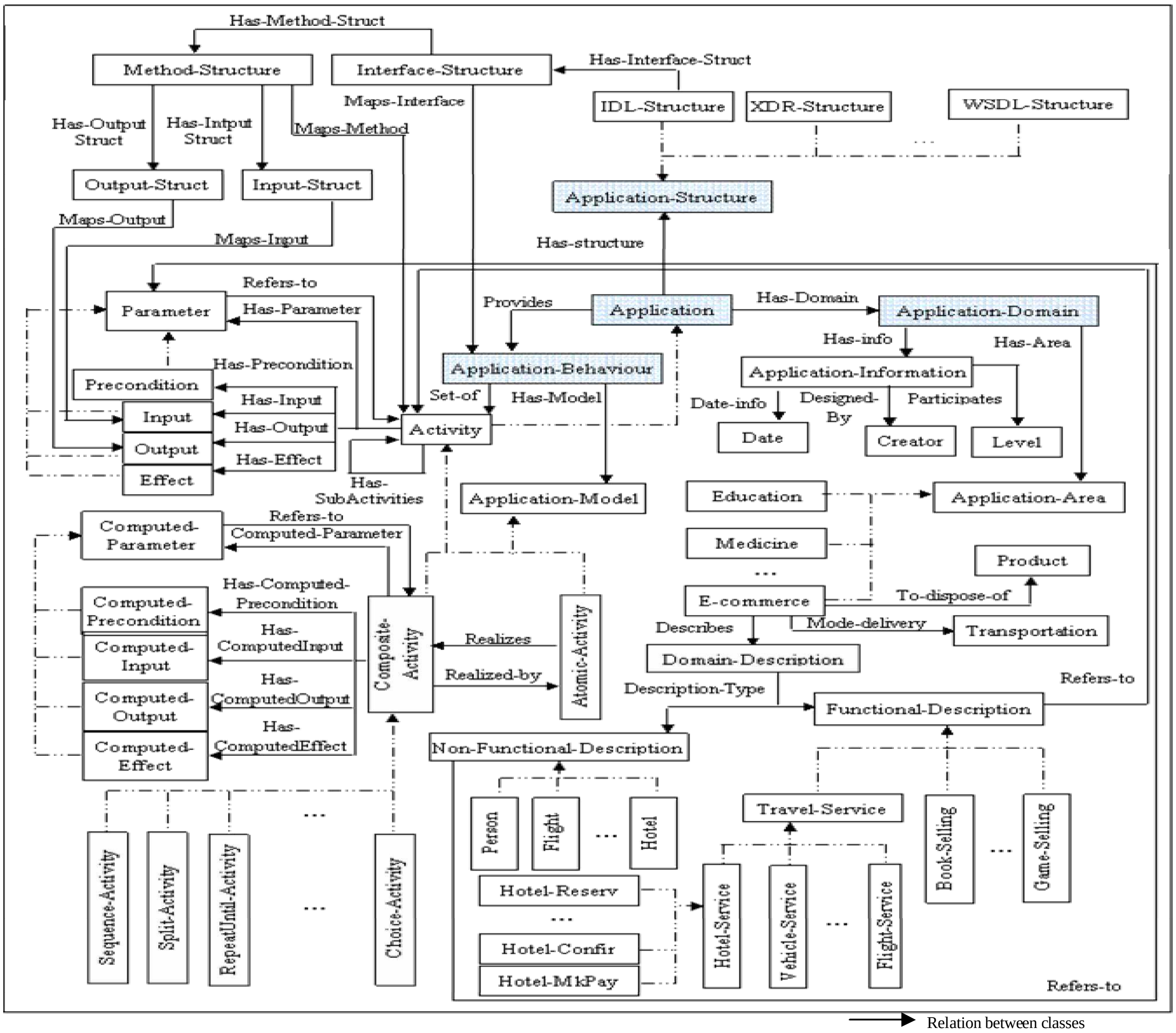

Fig. 3: binary-relations and concepts classification diagram

TABLE II
CONCEPTS DICTIONARY FOR THE DOMAIN « APPLICATION »

