CHAPTER 2
THEORETICAL BACKGROUND

This chapter introduces an overview of the theoretical background technology necessary for the proposed reference architecture and the metadata dictionary model. These related technologies encompass agent technology, ontology-based approach, XML technology, and information integration architectures. With the benefits of JDBC to provide all connection to various kinds of data sources. The JDBC will be used as the data source connectivity loaded by the resource agent to relief the client site.

2.1 Agent Technology

With the emergence of the promising agent technology, the notion of agent (Caglayan and Harrison, 1997; Knoblock and Ambite, 1997; Lange and Oshima, 1998, Li, Zhang and Swan 2000; Papastavrou, Samaras and Pitoura 2000) is well received in many subject areas such as Software Engineering and Artificial Intelligence. Agents can be found in computer operating systems, networks, databases, and so on. In contrast to software objects of object-oriented programming, agents are active entities that simply acts continuously in pursuit of its own goals. A property shared by all agents for all systems is that they live in some environment. They have the ability to interact with their execution environment, and to act asynchronously and autonomously upon it.

2.1.1 Agent Definitions

It has been heavily debated for several years now that what actually constitutes an agent, and how it differs from a normal program. The definition of agent can be classified in two broad perspectives as follows:

(1) End-User Perspective

An agent is a program that assists people and acts on their behalf. Agents function by allowing people to delegate work to them.

(2) System Perspective

An agent is a software object that is situated within an execution environment and possesses the following mandatory properties:

- **Delegation:** The agent performs a set of tasks on behalf of a user (or other agents) that are explicitly approved by the user (Caglayan and Harrison, 1997);
- **Communicative:** The agent needs to be able to interact with the user to receive task delegation instructions, to inform task status and completion, or to communicate with other agents (Caglayan and Harrison, 1997; Lange and Oshima, 1998);
- **Autonomous:** The agent has control over its own actions without direct intervention to the extent of the user's specified delegation;
- **Reactive:** The agent can sense changes in the environment and acts accordingly to those changes; and
- **Monitoring:** The agent needs to monitor its environment in order to perform tasks autonomously.

An agent may also possess any of the following orthogonal properties:

- **Mobility:** The agent can travel from one host to another;
- **Intelligence:** The agent needs to interpret the monitored events to make appropriate actuation decisions for autonomous operation or adapt in accordance with previous experience; and
- **Security:** The agent must appear believable to the end-user.

2.1.2 Agent Model

The agent model consists of two skill components, that is, task level skills, and communication skills. Both of these skills access to the knowledge.

(1) Task Level Skills

The task level skills comprise of the skills that the agent possesses to accomplish its goal. These tasks specify the functionality of an agent and require that an agent be able to

perceive its environment through sensors, and act upon the environment to perform these tasks.

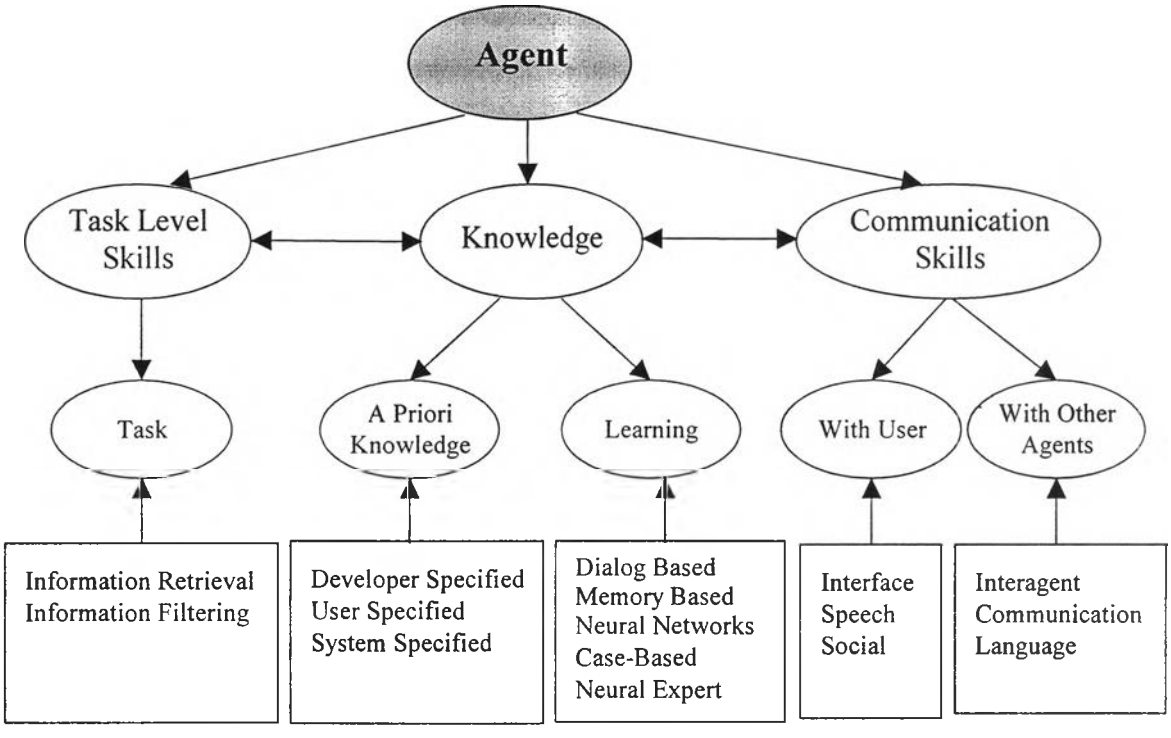


Figure 2.1 Agent model (Caglayan and Harrison, 1997).

