Despite the malaise presently affecting dot-com businesses, the ubiquity of the Internet will inevitably force the next generation of organizations to move away from their heritage legacy systems with deeply embedded business workflows and organize themselves into virtual enterprises. Virtual enterprises potentially shorten delivery times, increase product quality, deliver personalized products and services, and decrease transaction costs. Even more importantly, virtual enterprises gracefully accommodate short-term trading relationships, which can be as brief as a single business transaction.

This new paradigm of conducting business puts severe demands on both the extensibility and adaptability of enterprise information systems (EIS) infrastructures, and forces many brick-and-mortar organizations to shift from tightly interwoven and coarse-grained business applications to more flexible and loosely coupled ones. The new wave of enterprise systems should therefore allow dynamic composition of fine-grained business services into reinvented assemblies, in ways that previously could not be predicted in advance.

Web service technology is currently being touted as the ideal solution to meet the previously mentioned requirements for the dynamic composition of EISs [12]. Web services can be loosely defined as self-describing software components accessible through the Internet that support business processes [7]. Offered by multiple providers, Web services are usually published in a central repository with a brokering mechanism. The service-oriented computing paradigm migrates the traditional standalone hosting model to a networked one, by allowing Web services to dynamically discover and hook up to Web services offered by different providers. In this way, Web services can be blended to implement customized
virtual supply chains or smaller-scale business processes confined to a single enterprise. Existing Web service standards such as UDDI [8] and ebXML, however, only provide a first step toward achieving full EIS integration. Fundamental shortcomings of Web services include the following:

- Business process dynamics and nonfunctional properties of service-enabled processes are poorly addressed by existing service description languages and Web service flow languages. These languages seem to target service signatures and signature interactions only.
- Current standards do not put forth a methodology to assist designers in building Web services on top of legacy assets.
- Web service frameworks suffer from an object bias, neglecting the most essential aspect of B2B transactions: communication. Without communication, organizations cannot collaborate in an efficient way, assuming all participants communicate using the same ontology.

In order to overcome these shortcomings, we suggest leveraging Web services with techniques borrowed from other research domains, such as business object component technology (BOCT) and distributed artificial intelligence (DAI). Our aim is to create new types of Web services known as intelligent Web services (IWSs). In this article, we explain how to design IWSs, using a fictive scenario to illustrate how they work. We will discuss a business process redesign scenario for integrating shared business processes between a PC manufacturer and a semiconductor (integrated circuit board) manufacturer.

The computer hardware (subcomponent and PC) industry is under continuous pressure to deliver qualitatively superior products. To cope with these competitive forces, many manufacturers outsource part of their production operations, such as chip and motherboard production, and limit themselves to the core business: composition of computer configurations into various designs that can be tailored to individual customers. Manufacturers can now focus on what they do best, and let their partners do the rest. Outsourcing not only requires smooth internal business processes, but demands close partnerships with suppliers throughout the supply chain.

The new business model facilitates short-term, fluid relationships between customers and suppliers, so agreements can be renegotiated as new market conditions emerge. The PC manufacturer can search a centralized, international service UDDI repository for component suppliers matching its requirements. After the PC manufacturer has selected a trading partner from the list of available integrated circuit board (IC) manufacturers, a partner agreement recording the trading protocol must be stipulated. For example, the PC manufacturer may agree with the component supplier that he will deliver components for a period of three months on the basis of a release order contract. A release order contract specifies sales volume, delivery information, product quality, and price range. For example, chip AB234 may be procured every two months, with a one-day delivery time, for a price of $10. In this way, the contract governs the procurement process at the site of the semiconductor manufacturer, as well as the corresponding payment processes of the PC manufacturer.

