Ontology-driven enterprise application integration

G. Bucci, V. Sandrucci, E. Vicario
Dipartimento Sistemi e Informatica - Universitá di Firenze
{bucci, sandrucci, vicario}@dsi.unifi.it

Abstract—The hard problem of Enterprise Application Integration (EAI) is the semantic heterogeneity of data and applications. Today’s integration solutions focus mainly on the physical and syntactical aspect, providing little or no mechanism for semantic integration. No shared semantic concepts are explicitly used to define the semantics of different pieces of exchanged data, therefore the developer has to hard-code what to do with each data item from each application.

Ontologies are recognized as a most appropriate technology to overcome the limitations of current integration practices. Actually, they make possible the definition of a commonly agreed semantic model which can be reused and shared across applications.

We propose the use of an ontological model to capture both technical and semantic issues of a given integration project and to use that model to identify and validate integration components, as well as to validate the consistency of system configuration. Furthermore, since the ontological model can be accessed programmatically from conventional languages (e.g., Java), the contents of the messages exchanged among integration components are actually ontologies, guaranteeing the semantic consistency of exchanged data.

Keywords
Ontologies, knowledge base, semantic integration, enterprise integration patterns.

I. INTRODUCTION

The aim of Enterprise Application Integration (EAI) is the interconnection of multiple Enterprise Information Systems (EISs) to extend business processes throughout organizations.

Usually, EISs are made of heterogeneous, autonomous, and distributed systems [1], [2]. They are the result of a development occurred over time, in which individual application systems were developed independently, with little or no support for interoperability.

A consistent network of integration platforms, based on a variety of integration technologies (such as message brokers, Message Oriented Middlewares (MOMs), Web Services, etc.), are available of the shelf [3] [4], [3], [5], [6]. These solve the essential problem of information transport and point to point connectivity, but they do not address the problem of semantic heterogeneity. Therefore, the developer must know the meaning of the low-level data structures in order to implement semantically correct integration. As a result, mechanisms for interpreting and validating exchanged information are to be hard-coded [7], [8] and the correctness of the integration completely depends on the engineer’s knowledge [1].

The advent of semantic web technologies like ontology languages (such as RDF, RDF/S and OWL) enables formal description of the semantics of the data structures exchanged between enterprise application systems, thus opening the way to automated validation and reconciliation of exchanged data. As such, semantics-based technologies are envisaged as an essential part of integration solutions in the near future [7] [8]. Frameworks like Protégé [9] (a free ontology editor and knowledge-base framework) and Jena [10] [11] (a free semantic Web framework for Java) are already available to help the designer implement ontologies and use them via conventional programs.

The use of ontologies in EAI has been proposed by many authors. A largest share of the literature on the subject matter is essentially devoted to the semantic web, addressing issues such as service publication and discovering, and service composition. In that context, Service Oriented Architectures (SOA) and Web Services (WS) are the prevailing technologies.

An ontology-driven approach to integration based on ontology mediation is proposed in [2] and [12], where WS are chosen as the basic technology for integration. The resulting architecture is obtained augmenting SOA with a semantic layer that aims at enhancing service mediation in the context of EAI. Contrary to a static integration solution, where an integration scenario predefines every detail of any potentially connectable EIS, the approach supports also dynamic integration, in the sense that the binding services of target EISs can be performed at run time.

The potential use of ontologies and Semantic Web in systems and software engineering is the subject of a working draft of W3C [13].

The use of ontologies in a classical software engineering context is discussed in [14] and [15]. The goal of [14] was to show how ontologies can be defined that support the developer in creating new software or in running new components in a complex environment like an application server. Semantic metadata regarding software components and APIs are described through an ontology. This may be queried for finding APIs with certain capabilities (development time support) or for pre-loading components that are required by other components (run time support). The ontology is also used to perform validity checks aimed at avoiding inconsistent system configurations.

