

## **A CONTENT ORIENTED ARCHITECTURE FOR CONSUMER-TO-BUSINESS E-COMMERCE**

Joshua H. Greenberg<sup>1</sup>, Douglas Dankel II<sup>1</sup>, and Borko Furht<sup>2</sup>

<sup>1</sup>University of Florida, Gainesville, Florida, <sup>2</sup>Florida Atlantic University, Boca Raton, Florida

**Abstract.** Consumer-to-Business (C2B) systems represent the future of eCommerce. Using natural language as a basis, and remaining keenly aware of its potential pitfalls, we describe a software specific communication model based on a new concept called content-biased language (CBL). It is shown that the requirements of a C2B system cannot be satisfied with anything less than the stretchability of a CBL. Once this fact has been established, the remainder of this paper discusses a representation for a CBL, as well as an architecture for utilizing that representation. This effort results in the description of a new software quality measure called stretchability, as well as the introduction of perspective domain graphs (PDGs), external open ontological type systems (EOOTS), and global and constituent systems. Finally, the discussion closes with the definition of a new distributed system design called the Content Oriented Architecture (COA).

### **1. Consumer-To-Business: The New Acronym for eCommerce**

Business-to-Business, or B2B, is a well-known term designating the Internet-based supply-chain oriented transactions executed between corporations. Business-to-Consumer (B2C) has also entered the business executive's arsenal of Internet-based acronyms and generally signifies the set of activities surrounding the marketing and selling of goods by companies to individuals. Finally, the label Consumer-to-Consumer (C2C) is usually applied to online auctions and other non-corporate business activities. While all of these buzzwords designate important and profitable computing paradigms, it is a less frequently discussed model that may prove most revolutionary. Consumer-to-Business (C2B) is defined in [1] and [5] as the comparison shopping activities performed on-line by a user before purchasing a product. While this definition may accurately represent current implementations, it barely scratches the surface of what is possible. By enabling direct-marketing and self-marketing, the C2B concepts proposed and clarified in this paper, will allow consumers to do far more than simply compare prices and characteristics. It will place consumers on an equal footing with corporations.

If we move away from the notion of C2B as comparison-shopping, then at present, the most representative implementations are generally categorized as wallet software systems. Until very recently, the most formalized attempt at wallet software systems was the Electronic Commerce Modeling Language (ECML). ECML allows "consumers to enter personal details once into the wallet software, which could be called up as needed to make payments to retailers" [6]. Once the information has been entered, order forms for Internet transactions can be filled automatically with data, such as billing preferences, shipping information, identity, credit-card numbers, and digital certificates [3].

As recently as a year ago, Microsoft was discussing a new wallet based technology, code-named Hailstorm and later renamed as .NET MyServices. While the current status of the project is unclear, it is interesting to note that the wallet portion was to be called a safe-deposit box. While no complete version of Hailstorm is currently available, these naming choices, and the very existence of the product, demonstrate consensus concerning the evolution of C2B from comparison-shopping services to a complex consumer-based set of applications. This expanded view of the wallet more closely matches the domain of electronic commerce, which as described in [4], "involves everything one can do in the physical world: advertising, shopping, bartering, negotiating contracts and prices, bidding for contracts, ordering, billing, payment, settlement, accounting, loans, bonding, escrow, etc."

### **2. Example C2B Scenarios**

The following scenarios help clarify the intended role of C2B systems.

**Mobile Shopper:** A pedestrian is taking a walk through the streets of San Francisco. She passes by many shops and frequently pauses to peer through the windows and examine the various offerings. With so many stores to see, she rarely enters one unless something quite special catches her eye. But today, something different is about to happen. A clothing store just down the street from her current whereabouts recently installed a new C2B system. Sensing the young woman's C2B information on her personal digital assistant (PDA), the store's software requests her identity, and based on it, determines her clothing preferences. Reviewing current inventory, the software finds that the shop is stocking a number of items that seem to fit her profile. The pedestrian is notified of a special sale on these particular items by having a message pop-up on her (PDA). She is given directions to the store from her current location, and soon after becomes a new customer.

