Abstract

In this paper we propose an architecture for integrating heterogeneous information systems based on an agent messaging language called Knowledge Query Manipulation Language (KQML). We consider integration issues between relational database systems, object database systems, computer aided design systems, and compound document systems. Our solution consists of an integration architecture that consists of mediators (gateways) and facilitators (bridges). In addition, we propose extensions to KQML.

Introduction

In the market place, standards based on two different types of approaches exist for heterogeneous systems integration. They are:

- Message based integration
- Data flow based integration

Message based integration involves a client using the services of a server through messages sent from the client to the server. These messages are equivalent to program calls. The client issues synchronous or asynchronous messages to the server. The server responds to these messages by performing the requested service and optionally returning a value. Several of the compound document architectures proposed in the market place including Object Linking and Embedding (OLE)®[1] and OpenDoc®[2] use this style of integration. Additionally, distributed architectures based on CORBA®[3] use this style of integration between heterogeneous systems.

Another example of the same type of integration is the proposed Inter-framework Facility (OMG)[4] of Object Management Group (OMG). It uses messages with semantic content to support inter-operation of Electronic Computer Aided Design (ECAD) frameworks.

The data flow integration involves querying of a server by a client to extract a stream of information. The Tabular Data Stream (TDS)®[5] protocol of Microsoft Corporation and associated APIs (e.g. Open Database Connectivity - ODBC) to SQL database is an example of this kind of integration. In response to a query ,several responses may be returned (sometimes asynchronously). Therefore, the data stream protocol needs to support facilities to assemble multiple responses into one unit.

However, in real life, heterogeneous system integration needs to support both types of approaches. While the message based approach may be useful for integrating word processors with spell checkers, it is not general enough to address an application development where some of the servers are database servers returning a stream of responses. On the other hand, data streaming approaches are not designed to support the type of interaction expected between a word processor and a spell checker. Therefore, a hybrid approach that takes advantage of a data streaming protocol and a messaging protocol is needed.

However, the hybrid approach needs to deal with a number of problems when both of these approaches are mixed. These problems are similar to the problems faced by relays in inter-networking. They are:

- **Addressing:** Each of the market standards use different addressing scheme on their "subnets". The HTTP URL protocol uses a "/" delimited string with specific conventions for the protocol used and address of object to be accessed. OpenDoc and OLE use symbolic object names as addresses. These names do not include information on the format of the document. That

information is contained in the storage module of the compound document manager. Additionally, addresses may need to include information system "subnet" information such that optimal routing of packets can be achieved.

- **Data mapping:** Relational databases usually return their information in a tabular form. This is why Microsoft's TDS is sufficient as a data stream protocol for relational databases. However, CAD systems that support PDES/STEP SDAI return information as objects. The model of these objects is complex since PDES/STEP EXPRESS - the data modeling language of STEP, supports complex inheritance and associations.

  Similarly, a relational database can return a large number of small records in response to a query while a Engineering Data Management (EDM) system may return the TIFF image of a part (a large blob).

  In order to address both of these problems, the data stream protocol needs to be able to support both types of data transmission. In addition, the data stream protocol needs to support primitives that can be used to fragment and assemble different data formats.

- **Operation mapping:** Two different types of operation mapping need to be supported. The first kind is between similar systems. For example, accessing of a hypertext link in OpenDoc can be translated to a similar operation in HTTP. The second kind is between dissimilar systems. For example, mapping a SQL query to a hypertext access is a harder proposition. In order to support seamless mapping, the hybrid approach needs to support mapping between events and programs and vise versa. For example, accessing a hyper text link may generate an event within which is stored a script (a program) that gets executed on an SQL database. In the other direction, executing a script may generate a number of events which get mapped to following the links of a hyper text web.

- **Data encoding:** In order for heterogeneous systems to communicate with each other, they need a way to exchange meta data about the data they are exchanging. However, meta data comes in different forms. The Apple Event System used as the messaging interface of OpenDoc stores meta data as objects within the events. In addition the type of event (e.g. move, copy) is also used by programs to understand the action requested. This meta data is used by programs to understand the context within which the event happened. On the other hand, SQL 92 supports ways of exchanging schema information as part of the query language. The hybrid approach needs to support exchange of both types of meta data.

- **Packet transmission rate:** A far more important problem is that the speed of transmission of packets in the different database subnets (e.g. TDS subnet, HTTP subnet) may not be the same. Therefore, the proposed approach needs to be able to buffer data such that it can be moved from one "subnet" to another.