(2) Knowledge

The knowledge portion of an agent concerns the rules the agent follows to complete its task. The Agent relies on knowledge about the environment that has been built in by the designer. Some agents acquire their knowledge through learning.

Knowledge can be acquired by employing any of the following techniques:

- Developer specified,
- User specified,
- Derived from other knowledge sources, such as common agent languages that enable knowledge acquisition among agent communities, and
- Learned by the system, such as agents that derive knowledge from the user and environment.

(3) Communication Skills

The communication skills of the agent affect how it interacts with the user. In terms of interagent communication, a communication method is necessary for multiple agents to work together. Such communication typically involves a communication language and, in some cases, an intermediary application which serves as a request router. For example, Telescript (White, 1995), SmallTalkAgents (Gale, and Island, 1993), Agent Tcl (Tool Comand Language) (Gray, 1995), Knowledge Interface Format (KIF) (Genesereth and Fikes, 1994), and Agent Communication Language (ACL) (Genesereth, 1995) are languages which enable inter-agent communication.

2.1.3 Type of Agents

Agents can be categorized into various types from several orthogonal perspectives. From responsibility perspective, agents can be categorized into *Interface Agents*, *Intelligent Agents*, *Collaborative Agents* or *Multi-agents*, and *Information Agents*. From mobility perspective, agents can be divided into two different groups, namely, *stationary agents* or *static agents* and *mobile agents*. The stationary agent is bound to the system where it begins execution. If the required information is not on the system, or the agent itself needs to interact with an agent on a different system, the agent typically utilizes available communication mechanisms such as remote procedure calling (RPC) to accomplish the task. Mobile agent, on the other hand, has embedded code (behavior), execution state, and itinerary that allow agents to freely roam from one system to the next. Upon reaching the destination, the mobile agent autonomously performs its delegated tasks without any intervention from the originating host.

2.1.4 Mobile Agents

A Mobile Agent (Lange and Oshima, 1998; Papastavrou, Samaras and Pitoura, 2000) is not bound to the system where it begins execution. It has the unique ability to travel from one system to another in a network. This transporting ability allows the mobile agent to move to a system that contains an object it wants to interact, thus taking the advantages of being on the same host or network as the object. Mobile Agents roaming the Internet could search for information on goods and services, and interact with other agents that also roaming the networks (and meet at a pre-established rendezvous) or remain bound to a particular

machine. Having been created in one execution environment, the agent can transport its *state* and *code* to another execution environment in the network for greater operational flexibility and resource utilization. Such transportability is attributable to the states whose values determine what to do when execution resumes at destination. The code, on the other hand, holds the context necessary for execution.

A mobile agent model is not complete without defining a set of events that are of interest to the agent during its lifetime. For example, the event of arriving at a new location is a momentous one in the life of an agent and generally entails the invocation of one or more of its behaviors. The set of events varies from model to model. The following list outlines some of the most common events:

- **Creation:** This is analogous to the constructor of an object. A handler for this event should initialize states and prepare the agent for further instructions.
- **Disposal:** This is analogous to the destructor of an object. A handler for this event should free whatever resources the agent is using and prepare the agent for burial.
- **Dispatch:** Signals the agent to prepare for departing to a new location. This event can be generated explicitly by the agent itself upon requesting to migrate, or can be triggered by another agent that has explicitly requested this agent to move.
- **Arrival:** Signals the agent that it has successfully arrived at a new location and that it should commence performing its duties.
- **Communication:** Notifies the agent to handle incoming messages from other agents and is the primary means of inter-agent correspondence.

2.1.5 The Benefits of Mobile Agent

The above versatility makes mobile agent an ideal transport vehicle in distributed systems for several reasons:

- *Reduce bandwidth requirement*: Query transfer and transaction processing between client and server often occupy high bandwidth and time-consuming. Using mobile agent technology, an information package can be piggybagged with the agent and traverses to the destination where interrogating process is carried out locally. The

obtained results can also be filtered or preprocessed locally before being sent back to the request site. Considerable bandwidths and transmission overheads are therefore reduced;

- *Reduce unreliable network connection problems*: The fact that mobile agents can operate autonomously and asynchronously enables many task invocations to be scheduled and dispatched across network in the absence of full connectivity. In the wake of unreliable network connection, the mobile agents can still continue their work and return the desired results when the connection is reestablished;
- *Reduce distributed information management problems*: In manipulating the information that is distributed across the heterogeneous platforms, information integration is a major hurdle to overcome. As mobile agents provide the ability for resource discovery independent of the host system, they can be utilized as an assistant to automatically gather and consolidate dispersed information with minimal human intervention and supervision; and
- *Eliminate the overhead of traditional information sources connectivity*: Currently, the proposed methodologies (Greem, 1997) for handling, retrieving, and hooking up HIS to the Web overload the client and offer limited flexibility and scalability. Many commercial products employ Open DataBase Connectivity (ODBC) and Java DataBase Connectivity (JDBC) drivers to accomplish such tasks. This requires that the client must first set up the ODBC/JDBC driver to establish the connection, whereby causing a significant time delay in database access. This scheme overloads client's responsibility and performance. The mobile agents, on the contrary, connect between the client program and the server machine, to provide built-in connectivity similar to the above ODBC/JDBC interface, thus eliminating the burden of setting up client side driver interface classes. Much of the overheads incurred from connecting with traditional ODBC/JDBC interface are thus eliminated.

