# The Role of Semantic Web Technology in Enterprise Application Integration

Christoph Bussler
Digital Enterprise Research Institute (DERI)
Chris.Bussler@DERI.ie

**Abstract**

*Enterprise Application Integration (EAI) in today's form focuses on the syntactic integration of application interfaces on an implementation technology level. Current EAI technology focuses on the "plumbing" aspect by emphasizing different technologies like queuing [5, 8] or remote invocations [12, 14]. A formal description of the semantics of the interfaces necessary to make integration work flawlessly is not supported by applications or integration technology in any explicit form yet, for example, as process models or formal ontologies. Instead, implementing the correct integration semantics is a human designer task and the result is consequently subject to mistakes and mis-interpretations. This article introduces a first step toward making formal semantics explicit in EAI.*

## 1 Introduction

The main goal of enterprise application integration (EAI) [2] is the semantically correct integration of enterprise application systems like ERP (Enterprise Resource Planning) systems [7, 13]. Enterprise application systems have technical interfaces that are implemented in various forms. Examples are programming language-defined interfaces (e.g., Java [6]) or XML schema definitions [17].

While this allows the correct syntactic integration through passing valid instance data on a technical level between enterprise application systems, their semantics are not formally defined. Instead, the engineer integrating the enterprise application systems has to know the meaning of the low-level data structures in order to implement a semantically correct integration. This means that the correctness of the integration depends completely on the knowledge of the engineer and his flawless implementation of the integration. No formal definition of the interface data exists to assist him, let alone any formal automatic verification. Due to this lack of automated support in defining integration, it takes a long time for a human engineer to define semantically correct integration. In addition to the development effort, inconsistent business data due to faulty integration are costly for an organization due to potential loss of business caused by faulty downstream business decisions.

The advent of Semantic Web technology like ontology languages (such as RDF [10], RDF/S [11], OWL [9]) promises to improve this bleak situation significantly by enabling the formal semantic description of the semantics of the data structures exchanged with enterprise application systems. The concepts as well as the relationships between the concepts are defined through an ontology language. Based on this semantic definition


**Bulletin of the IEEE Computer Society Technical Committee on Data Engineering**

the knowledge about the meaning of the data is formally encoded and the engineer can rely on this. Instead of learning the meaning of the data based on experience and instead of interpreting the data manually the engineer has access to a formal definition of the data. The semantic definition of the data is part of the enterprise application system and therefore leaves the engineer with few, if any, mis-interpretations.

This paper outlines a general enterprise application integration architecture and its elements related to the semantic data definition of enterprise application interfaces. It focuses subsequently on the data interpretation and transformation problem which lies at the heart of every enterprise application integration effort. It defines a general schema for augmenting the enterprise application integration architecture with Semantic Web technology in order to support the semantically correct enterprise application system integration.

# 2 Enterprise Application Integration

This section characterizes the EAI problem briefly from the viewpoint of interfaces and the definition of data passed through them.

## 2.1 Applications are HAD systems

An appropriate characterization of application systems in context of EAI is that applications are HAD (heterogeneous, autonomous and distributed) systems:

- **Heterogeneous** refers to the fact that each application system implements its own data model defining its relevant concepts in its own way. One application might implement a customer as a separate object while another application system might implement customer as a special case of trading partner. In both cases there might be similar as well as dissimilar attributes of the concept customer.
- **Autonomous** means that application systems update their internal instance data independent of any other application systems. Furthermore, new versions of the application system software might become available at any point in time changing the underlying data model and consequently the instance data representations. An upgrade to this new version usually happens independently of upgrades of other systems.
- **Distributed** means that applications systems in general implement their data model in their own data repository (usually in form of a database management system) and these are not shared with other application systems.

In summary, application systems are standalone software entities that are defined in isolation and are operated autonomously. Any form of integration of one application systems with another application system happens "outside" of the application systems involved, meaning, that they are not aware of the fact that they are synchronized in any form with each other.

## 2.2 The Need for Integration

The need for integration of application systems arises when they are "concerned" about the same concepts and their instance data. This "concern" can have different forms:

- **Copy Synchronization.** If an organization deploys several application systems then it might be the case that the same business instance is maintained in several of them concurrently. For example, a particular customer may be represented in two different systems. A change of one has to be propagated to the other.
- **Replica Synchronization.** In the copy case the propagation is "non-transactional," meaning, that before the propagation of change two variants of an instance data are visible. In the replica synchronizing case this is avoided which ensures that all copies are synchronized before being made available (replica).

