TRADITIONALLY, SOFTWARE HAS BEEN PERCEIVED, developed, and delivered as a product. However, the Internet has created a recent surge of interest in developing software as a service delivered and consumed on demand. The benefit of this approach is seen in the looser coupling between business problems and the associated software solutions—hence the increased ability to evolve systems as business requirements change.

This shift toward a service-oriented model of software brings about a number of benefits as well as research challenges. The concept of delivering software as a service is relatively simple: “Do not buy software, simply use it as and when you need it.” Putting such a concept into practice, however, is far more complex and involves many issues. In this article, we address the questions: What are the characteristics of a market for software services? What are the characteristics of negotiating in such an environment? The work presented here is based upon ongoing research into service-oriented software and is designed to form a framework for the next generation of service-based software moving far beyond turnkey systems, outsourcing, ASPs, and current Web services technology.

The decision to buy or rent software usually depends on the characteristics of the needs and the way allocating available resources are chosen. For example, it is more likely that one would rent a car if the need is just temporary or the up-front cost is better allocated elsewhere. Delivering software as a service addresses customers with temporary or rapidly changing needs, where software can be rented over the Internet, composed to deliver the required outcome, and then discarded.

Logical separation of need from the need-fulfillment mechanism is at the heart of the service model. For instance, the different activities run by a bank could be considered as different services run by different commercial bodies and then combined to provide the customer with a certain service [9]. Customer credit analysis, money transfer, fund management, and Internet banking could all be done by different entities at the back end. The final service received by the customer is a combination of a number of these services.

A service-oriented model of software is one in which software functionality is delivered as a service, where each time functionality is required, service elements are identified, terms and conditions are negotiated, executed, and then discarded [2, 4]. This concept of “discarding” is crucial to the service model since it implies no further obligation to that
particular solution, either on the part of the consumer or the provider (unless, of course, they have chosen to negotiate otherwise). It is this concept that differentiates “service” from “product” and represents the special case of all service models of “use once only” and select a different solution next time.

Several business and technological drivers are converging to enable the software industry to move toward a service-oriented model of development and delivery. On the business side, providers have the potential of reduced unit costs arising from broader market reach. Also, the practice of outsourcing tasks that are not part of a company’s core competencies enables businesses to reduce up-front capital investment and to obtain market solutions faster [1].

On the technological side, the need for service-oriented models of software is motivated by both the inadequacy of current models to achieve software flexibility and the failure to exploit the potential of Internet-based computing to its logical end, that is, a full flexible, distributed software solution [2]. Examples include the limitations of configuration management and version control inflexibility.

**Negotiation as a means of tailoring software needs.** Applications are traditionally specified and implemented in terms of a single set of functional requirements. These requirements are generally uniform for all users (whether individual users or software entities). On the contrary, in a service-oriented model, the same service may be delivered with different qualities, prices, or delivery conditions to accommodate different modes of use.

In highly dynamic environments where the forces of supply and demand are continuously changing, fixing the price or any aspects of the service may not be possible. In such environments, the only way to receive a service is to negotiate what is possible at the time of the request. In other words, automated negotiation will be a form of interaction aimed at tailoring needs dynamically.

**Characteristics of the software services market.** A market is a place where buyers and sellers congregate to exchange values (goods or services). The basic elements of a market (whether digital or physical) are the participants, the exchanged commodities, and the rules for interaction. There is no one definition of each of these basic elements, but rather each market would have a different combination of attributes for its basic elements.

**The market is the cornerstone of the service-oriented model of software; it is the place where service providers, consumers, and third parties (such as brokers) can advertise the services they provide, request and negotiate service contracts, and execute contracts.** The domain characteristics of a software services market can be summarized in the following six key issues:

*Scale.* In a service-oriented model of software provision, granularity is a critical issue, which requires individual service elements to be sufficiently small to allow tailoring of fine-grained services to meet the specific needs of consumers [3]. Thus, the software services market is potentially very large, characterized by a large number of small services. And for a consumer to obtain “the best” deal may mean negotiating with all possible providers of the required service. However, from an implementation efficiency perspective, obtaining a good deal may be satisfactory.