Our business collaboration scenario requires the composition of several Web services from two different organizations. Leveraging these Web services with techniques from other research domains, such as DAI, offers organizations more flexibility and adaptability in carrying out their daily operations [5]. DAI, which includes distributed problem resolution and multi-agent system domains, aims to study issues related to the distribution and coordination of knowledge and actions in environments involving multiple entities called agents [3]. IWSs can benefit from agent characteristics such as autonomy, adaptability, and sociability. For instance, IWSs can cooperate according to specific negotiable agreements, collaborate to achieve complex operations, and commit themselves according to their workloads. Such approaches are crucial for proactively dealing with business change, allowing IWSs...
to dynamically adapt and renegotiate their trading protocols, as represented in BPEL, WSFL, or BPML. In the remainder of this article, we consider IWSs as components that act on the environment, as well as react to environmental stimuli.

When multiple IWSs must evolve in the same environment, it is important to define their behavior in order to regulate their interactions. Their behaviors may emerge as a result of their interactions with each other and/or with the external environment. Therefore, the following questions must be answered: How can we design collaborative IWSs? What types of information can these IWSs exchange? What types of infrastructure can support long-running business conversations? And how can these IWSs be associated with common goals?

Techniques borrowed from DAI provide answers to these questions, and result in the following equation: $\text{IWS} = \text{collaboration} + \text{adaptability} + \text{communication}$. Collaboration between IWSs can rely on contract-based coordination mechanisms such as scheduling using a task list, for example. Hence, the main characteristic IWSs should exhibit is their ability to collaborate by communicating, despite their differences.

As with business objects and components, an IWS encapsulates data and methods in order to provide a service to a requesting IWS. Methods specify the capabilities of an IWS in terms of result types, required input data, and type of input data. This assumes the designer is aware of all situations in which its IWS will be involved. As a result of their static behavior, Web services do not have the ability to dynamically adapt their operations according to current states and environmental characteristics. In contrast, the second characteristic IWSs should exhibit is the ability to adapt their behavior, according to the environments in which they evolve. This ability, enabling IWSs to decide with whom to collaborate, what operations to offer, what operations to request, and what behaviors to display requires that IWSs be equipped with some introspection and automatic adoption mechanisms.

When an IWS collaborates with other IWSs, it initially must identify and locate these IWSs. In particular, IWSs must discover each other based on the operations they provide and require. These operations are subject to mutual agreements that must be monitored, and possibly renegotiated. Therefore, the third characteristic of IWSs is their ability to communicate mutual agreements about their needs as consumers, and their capabilities as providers. This process should be mediated using contracts describing the commitments of IWSs according to their abilities.

**A Framework for Intelligent Web Services**

IWSs need not be developed from scratch. They can be drawn from existing information repositories and applications, and combined according to specific business rules and within specific business contexts. Through this approach, internetworked EISs shift from a passive role to more proactive one. Playing a proactive role, systems can make their own decisions according to functionality, ability, and availability. To achieve this, the behavior of an IWS can be captured in electronic contracts, denoted by trading partner agreements (TPAs) that serve as the basis for negotiations between IWSs, addressing factors such as the terms of delivery. The architecture of the IWS serving as the foundation for composition is represented in Figure 1.

The proposed framework has been borrowed from the domain of BOCT and is inherently object-oriented, with objects residing in one layer relying on objects residing at lower levels for functionality. The IWS framework promotes future reuse and dynamic modification of components at all levels of the architecture, by supporting the principles of separation of concerns, loose coupling, and dynamic binding in combination with interface evolution. The latter principle allows components at all levels to evolve, without affecting their implementation. The following are descriptions of the four layers illustrated in Figure 1, starting with the lowest layer. IWSs run on top of the highest layer, which is the business workflow layer.

**Distributed Object Layer.** This layer, which sits above the physical network, facilitates low-level interoperability functionality, such as ACID-compliant transactional support, concurrency, message brokering, load balancing, life-cycle management, and security facilities. Distributed objects facilitate communication between distributed applications, independent of their address space and implementation language. These objects can objectify legacy systems, offering them as self-describing components to the higher-order layers. In addition, this layer addresses semantic interoperability issues by defining domain-dependent ontologies that enormously ease the development and reuse of business objects and workflow applications. Distributed objects can be built upon existing backbone protocols such as CORBA, or the more lightweight SOAP [8].