- **Business Process Participation.** A business process changes the states of objects in a pre-defined sequence as it is executed. These are normally related since they are concerned about a business case (for example request for quotation, purchase order processing, invoicing). The integration therefore ensures that the appropriate application systems are involved as the business process progresses and therefore implements data flow across the application systems.

Independent of the particular reason for integrating application systems it is necessary to access the application systems' interfaces in order to manipulate their internal data instances in order to achieve the integration. This manipulation is the "classic" insert, update, delete access. However, in general, more than one data instance is modified and so their combined modification must leave a consistent state behind. Application systems implement different interface mechanisms on a technical level and these are discussed next.

## 2.3 Application Interfaces

Application systems provide a myriad of different forms of interface technologies through which they can be accessed. Some examples are persistent and transactional queuing systems, file systems, data base tables, or remote procedure invocation. Each of the interface technologies has its own interface description approach (if at all). For example, data base tables are defined through system tables that can be queried. In contrast, interfacing through a file system does not provide any interface definition at all; the meta data for the data within the interfacing files is not explicitly managed by the file system but outside of it, if at all. The reason is that file systems do not provide a mechanism to define the files' content type definition.

A variety of data formats for data instances can be found as well as different formats for meta-data definitions. Independent of the particular mechanism, today usually no mechanism exposing semantic meta-data about the interfaces or concepts behind interfaces is provided. All definitions are created from a syntactic programming viewpoint, not from a semantics viewpoint. For example, each interfacing message type in a queuing system is defined individually as a data structure. No shared concepts are explicitly called out that are used in different message types (like purchase order identifiers or addresses).

In general, data instances are passed to and from application systems by-value. All relevant data instances are part of the data structure that is passed. No references are passed which avoids the de-referencing problem once data leaves the application systems. However, this introduces the copy management problem since copies are passed around, not replicas. Any update of data extracted will not be visible by systems holding copies.

Application systems interfaces are fixed and in general cannot be changed to adjust to particular access needs by the integration technology. The same applies to the application code; it is immutable, too. Hence, integration technology has to cope with application systems as they are built by their respective vendors.

## 2.4 EAI is Integrating HAD systems

Fundamentally, EAI is integrating HAD application systems that are immutable. All integration functionality has to be implemented outside the application systems by the application integration technology. In general, the only way to access an application system is through its interfaces and they cannot be altered at all.

In consequence this means that the EAI technology itself is a HAD system that has to control other HAD systems by only accessing their interfaces. No state transition an application systems performs can be forced, delayed or avoided by the integration technology except by calling the application system interfaces appropriately. An application system does not ask the EAI technology for permission.

## 2.5 EAI Technology

EAI technology is implemented as a HAD system itself with specialized interfaces. These specialized interfaces are built with enormous flexibility. The reason is that these interfaces have to be able to adapt to any application system interface that can be potentially found when integrating application systems. EAI technology interfaces must be able to adapt to the interfaces of the enterprise application systems for it to integrate them.

However, not only the syntactical and technical integration must be possible. The semantics level must be addressed, too, and this is not focus of today's EAI technology. While this is not current focus, it is essential in order to achieve consistent integration so that the application systems that are integrated operate on a consistent set of all data involved.

Since the application system interfaces are immutable, both from a syntactic as well as semantics perspective, the EAI technology must be able to mediate between technical formats of different application systems on a data structure level as well as on a semantics level. On a technical level the EAI technology must be able to receive any particular data format from an application system as well as to pass any particular data format to an application system. If these are different, then the EAI technology must transform the data format representation into each other. This also has to happen on a semantics level. If the same concept is represented in different semantical terms then the EAI technology must transform between these. This semantic transformation is also called mediation. Mediation must ensure that the semantics of the involved concepts does not change at all, i.e., it must be semantics preserving.

## 3 Semantic Application Interface Integration

The syntactical integration of application system interfaces can be accomplished today using the already mentioned technologies like queueing systems, programming languages or XML. Therefore, it is not further discussed in the following. Each syntactical element (like a data structure in a message queue definition) has a semantic meaning in context of the application system it is part of. This meaning has to be preserved once the data structure leaves the application system through an application system interface and enters the EAI technology. In order to formally define this meaning the proposal is to encode the meaning of the data structures in formal ontology languages like [9] or [11]. If the meaning of the data structures passed through application system interfaces is encoded in a formal ontology then the EAI technology can refer to it. Any designer that is responsible for the integration of application systems can rely on these ontologies and does not have to understand the application system's internals any more.