Concept name	Synonyms	Acro-nyms	Ins-tances	Attributes	relations
Application	Program ; Software ; …	App	-	Name Text-description	Provides Has-structure Has-domain
Activity	Function; Service; Act; …	-	-	-	Has-parameter Has-Precondition Has-Input Has-Output Has-Effect Has-subactivities
Atomic-activity	Atomic-function;…	-	-	-	Realizes
Composite -activity	Composite-function; Composite-service; ; …	-	-	-	Has-Computed-parameter Has-Computed-Precondition Has-Computed-Input Has-Computed-Output Has-Computed-effect Realized-by
Split-activity	Split-function; Split-service ...	-	-	-	-
Choice-activity	Choice-function;.	-	-	-	-
Repeat-until-activity	Repeat-until-function; ..	-	-	-	-

TABLE III
RELATIONS TABLE FOR THE DOMAIN « APPLICATION »

Relation-name	Source concept	Target concept	Cardina-lity	Inverse relation
Provides	Application	Application-behavior	(1,1)	Provided-by
Has-structure	Application	Application-structure	(1,1)	-
Has-domain	Application	Application-domain	(1,1)	-
Has-parameter	Activity	Parameter	(0,n)	-
Has-Precondition	Activity	Precondition	(0,n)	-
Has-Input	Activity	Input	(0,n)	-
Has-Output	Activity	Output	(0,n)	-
Has-effect	Activity	Effect	(0,n)	-
Has-subactivities	Activity	Activity	(0,n)	Has-subactivities
Realizes	Atomic-activity	Composite-activity	(1,n)	Realized-by
Realized-by	Composite-activity	Atomic-activity	(1,n)	Realizes
Has-computed-precondition	Composite-activity	Computed-precondition	(0,n)	-
Has-Computed -input	Composite-activity	Computed-input	(0,n)	-
Has-Computed -output	Composite-activity	Computed-output	(1,n)	-
Has-Computed -effect	Composite-activity	Computed-effect	(1,n)	-
Has-computed -parameter	Composite-activity	Computed-parameter	(0,n)	-

5) Build the attributes-table: The attributes are properties which take its values in the predefined types (String, Integer, Boolean, Date…); for example the concept Parameter has attributes: Name, Text-description, and the type of the parameter.

For each attribute appearing in the concepts dictionary, we specify: its name, type and interval of its possible values and its cardinality; for example, the table 4 represents an attributes table for the domain « Parameter ».

6) Build the logical-axioms table: In this step, we will define the ontology concepts by using the logical expressions which are always true. In the table below, we define for each axiom, its description in natural language, the name of the concept to which the axiom refers, attributes used in the axiom and the logical expression; we specify some axioms as it is represented in table 5.

7) Build the instances-table: In this section, we will present a description of some ontology instances, for that, we will specify the instances' names and values of the attributes for each one of them; the table 6 illustrates some instances for each class.

C. Formalization

In this phase, we use the DL (Description Logic) [3] formalism to formalize the conceptual model that we obtained it at the conceptualization phase.

DL forms a language family of knowledge representation; it allows to represent knowledge relating to a specific area using "descriptions" which can be concepts, relations and instances. The relation of subsumption allows to organize concepts and relations in hierarchy; classification and instantiation are the basic operations of the reasoning on description logic, or terminological reasoning. Classification permits to determine the position of a concept and a relation in their respective hierarchies.

Description logic consists of two parts: terminological language TBOX in which we define concepts and relations; and an assertionnel language ABOX in which we introduce the instances.

1) TBOX construction: We define here concepts and relations relating to our domain, by using the constructors provided by description logic to give structured descriptions at concepts and relations; for example an activity must have a name and only one, a parameter in input, a parameter at output and produce an effect at the end of its execution.