The capabilities of mobile agent technology provide flexible means for transport mechanism in the heterogeneity environments of distributed systems.

2.2 Ontology-based Approach

In the last decade, ontology is a popular research topic in various communities such as knowledge engineering, natural language processing, cooperative information systems, intelligent information integration, and knowledge management (Fensel, 2001; Studer, Benjamins and Fensel, 1998). Ontology refers to the shared understanding of some domains of interest which may be used as a unifying framework to reduce or eliminate conceptual and terminological confusion (Uschold and Gruninger, 1996). An ontology also provides an explicit specification of a conceptualization that describes semantics of data. The conceptualization composes of objects, concepts, and other entities that are assumed to exist in some areas of interest, and the relationships that hold among them. Such a conceptualization may be implicit or explicit. For the information systems, or for the Internet, ontologies can be used to organize keywords and database concepts by capturing semantic relationships among keywords or among tables and fields in a database. The semantic relationships give users an abstract view of an information space for their domain of interest.

2.2.1 The Components of Ontologies

An ontology describes the subject matter using the notions of concepts, relations, functions, axioms, and instances (Benjamins and Gomez-Perez, 2000) as follows:

- **Concepts** can be anything about which something is said and could be the description of a task, function, action, strategy, reasoning process, etc.
- **Relations** represent a type of interaction between concepts of the domain. They are formally defined as any subset of a product of *n* sets, that is, R:C1 x C2 x ...x Cn
- **Functions** are special case of relations. For example, function Price-of-a-used-car calculates the price of a second-hand car depending on the car-model, manufacturing date, and mileage.
- **Axioms** do constrain the possible interpretations for the defined terms and are used to model sentence that is assumed to be true without proof.
- **Instances** are all of the terms in an ontology that have an associated definition.

2.2.2 Type of Ontologies

Depending on their generality level, different types of ontologies may be identified that fulfill different roles in the process of building a knowledge-based system. (Fensel, 2001; Guarino, 1998; Heijst, Schreiber and Wielinga, 1997). Ontologies can be distinguished broadly into different types as follows:

(1) Domain ontologies

Domain ontologies capture the knowledge valid for a particular type of domain, e.g., electronic, medical, mechanic, and digital domain, etc.

(2) Metadata ontologies

Metadata ontologies like Dublin Core (Weibel, Godby, Miller and Danierl, 1995) provide vocabularies for describing the content of on-line information sources.

(3) Representational ontologies

Representational ontologies do not commit themselves to any particular domain. Such ontologies provide representational entities without stating what should be represented. A well-known representational ontology is the *Frame Ontology* (Gruber, 1993), which defines concepts such as frames, slots, and slot constraints allowing the expression of knowledge in an object-oriented or frame-based manner.

(4) Generic or common sense ontologies

Generic ontologies aim at capturing general knowledge about the world, providing basic notions and concepts for things such as time, space, state, event, etc. (Fridman-Noy and Hafner, 1997). As a consequence, they are valid across several domains. For example, an ontology about mereology (part-of relations) is applicable in many technical domains (Borst and Akkermans, 1997).

2.2.3 Ontology Representation

An explicit ontology may take a variety of forms, but necessarily it will include a vocabulary of terms and some specification of their meaning, i.e., definitions. These vocabulary of terms and meaning so specified vary considerably, depending on the degree of formality as follows:

- **Highly informal**: expressed ontology loosely by natural language;
- **Semi-informal**: expressed by natural language, but increase clarity to reduce ambiguity, e.g., the text version of the "Enterprise Ontology" (Fraser, Tate and Uschold, 1995);
- **Semi-formal**: expressed by artificial formally defined languages, e.g. the Ontolingua (Farquhar, Fikes and Rice, 1997), and Frame Logic (Kifer, Lausen and Wu, 1995) as representatives for frame-based approaches. Both incorporate frame-based modeling primitives in a first-order logical framework; and
- **Rigorously formal**: meticulously defined terms with formal semantics, theorems, and proofs of such properties as soundness and completeness, e.g., the CycL and KIF (Genesereth and Fikes, 1994) as representatives for enriched first-order predicate logic and Description Logics (Borgida, 1995) that describe knowledge in terms of concepts and role restrictions that used to automatically derive classification taxonomies.

The best method for building an ontology will depend on the degree of formality required, which in turn depends a great deal on the intended purpose of the ontology.

2.2.4 Ontology Architectures

Ontology contributes to task integration that describes semantics of the information sources and makes the content explicit. Current ontology research can be classified into three approaches as follows:

(1) Single Ontology approach

This approach uses a global ontology to provide a shared vocabulary for the specification of the semantics as depicted in Figure 2.2 (a). All information sources are tied to this global ontology which can be a combination of several specialized ontologies. This single ontology approach can be applied to integration problems where all information sources to be integrated provide nearly the same view on a domain. This approach is used in SIMS (Arens, Chee, Hsu and Knoblock, 1993; Arens, Hsu, and Knoblock, 1996) model.

The drawback is that this approach is susceptible from changes in the information sources which can affect the conceptualization of the constituent domains represented in the ontology.