*Criticality of time.* In an extreme service-oriented model, services are negotiated with each request. This implies that negotiations must be brought to a conclusion during a very short time frame. However, for the sake of implementation efficiency considerations, such an assumption may be relaxed by making earlier bindings. An example of an early binding is one in which a contract covers a number of future uses of a service where services are “called off” within a contract framework.

*Provision of services in a supply chain style.* The service-oriented model creates the opportunity for providing value-added services, which are delivered by composing services. In other words, software will be supplied through a supply chain. The length of the supply chain may be as short as a single provider offering a simple service to a consumer, or it may be a network of interrelated services. For instance, a number of intermediary providers who offer value-

---

**THE ESSENCE OF SERVICE**

A service is defined as: “Any act or performance that one party can offer to another that is essentially intangible and does not result in the ownership of anything. Its production may or may not be tied to a physical product” [8]. **The essence of this definition is that the main interest in a service is its output, not the means by which the output is generated. This implies that the means of producing the service may change as long as the received output is satisfactory.**
added services may exist between the original provider of a service and the ultimate consumer. A case in point is a travel agent who provides a travel planning service by combining individual services such as flight reservation, hotel booking, and car rental services; where each of these individual services may be obtained from the original provider or from another intermediary provider.

**Imperfect knowledge.** In a very large market, negotiators (providers or consumers) are unlikely to have perfect knowledge about other parties or the environment; negotiators may not have a history of interaction about each other. The scale of such a market may also prevent the acquisition of perfect knowledge. This requires formulation of beliefs about trust. Such formulation may be done internally by the negotiator or may be shifted along the supply chain to be carried out by an independent body as a service, such as the certification performed by a credit reference agency on a new credit card customer.

**Volatility.** As service providers and consumers freely join and leave a market, the state of the market will continuously change. This influences the forces of supply and demand. The price of a service will be affected by the status of the market at any point in time, such as in the stock market. For instance, at time \( t_1 \) a provider of service \( S \) may survey the market for the availability and the price of other providers, and according to the information gathered he or she would be able to determine the price.

Consumers must also deal with the possibility that services might not be available all the time; they may not be able to establish point-to-point connections. A consumer might not be aware of which providers are available at any given moment; this creates a need for a third party acting as a directory for negotiators to find their partners. Also, negotiating a complex service in such an environment involves dealing with uncertainty. For instance, dynamic composition of a service through negotiation may fail as a result of a failure to contract one of the individual services comprising the complex service.

**Service differentiation.** In mature markets, it is often the case that providers base their service differentiation policies on a number of quality attributes rather than on just one attribute such as price. This requirement of multiple parameters must be accommodated in whatever negotiation protocol is selected. For instance, protocols that allow the negotiation of just one attribute only may not be as suitable as those that allow multiple attributes.

### Negotiating Software Services

Having established the key characteristics of a software services marketplace, we now turn to the negotiation process itself. Negotiating in a service-oriented environment can be seen as a three-phase process: Pre-negotiation, negotiation, and service delivery [4], (as illustrated by this figure).

**Pre-negotiation.** The pre-negotiation phase is concerned with producing the starting information for the main negotiation phase, within which there are three basic issues.

- **Service selection.** In a market for software services, service offerings may vary in their combination of attributes (functional and nonfunctional). While there may be differences in functional offerings, we allow these to be dealt with in the selection process, assuming all negotiation is based on services with equal functionality and thus concentrate on the problematic issue of nonfunctional differences.

  The issue is how can a number of offers of the same functionality, but with different combination of nonfunctional attributes, be evaluated and compared? One way to achieve this is to use a weighted utility function where each attribute value is assigned a certain arbitrary utility value and then each utility value is multiplied by a weight factor that denotes the importance of such attribute. All weights must add up to 1.

  In some situations it may not be possible to draw a utility graph and plot utility values corresponding to values of nonfunctional attributes. It may be more relevant to consider choices as packages of desirable and less desirable combinations of attributes [4]. Such cases arise when there is one critical attribute where its value is considered the dominant factor for choosing a combination of nonfunctional attributes. Accepting a combination of nonfunctional attributes is dependent on the value of the critical attribute.