**Emergency Room:** A businessman has just fallen ill after a dinner with some potential clientele in New York City. He is far away from his home and his family physician in California. The ambulance picks the gentleman up from his hotel room and rushes him to the emergency room at the nearest hospital. As a result of the food poisoning, the patient is in no condition to fill out insurance forms or to answer questions regarding family history or allergies to various medications. But today, nobody even asks. His identity is established, and authorized hospital personnel retrieve his medical information. In addition, the family doctor on the other coast is notified of his patient's condition. The local doctors prescribe the necessary treatment, and the following morning our businessman is recovering without complications.

**Insuring the Family:** For the Hendersons, the cost of car insurance just seems to keep increasing. Their youngest son has just received his permit to drive, and the rates have soared to new heights. With both parents working, there is really no time to deal with the hassle of calling twenty different insurance companies to find the best rate. But today they won't have to. Wanting a number of changes to their coverage, the Hendersons modify the description of their current policy. The description is not immutable and is created to include a number of equally acceptable alternatives. The utility function representing their requirements is sent onto the Internet and eventually matched with a willing insurer. Using public keys and digital signatures, the Hendersons are signed up for the new policy and receive their insurance cards the following morning.

A review of the scenarios above, as well as a number of others, leads to development of the following set of requirements.

1. Support mobile/non-mobile users in obtaining real-time, highly relevant information and personalized attention.
2. Support mobile/non-mobile users in the maintenance and access of their personal data for a variety of real-life situations.
3. Enable creation of semantically enriched "forms" for automated information extraction/completion from personal information sources.
4. Enable matchmaking between personal requirements and corporate offerings.
5. Enable automated update of personal information at the completion of specified transactions.
6. Enable registration of interest though profile description along with means for anonymous notification of discovered matches.
7. Enable system creation in the absence of well-specified type definitions or well-formulated requirements.
8. Enable automated system evolution in support of new, custom data.

### **3. Stretchability as a Software Quality Measure**

The problems that arise when attempting to design a C2B supportive architecture are best described through the introduction of a new software quality measure called stretchability. Stretchability is best defined as an internal software quality akin to evolvability and reusability but focused primarily on type definitions rather than whole systems or modules. Evolvability is a property attributed to systems designed such that the addition of new functionality can be performed with minimal effort. Reusability is a software quality that refers to systems whose modules or components may be reused with minimal modification to help create an entirely new system.

Stretchability refers to a system's capacity to absorb changes to underlying type definitions and to accept entities of an unknown type definition from external sources. Since the evolution of a system often requires modifications to type definitions, it is clear that stretchability assists in evolvability. Furthermore, since many systems differ little except in their utilization of different domain data, stretchability can be considered a support mechanism for reusability. Finally, and very importantly, it should be noted that stretchability clearly differs from other software qualities in that it involves immediate consideration of external sources. This clearly reflects the intent and importance of this metric as a measure of the suitability of designs and implementations of ubiquitous systems. In closing, it should be mentioned that stretchability is an attribute that can be applied with equal import to process as well as product.

It is our belief that future systems must exhibit ever-increasing degrees of stretchability to succeed in a ubiquitous environment. These new systems will be developed under the constraints of incomplete and unstable requirements and will enable the creation, communication, and externalization of data along with completely supportive semantic interpretations.

Three useful tools can be applied to help clarify the required behavior of a stretchable system. First, a six-faceted requirement (6FR) structure provides a framework that helps classify the requirements of a system based on the facets who, what, where, when, why, and how. To help provide answers for each facet, a number of software specification axioms (SSAs) are defined. Each SSA provides a simple, well-defined, and reusable description of a particular aspect of software design. Finally, the combination of a number of SSAs into a requirements pattern (RP) provides a single semi-formal description of system requirements.

