Semantic Trading: Tackling Interoperability Problems during System Integration

S.Terzis and P. Nixon

Computer Science Department, University of Dublin, Trinity College, Dublin 2, Ireland
{Sotirios.Terzis, Paddy.Nixon}@cs.tcd.ie

Abstract: With the emergence of the Virtual Enterprise (VE) model for business organization, new enterprise systems are constructed by integrating existing one. The integration is achieved through the composition of components selected from either a collection of components provided by the participating enterprise systems or external trusted sources. The process of component selection and composition is hindered by numerous problems at all levels of interoperability. This paper proposed the use of semantic trading to tackle interoperability problems mainly at the semantic level.

1. Motivation

The convergence of several threads of technology and business imperatives makes the Virtual Enterprise model for business organization both feasible and appealing. The effectiveness and efficiency of the model depends on the development of flexible global information infrastructures. These infrastructures should support the dynamic composition and configuration of components. They should also support on-the-fly re-composition and reconfiguration of components. Finally, since the number of these components will be significant, these infrastructures should also be scalable (see [1] for a more detailed discussion).

With the support of these infrastructures the development of new enterprise systems is a process of dynamically selecting the appropriate components, and composing them into a new configuration. This process presents a number of challenges to traditional development techniques. First, the VE model changes the basic assumption of equally trusted, and equally under control, system parts that drives traditional system development. Under the VE model that is simply not the case. Second, severe interoperability problems arise since different systems were developed using different languages and protocols, on different platforms, under different models. Although, current development frameworks like CORBA [2] and DCOM [3] provide sufficient support for overcoming interoperability problems at the syntactic level, they are insufficient in supporting semantically rich interactions and interoperability at the protocol and semantic level.

This paper proposes the use of semantic trading to support the development of the new enterprise systems and to tackle interoperability problems mainly at the semantic level. Semantic trading is a fundamentally different kind of trading based on, one hand, a component description language that can capture behavioral characteristics of

components and on the use of ontologies, which allow the precise description of enterprise system models. A community of semantic traders with the support of a composition service could tackle interoperability problems:

- first at the semantic level by forming a common understanding (common ontology) between the different system models,
- second at the protocol level by making explicit protocol dependencies, and
- third by seamlessly wiring components through proxies, adapters, mediators and brokers using the composition service.

2. Supporting Interoperability through Trading

There are three levels of interoperability (a) syntactic, (b) protocol and (c) semantic. In this section we discuss how current trading addresses interoperability issues at the syntactic level and how can semantic trading tackle interoperability problems at other two levels.

2.1. Current Trading

A definition of current trading is the following: "the activity of choosing services, such that they match some service requirements. The choice is based on the comparison of the specification of a service required (provided by a prospective consumer) and the service specification supplied by service providers or their agents" [4]. Trading is based on the notion of matching service offers and service requests[1]. The component responsible for the maintenance of the trading space and the matching of offers and requests is called a trader. Current traders [4, 5, 6] organize services in a service type hierarchy. Each service type defines an interface that prescribes the operations available for the interaction between the service provider and the service consumer. They also allow the association of a number of properties as attribute value pairs with each service type.

Central to the service type matching is the notion of type conformance[2]. Type conformance is based on the "no surprise rule" on the interaction interface, which means that no unexpected input or output requirements or operation termination conditions will emerge guaranteeing in this way interoperability between service providers and consumers (see [4] for a more detailed discussion about type conformance). Since type conformance is determined by the interaction interface, the description of interfaces is crucial in current trading. For the specification of interfaces interface definition languages (IDLs) have been developed (e.g. CORBA/IDL [7]). So, trading is currently IDL-based. This means that the foundation

[1] A service is a function provided by a component at a computational interface [5].

[2] If TypeA conforms to TypeB then instances of TypeA can be substituted for instances of TypeB and the clients will not be able to detect that the interface being used is not of the exact requested type.

of trading is answering questions of the kind: "*Find me a service **that conforms** to the following service type and has the following attributes*".