(2) Multiple Ontologies approach

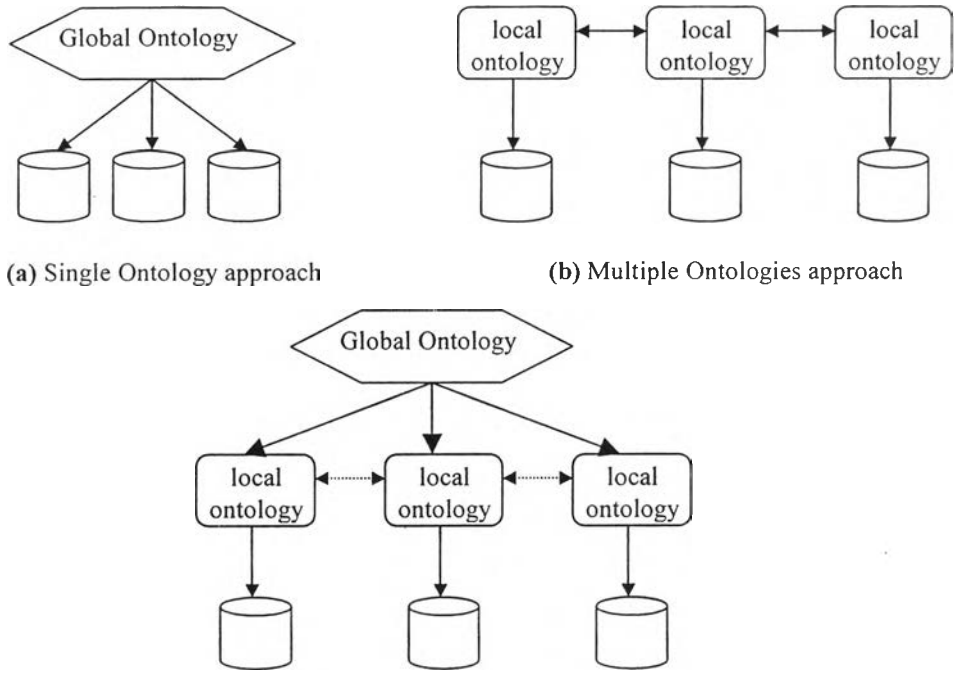
In this approach (see Figure 2.2 (b)), each information source is described by its own ontology. The semantics of an information source is described by a separated ontology localizing to that source. The information source ontology may be a combination of several ontologies, but it cannot be assumed to be a shared vocabulary source of these ontologies. This approach is used in OBSERVER (Mena, Kashyap, Sheth, and Illarramendi, 1996).

The advantage of this approach is that each source ontology can be developed independent of other source ontologies. This approach simplifies the integration task and supports incremental change, that is, adding and removing from sources. The drawback of this approach is that it is difficult to compare different source ontologies due to the lack of a common vocabulary.

(3) Hybrid approach

This approach (see Figure 2.2 (c)) attempts to overcome the drawbacks of the aforementioned approaches. The hybrid approach mixes the advantages of single ontology and multiple ontologies approaches by allowing each source to describe its own ontology. The local ontologies are then combined to form a globally shared vocabulary. This approach is used in BUSTER Stuckenschmidt, Wache, Vogele and Visser, 2000).

The advantage of this approach is that new sources can easily be added without the need of modification. The use of a shared vocabulary makes local source ontologies comparable, thus avoiding disadvantages of multiple ontology approach. However, the drawback of this hybrid approach is that existing ontologies cannot easily be reused and must be individually redeveloped from scratch.



(a) Single Ontology approach

(b) Multiple Ontologies approach

(c) Hybrid Ontologies approach

Figure 2.2 The ontology architectures (Wache, Vögele, Visser, Stuckenschmidt, Schuster, Neumann and Hübner, 2001).

2.2.5 Applications of Ontologies

Part of the research on ontologies is concerned with envisioning and building enabling technology for the large-scale reuse of ontologies at a world-wide level. In order to enable as much reuse as possible, ontologies should be small modules with high internal coherence and limited amount of interactions between the modules (Fensel, 2001). This requirement and others are expressed in the design principles of ontologies (Gruber, 1995; Guarino, 1995; Uschold and Gruninger, 1996).

The uses of ontologies can be identified in three main categories as follows:

(1) Communication

Ontologies enable shared understanding and communication between people with different needs and viewpoints arising from their particular contexts.

(2) Interoperability

Many applications of ontologies address the issue of interoperability among systems in which different users may need to exchange their data. There are need to translate between modeling methods, paradigms, languages and software tools. The ontology is used as an interchange format.

(3) Systems Engineering

The applications of ontologies have focussed on the role that ontologies play in the operation of software systems. To be effective, ontologies must support the following:

- **Re-usability:** The ontology is the basis for a formal encoding of the important entities, attributes, processes, and their inter-relationships in the domain of interest. This formal representation may be a reusable and/or shared component in a software system.
- **Knowledge Acquisition:** Using ontology as the starting point would increase speed and reliability in knowledge acquisition.
- **Reliability:** A formal representation also makes possible the automation of consistency checking of results in more reliable software.
- **Specification:** The ontology can assist the process of identifying requirements and defining specifications for an IT system.

2.3 XML Technology

XML (eXtensible Markup Language) is an open, text-based markup language that provides structural and semantic information to data. XML is a standard language for data communication over Internet, since it can be used to structure and describe data that can be understood by different applications. XML has been emerged to overcome the weaknesses of HTML by separating the data representation and organization from displaying. The data is represented with meaningful structure and semantics that are easy for human and computers to understand and manipulate it. An example of HTML and XML documents is illustrated in Figure 2.3. Both documents express the same result on web browser. XML covers what the data means whereas HTML only tells how the data should look. With the

structure of XML, searching the information is much more accurate and useful than that of HTML document. More importantly, XML enables business to business communication. Its portability eliminates the needs for user to write code to exchange data with the other's system or different applications.