  Consider a service that transmits telephone calls over the Internet. For simplicity, consider the nonfunctional characteristics of such a service are price, voice quality (measured in the number of lost pack-
Several business and technological drivers are converging to enable the software industry to move toward a service-oriented model of development and delivery.

ets), and reliability (measured in the number of times the service is unavailable per week). In this example, if the price is the determinant factor, then any decision to accept a combination of the other two quality attributes will be based on the value of the price attribute. A consumer willing to pay a high price per minute will link such a price to a maximum acceptable value of lost packets and a maximum value for service failure per week. It is worth noting the critical attribute may be described as a combination of a number of other attributes.

Provider selection. In the potentially very large market, each service invocation may involve a new provider. However, for the sake of implementation efficiency (search efficiency), a consumer may keep a short list of preferred providers. Such a list is updated continuously based on the feedback gained from both the negotiation phase and the delivery phase [4].

Predicting service usage. In a service-based application where most or all of the functionality is delivered as services, negotiation results in a high communication overhead and therefore can greatly affect performance. Among other approaches to address implementation efficiency problems, it is possible to predict the way software is used and to create temporary contracts that can be signed and enacted if and when the considered service is invoked. For example, in a service-based word processor, the application may proactively contact a number of spellcheck service providers and negotiate terms and conditions for such service as soon as the user starts editing a document. When the user invokes the service “spellcheck” a chosen contract is signed and enacted.

Negotiation phase. In its simplest form, automated negotiation can be regarded as the exchange of messages between negotiating parties with the aim of reaching an agreement. The term agent is used here in its broadest sense to describe any participant in the interaction.

The negotiating agents exchange a number of contract templates until an agreement over a contract whose values are acceptable by both parties is reached. Each contract contains a description of the negotiated service in addition to a number of business conditions. The interaction is governed by the rules of a negotiation protocol.

An agent is a negotiation participant composed of a collection of negotiation capabilities (roles), plans (strategies), and needs (services). Roles are collections of negotiation competencies that comply with a specific protocol. While an agent can play a number of roles such as a provider, consumer, or arbitrator, an instance of an agent will involve only one role. Negotiation strategies are models that describe the reasoning procedure negotiators use to evaluate offers, work out counteroffers, and decide when to accept proposals [7]. Service profiles describe different aspects related to a negotiated service such as functional aspects, nonfunctional aspects, and methods of evaluating service attributes.

Functional composition. In a large market for software services, matching functionality is likely to be an issue. Given a set of functional requirements and a number of software services, negotiating the best functional combination that meets the desired requirements can be a topic for negotiation. Addressing such a problem requires, among other things, the consideration of the implications of assembling functionality from a number of different sources and the need to consider the effect on the ultimate quality of the required functionality.

Management of dependencies and uncertainty. Unlike traditional business processes normally executed in a relatively predictable and repetitive way, composing a complex service through negotiation must deal with the issues of uncertainty and the management of dependencies. Although simple services are delivered point to point, a number of simple services could be combined together in a meaningful way to provide a composite service. However, the creation of a composite service is highly affected by the dynamics of the service-oriented environment.

An example of service dependencies is a travel planning service consisting of flight booking, hotel
reservation, and car rental services. While these services may be independently contracted, they are dependent on each other in the sense they are all contributing to the goal of making an appropriate journey. For instance, it may be risky to first book a hotel and then fail to make a flight reservation. If the hotel reservation and the flight-booking services are provided by independent providers, the negotiator must take into account the synchronization implied in contracting these two services. Because the car rental may by substituted by using public transportation, failure to complete such service may not require the cancellation of the composite service. Such aspects have to be taken into consideration during the design of a negotiation-oriented system that will work in an environment where service dependency and uncertainty are key attributes.

**Negotiation convergence.** In an environment where time is critical and negotiations must be brought to an end in a short period of time, two mechanisms can help achieve this objective: negotiation protocols and negotiation strategies.

Negotiation protocols can be used to tune the behavior of the participating parties in such a way that time-wasting activities can be minimized and negotiations can be brought to a conclusion as early as necessary. Such rules may be as simple as setting a maximum duration for negotiation or as complex as penalizing agents for any time-wasting behaviors.

The negotiation strategies approach incorporates two techniques that can help achieve better convergence: the heuristic approach and the argumentation approach have been discussed in literature. The heuristic approach uses different learning techniques that enable an agent to produce offers likely to be accepted by the other party based on a history of negotiation. The argumentation approach depends on using techniques that can affect the state of mind of the other party and hence persuade that party to accept the proposed offer [5].