We can describe this sentence in description logic by:

$$\text{Activity} = (\exists\ \text{Name.String}) \cap (=1\ \text{Name.String}) \cap (\geq 0\ \text{Has-input.Input}) \cap (\geq 0\ \text{Has-output.Output}) \cap (\geq 0\ \text{Has-precondition.Precondition}) \cap (\geq 0\ \text{Has-effect.Effect}).$$

Moreover, we specify subsumption relations which exist between various concepts; for example to specify that the Activity class is subsumed by the Application class we write: Activity $\subseteq$ Application

Table 7 represents the definitions of some concepts

However, we define relations by giving the couples of concepts source and concepts target of each one, and/or by specifying its inverse relation; for example the Has-parameter relation which connects an activity with its parameters is specified by:

Has-parameter: (Activity, Parameter)

Has-parameter: Refers-to-

Table 8 represents the definitions of different relations of our ontology.

TABLE IV
ATTRIBUTES TABLE FOR THE DOMAIN « PARAMETER »

Attribute name	Type values	values Arrange	Cardinality
Name	String	-	(1,1)
Text-description	String	-	(1,n)
Type	Thing	-	(1,1)

TABLE V
LOGICAL-AXIOMS TABLE

Concept name	Axiom description	Attributes used	Logical expression
Application	An application must have a domain	Has-domain	∀ X Application (X) ∃ Y Application-domain(Y) Has-domain (X,Y).
Activity	An activity must produce at the end of its execution a results	Has-Output	∀ X Activity(X) ∃ Y Has-output (Y) Has-output(X,Y).
Application-information	An application can interact at the interior or at the exterior of the company.	Participates	∀ X Application-information(X) ∃ Y participates (Y) participates(X,Y).
Parameter	A parameter refers to an activity	Refers-to	∀ X Parameter (X) ∃ Y Activity (Y) Refers-to(X,Y).
Composite-activity	A composite activity can be a sequence of activities, Split activities or Choice activities…	-	∀ X Composite-activity (X) ⇒ (Sequence-activity (X) ∨ Split-activity(X) ∨ Choice-activity (X) ∨ Repeat-until-activity(X))
IDL-structure	An application must have an interface with the middleware	Has-interface-structure	∀ X IDL-structure (X) ∃ Y Interface-structure (Y) / Has-interface-structure (X,Y).
Creator	An application is conceived or published by a company.	Design-by	∀ X Application-information (X) ∃ Y Creator (Y) / Design-by (X,Y).
Domain-description	A domain is described by functional and non-functional properties.		∀ X Domain-description (X) ⇒ (Functional-description (X) ∧ Non-functional-description (X))
Functional-description	A functional entity must refer to an activity	Refers-to	∀ X Functional-description (X) ∃ Y activity (Y) Refers-to (X,Y).
Non-functional-description	A non-functional entity must refer to a parameter.	Refers-to	∀ X Non-functional-description (X) ∃ Y parameter (Y) Refers-to (X,Y).
(…)	(…)	(…)	(…)