The success of XML is primarily based on its flexibility, that is, anyone can create XML-DTD (Document Type Definition) to define the structure of XML document that conforms to the set of rules in DTD. This set of rules can be reused in a new document, where additional DTDs can be created on top of the existing ones. Two of XML's most valuable features are its ability to provide structure to document and to make data self-describing. A valid XML document always obeys all validity constraints identified in the XML specification and DTD. Any violations with the rules would be reported to the XML application by the XML processor. Nonetheless, validation might not always be necessary since XML supports well-formed document. The XML specification addresses the concept of a well-formed document that is not validated. This means that some rules must be obeyed, but not nearly as strict as those constraints required for validity.

HTML document	**XML document**
Apple 48119 Apple Computer</br> 1111112</br>	<product> <name>Apple</name> <model>48119</model> <dealer>AppleComputer</dealer> <telephone>1111112</telephone> </product>

Figure 2.3 Examples of HTML and XML documents.

The salient characteristics of XML render itself attractive to be incorporated as an assistant mechanism for information integration and representation of the metadata dictionary contents as follows:

- *XML for information integration*: Such an ease of implementation and manipulation permits XML to be used as a means for HIS transaction interoperability, including data integration, exchange, and transformation. Consequently, data in different formats can be retrieved and combined into one multi-purpose document by means of XML supports. In addition, implementers

are free to write applications that transform data incompatibility from HIS into a unified format for subsequent processing.

- *XML for metadata representation*: Metadata approach enables collections of data about data and documents dispersed over several hosts or networks to co-exist in the same environment. In general, metadata is designed to support many functional requirements, such as library catalog, application integration, and information resource discovery. Owing to XML's strength in well-formedness, validity, and schema representation, it is more flexible to implement the metadata in XML than existing HTML technology. Some examples of ontology-based knowledge representation in XML metadata languages that use XML as a means for building and exchanging ontologies on the WWW are XOL (Ontology Exchange Language) (OEL-url), OML/CKML (Ontology Markup Language/Conceptual Knowledge Markup Language) (OML+-url), and OIL (Fensel, Harmelen, Horrocks, McGuinness and Patel-Schneider, 2001) (Ontology Interchange Language) which is based on RDF (Lassila and Swick, 1999) and RDF Schema (Brickley and Guha, 2000) .

Hence, the XML technology will be considered, in many respects, as an effective multiple information sources access and interchange vehicle for web-based enterprises.

2.4 Information Integration Architectures

The integration of heterogeneous information sources over Internet is an active research topic and is one of the challenging tasks to overcome. The task of an information integration system is to answer enquiries that may require extracting and combining data from multiple data sources. The traditional web architecture that supports database access and information integration is a three-tier client/server/application architecture (see Figure 2.4) that uses the standard communication protocol HTTP (HyperText Transfer Protocol) implemented on top of TCP/IP. With this architecture, an application server is responsible to run application programs. The web client browser only provides graphical interface and can download a small application program called applets from the application server to carry out some simple tasks such as validating data entry, formatting, and display. The web client communicates with the web server via HTTP. Server site technology focuses on

query optimization, query evaluation, and storage-related issues (clustering, indexes, etc.). The connection between a client and a server is usually tight, and the two modules are provided by the same vendor.

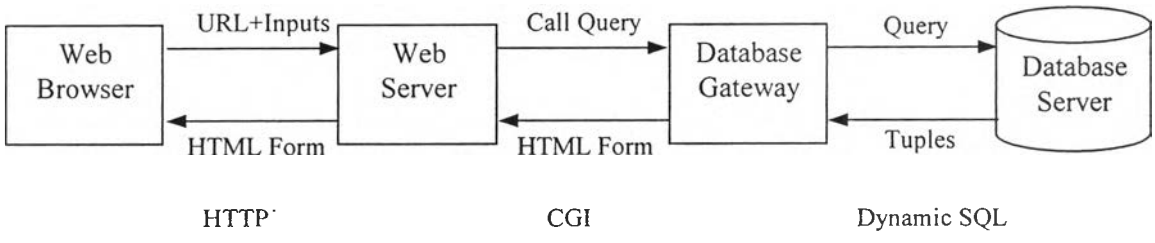


Figure 2.4 Database access from a Web browser.

To access databases, generally, the structure of databases is unfriendly to the Web. The users have to know the database schema and go through a rigid protocol or DBMS. Several methodologies have been proposed to hook databases and the Web. One popular technique is CGI (Common Gateway Interface) applications that provide mechanisms to link the databases and the Web in a simple and straightforward way. However, CGI still has some drawbacks, for example, in a large-scale service environment, where handling a large number of simultaneous requests to connect database systems often leads to high communication costs between the database gateways and the web server. It is sometimes required to recompile the application before database processing.

Moreover, the need to access database over the Internet has given rise to new techniques and architectures. Two new architectures have been proposed, that is, data warehouse and mediator.

2.4.1 Data Warehouse Architecture

A data warehouse acts as a centralized repository of an organization's data, ultimately providing a comprehensive and homogenized view of the organization. The architecture of a data warehouse is illustrated in Figure 2.5. A new layer called the *data warehouse* has been augmented between a client and data sources (or servers). It can be a relational database, multidimensional database, flat file, hierarchical database, object database, etc. This architecture is used to load data from several data sources into a warehouse which serves as a data integration center. All queries can thus be applied to the warehoused data. The advantages of the data warehouse are that adequate performance can be guaranteed at

query time (Florescu, Levy and Mendelzon, 1998). However, this approach requires updating the warehouse when data changes which, in turn, calls for increasing efficient data loading and updating maintenance. Thus, data warehouse is suitable for applications where accessing to physical data sources is slow and the data sources are rarely changed.

