Building Trust in E-Commerce

A network of Internet-based intermediaries that guarantee delivery and payment in e-commerce could help bolster consumer and merchant confidence.

Yacine Atif
United Arab Emirates University

Most of us who have purchased items via the Internet have felt reluctant about the transaction at some point — usually when entering our credit card number or receiving unexpected goods. As customers, we need guarantees that the other party will not misuse confidential information, and merchants need guarantees that they will receive payment for the goods delivered. While security protocols such as the Secure Sockets Layer (SSL) and Secure Electronic Transactions (SET) ensure that a credit card number will not be intercepted during transmission, they provide no guarantee against its misuse by the receiving party, nor against fraud by the transmitting party.

To increase confidence in commercial transactions over the Web, where the transacting parties are invisible to each other, we need not just new protocols but also new transaction processes. One solution is to enlist a third party, referred to here as a *trust service provider* (TSP), to act as an Internet-based intermediary that assumes responsibility for a smooth transaction. The TSP is known and trusted by both customer and merchant and makes purchases on behalf of the one and conveys the goods on behalf of the other.

This article describes my proposal for a *trust web* model based on a distributed search algorithm and a network of trusted intermediaries that can establish a trusted channel through which *terminal* transacting parties deal virtually directly and risk-free with each other. I have developed a CORBA-based implementation of the trust-path building algorithm and am currently testing its performance. The actual version of the system can be found at http://faculty.uaeu.ac.ae/~atif/research/ecommerce/ec.html.

Building Trust

The *trust web* features three essential components for establishing confidence between e-commerce parties:

- A simple model composed of a network of TSP entities and an algorithm that establishes a trust path prior to carrying out an e-commerce transaction.
- An open extensible TSP architecture that can include additional e-commerce mediating services and an

1089-7801/02/$17.00 ©2002 IEEE

application programming interface (API) to run the trust-path building algorithm.
- A parallel implementation of the trust-path search algorithm, which contributes to a balance between time and network traffic complexity.

Once the TSP network is in place, a commercial organization can easily access trust services by registering with a selected TSP. The organization needs a minimum Common Object Request Broker Architecture (CORBA) interface to connect to the object request broker (ORB), which conveys its initiated trust requests to a trust manager object running on the TSP host. There is no need to purchase expensive CORBA implementations; the new releases of Netscape Navigator and the Java Developer's Kit already support CORBA.

Establishing Relationships

A trust connection is established by an initiating entity that wishes to build a relationship with another selected entity by some means, such as a private favorable relationship, positive past experience, or simply by reputation. Once the approached entity approves the trust-connection building request, it becomes fully liable for carrying out incoming transactions issued by the initiating entity. The process somewhat resembles opening a bank account, wherein the bank is bound to carry out the customer's incoming requests and be responsible for their outcome. This liability level eliminates transaction risks.

This form of Internet-based mediation can be iteratively extended when the customer-contracted broker does not have a direct trust link with the targeted merchant. Inter-TSP connectivity, along with the customer and the merchants, thus forms a web of trusted entities. Figure 1 shows a graph-theoretic representation of the trust web; vertices denote the intervening entities and edges reflect trust relationships.

Finding a Path

The process of reaching a targeted merchant within the trust web can be framed as a distributed search problem aimed at identifying a *trust path*. This path links the customer with the targeted merchant across a chain of TSPs that each trust their immediate neighbors. The trust-path building process remains invisible to the transacting parties because it is carried out automatically (that is, with no offline delay) to let the parties deal virtually directly with each other. There are three major challenges in the search for a trust path.

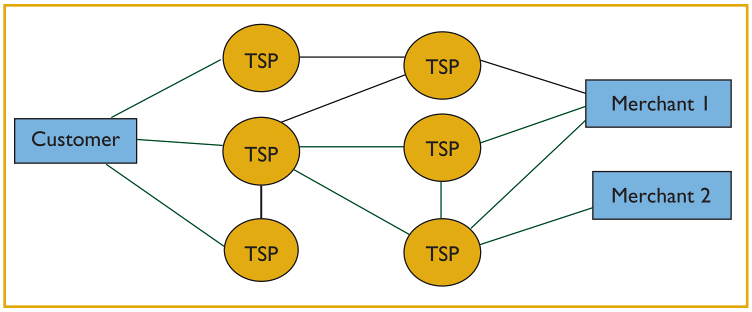

Figure 1. Graph-theoretic representation of the "trust web." Vertices denote the intervening trust service providers, and edges reflect trust relationships.