#### **4. The Importance of Content**

It is possible to categorize the intent of inter-agent communication into three major types: data transfer, content transfer, and knowledge transfer, as shown in Table 1. Data transfer is the act of moving data from one agent to another without any prior contract between the sender and receiver regarding the content or meaning of the transferred data. This type of communication is very simple but can still be useful for such tasks as database loading, application logging, and file transfer protocols. It should be immediately apparent that we are only concerning ourselves with negotiation at or above the presentation layer of the OSI model. There may be quite a bit of a priori knowledge involved at the lower levels to ensure that the data are properly transmitted (i.e. packetized, CRCed, etc.) and formatted (i.e. comma separated), but this knowledge is exclusively concerned with form, not meaning or function.

Table 1. SSA-Communication Intent

<b>Value</b>	<b>Description</b>
None	No data will be transferred outside of the agent.
Data-Transfer	Act of moving data from sender to receiver without any prior, direct or indirect contract between the parties regarding content or meaning.
Content-Transfer	Act of moving data from sender to receiver such that the receiver can apply simple conditional logic to discern the context based on an a priori contract with the sender, an a priori contract with a third-party facilitator, and/or a transferred metadata manifest.
Knowledge-Transfer	Act of moving data from sender to the receiver such that receiver can apply simple conditional logic to discern the context based on an a priori contract with the sender, an a priori contract with a third-party facilitator, and/or a transferred metadata manifest.

Moving up the ladder of communication complexity, we arrive at content transfer. Content transfer depends on the ability to perform data transfer but adds the additional requirement of contextual agreement. That is, the sender and the receiver must have previously agreed upon the meaning of the transferred data such that the receiver can determine what data it has received. Furthermore, the receiver should be able to perform simple conditional logic based on the content. Whereas the recipient of a data transfer simply performs a single function upon receipt of the data, the content recipient can pass the data through a state machine and perform varied behaviors depending on what was received. Content transfer has recently become a widely discussed topic, and the extensible Markup Language (XML) was introduced largely to

enable such goals. With a simple, standardized syntax in place, developers using XML are able to spend more time considering the semantics of their communications instead of their format.

Knowledge transfer represents the pinnacle of our simple classification of communications. Just as content transfer required data transfer, so knowledge transfer requires content transfer. This time, however, we expect the recipient not only to accept and process the content, but also to actually know when it has learned something new and proceed to reason about the consequences of the additional knowledge. The consequences may include such high-level notions as the mental state of the sender. The important distinction here is that at least one of the communicating parties is capable of reasoning.

It is important to note that for stretchability it is sufficient to enable content-transfer. The additional requirements implied by knowledge transfer, especially that of mental state, are not necessary to engage in useful communication. In fact, we submit, that if we engender our agents with mental states, then we endanger our prospects for honest, purposeful communications. We further suggest that a considerable amount of human language is dedicated to the purposes of evasion and dishonesty, with secondary importance placed on comforting our kinsmen. The additional purpose of transferring relevant information ranks far lower when measuring the motivations for natural language design. However, in the context of C2B, it is the low-ranked goal of content-transfer that acquires ultimate importance.

What we need then is a new content biased language (CBL) that allows software agents to successfully communicate details regarding their requirements within a transaction. This new language must achieve the following goals.

1. Obtain agreement on the semantic meaning of lexemes.
2. Establish the unique identity of a referent.
3. Represent any concept at any level of granularity.
4. Represent any relationships that may occur between concepts, including relational, non-relational, Cambridge, and comparative relations.
5. Distinguish the use of the same concept in different contexts without always discussing all aspects of the concept and without losing the importance of the specific context.
6. Factor time and object evolution into the model.
7. Construct the scaffolding such that any model of any physical-behavioral unit can be modified and reused, including direct reference from any other model of any other physical-behavioral unit.
8. Enable a means of expressing the rationale or intentions behind an individual's decision to make data accessible.
9. Ensure that the description of intentions with content is orthogonal to the content itself.
10. Ensure that partial content is understood as a subset of total content, and the functionality of an agent is not inexorably halted when presented with partial content.