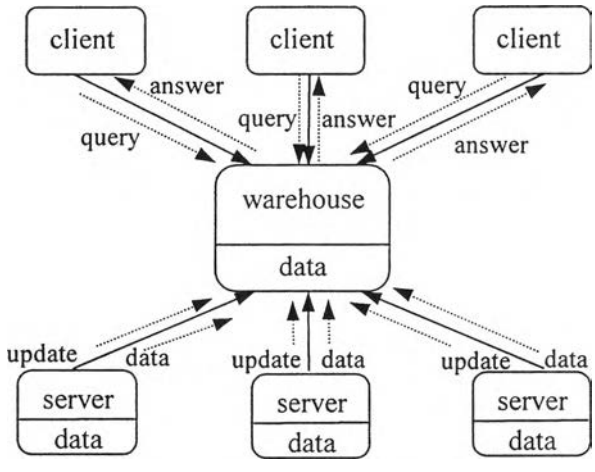


Figure 2.5 A data warehouse architecture.

2.4.2 Mediator-Wrapper Architecture

A mediator-wrapper architecture is a virtual approach that mostly is used in data integration where data freshness is critical. The data remain in the data sources and no data is actually stored at all in the middle tier such that the data freshness is guaranteed at query time. This architecture is suitable for integrating data from several sources where it is impossible or difficult to load the entire data from the sources. A wrapper, which is a program translating data in the data source to a form that can be further processed by the data integration system (or the mediator), connects each data source to the mediator. Such a set up permits exporting information about the data source schema, data, and query processing capabilities (Cluet, Delobel, Simeon and Smaga, 1997) to a mediator. The mediator in turn centralizes the information of all available data in a unified view. When a user submits a query, the mediator has to process a query at run time. First, the mediator has to decide which sources associate to the user's query. Second, the mediator has to perform query transformation or query rewriting into sub-queries that appropriate to the target data source(s). Third, the mediator needs to generate a query plan to optimize the query access time. Finally, the returned answers from executing each sub-query are integrated at the mediator into the final

result before forwarding to the user. Hence, this virtual approach is more appropriate for building a system where the number of sources is large, the data are frequently updated, and the data sources are autonomous. A mediator-wrapper architecture is shown in Figure 2.6.

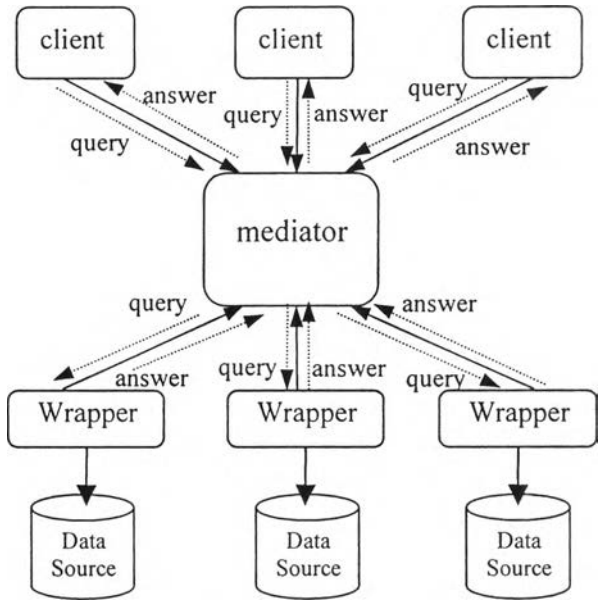


Figure 2.6 A mediator-wrapper architecture.

The mediator-wrapper architecture requires creating of integrated view to allow for distributed query. The information from the HIS can be conceptually integrated to form a single *global conceptual schema* or *integrated schema.*

To create the global conceptual schema, the integration process can occur in two steps, namely, *schema translation* and *schema integration.*

(1) *Schema Translation* involves with translating the local schema of each information source into a common intermediate canonical representation. This requires the specification of a target data model for a global conceptual schema definition; and

(2) *Schema Integration* follows the translation process by integrating the intermediate schemas into the global conceptual schema. Schema integration involves two tasks, that is, homogenization and integration.

- *Homogenization* solves the structural and semantic heterogeneity of the HIS.

- *Integration* merges all intermediate schemas into a single database schema and then restructured to create a global conceptual schema.

There are many issues to be considered for information integration, that is,

(1) *The number of data sources*: If the number of data sources is high, creating a schema integration view and solving conflicts are very difficult;

(2) *The domain of data sources*: The domain of data sources is very dynamic, adding or dropping a data source should have minimal impact on the integrated schema; and

(3) *The structure of data sources*: The integrated schema should support data sources that may be structured, semi-structured, to unstructured.

A number of systems has been emerged based on the mediator-wrapper architecture that can be broadly classified into two groups: static integration systems, and dynamic integration systems.

Static Integration Systems

When a new component database is incorporated into the system, heterogeneity problems arise as to where and how this foreign component can be seamlessly integrated to the existing HIS. In a static integration (or tightly-coupled) system, these problems can be solved in lieu of the users' perspectives. There are three categories of users in the system, namely, *domain integrator*, *mediator author*, and *end-user*. When a new component database is added, the domain integrator and mediator author are responsible for carrying out the schema integration process, in order to establish an integrated schema. The integrated schema hides the semantic and structural differences of the underlying component databases from end-users and thus solves the heterogeneity problems. This process is illustrated in Figure 2.7.