Currently, a trader with the support of an object request broker (ORB) [2] solves interoperability problems by first locating all the components that are compatible and then, the ORB plugs them together. At this level compatibility is translated into interface conformance and interoperation into message forwarding (method calls). "*Unfortunately, …behavior of methods is clear only by explanation or documentation, and therefore susceptible to misinterpretation and misunderstanding. …CORBA [IDL] provides only the syntax… semantics are implicit and unenforced*" [8]. So, the nature of current trading is syntactic. Thus, it is unable to address interoperability issues at the semantic level because "*An interface definition provides tremendous leverage in defining a contract but is insufficient for frameworks that support semantic concepts beyond those provided by a messaging infrastructure*" [9]. Additionally, even for a messaging infrastructure the support is limited since issues of message ordering (protocol interoperability) are not addressed.

In order to address interoperability issues at the semantic level what is needed is "*The inclusion of semantics …[to] IDL: information about the meaning of a component, information about what the component will accomplish, and information about the method's inputs and outputs* " [10]. In other words what is needed is a new kind of semantically rich trading, semantic trading. Semantic Trading is a fundamentally different type of trading because its foundation is answering questions of the type: "*Find me a component/service **that does** the following*". So, the focus shifts from the appearance (syntax) to the functionality (semantics) of the components.

2.2. Semantic Trading

Semantic trading differs from current trading in a number of aspects. First, the basis of semantic trading is a Component Description Language (CDL) instead of an IDL. CDL besides the interface is also able to capture behavioral characteristics, like invariants and pre-/post- conditions, context dependencies, like required interface consumed and produced event and relationships between components (for example see [8]). Second, the matching process in semantic trading is based on the notion of component compatibility instead of type conformance. Component compatibility is determined by substitutability according to the different reuse patterns as they are described in [11]. Finally, the trading process is supported by an ontology of concepts that captures semantic relations instead of a type hierarchy.

Semantic trading uses information about the behavior of the components, which is extracted either by the knowledge acquired at system design or by reverse knowledge engineering (see [12] for a discussion about reverse knowledge engineering). This information is used to construct an ontology of the system's domain. In fact, if a design process like the one described in [13] was followed, then the concepts of the ontology are directly the concepts (types) identified during the design and their relationships can be directly mapped into the ontology. The ontology is the basis for the construction of a knowledge base that encodes the design of the system and

provides a common vocabulary between the components, which ensures that there are no misconception hiding [14]. During the trading phase design knowledge in combination with a number of rules, which capture various composition patterns[3] that appear in different architectural styles (see [16] for an introduction on architectural styles) determines semantic interoperability constraints and a composition strategy that overcomes them. This strategy is used by a composition service in order to create the necessary proxies, adapters, mediators or brokers that will allow seamless component wiring. The knowledge for overcoming interoperability problems is encoded into the composition rules and a number of components that have to be available in the system. If that is not the case then the semantic trader will be unable to determine a complete strategy leading to a failure of the composition phase. In these cases human intervention is unavoidable. So, the trader will prompt the developer reporting the situation (currently determined strategy) in which case the developer will have to either provide the components missing or transform some of the available ones. Finally, when the user identifies a new composition pattern he could add a new matching rule to the system.

Although semantic trading is not based on interface matching, the use of a composition service in combination with an ORB allows the semantic trader to maintain syntactic level interoperability. In order to achieve that the composition service adapts located components so that they present the required interface. In the simple case this adaptation might involve a type mapping function similar to the one described in [17]. The difference in semantic trading is that type-mapping components are located through the trader from a collection of available ones. Finally, the semantic trading could resolve some protocol level interoperability problems by extracting call-sequencing information from the knowledge base. This requires that information about protocol dependencies was made explicit during system's design either through the use of temporal constructs provided by the CDL (for example the `leads to` construct of [8]) or through the definition of special relationships between the components captured by the ontology.