- Each TSP knows only its immediate trusted neighbors rather than each *principal* in the global trust web.
- Each TSP might be running on different operating systems and hardware and following a variety of software standards.
- The trust web's connectivity might be high — that is, each TSP entity might be linked to several other TSPs — increasing the complexity of the search process.

The distributed search algorithm I propose in this article addresses the first issue by allowing only local information related to neighboring TSPs to be available to each TSP. Maintaining platform independence in the system will handle the issue of heterogeneity among TSPs, and implementing a parallel Web search version of the algorithm will decrease search complexity, as will constraining the search process using cost functions.

Trust-Path Building Algorithm

The TSP — which is basically a software agent — automatically relays a customer trust-building request to the corresponding merchant if it knows and trusts the merchant directly. Otherwise, the TSP approaches other trusted brokers until it establishes a trusted channel across which the confident exchange of goods and money can occur. The trust web represents the algorithm's search space, and the algorithm requires the cooperation of multiple principals to find a trust path.

When the customer initiates the search process to acquire commodities from a set of targeted merchants, it relays these targets automatically to the nearest trusted TSPs in hopes of finding the targets. If the approached TSPs do not have trust connections with the targets, they forward the customer's request to adjacent TSPs to identify targets deeper within the trust web. The parallel nature of trust invocations and trust web connectivity means a TSP

1	The customer A prepares the initial message M, and sends it in parallel to all of its TSPs.			
2	Forward messages: On receiving a message, each trusted intermediary t_i performs the following steps.			
	A TSP receives a trust-path building invocation from the customer or from another TSP.	2.1	t_i retrieves its set T_i of trusted neighboring principals.	
		2.2	t_i stores the ID of the sending principal in t_{sender}.	
		2.3	t_i removes from T_i the excluded principals that are on the path constraint list ($T_i^{(1)} \leftarrow T_i \backslash (T_i \cap E_p)$).	
		2.4	If the resulting $T_i^{(1)}$ is empty, all further processing on *M* stops. This is a dead end.	
		2.5	t_i removes the visited TSPs from its neighbors and builds an updated set of trusted neighbors ($T_i^{(2)} \leftarrow T_i^{(1)}) \backslash (T_i^{(1} \cap P)$).	
		2.6	If the resulting $T_i^{(2)}$ is empty, all further processing on *M* stops. This is a dead end.	
		2.7	t_i retrieves set P_i of the target principals that are in its neighborhood ($P_i \leftarrow P \cap T$).	
	Found targets are added to the trust path. Backward message is initiated if all targets have been located.	2.8	If some target principals are found (i.e. $P_i \neq Ø$) then:	
			2.8.1	t_i adds the found targets P_i to the trust path *R* and updates $R(R^{(1)} \leftarrow R \cup P_i)$
			2.8.2	t_i builds the set of target principals $P^{(1)}$ left to locate ($P^{(1)} \leftarrow P \setminus P_i$).
			2.8.3	If all target principals have been located ($P^{(1)} \neq Ø$), t_i prepares and sends a backward message $M^{(1)}$ in which trust path *R* is replaced by the newly updated path $R^{(1)}$, and moves on to step 3.
	The TSP prepares to send a new forward message to locate the remaining targets.	2.9	t_i retrieves the message-recipients $T_i^{(3)}$ to which the message is to be forwarded by applying a cost function on each TSP and filtering out the resulting TSPs ($T_i^{(2)} \leftarrow$ *apply-cost-constraints*$(C,D, T_i^{(2)})$).	
		2.10	If the resulting TSP set $T_i^{(3)}$ is empty, all processing on *M* stops. This is a dead end.	
		2.11	t_i adds itself to the trust path *R* and updates *R* ($R^{(2)} ..\ R >> [t_i]$).	
		2.12	t_i prepares the new forward message $M^{(2)}$ in which the trust path R is replaced by newly augmented path $R^{(2)}$.	
		2.13	t_i sends $M^{(2)}$ to all TSPs in $T_i^{(3)}$ in parallel ($\forall\ t_k \mid t_x \in T_i^{(3)}$, Send $(M^{(2)}, t_k)$).	
3	Backward messages: On receiving a backward message, each t_i sends it to the associated t_{sender}.			

Figure 2. Trust-path building algorithm. Each principal executes the algorithm when it receives a message from another.

Table 1. Message structure.