In so doing, the domain integrators utilize system-supplied *translation tools* to implement a *wrapper* for schema translation. The resulting intermediate schemas generated from wrappers are called *component schemas*. The mediator authors then use system-supplied *integration tools* to implement a *mediator* that combines multiple component schemas into an integrated schema, in which inconsistencies are resolved and duplicates are removed. In some systems, all component schemas are integrated into a global schema,

while in others, multiple integrated schemas exist for different classes of end-users. When a user poses a query to access the sources, the mediator has to decide which sources contribute to the given query. Then, the mediator has to process the query transformation appropriate to each individual source.

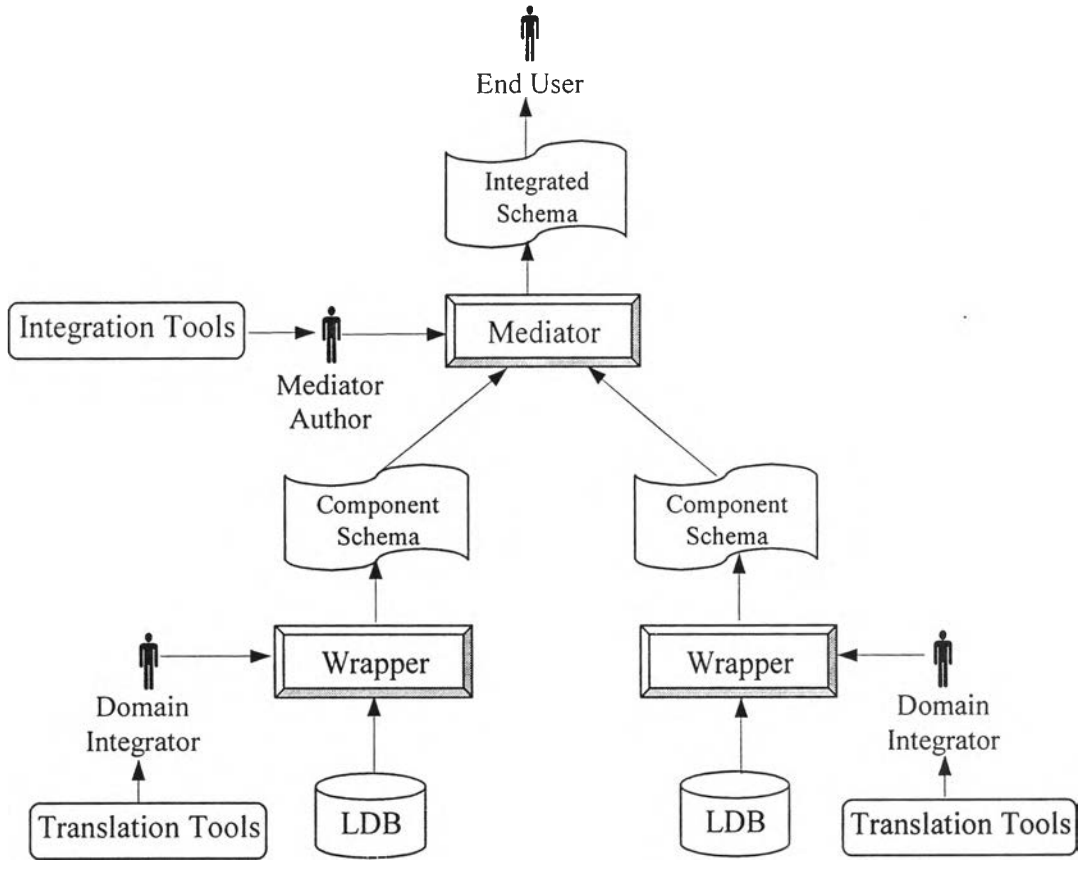


Figure 2.7 Static integration architecture (Jakobovits, 1997).

Dynamic Integration Systems

In dynamic integration systems (or loosely-coupled systems), shown in Figure 2.8, the end-users are not provided with a predefined view. Instead, they are granted direct access to the component schemas at query time by means of a multidatabase query language or a graphical user interface. A metadata resource dictionary (Cheung and Cheng, 1996) contains descriptions of the component schemas (including conversion information). Using the metadata, the end-user constructs an integrated schema on the fly, and poses queries directly against the custom-built view.