An examination of several ontology-based approaches for data integration is in [16]. A thorough review of the potential benefits that software engineering can achieve by applying ontologies in the various stages of the software development life-cycle is [17].

The problem of developing software architectures, based on formal domain models (ontologies), consisting of conventional components written in languages such as Java, is treated in [18]. The point is the construction of the ontological domain
model and the instantiation of the classes from the ontology into Java classes so as to be manipulated by conventional programs. This is made possible through the use of Protégé (see also [19] and [20]) and Jena library, the former as a rapid prototyping environment for building ontological domain models and to test how the system behaves in response to ontology changes, the latter as the API toward plain Java programs.

The transformation of an ontology in RDF Schema into a familiar, object-oriented Java API is described in [21]. The mapping of an OWL ontology into Java is also addressed in [22], where a set of Java interfaces and classes are created from an OWL ontology such that an instance of a Java class represents an instance of a single class of the ontology, with most of its properties, class relationships and restriction definitions maintained. A framework that translates ontology constructs into Enterprise Java Beans and connects them to relational database persistent storage through the generation of Hibernate object-relational mappings is presented in [23].

In this paper we address the definition of an ontology capable of relating technical integration aspects to application-domain concepts, so as to augment integration components with explicit semantics.

Semantic integration is obtained through the resulting ontological knowledge-base, which is used: (a) during analysis and design stages to search for components with certain capabilities and/or to validate configurations; and (b) to provide the semantic domain model from which the programmer instantiates Java objects that are therefore consistent with their intended meaning.

Physical integration is obtained through a middleware built on top of a communication bus (e.g. JMS, [24]). The middleware is made of number of Enterprise Integration Components (EICs) designed to implement the functionalities required by the given integration target. EICs are modeled after Enterprise Integration Patterns (EIPs) [25], that is patterns of solutions to recurring integration problems. As such, EICs are essentially based on asynchronous messaging and the resulting middleware leaves existing applications unchanged. Through ontologies and patterns, the exchange of consistent information between business functions occurs in a manner that appears to be seamless.

The proposed integration method has been applied in a pilot project carried out for the largest hospital of Florence, Italy.

The rest of the paper is organized in the following manner. Section II defines the structure of an ontology serving the purpose application integration. Section III discuss how EICs are represented in the ontology and proposes a solution to the target integration problem. Sections V and VI illustrate the construction of integration components. Conclusions are drawn in section VII.

II. DEFINING THE ONTOLOGY FOR INTEGRATION

Before introducing the structure of our ontology, let us define the simple integration problem that will lead us in the sequel. The problem is as follows.

There are several applicative programs (called App1, App2, App3, ...) installed at distinct locations (Loc1, Loc2, Loc3, ..., respectively), all of which have their own database storing patients’ medical data.

Assume App1 stores the following attributes for each patient: name, surname, date and place of birth, social security number, telephone number and home address, while App2 stores the same attributes as App1, with the exception of patient’s home address which is replaced by patient’s profession. Other applications, store the same sort of data with possible variations as above. Each application assigns a unique identifier to any patient in the local database, but there is no global identifier valid across all the databases.

The integration target is twofold:

1) to ensure consistency among the data bases when one application modifies some data in its local database;
2) to allow execution of distributed queries, that is queries involving several databases; that could be the case of a query generated by App3 for the data describing a given patient (this requires the union of data from App1 and App2).

We want to build an ontology which serves two purposes:

- to support “ontology-based specification” ([8], [14], [17]), aiding the integrator in: (a) finding software components and their properties; (b) validating consistency of component configuration; and (c) deriving further knowledge about the integration scenario;
- to support “common access to information” ([8], [14], [17]), allowing access the semantic model from conventional programs. This makes possible the construction of integration components which exchange ontologies rather than raw data, avoid hard-coding of mechanisms for data generation, interpretation and validation.