TABLE VI
INSTANCES TABLE

Concept	Instance	Property	Value
Application	FLIGHT_RESERVATION	Name	FLIGHT_RESERVATION
		Text-description	flight reservation
Activity	GET_FLIGHT_DETAILS	Name	GET_FLIGHT_DETAILS
		Text-description	Take all details of the flight.
Atomic-activity	CONFIRM_RESERVATION	Name	CONFIRM_RESERVATION
		Text-description	Send confirmation to customer.
Composite-activity	BOOK_FLIGHT	Name	BOOK_FLIGHT
		Text-description	BOOK_FLIGHT=SEQUENCE_ACTIVITY{ GET_FLIGHT_DETAILS, GET_CONTACT_DETAILS, RESERVE_FLIGHT, CONFIRM_RESERVATION }
Computed-Input	ACCOUNT_NAME	Name	ACCOUNT_NAME
		Text-description	Name of the account
		Type	String
Computed-Input	PASSWORD	Name	PASSWORD
		Text-description	Account password.
		Type	String
Computed-output	ACK	Name	ACK
		Text-description	Return “ACK” to user.
		Type	Boolean
Compute-Precondition	IS_MEMBER(ACCOUNT_NAME)	Name	IS_MEMBER(ACCOUNT_NAME)
		Text-description	Is account name valid?
		Type	Boolean
Computed-effect	LOGGED_IN(ACCOUNT_NAME, PASSWORD)	Name	LOGGED_IN(ACCOUNT_NAME, PASSWORD)
		Text-description	Open session of ACCOUNT_NAME with the provided password
		Type	Activity
Creator	ALGERIA AIRLINES	Name	ALGERIA AIRLINES
		Phone	00213 31 92 45 74 00213 31 92 46 90
		Fax	00213 31 95 55 84 00213 31 92 48 98
		e-mail	contact@algeria-airlines.dz infos@algeria-airlines.dz
		Site-web	www.algeria-air-lines.dz
		Physical-@	Mohammed Boudhiaf international Airport, Ain-Bey Constantine (Algeria)
(…)	(…)	(…)	(…)

2) ABOX construction: The assertionnel language is dedicated to the description of facts, by specifying the instances (with their classes) and the relations between them in the following way:

A: C to indicate that A is an instance of class C;

For example: FLIGHT_RESERVATION: Application.

(A1, A2): R to indicate that the two instances A1 and A2 are connected by the relation R;

For example: (ALGERIA_AIRLINES, FLIGHT_RESERVATION_INFOS): Creator

In tables 9 and 10 we define some assertions.

TABLE VII
CONCEPTS DEFINITION IN TBOX

Concept	Definition	subsomption relation
Application	-(∃ Name.String) ∩ (=1 provides.application-behavior) ∩ (=1 Has-structure.Application-structure) ∩ (=1 Has-domain-Application-domain)	Application ⊆ Thing.
Activity	= (∃ Name.String) ∩ (=1 Name.String) ∩ (≥1 Has-input.Input) ∩ (≥1 Has-output.Output) ∩ (≥0 Has-precondition.Precondition) ∩ (≥1 Has-effect.Effect).	Activity ⊆ Application
Atomic-activity	Activity ∩ Application-model ∩ (≥1 Realizes-Composite-activity)	Atomic-activity ⊆ Activity Atomic-activity ⊆ Application-model
Sequence-activity	Composite-activity	Sequence-activity ⊆ Composite-activity
Application-domain	-(∃ Name.String) ∩ (=1 Name.String) ∩ (=1 Has-info.Application-information) ∩ (=1 Has-area.Application-area)	Application-domain ⊆ Thing
Application-information	Application-domain ∩ (=1 Participates.Level) ∩ (=1 designed-by.Level) ∩ (=1 Date-info.Date)	Application-information ⊆ Application-domain
Application-area	Education ∪ Medicine ∪ … ∪ E-commerce	Application-area ⊆ Application-domain
Parameter	-(∃ Name.String) ∩ (∃ Type.Thing) ∩ (=1 Name.String) ∩ (=1 Type.Thing) ∩ (≥1 Refers-to.Activity)	Parameter ⊆ Thing
Computed-parameter	-(∃ Name.String) ∩ (∃ Type.Thing) ∩ (=1 Name.String) ∩ (=1 Type.Thing) ∩ (≥1 Refers-to.Composite-activity)	Computed-parameter ⊆ Thing
Application-structure	-(∃ Name.String) ∩(=1 Name.String) ∩ (∀ Structure-type{IDL-Structure, XDR-Structure, … WSDL-Structure})	Application-structure ⊆ Thing
IDL-Structure	Application-structure ∩ (≥1 Has-interface-struct.Interface-structure)	IDL-Structure ⊆ Application-structure
Application-behavior	-(∃ Name.String) ∩ (=1 Name.String) ∩ (=1 Hs-model.Application-model) ∩ (≥1 Set-of.Activity)	Application-behavior⊆ Thing
Interface-structure	(=1 Name.String) ∩ (≥1 Has-method-structure.Method-structure) ∩ (=1Maps-interface.Application-behavior)	Interface-structure ⊆ Thing
Method-structure	(=1 Name.String) ∩ (≥1 Has-input-struct.Input-structure) ∩ (≥1 Has-output-struct.Output-structure) ∩ (=1Maps-method.Activity)	Method- structure ⊆ Thing
Input-structure	(=1 Name.String) ∩ (=1Maps-input.Input)	Input- structure ⊆ Thing
Level	(∀ Name.{ Collaborative-level, Applicative-level })	Level ⊆ Thing
Product	(=1 Name.String) ∩ (=1 Code.String) ∩ (=1 Classification.String)	Product ⊆ Thing
Creator	-(=1 Name.String) ∩ (≥1 Phone.String) ∩ (≥1 Fax.String) ∩ (=1 e-mail.String) ∩ (=1 Site-web.String) ∩ (=1 Phisycal@.String).	Creator ⊆ Thing
(…)	(…)	(…)