**Business Entity Layer.** We envision that business objects are the ideal implementation technology for IWSs, as they provide a natural way to describe appli-
cation-independent concepts such as product and release order. Business objects play a central role in capturing the semantics of actual business entities and processes, in a way that is easily understandable [6]. Business entities serve as persistent data containers to store the state of business processes, tasks, and workflows deployed on top of this layer. This category of business objects tends to evolve slowly, and forms the foundation of the integrated enterprise. The business methods encapsulated in the business entities serve to manipulate "hidden" enterprise data and are transactional in nature by supporting the two-phase commit protocol.

The interface of business entities, as well as the higher-order task and workflow objects, should be textually specified with some standardized interface description language, such as IDL or WSDL. For reasons of simplicity, we exemplify this process with the Component Definition Language (CDL) developed by the Business Object Domain Task Force (BODTF) within the OMG, but in principle any language can be applied here. CDL is a declarative language to specify the services of collections of business objects, their relations, dynamics, business constraints, and attributes. Business objects are not written in CDL, but in programming models for which language mappings are available, such as Enterprise JavaBeans. This specification language extends IDL by adding several high-level constructs to capture more business semantics. The CDL excerpt in Figure 2a illustrates a supplier business entity associated with the Release_Order_doc business entity.

**Business Task Layer.** Business tasks comprise a set of interrelated activities of relatively short duration that collectively produce a useful artifact, which can be reused within the business process, according to a set of prespecified policies. Business tasks are initiated on the basis of an incoming event, such as a customer requesting a product delivery, and result in an outgoing event, such as notification of product shipment. Business tasks rely on business entities, as illustrated in Figure 1, to perform their operations and store intermediate results. Business task objects are transactional in the sense that they are atomic, result in a consistent end-state, are able to execute independently, and can survive system failures. This transactional support is essential for constructing scalable, distributed business workflows.

**Business Workflow Layer.** Objects in this layer assign business tasks to roles according to the state of each business task in progress, and move the process forward from one task-performing role to the next. Workflow-enabled business processes can track transactions across departments and even across enterprise boundaries. The workflow layer essentially prescribes the sequence of internal business activities, arranges for the delivery of work to the appropriate inter-organizational resources, tracks the status of business activities, and coordinates the information flow of inter- and intra-organizational activities. Proper deployment of business workflow requires scripting business task objects into transactional workflow objects while relaxing traditional ACID properties using workflow compensation and atomic spheres.

The CDL excerpt shown in Figure 2b specifies the componentShipments workflow that handles the delivery process of the IC manufacturer to the PC Manufacturer. The workflow is initiated by an external event init_component_shipment; this event will be linked to the contract, described later. The state transition rules (lot_shipment and credit_limit) specify how the workflow shifts from one business task, such as process_release_order, to the next, such as check_credit_limit.

---

**Figure 2a.** A CDL excerpt of the supplier business entity.

```csharp
@Entity
		description of the entity object "Supplier"
	static aspects
relationship theSupplier;  
attribute Int supplierID, String supplierType, Dollar creditLimit, ContactInfo contact_information;  
// dynamic aspects
void createSupplier(in Int supplierID, in String supplierType, in Dollar creditLimit, in Bag location);  
void removeSupplier(in Int supplierID, in String supplierType, in Dollar creditLimit, in Bag location);  
}; # end entity Supplier
```