Of course, the ontologies have to be expressed in such a way that the meaning can be inferred. A bad example would be to introduce a concept "Address" with two attributes, "address–line–1" and "address–line–2", both of type string. In this case the structure of the two attributes would not be made explicit (i.e., not normalized) and the internal structure of them is anybody's guess. However, ontologies are meant to leave out the guessing part. Instead, several attributes, each referring to other address concepts might be the better design, e.g., "street–number", "city", etc., each referring to a separate concept "Street–Number" and "City." For those concepts that can have enumerable instances, the attributes domain has to range over those, tying down the semantics even more. In addition, there are constraints between attributes, like cities have specific zip codes (or no zip codes, like in Ireland). These constraints, when expressed explicitly, will make the interface definition semantically more precise.

Application systems themselves, however, since they are immutable, will not present ontologies as part of their syntactical interface definitions today. While this can be expected in the future, today this cannot be found at all. Therefore, the EAI technology must store the application systems' semantic interface definitions within its own data model. In that sense the application system interfaces' definition is stored as a proxy interface definition. Once the application systems' interfaces are stored, they can be reused in any integration defined within the EAI technology.

In summary, EAI technology has to be able to connect to application systems' interfaces syntactically as well as define those semantically. Once this is achieved, different types of integrations can be defined based on the stored definitions. In general, as discussed above, different application systems will express the same or similar concepts in very different ways. Since only the EAI technology is aware of these differences (not the applications themselves due to their autonomy) it has the difficult task to mediate between the differences in case one application system has to be integrated with another one.

# 4 Mediation

In general, every application system defines its own concepts and stores them in its own data model. The providers of application systems do not coordinate the data model design of their applications, but define them independently of each other. This approach results in the situation that every application system has a different implementation of meta-data in its data model even though they may be implementing the same concepts like "customer". Basically, concepts might be shared, but not their specific way of implementation.

For example, one system implements the "customer" concept directly while in another system the "customer" is a subconcept of "trading partner" inheriting all its attributes. While both are representing customers, their attributes might be very different. If both application systems have to be integrated and a customer has to be copied over from one system into another system then the concepts have to be mapped onto each other in all detail in order to ensure that both customer representations are consistent with each other.

In terms of ontologies, mediation is the re-representation of concepts from one ontology in terms of another ontology. If both ontologies are the same, then the mediation reduces itself to a copy functionality. If the ontologies are different, rules have to be put in place that allow the re-representation of concepts. For example, if one ontology has an explicit address concept whereas the other ontology adds the address attributes directly to the customer concept then the rules have to map the attributes of different concepts to each other in order to ensure semantic consistency.

The role of ontologies is therefore to provide a sound and formal as well as explicit representation based on which the mapping between the concepts can be achieved. While ontologies by themselves do not achieve mediation directly, their availability makes mediation easier and more reliable in terms of consistency.

# 5 Shared Integration Ontologies

A huge improvement over the current situation would be if application systems would share ontologies between each other for the definition and implementation of concepts. If they not only share ontologies, but also use explicit and formal representations then the integration task would be a lot easier. First, ontologies would not have to be extracted from application systems after the fact. Second, since the ontologies are shared, the mediation problem will be reduced (or will not be there any more in the best case) since the same concepts are represented in exactly the same way. Both cases improve interoperability since the uncertainty of establishing semantic correctness would be removed.

However, this almost ideal situation cannot be expected to happen anytime soon. First of all, application systems are in use today and will not be replaced any time soon on a global scale. Second, the sharing of ontologies is not current practise at all. However, describing the semantics of interfaces using ontologies can be expected in the near future and this is a very good start.

# 6 Web Services and EAI

Web Services, defined as a combination of WSDL [16] and SOAP [14] are presently the proposed method for remote invocation of application systems. Web Services provide a means to synchronously invoke application logic remotely over the HTTP protocol with XML as the transmission format for the transmitted data. Web Services have two major drawbacks: first, they do not provide any ontology support, and second, they do not provide support for modeling application system interface behavior. As argued earlier, ontology support is extremely important to provide system support for integration. Application system interface behavior is relevant to invoke the exposed interfaces in the correct order. Otherwise errors would occur prohibiting interoperability. Application system interface behavior is discussed next.