## **5. Designing a Representation for a Content Biased Language (CBL)**

In general, a language is composed of two distinct components, the vocabulary and the grammar. The vocabulary defines the lexical elements of the language, and the grammar defines the syntactical rules for combining the elements. The semantics implied by certain syntactically valid constructions form yet another important dimension of language. In the case of our content biased language, the vocabulary will be referred to as an External Open Ontological Type System (EOTS). This follows directly from the fact that the language will serve as a type system but also has properties of an ontology.

When an agent receives a request-for-content, it must understand what information the sender is requesting. Likewise, when that agent sends back a reply-with-content, the original sender must understand the response. There are a number of approaches that can be applied to achieve this goal. The immediate solution is to create a standardized content representation vocabulary that all communicating agents must use. Note that a markup language such as XML is not, in and of itself, sufficient to achieve this goal. Simply because an agent can utilize a standard XML parser to extract the data from the XML document, does not imply that the data points have any particular significance. Thus, the problem we are trying to solve is not one of simply parsing out the individual data values within a message, but rather one of comprehending the significance of those values. That significance is conveyed by the semantic layer logically situated atop the EOTS.

The English language, or more accurately an English Language Dictionary, may be considered a repository of words. Each word represents one or more concepts, as described by the definition of the word, and made real by acceptance and use in everyday dialog. In much the same way, the content biased language (CBL) described by the EOOTs can be considered to have an underlying repository of well-known concepts. This dictionary, or Global Type Repository (GTR), represents the complete set of concepts that are globally accepted as parts of the language. The GTR (pronounced Gator) is composed of a set of concept aggregates represented using EOOTs. Based on this description, it should be clear that a CBL is as much defined by its GTR as English is defined by an English dictionary. Furthermore, if two different GTRs were created then two different CBLs would result (just as an English Dictionary defines English and a Spanish Dictionary defines Spanish).

When a new set of concepts needs to be added to a CBL, those concepts will be represented using EOOTs. Before those concepts are registered in the GTR, they are called wild EOOTs. Wild EOOTs are not part of a CBL, since they are not accessible to anyone other than their creator. If the creator of the wild EOOTs wishes to integrate his new concepts into a CBL, he must register his wild EOOTs with the GTR for that CBL. This process is called "Sewing Your Wild EOOTs". In designing a representation for the EOOTs, the following objectives must be considered.

1. Public Standard - no requirement of a global standardizing committee.
2. Open Standard - open specification allowing constituent system designers to freely add new concepts to the language.
3. User-Friendly - minimal thought required for determining the appropriate positioning of the new concept in the existing language structure.
4. Self-Administering - unprompted self-administration through a natural selection mechanism.
5. Explicit Relationships - clear representation of a concept's relationship to other concepts.
6. Atomic Values - normalized, atomic data values unambiguously representing a given concept.
7. Single Source - all constituent systems map their concepts to a central type repository rather than to each other.
8. Extensible References - concepts should be given some means of referencing multiple, distinct concepts when each of those concepts make sense in the context of the concept, even if all referenced concepts are not known beforehand.
9. Differential Definition - some form of content-sharing should be possible but should not be limited to any particular relationship.
10. Contextual Independence - the designation of a concept, as well as its properties, are not affected by the relationships in which the concept participates.
11. Perspective Support - a single concept should support representation from many different perspectives.

## **6. Perspective Domain Graphs: An EOOTs Representation**

An EOOTs model is a software-oriented structure for encoding the communication metadata related to a physical-behavioral unit (PBU), or more specifically a granular partition corresponding to some PBU. A Perspective Domain Graph (PDG) represents a subset of the universe as seen from the perspective of a PBU. The PDG encodes the important details of a PBU in such a way that the details can be easily utilized as topics of discourse. A PDG is a directed acyclic graph (DAG) with four node types.

1. Concept Node - Represents an identifiable entity in the PBU.
2. Relationship Node - Specifies a relationship between sets of concepts.
3. Reference Node - Used to refer to the subjects or concepts that take part in some relationship.
4. Subject Node - Used to express similarity across a set of concepts.