**Figure 2b.** A CDL excerpt of the delivery process workflow.

```csharp
// CDL-Descriptions of the  business workflow object
BusinessWorkflow componentShipments {  
	cdl description of the workflow *componentShipments*  
	// static aspects
attribute ActiveProcess active_task, Actor actor;  
attribute Int purchaseOrderNumber, Int quantity, Price price,  
Date deliveryDate, Date releaseDate, Dollar deliveryCosts;  
// dynamic aspects
state{initial, process_release_order, check_credit_limit,  
pick_order, ship_order}  
apply StateTransitionRule lot_shipment{  
trigger=init_component_shipment
source=initial;  
target=process_release_order  
};  
apply StateTransitionRule credit_limit {  
trigger = release_order_finished();  
source = process_release_order;  
target = check_credit_limit;  
}; # end BW CSP
```
Intelligent Web Services. Business objects are helpful for integrating internal business processes in enterprises. However, business objects cannot proactively engage in cross-organizational business transactions and act in goal-oriented ways. In addition, business objects do not survive in heterogeneous system environments with loosely coupled EISs. Proactive behavior assumes some built-in intelligence in business objects, thereby allocating this technology to a higher level: that of IWSs. An IWS can be loosely defined as a software component that considers its internal state as well as the state of its environment before it performs any business-related activities. As stated earlier, an IWS should at least display collaboration, adaptability, and communication features via the following goal-oriented behaviors:

- Collaborating with other IWSs to achieve complex and long-running business transactions. Such collaboration can occur at the business workflow layer, in which different processes must be integrated into a common workflow.
- Adapting its behavior in case collaboration commitments from other IWSs are not honored. This means reviewing the processes to be executed according to the requirements of the new situation.
- Communicating with other IWSs in case it lacks resources or requires the capabilities of these IWSs. Such communication can be achieved through the distributed object layer.

Collaboration, adaptability, and communication features are present in our descriptive scenario. The PC-assembler company, represented by its IWS, must communicate its needs to potential providers. Then the IWS must collaborate with these providers to define what is expected from them and how they can complete it. Finally, the IWS may have to adapt its working mechanisms and goals in terms of its commitments in case unexpected events occur, such as poor provider delivery times. IWSs will thus trade with each other at some e-marketplace on behalf of their physical counterparts, and can be constructed on top of emerging Web service and distributed object technology.

Trading Partner Agreements
IWSs tend to give a rather biased perspective of an electronic transaction, with the analyst taking the viewpoint of either the buyer or the seller. Since business transactions are in fact exchange processes, we have defined contracts to represent this reciprocity. Contracts can be orchestrated as two interwoven intelligent Web services, such as a customer IWS requesting a product and a supplier IWS requesting compensation in return. Depending on the trust between parties engaged in an e-commerce relationship, they use various contracts while conducting a business transaction. IWSs would respectively represent customers and providers.

Even at a higher level, contracts can be organized in scenarios that denote a sequence of business transactions in a supply chain. Generally speaking, e-business scenarios encompass the following three phases: information, negotiation, and fulfillment. During the information phase, prospective buyers identify and evaluate their requirements as well as alternative sellers to fulfill them. The sellers offer their products and services to the potential customers by means of an IWS catalog. Next, the prospective customers and sellers negotiate mutual terms and conditions, which are represented in contracts. Contracts, known as trading partner agreements (TPAs), mandate the shared goals and policies of the virtual enterprise. Moreover, TPAs capture the communication aspects, and thus the authorizations and obligations, regarding business protocols between trading partners engaged in a business transaction. They also support long-running conversations between interacting trading partners. Eventually, the contract is executed within the prescribed period, and the objects of the transaction are exchanged according to the conditions of the contract. This last phase typically consists of fulfillment of goods or services, and payment from the customer to the seller.

In the framework we propose, the TPA captures critical information to coordinate execution of the IWSs, and the underlying business entity, task, and workflow objects in terms of goals, deadlines, and exception scenarios. TPAs represent the shared goals IWSs pursue while engaged in a business transaction. These goals are achieved by the interacting IWSs when all mutual commitments are met. Commitments are described with deontic clauses, incorporating messages that are in turn linked to the signature of underlying business objects.