- **Plug and Play:** Additionally, when multiple subnets are connected together, users want a way for them to be connected without any effort. In addition, they want these heterogeneous information systems to automatically re-configure based on demand[6]. The users do not want to update different address tables, down load information, etc. We call this plug and play between various heterogeneous

information system networks.
This need to establish, on demand, virtual connections between systems that are most capable of dealing with the service being requested needs to be addressed by the hybrid approach. The hybrid approach needs to *facilitate* establishment of such virtual networks. It also needs to support exchange of capability information between clients, servers, gateways, and bridges such that these virtual connections can be established intelligently.

- **Presentation:** End user agents and tools access the heterogeneous integration framework in different ways. Some tools may require a file system like interface, while others may be capable of dealing with event like interfaces. Additionally, users may want to use tools such as Query By Example to directly extract information or to spawn of agents. The hybrid approach should provide the appropriate support for each of these types of access of its facilities.

In order to address these requirements, several key technologies need to be developed. First, a layered architecture needs to be developed. The layered architecture not only supports flexibility and extensibility, but also provides a reference model for building bridges and gateways. Second, a protocol needs to be developed such that the rules of inter-operation are explicitly specified. We are using an agent messaging language called KQML as the protocol. Third, a set of tools (servers, gateways, and bridges) need to be developed to support such integration. Finally, the services provided by these tools need to be specified in an open and standard way.

In the following sections, we describe our layered architecture and the tools we are in the process of developing. KQML has been described in more detail in other places[7].

Layered Architecture

In order to support the requirements expressed above and in order to reduce the complexity of designing and maintaining software, we have designed a layered model. The following figure presents the layered model:

Interface layer
Domain layer
Transaction layer
Facilitation layer
Network layer
Message layer
Transport layer

The transport layer, message layer, network layer, and the facilitation layer mask from higher layers the vagaries of the underlying transport mechanism and the configuration of the virtual information circuit. The transaction layer, domain layer, and the interface layer jointly achieve the integration functionality.

In the following sections we describe each of these layers:

Transport layer

The transport layer is responsible for connecting to the underlying transport mechanism and passing the information according to the format expected by the underlying transport infrastructure. The underlying infrastructure could be based on:

- Basic internet facilities such as sockets, netinfo addresses, etc.,
- Higher level RPC facilities, associated security mechanisms, etc. (e.g. OSF's DCE)

- CORBA and CORBA advertisement repository
- COSS events
- HTTP messages,
- OLE or OpenDoc server requests or
- SMTP mail messages

In addition, the transport layer provides a uniform set of services to the message layer even though the underlying transport mechanisms may provide varying amounts of functionality.

Message layer

The message layer is responsible for creating the appropriate message or event or script to communicate with the transport layer. The transport layer provides the appropriate drivers for communicating with the underlying communication infrastructure, while the message layer creates the appropriate *packets.* The message layer takes KQML packets and passes them as semantic events (OpenDoc), OLE server calls, CORBA messages, a stream of text over a socket connection, etc.

Network layer

The network layer is responsible for establishing the network topology of the heterogeneous information systems. It supports ways for information systems to register their capability, declare the kinds of networking functions they can perform (e.g. forward), etc. The network layer stores these kinds of information such that the facilitation layer can take advantage of it.

Facilitation layer

The facilitation layer takes advantage of the information maintained by the networking layer to assemble virtual circuits. The facilitation layer is used to map data and operations, by converting events, messages, etc. to a set of KQML performatives. KQML performatives are used as the specification language for performing this conversion. In addition, it translates names to addresses that are used by the transport layer. Further, congestion problems due to the difference in packet rate transmission between the sub-nets is managed at this level. Finally, it has facilities to split and join information entities. It uses the information about the virtual circuit that has been created to properly split the information entities.

Transaction layer

The transaction layer controls the dialogue between various information entities. It manages transactions between various information entities in a heterogeneous information system network. Additionally, it prevents dead locks and supports rollback.

Domain layer

The domain layer has domain specific knowledge. It uses this knowledge to translate and/or compress domain entities to the underlying protocol used for communication.

Interface layer

The interface layer supports different API mechanisms. These APIs are used at the back end of systems. These include a file system like API, a socket like API, scripting API, and event based API. Using the file system like API, the applications can open a file, write and read from it. In addition, they can group files into a directory and so on.

Calls to this API will be converted to the appropriate KQML performative set. The socket like API provides an open socket to the client. The client can read and write to the socket. The scripting API supports a way for the client to send or receive a program. The event API supports push and pull style event generation and event handling registration.

In addition, it defines two different domain independent mechanisms for graphical interaction between a end user and the framework. The first is support for a navigational metaphor (much like HTTP) and the second is support for Query By Example (QBE) like interaction.