3. Semantic Trading Architecture

Semantic trading will support system integration in the following way (see figure below): At the start there are a number of existing enterprise systems. Each of these systems consists of a number of connected components[4] and a semantic trader (the black circle inside each system). The creation of the new system (the system in the middle) will be driven by a new semantic trader, which is formed by composing the existing ones. The new trader is used to select the components that will form the new enterprise system while the composition will be supported by the composition service

[3] The term pattern here is in the sense of Design Patterns as described in [15]. Of particular interest are the adapter, bridge, proxy and mediator ones.

[4] In this paper we adopt the following definition of components: "*A software component is a unit of composition with contractually specified interfaces and explicit context dependencies only. A software component can be deployed independently and is subject to composition by third parties*" [18].

(explained below) associated with the trader, which creates the new configuration. Some times the existing systems might not provide all the necessary components in which case the component trader will try to retrieve missing components from a world-wide pool of available ones.

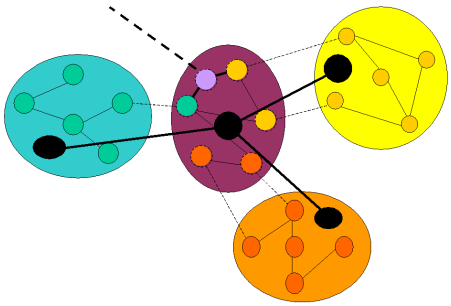

Fig. 1. Enterprise System Development using Semantic Trading

In this context and in order to play the role prescribed in the previous section a semantic trading architecture is defined as follows (see figure below): The trader receives service requests and replies by providing a composition of the retrieved components. The requests go through some pre-processing component. A number of different pre-processing components allow the trader to support a number of different formats for expressing requests. This makes the trader more flexible since it can support a number of different languages for expressing requests. After the requests have been transformed to the basic format, which depends on the component description language selected then, the requests are forwarded to the "translator".

The "translator" is the heart of the semantic trading process. Its aim is to transform the requests from the component description language into a number of alternative composition strategies, which will drive the component retrieval phase. Each strategy is determined with the use of one of the composition rules of the trader maintains specifies a container where the retrieved components have to be plugged-in. For each strategy a number of queries for the component repository (the trading space) is formed. The query formation process is supported by the terms of the ontology and ensures semantic interoperability through query expansion or contraction. The query expansion / contraction phase determines which of the "semantically close" alternatives should be included or excluded. The ontology is a number of terms with their definitions and relationships between these terms and is kept in the form of a semantic network (see [19] for an introduction into semantic networks). The semantic network is stored in a relationship service [20].

The interface of the components plays a secondary role in the retrieval phase. When the retrieval phase is completed the trader has to prepare the reply. The preparation of the reply is done with the support of the composition service, which

takes the composition strategy and the associated retrieved components provided by the trader and created a container. The composition strategy determines how the wiring of the components should be performed, how the necessary proxies, adapters, mediators and brokers are going to be created and composed to form the container. A scripting language (in the sense described in [21]) supports this phase. Finally, the trader returns the container to the client. In fact, the trading might lead to the generation of a set of containers in which case either one is selected according to the client's preferences or the whole set is returned.