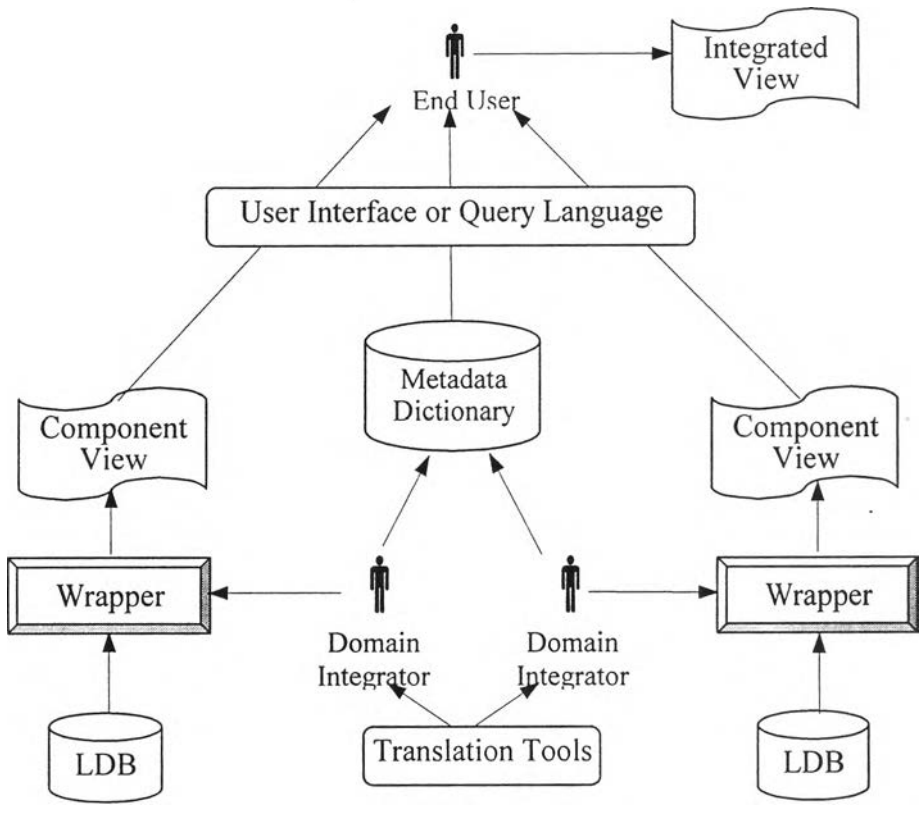


Figure 2.8 Dynamic integration architecture (Jakobovits, 1997).

Because integrated schemas can be promptly created and dropped on the fly, dynamic integration systems allow the end-user more flexibility and are much easier to maintain in the presence of evolving component schemas. However, they require a more sophisticated end-user who understands semantics of the component schemas and is responsible for carrying the integration. However, the development of new integration techniques presented in this work entails static systems to become more dynamic in nature.

Both information integration architectures will be applied in the design of the HIS reference architecture.

2.5 The Information Source Connectivity

In order to establish the connection with the information source, the interface wrapper must provide the middleware mechanisms, such as ODBC/JDBC, HTTPD, C++ Interface, etc. Some important characteristics of Java (Cornel and Horstman, 1996) such as portability, robustness, and security control system contribute Java to be the leading integration tool in client/server programming and mobile computing (Pitoura and Samaras, 1997). Currently, the Java Database Connectivity (JDBC) (Hamilton, Cattell and Fisher, 1997; Jepson, 1997) is the Java standard specification for accessing and manipulating databases through the XML data source (Sun-url). The JDBC consists of two layers, namely, the JDBC API which provides a Java application interface to access and manipulate data sources, and the JDBC driver API which executes/implements this interface. A client that employs the JDBC API must first download a JDBC driver to its environment before accessing the data source. JDBC offers the ability to connect to almost all data sources on numerous platforms (Hamilton, Cattell and Fisher, 1997). With only one application, it is not necessary to write difference applications to access a variety of different data sources, e.g., Informix, Oracle, Sybase, XML, and so on. Moreover, hooking a database system and the Web can be accomplished via these methodologies as illustrated in Figure 2.9.

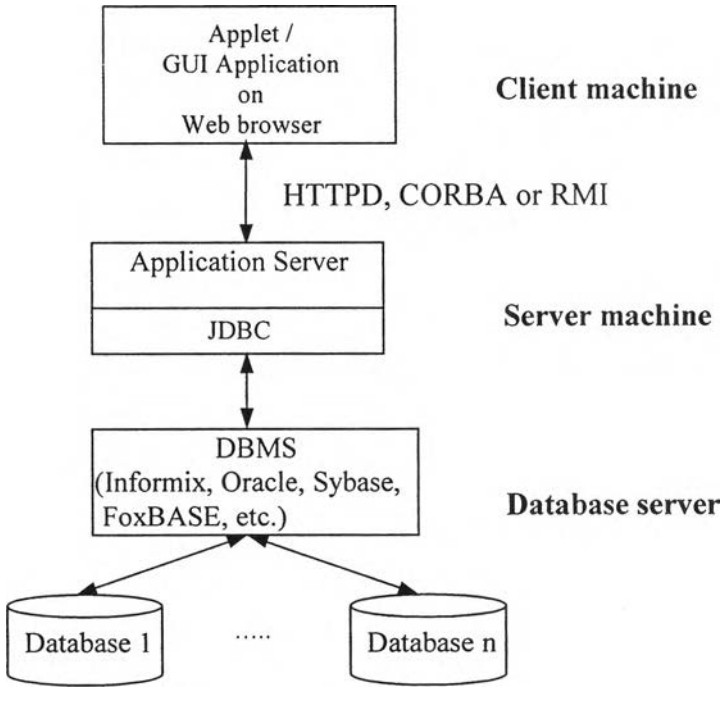


Figure 2.9 Hooking database systems and the Web (Hamilton, Cattell and Fisher, 1997).

Initially, the users send request to the application server via Applet or GUI Application on the web browser; the request then translated to an SQL command at the application server. The JDBC establishes a connection with the database server before sending the SQL command to process at the database server. The database server in turn collects results and sends them back to the application server, which then returns them back to the users.

One drawback of using JDBC is that the client, which employs these drivers, must first download and initiate the JDBC driver on the client machine before accessing a particular database. This will overload the client and offer limited flexibility and scalability. Moreover, JDBC usually imposes a significant time delay in database access.