TABLE VIII
RELATIONS DEFINITION IN TBOX

Relation	Couple (domain, co-domain)	Inverse-relation
Provides	(Application, Application-behavior)	Provided-by
Has-structure	(Application, Application-structure)	-
Has-domain	(Application, Application-domain)	-
Has-parameter	(Activity, Parameter)	Refers-to
Has-Precondition	(Activity, Precondition)	-
Has-Input	(Activity, Input)	-
Has-Output	(Activity, Output)	-
Has-subactivities	(Activity, Activity)	-
Realizes	(Atomic-activity, Composite-activity)	Realized-by
Realized-by	(Composite-activity, Atomic-activity)	Realizes
Has-Computed-input	(Composite-activity, Computed-input)	-
Has-Computed-output	(Composite-activity, Computed-output)	-
Has-Computed-effect	(Composite-activity, Computed-effect)	-
Has-info	(Application-domain, Application-information)	-
Has-area	(Application-domain, Application-area)	-
Participates	(Application-information, Level)	-
Designed-by	(Application-information, Creator)	-
Date-info	(Application-information, Date)	-
Describes	(E-commerce, Domain-description)	-
To-dispose-of	(E-commerce, Product)	-
Mode-delivery	(E-commerce, Transportation)	-
Refers-to	(Parameter, Activity)	Has-parameter
Refers-to	(Computed-parameter, Composite-activity)	Has-Computed-parameter
Computed-parameter	(Composite-activity, Computed-parameter)	Refers-to
Has-interface-struct	(IDL-structure, Interface-structure)	-
Maps-interface	(Interface-structure, Application-behavior)	-
Has-method-struct	(Interface-structure, Method-structure)	-
Maps-method	(Method-structure, activity)	-
Has-input-struct	(Method-structure, Input-struct)	-
Has-output-struct	(Method-structure, Output-struct)	-
Maps-input	(Input-struct, Input)	-
Maps-output	(Output-struct, Output)	-
Design-by	(Application-information, Creator)	-
Provided-by	(Application-behavior, Application)	Provides

TABLE IX
ASSERTIONNEL DESCRIPTION OF CONCEPTS

Concept	Description
Application	FLIGHT_RESERVATION : Application HOTEL_RESERVATION : Application VEHICLE_RESERVATION : Application
Activity	GET_FLIGHT_DETAILS: Activity RESERVE_FLIGHT: Activity; …
Atomic-activity	CONFIRM_RESERVATION: Atomic-activity; …
Composite-activity	BOOK_FLIGHT: Composite-activity
Computed-Input	ACCOUNT_NAME: Computed-Input
Computed-Input	PASSWORD: Computed-Input
Computed-output	ACK: Computed-output
Computed-Precondition	IS_MEMBER(ACCOUNT_NAME): Computed-Precondition
Computed-effect	LOGGED_IN(ACCOUNT_NAME, PASSWORD): Computed-effect
Creator	ALGERIA_AIRLINES : Creator
(…)	(…)