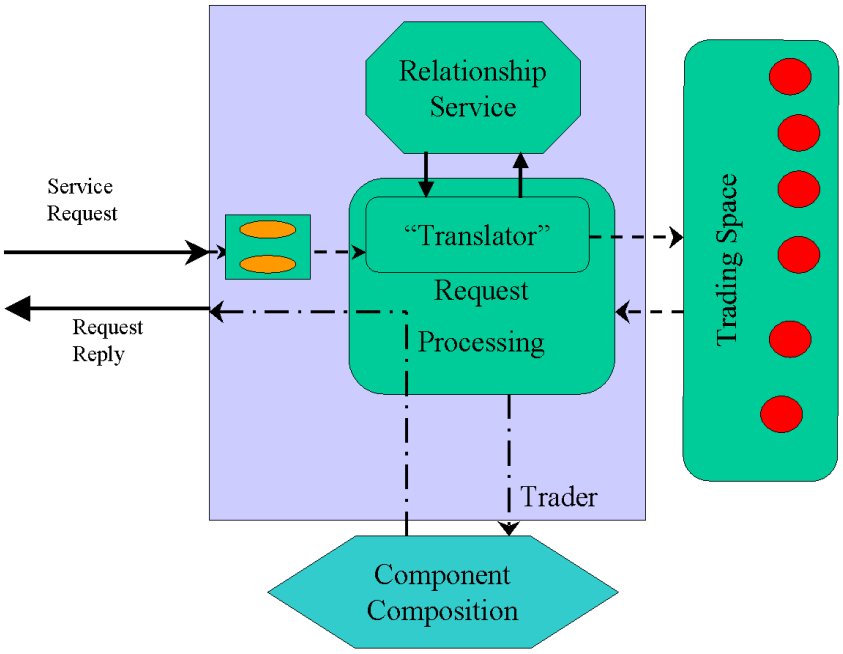

Fig. 2. Semantic Trading Architecture

As mentioned above, the use of semantic trading for system integration requires the composition of existing semantic traders. This composition involves the exchange of knowledge between the traders and the creation of a common ontology. The common ontology guarantees semantic interoperability between the components. The issue of exchanging knowledge is quite central in the multi-agent systems development and standards like KIF [22] and KQML [23] have been developed in order to support it.

4. Open Issues and Discussion

The above presentation shows how semantic trading could be used in a system integration environment to tackle interoperability problems and describes the basic

architecture for a semantic trader. The advantage of the method is that although it mainly deals with interoperability problems at the semantic level, it could provide the basis for dealing with interoperability problems at all level inside a single framework. That is because the semantic trading architecture presented above provides:

- Protocol interoperability by making component dependencies explicit through the use of information about the behavior of components,
- Semantic interoperability by providing a common ontology for the expression of semantic relationships and
- Syntactic interoperability by creating a container and wiring the retrieved components.

Although the issues of component interoperability and composition have attracted a lot of attention in recent years, general solutions have not yet been proposed. Our approach aims to provide a framework where the different solution could be combined. Of course as it is clear, there are a number of open issues, which need to be dealt before the framework can be realized. These issues are:

1. The construction of an ontology from design knowledge in order to enable interoperability. In particular how formal should the specification notation be, especially since it seems to be a tradeoff between formality and ease of use? Can different levels of formality for different aspects of the system be deployed in the same framework? Can a relatively informal notation like UML be sufficient to express the necessary information? Related is also the issue of how can we automate the knowledge extraction form the specification notation and the ontology construction?
2. The component composition rules. In particular what are the various composition patterns and how the different architectural styles affect them? How general can we assume the composition to be, especially in the light of component frameworks and architectures?
3. The definition of the scripting language for the composition service. In particular how can we create a scripting language so that it will allow automation but it will also be efficient? Are techniques like the dynamic skeleton invocation [7] enough or not? Is human intervention unavoidable?
4. Component annotation. How do we annotate components in order to expose their interoperability constraints? According to [24] it is impossible to capture all the different aspects of a component's specification that might be of interest. How can component annotation deal with this problem?
5. Finally, the knowledge exchange phase. How can the different composition rules along with the respective composition strategies be exchanged? What effects this knowledge exchange has to the trading federation?