Protocol

In order to address these problems ,we are developing a protocol called Heterogeneous Information System Messaging Language (HISML - also called Agent Messaging Language - AML). HISML is based on KQML. We follow KQML's syntax specification and the concept of messages (performatives) delimited by ":" keywords. We also follow the notion of embedding different performatives in a list to support the layered hierarchy.

Currently, KQML provides functionalities in several of the layers presented above. The following figure shows the layers in which KQML provides functionalities:

Layer	KQML functionality
Transport	None
Message	KQML performative syntax
Network	Network performatives
Facilitation	Facilitation, evaluation, and query performatives
Transaction	None
Domain	administrative support for ontology
Interface	None

HISML extends the KQML in the following ways:

- **Addressing:** Currently, KQML supports words as addresses for sender and receiver. We extend this and allow lists. Lists allow us to map addresses that contain not only identification information, but also other information (e.g. HTTP URL - which contains information about protocols). In addition, a list based addressing scheme can be used by routers and bridges to add layer specific information to the packet.
- **Capability definition:** KQML supports performatives to *advertise* capability information. The capability that is advertised is the type of KQML performative that an agent is best suited to processing. We extend and standardize this list. Additionally, we introduce new performatives such as *brag* for selective broadcasting and *bid* to support negotiation.

In addition, we introduce new functionality in HISML with respect to the transaction layer, domain layer, and interface layer.

We are currently in the process of designing the transaction management functionalities.

The domain layer extension involves two parts. The KQML is extended with an object oriented, functional language. We call this language SLOTH. This language, in addition to KQML, (and our extensions to it) becomes the *wire protocol*. All information is transmitted using SLOTH and KQML. The second part involves supporting concepts and operation in the language such that domain specific information and operations can be translated into SLOTH from other descriptions. The concepts include classes, instances, event objects, rules, etc.

Further, for specific domains the language contains ontologies that reflect standard ways

of organizing the information. For example, the language represents PDES/STEP application protocol entities as classes in the language. OpenDoc events are also modeled as objects in the language. In order to support SQL databases records and tables are supported in the language.

Operations also are modeled in the language. For example, it supports classes which model operations found in PDES/STEP Standard Data Access Interface (SDAI) and SQL queries.

Scripts can be translated into sloth because it is a Turing complete programming language. Additionally, scripts expressed in declarative form can be translated into rules.

Additionally, we plan on developing a P-code interpreter for this language. Using this feature not only can we support multi-tasking (an essential feature for a KQML server), but also compress the volume of data shipped across the network. In addition, generating and transmitting the P-code is a simple form of encryption which adds to the security of transmission.

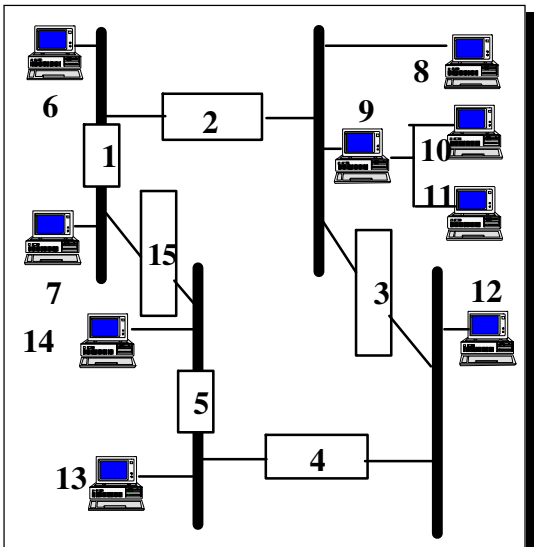
Finally, we are developing several extensions to support the interface layer. First, we are specifying an event based API based on the COSS standard of OMG. The COSS standard specifies the interface for event management systems using IDL. We are using the type less portion of the COSS event management standard to specify an API. Second, we are in the process of specifying a Query By Example (QBE) interface. This interface is specified in HTML and OMG's IDL. The HTML specification can be interpreted by Mosaic clients such that queries can be issued to HISML servers from Mosaic. Second, the IDL descriptions can be "compiled" into libraries that can be loaded into GUI builders. Finally, we are looking into building a file system like interface and a socket like interface to HISML servers.

Networking Architecture

We are in the process of developing servers, gateways (mediators), and bridges based on the protocol described above. Servers are used by human agents or autonomous programs to communicate with other systems on the heterogeneous information network. Mediators are used to map data or operations between dissimilar subnets. Bridges are used to map between similar subnets.

The similarity of the heterogeneous information sub-nets are characterized according to the mode of interaction they have (e.g. event based vs. data stream based) and according to the type of data transmitted (e.g. tables vs. objects).