To satisfy the requirements of the previous two points, an ontology must be built around:

a) concepts belonging to technical and architectural aspects, such as hardware component, software component, component allocation, etc.;
b) concepts relative to the domain of interest such as patient, physician, therapy, etc.;
c) individuals representing realizations of the concepts of the two previous points.

This leads to the structure of Figure 1 where EAI Ontology Schema is the ontological model pertaining to the technical aspects, Domain Ontology Schema is the ontological model pertaining to the domain of interest and EAI Ontology Data are the individuals instantiated from EAI Ontology Schema.

![Fig. 1. General view of the ontological model supporting EAI.](image-url)
A. EAI Ontology Schema

The EAI Ontology Schema describes the concepts that are useful for integration. It is essentially composed of three classes, and their subclasses, representing respectively: (a) hardware components (i.e., machines); (b) software components; and (c) locations. Needless to say, the model also includes the associations among them.

As regards software, initially the integration scenario will only contain existing applications. In the course of integration, new sub-classes of class Software are added. For instance, when a given type of EIC, say an Adapter, is identified, the corresponding class is added to the EAI Ontology Schema, while individuals of that class are instantiated in the EAI Ontology Data (see below) to represent the actual adapters entering in the integration solution. In this way the EAI Ontology Schema grows so as to describe any software class, while EAI Ontology Data grows so as to describe any actual software component comprised in the integrated system.

B. Domain Ontology Schema

The Domain Ontology Schema contains domain concepts and domain predicates. It corresponds to the so-called “canonical data model” of [25], that is a data representation to/from which all the different data formats are converted. The essential difference with respect to [25] is that with the Domain Ontology Schema the developer obtains the payloads of exchanged messages by instantiating ontological classes in the Schema. This makes the system more resilient to the evolution of domain concepts. In fact, ontologies, provide a comprehensive and flexible conceptualization of domain data [26] which is inherently extensible and sharable across applications. This is the key concept to avoid hard-coding of data interpretation. (In section V-B we shortly describe how this is done with the support of Jena.)

Referring to our integration target, the domain ontology must contain the class Patient and the following predicates: hasName, hasSurname, hasPlaceOfBirth, hasDateOfBirth, hasSocSecN, hasAddress, hasProfession and hasID. These predicates correspond to the union of the sets of attributes managed by App1 and App2. Furthermore, for each ID we must know the value of the local ID and the identity of the system to which that ID is related. This means that the rank of predicate hasID will be made of couples such as (systemName, value).

C. EAI Ontology Data

EAI Ontology Data contains individuals that are instantiated from EAI Ontology Schema. These individuals describe the actual (hardware and) software components; for instance, App1, AdapterK, etc.. Any instance is augmented with references to the domain concepts (e.g. Patient) classifying entities that are related to the component itself. In Figure 1, this is schematized through the dependency labeled <<refer>>.

EAI Ontology Data is populated in the course of integration activities. It is the duty of integrators and domain experts to establish the appropriate relations between the instantiated components and the concepts of the Domain Ontology Schema.

III. INTEGRATION COMPONENTS

Each EICs is considered to be a black box with its own inputs and outputs. For each input (output) we specify the types of messages that are accepted (generated). Figure 2 shows that software components are related to exchanged messages which, in turn, are associated to domain concepts (refer to the discussion of section II-C). To avoid cluttering Figure 2, certain details have been omitted (for instance, communication channels associated with ports are not shown).

![Figure 2. Part of the model describing integration components as black boxes.](image)
may also send commands to the adapters ordering them to update the representation of the patient.

- **Query Manager** is in charge of solving distributed queries, i.e., queries that involve the search of a given Patient in more than one database. Specifically, the Query Manager: (a) listens for a distributed query; (b) converts the distributed query into a query for the given Patient; (c) sends the query over the communication bus, so that all adapters can pass it to the associated data base; (d) collects the answers from adapters (i.e., from applications/databases); (e) uses the database maintained by the Identity Manager to eliminate duplications; (f) builds the body of the response to the distributed query; (g) sends the response over the communication bus.