References

1. S.Terzis, P. Nixon, V. Wade, S. Dobson and J. Fuller, "The Future of Enterprise Groupware Applications", in proceedings of the 1st International Conference on Enterprise Information Systems (ICEIS'99), Setubal, Portugal, March 1999.
2. OMG, "CORBA/IIOP 2.2 Specification", July 1998.
3. Microsoft DCOM, http://www.microsoft.com/com/

4. Jean-Pierre Deschrevel, “The ANSA Model for Trading and Federation”, Architecture Report APM. 1005.01, Approved 15 July 1993.
5. Draft Rec. X.950-1| ISO/IEC 13235-1 – ODP Trading Function: Specification.
6. OMG, “Trading Object Service Specification”, in CORBAservices: Common Object Services Specification, December 1997.
7. J. Siegel, “CORBA fundamentals and programming”, Wiley 1996.
8. J.R. Kiniry, “CDL: A Component Description Language”, submitted to the Advanced Topics Workshop (ATW) of the 5th USENIX Conference on Object-Oriented Technologies and Systems (COOTS’99), March 1999, available at http://www.cs.caltech.edu/~kiniry/projects/papers/kiniry/COOTS99_ATW/cdl/cdl.html
9. T. Digre, “Business Object Component Architecture”, IEEE Software, vol. 15, no. 5, September/October 1998.
10. J.H. Gennari, A.R. Stein and M.A. Musen, “Reuse For Knowledge-Based Systems and CORBA Components”, in proceedings of the 10th Knowledge Acquisition for Knowledge-Based Systems Workshop (KAW’96), Banff Canada, November 1996, available at http://ksi.cpsc.ucalcary.ca/KAW/KAW96/gennari/
11. K. De Hondt, C. Lucas and P. Steyaert, “Reuse Contracts as Component Interface Descriptions”, in proceeding of the 2nd International Workshop on Component-Oriented Programming (WCOP’97), Jyvaskyla, Finland, June 1997.
12. P. Devanbu, R. Brachman, P. Selfridge and B. Ballard, “LaSSIE: A Knowledge-based Software Information System”, Communication of the ACM, vol. 34, no. 5, May 1991.
13. J. Martin and J.J. Odell, “Object-Oriented Methods: A Foundation”, UML edition, Prentice-Hall, 1998.
14. M. Uschold and M. Grununger, “Ontologies: principles, methods and applications”, The knowledge Engineering Review, vol. 11, no. 2, June 1996.
15. E. Gamma, R. Helm, R. Johnson and J. Vlissides, “Design Patterns: Elements of Reusable Object-Oriented Software”, Addison-Wesley, 1995.
16. D. Garlan and M. Shaw, “An Introduction to Software Architecture”, in V. Ambriola and G. Tortora (ed.), Advances in Software Engineering and Knowledge Engineering, Series on Software Engineering and Knowledge Engineering, vol. 2, World Scientific Publishing Company, 1993.
17. D. Konstantas, “ Object Oriented Interoperability”, in proceedings of the 7th European Conference on Object-Oriented Programming (ECOOP’93), Kaiserslauten, Germany, July 1993.
18. C. Szyperski, “Component Software: Beyond Object-Oriented Programming”, Addison-Wesley, 1998.
19. J.F. Sowa (ed.), “Principles of Semantic Networks: explorations in the representation of knowledge”, Morgan Kaufmann, 1991.
20. OMG, “Relationship Service Specification”, in CORBAservices: Common Object Services Specification, December 1997.
21. O. Nierstrasz, D. Tsichritzis, V. de Mey, and M. Stadelmann, “Objects + Scripts = Applications”, in proceedings of the Esprit 1991 Conference, Dordrecht Germany, 1991.
22. ARPA Knowledge Sharing Effort, “Knowledge Interchange Format (KIF)”, information available at http://ksl-web.stanford.edu/knowledge-sharing/kif/
23. T. Finin, Y. Labrou, and J. Mayfield, “KQML as an agent communication language”, in J. Bradshaw (ed.), ``Software Agents'', MIT Press, Cambridge, (1997).
24. M. Shaw, “Truth vs. Knowledge: The Difference Between What a Component Does and What We Know It Does”, in proceedings of the 8th International Workshop on Software Specification and Design, March 1996.