### 6.1 Application System Interface Behavior

Using ontologies as a formal and explicit way of defining interfaces is a very important and essential first step. However, this by itself does not ensure data consistency across the integrated application systems. Different systems require different interface invocation protocols (i.e., which application interface has to be invoked when) in order to achieve a consistent state transition within the application system. While one application system might provide one interface to create a new customer, another might provide different interfaces, one for creating the customer object, one for inserting the name, another one for inserting the address and one for activating the customer in the sense that he can start purchasing goods. In the first case, one invocation completely defines a customer whereas in the second case several interface invocations are necessary. However, these invocations have to follow a specific order: activating the customer cannot be the first invocation, but must be the last after the customer is established.

Like in case of the concepts, different application systems can require different invocation protocols and this might result into an invocation heterogeneity. As in the example of the previous paragraph, one application might provide one interface for customer creation whereas the other application requires several invocations. If both are integrated, behavior heterogeneity occurs (one vs. several interface invocations). This so-called behavior heterogeneity has to be addressed, too, and this is done through an approach called behavior mediation [4]. Behavior mediation ensures that the invocation sequence of one application system is obeyed and so is the one of the other application system. Any mismatch will be mediated through additional invocations or invocations that will not be propagated across systems.

### 6.2 Process Languages

The definition of behavior is focus of several efforts like DAML-S [3] or BPEL [1]. These efforts try to address the lack of behavior definition in Web Services through process definition languages. They provide language constructs that support the ordering of Web Service invocations. Most of them, however, treat the behavior problem like a workflow problem not distinguishing the external interface behavior ("public process") of application systems from the internal implementations ("private process"). In addition, most of them do not provide language constructs for either data or behavior mediation. In light of the previously discussed properties and requirements in context of application systems and their integration these process languages can only be seens as a first step towards a practical solution.

## 7 Outlook

The semantic integration of application systems (EAI) is a hard problem due to the need of semantic description of application interfaces. Not only the data have to be described semantically meaningful, but also the behavior of application system interfaces. First steps toward the semantic description of data communicated by application interfaces can be achieved using ontologies. In addition, the external or public behavior of application system interfaces needs to be established. While for the data aspect ontologies from the Semantic Web research community are available, the process aspect is far from being solved. However, efforts like WSMF [4] or SWSI [15] realized the lack of semantic interface behavior support and are working towards practical solutions.

## 8 Research at the Digital Enterprise Research Institute (DERI)

The mission of the Digital Enterprise Research Institute (DERI, www.deri.ie) is to combine two fundamental technologies: Semantic Web technology and Web Service technology. The result, called Semantic Web Services, is the foundation for various application areas like Knowledge Management, Enterprise Application Integration and e-Business. For each of these five areas a dedicated group, called cluster, is established focusing on research in the respective area of expertise as well as applying the research results in real-world applications provided through industrial partners. DERI is funded nationally by the Science Foundation Ireland. In addition to Irish funding, international funding is acquired through international projects like European Union 6th Framework projects (see www.deri.ie for the current list).

## References


[1] Business Process Execution Language for Web Services, Version 1.1, 5 May 2003, BEA Systems, International Business Machines Corporation, Microsoft Corporation, SAP AG, Siebel Systems

[2] Bussler, c.: B2B Integration. Springer Verlag, 2003

[3] www.daml.org/services/

[4] Fensel, D.; Bussler, C.: The Web Service Modeling Framework WSMF. In: Electronic Commerce Research and Applications, Vol. 1, Issue 2, Elsevier Science B.V., Summer 2002

[5] IBM Webshpere MQ. www-3.ibm.com/software/integration/wmq/

[6] Java. java.sun.com/

[7] www.oracle.com/applications/index.html

[8] Oracle9i. Application Developers Guide - Advanced Queuing, Release 2 (9.2), March 2002, Part No. A96587-01

[9] OWL Web Ontology Language Overview, W3C Candidate Recommendation, 18 August 2003

[10] Resource Description Framework(RDF), Model and Syntax Specification, W3C Recommendation, 22 February 1999

[11] RDF Vocabulary Description Language 1.0: RDF Schema, W3C Working Draft, 10 October 2003

[12] RPC: Remote Procedure Call, Protocol Specification, Version 2, www.ietf.org/rfc/rfc1057.txt?number=1057

[13] SAP. www.sap.com

[14] SOAP Version 1.2 Part 0: Primer, W3C Recommendation, 24 June 2003

[15] Semantic Web Services Initiative. www.swsi.org/

[16] Web Services Description Language (WSDL) Version 1.2., W3C Working Draft, 3 March 2003.

[17] XML Schema Part 0: Primer, W3C Recommendation, 2 May 2001