Consider a simple example of car ownership illustrated in Figure 1. In this PDG, two primary concepts are identifiable, the car and the owner. For this example, we will assume the owner is a person, though that need not always be the case. On the right hand side of the figure, a legend clearly designates the different levels of the graph. The source (root) node of a PDG is always a concept node and is therefore situated in a concept level (CL). Following all but the final concept level, there will always appear a Relationship level (RL), Reference level (XL), and Subject level (SL) in that order. As expected, an RL only contains relationship nodes, an XL contains only reference nodes, and an SL contains only subject nodes. Furthermore, the sequence of levels (CL, RL, XL, SL) repeats as many times as necessary to represent all

required concepts. For this particular example, the subject level is empty, but the dotted box (explained later) shows where such nodes would appear.

Edges in the graph are unlabeled. This is because all relationships are expressed as nodes in the relationship levels. In the case of Car-Ownership, there is only one user-defined relationship, <owns>. The <owns> relationship has two reference nodes as children. Each reference node is identified by a name and specifies the content-type (discussed later) of its referent. Thus, the <owns> relationship clearly relates an owner-person to an owned-car. The <ctx> node is a special relationship that is always present. In addition, there is always one child reference node under the <ctx> node for each concept at the following concept level. The child reference nodes of <ctx> have no name and no referent concept-type. The <ctx> node is best explained by equating it to the phrase “in the context of”. Thus, <person> in the context of <Car-Ownership> accurately signifies the <person> node in the graph.

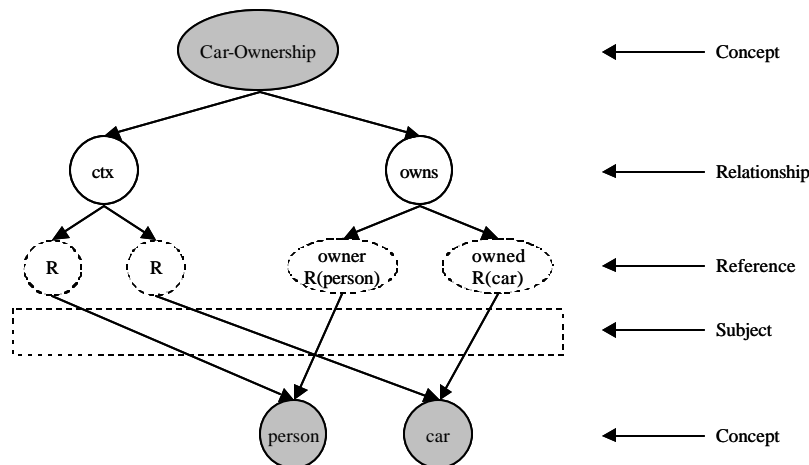


Figure 1. A Perspective Domain Graph.

## 7. The Content Oriented Architecture

It is useful to introduce the concept of a global system. A global system is not developed by any single entity; it is instead, a conglomeration of the efforts of many autonomously functioning entities and their implicit interactions. Another useful concept is that of a constituent system, which could be considered nothing more than a set of components housed within the framework of the global system. In general, a constituent system is written by a single entity. If multiple entities are involved in the development effort, it is assumed they share close communication and common goals. The constituent system itself will be composed of many components (modules) only a subset of which will interact outside the context of the constituent system (with other components of the global system). The remaining components will serve to provide the functionality we presently associate with enterprise-based software. Thus, a constituent system really has two separate sets of requirements. The first set defines the business (or personal) needs of the owner, while the second defines the level of stretchability supported to enable interaction within the global system. Somewhere in the middle there may also be a translation layer used to marshal information in and out.