Concerning point (e), the Query Manager and the Identity Manager interact on the basis of the integration pattern **Shared Database** [25]. The shared database is the database of equivalences maintained by the Identity Manager. (The dependency of the Query Manager on the Identity Manager is explicitly shown Figure 3.)

**V. BASIC INTEGRATION COMPONENTS**

We distinguish between basic EICs and composed EICs. The former are those whose operations cannot be broken into a number of simpler operations carried out by other components; the latter are those whose functions are obtained by appropriate composition of functions carried out by other components.

In this section we discuss the construction of a basic EIC (an adapter). In so doing we take the opportunity for explaining how ontologies help structuring and validating exchanged information. Next section will describe the construction of a composed EIC.

**A. Component definition**

In designing any component, the first step is the specification of its interface. For an Adapter, this is specified in terms of interactions with both the communication bus and the associated application. In Figure 4, the former are represented as solid arrows, the latter as a dashed (bidirectional) arrow. Specifically:

- **Patient Insert, Update and Delete Events** are notifications messages elicited by the local application. They respectively inform that something happened in the local database: (a) a new patient has been added; (b) a patient has been updated; and (c) a patient has been deleted. In our simple system, they are observed only by the Identity Manager, which initiates a sequence of operation that aligns the equivalence database maintained by the Identity Manager itself and all the databases containing data relative to the involved patient.

- **Patient Insert, Update and Delete Commands** determine execution of the corresponding operation against the data base managed by App. In the system of Figure 3 these messages arrive to the Adapter sent by the Identity Manager.

- **Patient Query Command** is converted to a query to the data base managed by App. In the system of Figure 3 this message is sent either by the Identity Manager or the Query Manager. The answer of the database is the used to build the **Patient Query Response**.

- Interactions with the associated application/data may take place in different ways, including: (a) use of services exposed by the application; (b) direct interaction with the local data base; and (c) use of conventional application interfaces.

An adapter is activated either by the associated application or by an appropriate message coming from the bus. The first case occurs when (a) the associated application needs to perform a query across the distributed system, or (b) the associated application has made a change to its local data that is to be communicated to the other application (change discovering is usually obtained via a trigger on the database). The second case is triggered by incoming command messages.

**B. Ontology-supported component implementation**

Consider the case in which the data relative to representation of patient “Mario Rossi” is modified by App1. Then, Adp1 will publish an appropriate notification. To set up the payload of the notification, the developer:

a) uses the ontology to instantiate the patient. This is done with the support of library like Jena, by instantiating class OntModel and setting the due properties;

b) converts the model to a transportable RDF string;

c) pack the string in a notification message and sends it.
In building a listener (e.g. the Identity Manager), the developer:

a) obtains the OntModel from the the RDF string constituting the payload of the received message. By construction the content of OntModel is consistent with its intended meaning. There is no need for hard coding specific validation validation mechanisms since there are automatically done by the supporting library;

b) process the message content on the basis of on component’s specialization.

VI. COMPOSITION OF EICs

In the following, we refer to the Query Manager as an example of construction of a composed EIC.

The Query Manager recognizes a Distributed Query Command (generated by any application), interrogates the individual databases and recombines their answers to build the response to the command. Of course it could be programmed from scratch, though a better solution is to obtain it from the cooperation of simpler components. In Figure 5, the Query Manager results form cooperative work of two further basic components, i.e. the Aggregator and the Query Process Manager, both shaped as the EIPs [25] of the same name.

- Aggregator produces a message containing a list of patients obtained through aggregation of input messages.
- Query Process Manager is the adaption to our needs of the integration pattern Process Manager of [25]. This EIC has an internal state to keep track of the advancement of processing.

Fig. 5. How the Query Manager is obtained from simpler components.