**Service delivery.** If negotiation results in a contract, the next phase is service delivery, which involves implementing what the parties have agreed in the contract. It is the phase where the fulfillment of promises is tested and trust values are updated. The information gained at this phase feeds back into the agent’s knowledge repository to be considered the next time the agent decides to engage in negotiation with the same partner.

**Contract enactment.** A contract describes service functionality and its associated nonfunctional attributes (price and quality attributes), service pre- and post-conditions, and contract pre- and post-conditions. The difference between service and contract conditions (pre and post) is that service conditions are technical in nature, specific to a certain service, and describe the requirements and the consequences of using a particular service. While contract pre- and post-conditions are business oriented and can apply to more than one service. An example of a contract precondition is the requirement that a user of a library service must have a student ID or that an application requesting a service call must from within a specific IP range. A contract post-condition describes the consequences of providing a service. For instance, a contract precondition may state that a payment is due within a specified period of time after the service has been delivered or that a minimum charge for usage applies. A contract can be modeled as a process that describes the phases, the events, and the data produced as a result of delivering a service.

**Trust.** In an environment where consumers have the choice to change the source of the service each time a need arises, trust becomes a key issue in selecting a service provider. In the context of service provision, trust is the perception of the ability of a service provider to offer what has been agreed upon in the service contract. It is worth noting that trust is a relative concept in the sense a provider may be trusted in one context but not the other. For instance, an ISP may be trusted to provide fast download rate during off-peak hours but not during peak hours.

Formulation of trust may be developed internally from past experience in dealing with a specific provider or it may be outsourced to a third party. An example of a trust service provider is a company that provides credit ratings for individuals and can make such a service available for commercial bodies to assess the credit rating of their customers.

**Implementation Technologies**

How much of this vision is in place today? Web services are the technical apparatus for achieving an Internet-based, service-oriented computing vision. The term “Web services” refers to a group of closely related, emerging technologies aimed at turning the Web into a collection of computational resources each with well-defined interface for its invocation. Web services are designed to be platform and lan-
A Web services architecture describes a layered architecture and various standards used in different layers to achieve interoperability between its three basic operations: publish, find, and bind [6]. As shown in the accompanying table, the bottom layer of the Web services depends on the ubiquity of the existing Internet protocols such as HTTP and FTP. Almost very other layer of the Web services stack depends on XML.

Web Services Description Language (WSDL) is an XML-based description of how to connect to a particular Web service. WSDL uses XML format for describing network services as a set of endpoints (addresses and protocol) operating on messages containing either document-oriented or procedure-oriented information. The operations and messages are described abstractly, and then bound to a concrete network protocol and message format to define an endpoint.

Universal Description, Discovery, and Integration (UDDI) is an industry initiative project aimed at creating a platform-independent, open framework for describing services, discovering businesses, and integrating business services using the Internet.

Web Services Flow Language (WSFL) defines a framework that implementers of Web services can use to describe the business logic required to assemble various services into an end-to-end business process. WSFL is still in its early phases of development.

On top of the Web services stack, a negotiation layer can define the mechanisms by which a number of parties agree over protocols, terms, and conditions for a service provision. The importance of such a layer is that it will enable applications to tailor their needs dynamically beyond the “invoke-reply” model.

Web services provide an essential building block for realizing dynamic e-business by facilitating standards for program-to-program interaction. However, to achieve a fully functional market for software services, it is necessary to address a number of business issues brought about as a result of delivering software as a service such as business partner selection, service-level agreement, contract frameworks, and negotiation.

**Conclusion**

The shift toward delivering software as a service necessitates the ability to tailor needs through negotiating. This article highlights the characteristics of the potential software services market, the implications of such characteristics for negotiation, and presents a three-phase classification of negotiation issues that must be dealt with through different phases of negotiation.

To address the issues described in the classification presented in this article, we have developed a Negotiation Description Language to provide a concrete representation of the concepts that underlie service negotiation in the context of the service-oriented model. A full description of the language can be found in [4].

**References**


Ahmed M. Elfatatry (Elfatatry@G5online.com) is a lecturer of software engineering in the Department of Information Technology at the University of Alexandria, Egypt. Paul J. Layzell (paul.layzell@umist.ac.uk) is a professor of software management in the Department of Computation at the University of Manchester Institute of Science and Technology, Manchester, U.K.