Applications built using SOAP, WSDL, and UDDI often conform to what is called the Service-Oriented Architecture (SOA). As illustrated in Figure 2a, a system based on SOA has three major components: the service registry, the service consumer, and the service provider. The service registry is usually realized as a UDDI operator node. Service providers will register WSDL descriptions of their services with the service registry. Later on, service consumers will discover these services by performing searches on the UDDI operator cloud. Once a service is located, the service consumer will extract the WSDL service description, including both the service interface and service implementation, and use it to generate code for accessing the service. In almost all cases, the code generation is performed at design time. This is necessary since the service description requires a human intellect to understand the required message parameters and ensure that the service consumer’s code provides the appropriate arguments to the service provider’s interface.

Any changes to the service provider's interface will require, in addition to re-registration with the service registry, a recompile (or redesign) at each service consumer's site.

To combat both the design time dependence and interface fragility problems inherent in SOA, we propose the Content-Oriented Architecture (COA) illustrated in Figure 2b. The similarities between SOA and COA are not accidental. SOA, and the technologies used to support it, form the foundation for COA. In COA, the service registry has been replaced with a matchmaker. In addition, a new component, the Global Type Repository, has been added. The GTR will contain the specification of a content biased language (CBL).

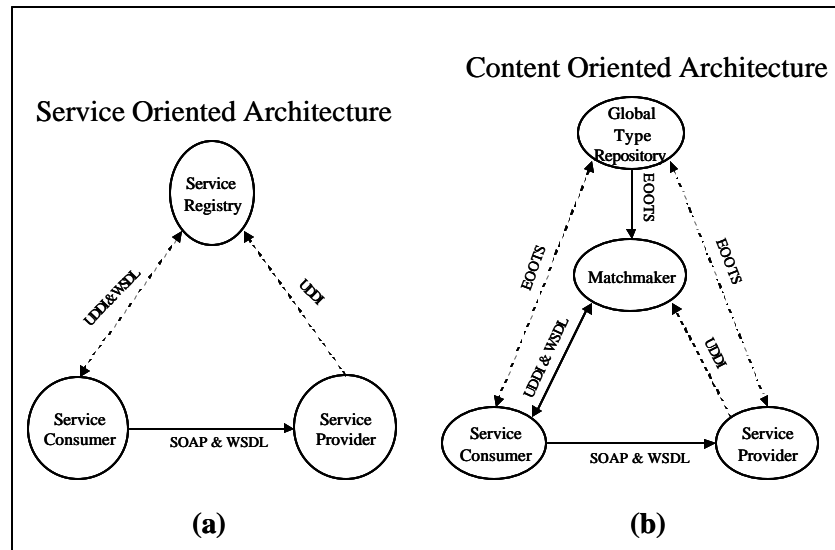


Figure 2. (a) Service-Oriented Architecture, and (b) Content-Oriented Architecture.

The service consumer and service provider have been replaced with peer-to-peer constituent system agents. This supports an important goal of C2B, which is placing the consumer on equal footing with the business. When the constituent system agents in COA wish to communicate, they do so using the CBL contained in the GTR. If the two agents speak different dialects, then messages can be routed through the matchmaker. Using cross-dialect mappings (PDG mappings) from the GTR, the matchmaker can provide a translation service to the agents to increase the probability of a successful communication.

Notice that some lines in Figures 2a and Figure 2b are dotted, while others are solid. The dotted lines represent activities that occur at design time, while the solid lines represent run-time actions. Through the addition of the GTR, it is now possible to make all activities supported by SOA function at run-time. In this way, COA provides a top layer that is currently missing in the SOA hierarchy. The comparison is illustrated in Figure 3. The acronym CDL represents a technology called content description language, which is an XML-based representation of a PDG.

One final important point concerns the style of interfaces in COA. Whereas an SOA interface is generally composed of arbitrary, multiple argument method calls, a COA interface could consist of a fixed set of well-known single argument method calls. A good analogy is human communication. A human has only one set of ears (one interface) that handles all audible communication. Once the communication has entered the brain (agent), it is dispatched to the appropriate handler based on the message content. Thus the COA, and its associated CBL, clearly represent a more flexible, anthropomorphic model of software design. Now, if an agent design is changed such that it requires different information (different arguments), the calling agent does not need to be recompiled. Instead, the new callee version simply asks the caller for the additional information. Given the existence of a CBL, this request is unambiguous, and may succeed though it had no previous precedent. Furthermore, if the caller cannot immediately supply the additional information, an ancillary GUI agent could be spawned to solicit help from a human operator.