In building the response to a given distributed query, the Query Manager must use only the database responses that are relative to that query, avoiding to mix up responses that relative to other queries. This implies that messages must have a correlation ID so as to determine which message goes with which. The Query Manager behaves as follows:

1) recognizes a Distributed Patient Query Command; the message could say “send me the data relative to Mario Rossi”;

2) transforms the received message into a Patient Query Command. This message will be observed by any adapter which will obtain the data relative to the identified patient (i.e., Mario Rossi) from its local store, so as to answer with a Patient Query Response;

3) takes the Patient Query Data which are built by the Aggregator through composition of the Patient Query Responses coming from application adapters in response to the Patient Query Command;

4) analyzes the Patient Query Data eliminating possible duplicates so as to build the Distributed Patient Query Response (this will be observed by the adapter of the application that issued the distributed query).

(For completeness, we must say that Figures 5 has been drawn in a simplified manner. For instance, it does not show how the Aggregator is initialized to properly group incoming messages.)

The Query Manager of Figure 5 is a logical component rather than a real one, in the sense that its functionalities results from proper composition of the functionalities of the Query Process Manager and the Aggregator, which are the actual, concrete components. In other words, the Query Manager is obtained without writing a single line of code, the only requirement being that the installed individuals are properly configured so as to operate in concert.

Of course the process of composition can be iterated to build further, more complex components. For instance, we may want a component capable of returning all and only the patients that satisfy a given condition. This can be done combining the Query Manager with additional components capable of splitting and filtering the responses to Patient Query Command coming from individual applications.

In the stage of component definition, the ontological knowledge base is used to identify the existence of already defined components with certain capabilities. A reasoner can be used to validate the consistency of a given composition. For instance, we can verify whether messages Patient Query Data that the Query Process Manager accepts in input are consistent with those generated by the Aggregator. Before deployment, we can also validate whether each individual component has been properly configured, as well as validate overall configuration consistency.

VII. CONCLUSIONS

We defined an ontological knowledge-base to describe both the technical and the semantic aspects of integration. Technical aspects are captured by an ontological model (EAI Ontology Schema) describing the overall hardware/software system. This includes the description of both applications to be integrated and integration components, as well as relations and dependencies among them. Starting from the initial representation of the application systems to be integrated, the ontology grows so as to include any EIC that is defined and validated.

Semantic aspects are captured by an ontological model (Domain Ontology Schema) describing concepts, relations and dependencies relative to the specific application domain. Instances of entities of the EAI Ontology Schema are associated with concepts of the Domain Ontology Schema giving rise to an ontology that relates software components to the domain concepts they deal with.

The developer uses the knowledge-base to examine the properties of the applications that are involved in a given integration target. This is followed by the definition of the integration components that are required to achieve the specific
target. In this process the developer resorts to Enterprise Integration Patterns, structured solutions to almost any integration problem. In laying down the integration solution, he can consult the knowledge-base in search for the already defined integration components which possess (either partially or completely) the functionalities that are to be implemented. This is particularly useful in order to obtain new functionalities by aggregating existing integration components (either concrete or logical), so as to give rise to additional behaviors which correspond to new (logical) components. The same ontological model is the tool through which the semantics of EIC compositions, EIC configurations as well as system configurations is validated.

In programming EICs, the developer uses ontologies to derive the information contained in transmitted messages. As a result, the payloads of exchanged messages are actually ontologies, so that the receiving components can automatically rebuild information from those ontologies. This also guarantees that information circulating among integration components is always consistent with its intended meaning, avoiding a typical problem of today’s enterprise integration platforms, that is the proliferation of configuration files.

ACKNOWLEDGEMENTS

We are grateful to A. Tarocchi who helped defining and implementing the integration software. We are also grateful to C. Berdondini and L. Bartoli for their endorsement of the project within the Ospedale di Careggi, Florence, Italy.

REFERENCES