TABLE X
ASSERTIONNEL DESCRIPTION OF RELATIONS

Relation	Description
Creator	(ALGERIA_AIRLINES, FLIGHT_RESERVATION_INFOS) :Creator ; …
Computed-precondition	(BOOK_FLIGHT, IS_MEMBER(ACCOUNT_NAME) : Computed-precondition ; …
Computed-Input	(BOOK_FLIGHT, ACCOUNT_NAME): Computed-Input; …
Computed-output	(BOOK_FLIGHT, ACK): Computed-output; …
Computed-effect	(BOOK_FLIGHT, LOGGED_IN(ACCOUNT_NAME, PASSWORD): Computed-effect; …
(…)	(…)

D. Implementation

The implementation deals with building a computable model. The effort is concentrated on the suitability of the OWL DL [11], which is equivalent to the SHOQ (D) [9]. For checking, we need to use the inference services provided by many systems such as RACER [8] and DLP [11]. These systems have shown to work well with large ontologies. The use of the RACER system can make possible to read OWL file and to convert it in the form of a DL knowledge bases. It can also provide inference services. We use that to manipulate the application ontology and PROTEGE-2000 [9] which offers a convivial graphical user interface. Additionally, PROTEGE-2000 provides facilities to impose constraints to concepts and relations.

To evaluate correctness and completeness of application ontology, we use query and visualization provided by PROTEGE-2000. We use the built-in query engine for simple query searches and query plug-in to create more sophisticated searches. We also use visualization plug-ins to browse the application ontology and ensure its consistency.

Discuss

In the literature, many approaches have proposed to integrate the applications enterprise. Wasserman [23] has classified the integration approaches in four classes which are principally: applications integration using data, treatments (function), presentations (interface) and processes.

In our work, we developed an application ontology in order to integrate the enterprise applications. The construction of this ontology must be based on a model capturing structural and behavioral properties of an application. In addition, the properties on the domain to which the application belongs. The behavioral properties of an application were modeled by sub-ontology *Application-behavior* , which maintains the two integration approaches : by treatments and processes, the concepts of ontology *Application-bahavior* do not have any positive impact on integration without recourse to ontology *Application-domain* and Application-structure which also make it possible to define a concepts set in order to enrich or to increase the integration capacity and this one by providing a properties modeling of the application as well as properties on the application interfaces, i.e., the concepts set which facilitate to define how the application is connected to the middleware, for example: ‘IDL-structure’, ‘WSDL-structure’, etc.…

In conclusion, the ontology is often seen as new solution providing exchange facilities of semantically enriched information, which can resolve the heterogeneity problem and ensure greater interoperability between the integrated applications.

V. Conclusion

The problem dealing with heterogeneity, even semantic has been deeply investigated in the field of ontology.

We proposed an ontological approach for building application ontology. This approach enhances application integration at both applicative and collaborative levels. The important benefit of our work is that the communicator can reuse the mapping information for managing interaction between applications.

Future work will focus on the development of global ontology by integrating the application ontologies for managing the enterprise information heterogeneity based on the semantic bridges concept [5]. A semantic bridge encapsulates all required information to translate instances of the source entity to instances of the target entity. So, the integrated applications can successfully and efficiently communicate and exchange information as well as services through the mapper component.

References

[1] D. Oberle, M. Sabou, D. Richards, “ An ontology for semantic middleware : extending DAML-S beyond Web services” International Conference on Ontologies, Databases and Applications of SEmantics (ODBASE), Catania, Sicily (Italy), Workshops, 3-7 November 2003.

[2] H. Panetto, M. Scannapieco, M. Zelm, “ INTEROP NoE : Interoperability research for networked enterprise application and software”, OTM Workshop, Lecture Notes in Computer Science, vol. 3292, Springer-Verlag, Heidelberg R. Meersman et al. edition, Berlin, pp. 866-882, 2004.