<b>SOA Interop Stack</b>	<b>Undefined</b>	<b>Content Biased Communication (CDL)</b>	<b>COA Interop Stack</b>
	UDDI	GTR and Matchmakers	
	Endpoint Access Protocol (SOAP)		
	eXtensible Markup Language (XML)		
	Internet Protocols (HTTP, TCP/IP)		

Figure 3. Interop Stacks: Adapted from the UDDI Technical White Paper [9].

## 8. Conclusions

The natural languages employed in human communications represent the pinnacle of flexibility and extensibility for the transfer of information. Unfortunately, they are also often wrought with problems of ambiguity and misinterpretation. Using natural language as a basis, and remaining keenly aware of its potential pitfalls, we have described a new communication model based on what we have termed content biased language (CBL). A CBL consists of a well-defined set of concepts and is capable of expressing relationships and properties of those concepts. The representation of a CBL as a set of EOOTs in a GTR enables extensibility and also ensures flexibility as different dialects can be used interchangeably through the creation of appropriate mappings.

The requirements of a C2B system could be satisfied with nothing less than the flexibility of content biased communication. The sheer number of potential subject areas, and the multitude of possible, and meaningful communication patterns, made it absolutely mandatory to establish a more flexible, more stretchable form of content transfer. This notion of stretchability, and in particular its implications for content transfer, provided the primary motivation for the ideas presented in this paper. Once this underlying problem was identified, the enabling research on perspective domain graphs, external open ontological type systems, and ultimately the content oriented architecture could be developed. The introduction of the content oriented architecture as the culmination of the research helped to clearly relate the many facets of the consumer-to-business problem and also served to situate the topic clearly in the realm of cutting-edge computer science research. The business of eCommerce is the business of the future, and the foundation of eCommerce will be stretchable COA based systems.

## References

- [1] Brown, K., & Taylor, W., "Inktomi Acquires Online Shopping Developer C2B Technologies," <http://www.inktom.com/new/press/1998/c2b.html>, 1998.
- [2] ECML.org., "Electronic Commerce Modeling Language: Wallet/Merchant Test Form," [http://www.ecml.org/wallet\\_test.html](http://www.ecml.org/wallet_test.html)
- [3] Frantz, L., "Easy Money," UPSIDE Today, Upside Media, June 14, 1999.
- [4] Honeyman, P., "Digest of the First USENIX Workshop on Electronic Commerce (EC 95)," New York, New York, <http://www.usenix.org/publications/library/proceedings/ec95/digest.html>, July 11-12, 1995.
- [5] Livraghi, G., "B2B B2C C2C C2B," *off-line*, <http://www.gandalf.it/offline/off26-en.htm>, May 2000.
- [6] Power, C., "Digital Frontiers Electronic Commerce: MasterCard, Visa, and Amex Spearhead Standard Format For Digital Wallet Software," *American Banker*, pg. 19, 1999.
- [7] Smith, B., "Mereotopology: A Theory of Parts and Boundaries," *Data and Knowledge Engineering*, 20, 287—303, 1996.
- [8] Smith, B., "Objects and their Environments: From Aristotle to Ecological Ontology," Department of Philosophy, Center for Cognitive Science and National Center for Geographic Information and Analysis, University at Buffalo. [http://ontology.buffalo.edu/smith/articles/napflion.html#N\\_1\\_](http://ontology.buffalo.edu/smith/articles/napflion.html#N_1_)
- [9] UDDI.org., "UDDI Technical White Paper," [http://www.uddi.org/pubs/Iru\\_UDDI\\_Technical\\_White\\_Paper.pdf](http://www.uddi.org/pubs/Iru_UDDI_Technical_White_Paper.pdf), Sep 6, 2000.