The following diagram presents an illustrative example of the computing architecture:



Example Interoperability Computing Architecture

1. OLE server to OpenDoc server bridge
2. Gateway between event based network and data stream based network.

3. Gateway between tabular data stream based sub-net and object data stream based sub-net.
4. Bridge between object stream sub-net and PDES/STEP data stream sub-net
5. Gateway between PDES/STEP repository and OpenDoc server
6. OpenDoc server
7. OLE server
8. SQL database
9. SQL server
10. SQL database managed by SQL server
11. SQL database managed by SQL server
12. OQL server
13. PDES/STEP SDAI database
14. OpenDoc server
15. HISML server

For brevity's sake we present the key functionalities of the server (15), gateways (2, 3, 5) and bridges (1, 4). For a more detailed description of the others, please use the references provided.

The HISML server is used by agents that can generate scripts (programs). These could be humans using a GUI based tool or knowledge based systems.

Inter-networking

Inter-networking between heterogeneous information systems is achieved through the use of bridges and gateways..

	Subnet A		Bridge		Subnet B
Facilitate	Create KQML packet with performatives and routing information		Convert from (e.g. evaluate) to (e.g. stream all) Establish virtual circuit		Process KQML packet
Network	Add network address		Add and remove network address		Remove network address
Message	Create appropriate packet for the underlying transport mechanism		Convert transport layer packet to KQML packet		Convert transport layer packet to KQML packet
Transport	Load the appropriate drivers and send packet		Load the appropriate drivers, send and receive packet		Load the appropriate drivers and receive packet
TCP/IP Socket				CORBA	

The above figure shows the "protocol stack" of a simple bridge. The facilitation layer below Subnet A gets the information and creates KQML packets with the appropriate performatives. The selection of performative will depend on the characteristics of the sub-net from which the request is coming from and the characteristics of the route (virtual circuit) assembled by the facilitation layer. This packet is then passed on to the networking layer where actual symbolic addresses are added and other network performatives are also added.. This packet is then passed on to the message layer which creates the appropriate messages (e.g. OpenDoc semantic events or ODBC SQL calls, etc.) based on the underlying transport layer. The transport layer loads the appropriate drivers for the underlying transport mechanism and sends the packet.

This packet eventually reaches a bridge which passes the packet to the facilitation layer of the bridge. The facilitation layer converts the incoming performatives to different ones based on the sub-net to which this packet is being sent to. For example, an evaluate performative may be converted to a stream-all performative if the sub-net from which the packet arrived was based on message passing and the sub-net to which it is going to is a data stream based sub-net. Suitable routing and networking information is added by the facilitation and networking layers before being sent using the second sub-net infrastructure.

The OLE to OpenDoc bridge and the Object Data Stream to PDES/STEP data stream bridge map similar concepts from one to the other. For example, the OLE to OpenDoc gateway maps Apple Event data structures to data structures accepted by OLE servers. In addition, they map OpenDoc semantic events (essentially RPC calls) to messages sent to OLE servers.

The gateways not only do the above tasks, but also map one operation in one of the sub-nets to several operations in the other sub-net and vise versa. In addition, they split, buffer and assemble data from one sub-net to another.

Future Directions

We need to validate the proposed layered architecture by building various gateways, bridges and servers between heterogeneous systems. We have so far implemented a server. Additionally, we need to design and specify the transaction management functionalities. Finally, we need to build the transport layer connections. We have so far explored CORBA and TCP/IP sockets. In addition, we are also exploring connections to CAD tools based on PDES/STEP and relational databases using Microsoft's ODBC.

[1] OLE is a registered trademark of Microsoft

"Inside OLE 2", K. Brockschmidt, Microsoft Press, 1994.

[2] OpenDoc is a registered trademark of Apple and Component Integration Labs.

"OpenDoc - The Alpha release", Apple Corporation, 1994.

[3] CORBA is a registered trademark of Object Management Group.

"Common Object Request Broker Architecture", version 1.1, Object Management Group, 1994.

[4] "Inter-framework Facility", David Zenie, Draft Proposal, EDA Labs, IBM Microelectronics. 1994.

[5] TDS and ODBC are registered trademarks of Microsoft corporation.

"ODBC 2.0 Programmer's reference and SDK guide", Microsoft Press, 1994.

[6] "Distributed Collaborative Enterprises", White Paper, Bob Neches, ARPA program management.

[7] "Software agents", Michael R. Geneserth and Steven P. Ketchpel, ACM Communications, July 1994.

"Knowledge Query Manipulation Language - Draft specification", Tim Finin, et. al., 1993.