[3] F. Baader, P. Patel-Schneider, D. Calvanese, L. D. McGuinness, D. Nardi, editors. “The Description Logic Handbook”, Cambridge University Press, 2003.

[4] J. Chauvet, The book, Services Web avec SOAP, WSDL, UDDI, ebXML. Eyrolles edition, 2002.

[5] E. Gahleitner, W. Wob, “Enabling Distribution and Reuse of Ontology Mapping Information for Semantically Enriched Communication Services”, In: 15th International Workshop on Database and Expert Systems Applications. IEEE Computer Science, Zaragoza, Spain, 2004.

[6] H. Chalupsky, “OntoMorph: A Translation System for Symbolic Knowledge”, In the Seventh International Conference of Principles of Knowledge Representation and Reasoning, San Francisco, USA, 2000.

[7] D. Dou, D. Mcdermott, P Qi, “Ontology Translation by Ontology Merging and Automated Reasoning”, In EKAW, Workshop on Ontologies for Multi-Agent Systems (OMAS), Spain, 2002.

[8] V. Haarslev, R. Möller, “RACER System Description”, In IJCAR’01, 2001.

[9] Protégé OWL, version 3.1.1, 2005. http://protege.stanford.edu.

[10] R. Driouche, Z. Boufaida, F. Kordon, “An Ontology Based Architecture for Integrating Enterprise Applications”, Modelling, Simulation, Verification and Validation of Enterprise Information Systems’06 Workshop, Cyprus, Paphos, INSTICC Press, pp. 26-37, 2006.

[11] I. Horrocks, P. F. Patel-Schneider, F. V. Harmelen, “From SHIQ and RDF to OWL: The making of a web ontology language”, In Journal of Web Semantic, Vol. 1, N° 1, pp. 7-26, 2003.

[12] D. S. Linthicum. “Enterprise Application Integration”. Edition Addison-Wesley, Boston et al. 2003.

[13] R. T. Gruber, “A Translation Approach to Portable Ontology Specification”, Knowledge Acquisition, Vol 5, N° 2, pp 199-220, 1993.

[14] S. Izza, L. Vincent, P. Burlat, “Unified Framework for Application Integration”. In 7th International Conference on Enterprise Information Systems’05, USA, 2005.

[15] M. F. Lopez, A.G. Perez, “Overview and Analysis of Methodologies for Building Ontologies”, In Knowledge Engineering Review, 17(2), 2002.

[16] M. Uschold, M. Grüninger, “Ontologies Principles Methods and Applications”, In Knowledge Engineering Review, 11(2), 1996.

[17] M. Grüninger, M. S. Fox, “Methodology for the Design and Evaluation of Ontologies”. In IJCAI’95, Workshop on Basic Ontological Issues in Knowledge Sharing, Montreal, 1995.

[18] M. Fernandez, A. Gomez-Perez, N. Juristo, “Methontology from Ontological Art Toward Ontological Engineering”, In AAAI’97, Spring Symposium Series on Ontological Engineering, USA, 1997.

[19] M. Hemam, Z. Boufaida “An Ontology Development Process for the Semantic Web”, EKAW’04, 14th International Conference on Knowledge Engineering and Knowledge Management Whittlebury Hall, Northamptonshire, UK, 5-8 October 2004

[20] A. Kotok, D. R. R. Webber, “ebXML: The New Global Standard for Doing Business over the Internet”. New Riders, Indianapolis, 2002

[21] K. Sycara, M. Stollberg, S. Galizia “Semantic Web Service Tutorial” Kauai 2006.

[22] R. Driouche, Z.Boufaida, F. Kordon, 2006 “Towards integrating collaborative business process based on process ontology and ebXML collaboration scenario” in 6th International Workshop on Web Based Collaboration’06, IEEE Computer Society Krakow, Poland 4-6 September.

[23] Wasserman A I., “Tool Integration in Software Engineering Environments”. In software Engineering Environments; Workshop on Environments. Berlin 1990.