In addition, contracts record specific technical parameters for conducting electronic business, such as visibility of entities or task objects, middleware technology in use, network address, encryption protocol, identity, and communication protocol. To meet these requirements for contracts, we rely on the XML Language for Business Communication (XLBC) [10], which represents messages as a sequence of speech acts. In order to facilitate future reuse, the contracts are organized in a layered archi-
tecture, ranging from low-level, communicational building blocks, or speech acts, to transactions and contracts.

Speech acts define what actions people take while communicating [1]. They represent a performative act like a request or a promise, or a commit, which in itself constitutes an action. A message can contain more than one speech act. IWSs can initiate performative actions, such as a workflow, or a single business task represented as a set of obligations and authorizations. In this way, IWSs can incur or fulfill an obligation to another IWS, and require authorization from other IWSs to perform an action. Speech acts usually only have meaning when they come in pairs, such as a speech act incorporating a request followed by another speech act signifying a commit. We call these pairs of messages a transaction1. For a detailed description of the stratified formal business communication language, see [11].

Figure 3 is an example of an XLBC contract, called Release_Order_Contract, between a PC manufacturer such as IBM and one of its main part suppliers, Philips-Conductors. This release order contract is a special category of sales document that states fixed delivery conditions, such as the number of parts procured in a period. The sample contract describes the obligation of Philips-Conductors to deliver ICs by executing the componentShipment workflow (see Figure 3, service w003), as well as the obligation of IBM to pay for this (service w004).

In the example, the ACCOUNT-PAYMENT service is only mentioned, but not elaborated. This could be sufficient if this service is a reusable Web service defined and stored elsewhere, such as in a UDDI service repository system. The RELEASE_ORDER service is defined in more detail but we have abbreviated it due to space constraints. Of the various states, only one is worked out: the state in which there exists an obligation for Philips-Conductors to deliver motherboards within five days. During this deontic state, invoking the internal workflow componentShipment (see Figure 2b) fulfills the obligation of the supplier. This workflow is initiated by execution of the transaction “t1”, and terminated by either transaction “t2” or “t3” depending on the evaluation of its result.

The contract thus shows the individual workflows with deontic statements that control behavior by linking the event mechanism of the workflow objects to the transactions. These transactions are goal-oriented. In fact, the workflow is regarded as a holistic entity, which retains its own control. This assumption is acceptable in practice, as many organizations prefer to preserve autonomy and control over their internal business processes, especially in the case of marginal trust between the trading partners. The security section of the contract specifies visibility of business objects and workflows, but could also incorporate other information, such as security protocol (SSL) details.

**Conclusion**

We have proposed a framework for contract-based support to establish virtual collaboration using loosely coupled and heterogeneous intelligent Web services. Contracts encapsulate the control information for IWSs engaged in e-business transactions. Recently, several similar initiatives have been launched to introduce contracts as a means to script workflows into an integrated value chain. A recent example is the CrossFlow project [9], and IBM’s TPA [2]. However, these contract specification lan-
guages are equipped with less business semantics, and are not typically linked to a business object component framework. In the domain of Web service technology, initiatives such as BPEL and WSFL [4] only focus on low-level plumbing of Web service interfaces while disregarding complex business interactions in terms of mutual commitments.

IWS technology is still in its infancy. Many important issues must be addressed before “agentified” Web services can be successfully deployed at e-marketplaces, such as the integration of IWSs with wrapped legacy systems, the semantic integration of Web services, the choreography of Web services into e-commerce transactions while relaxing traditional ACIDity, and legal issues such as those surrounding mobile IWS technology.

REFERENCES

Willem-Jan Van Den Heuvel (wjheuvel@kub.nl) is an assistant professor in the Infolab at Tilburg University, The Netherlands.
Zakaria Maamar (zakaria.maamar@zu.ac.ae) is an associate professor in the college of information systems at Zayed University, Dubai, U.A.E.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

© 2003 ACM 0002-0782/03/1000 $5.00