Label	Legend
S	Unique session ID to identify the request
t_s	Time stamp
R	Return path initially containing one entry (R=[a])
P	List of target principals. (P=[p1 ,p2,... ,pn])
E_P	Path constraints: a list of principals to be excluded from the trust path
C	Cost constraints, such as number of TSPs on the path, maximum charges that the customer will pay, communication costs, or time after which the search should end
D	Accumulated costs along the path

might receive and process the same request twice. To avoid this, each request includes a unique session ID and time stamp.[1] All received requests are backlogged, and duplicate requests are discarded.

Reducing Complexity

The search process proceeds from the root node representing the customer to a leaf node representing the merchant. A TSP simply stops the search if it faces a dead end because it has no trust connections or all its connections lead to already explored TSPs. We can reduce the search space complexity by pruning the alternatives that do not satisfy some constraints.

A customer can specify certain TSPs not to include on the path, for example, by indicating individual TSPs' identities or geographical regions to exclude. Other constraints could specify the maximum cost the customer will pay for TSP services, the number of intervening TSPs, or even the communication cost. Additional constraints increase message size but reduce the number of messages exchanged between principals. Combining trust-path building constraints with a parallel search for trusted partners provides a tradeoff between time and network traffic complexity.

Locating Targets

Each principal executes the distributed search algorithm described in Figure 2 whenever it receives a message. (Later, in the algorithm's object-oriented implementation, messages take the form of remote object invocations.) Initially, customer *A* sends a message to its immediate TSP's set: the TSPs with which the customer has direct trust relationship. Table 1 describes the structure of a message M = $\{S, t_s, R, P, E_P, C, D\}$, sent to the trusted intermediaries after being encrypted using each TSP's public key.

As Figure 2 indicates, the algorithm recognizes two message types: *forward* and *backward*. The search for targets generates forward messages. When targets have been located, backward messages trace the path of the forward messages back to the trust-building request initiator. No backward messages are generated if the path could not be found. If a starting principal does not receive a backward message after a certain time, it assumes that no path was found.

The algorithm starts by sending messages from the customer (*A*) to all of its directly connected principals t_i in the trust web. On receiving a message, a TSP forwards it to its direct

Related Work in Trust

Early work on trust-path evaluation originated in the problem of public-key exchange and authentication.[1] Most of the research focused on interconnected "authentication" servers that were used to obtain public keys with little or no risk. This research inspired the work proposed in this article on developing a network of trusted intermediaries for Internet-based commerce.

More recently, Borcherding and Borcherding suggested a centralized algorithm, based on the Dijkstra shortest-path algorithm, for calculating a trust path for safely transmitting public keys.[2] I have incorporated aspects of this approach in the trust web proposal described in the main text.

A New Dimension

Much of the related research focuses on network security and authentication, but e-commerce domains add a new dimension of trust requirements. In an e-commerce system, trust stems from consumers' and merchants' confidence in the competence, dependability, and security of the system under risk conditions.[3]

A typical e-commerce transaction follows three steps:

- locating business parties,
- establishing a trusted path between parties, and
- executing the transaction along the trust path.

The buyer usually seeks out the business party, but a merchant could also initiate the search by mining for potential customers. In most cases, customers do know the targeted merchants before issuing purchase requests or looking through Web-based catalogue servers. In either case, the search phase results in a list of targeted merchants the customer wishes to deal with. This list represents the input to the trust-path building algorithm discussed in this article.

Trusted Intermediaries

Successful e-commerce systems depend heavily on the second step, but the Internet fosters distrust because the involved parties are invisible to each other. This dilemma can be solved by using a *trusted intermediary (TI)*,[4] which is basically a broker trusted by and accountable to both customer and merchant. The TI verifies that the goods match the specifications, and then forwards the goods to the customer and the payment to the merchant. Unless transacting parties know who is liable for guaranteeing transactions, confidence in e-commerce will remain weak.[5]

Ketchpel and Garcia-Molina first suggested a solution for conducting risk-free transactions with the aid of trusted intermediaries,[4] but they did not describe how to build a "trust path" between unknown parties. Su and Manchala proposed one such algorithm for evaluating trust paths with the help of trusted intermediaries.[6] In their proposal, a central server calculates the minimum-length Steiner tree for connecting selected vertices with customer and merchants as leaf nodes. This approach, however, requires a centralized execution of the algorithm and assumes that the central server on which the algorithm is running knows the entire solution space.

References

1. R. Yahalom, B. Klein, and T. Beth, "Trust-Based Navigation in Distributed Systems," *J. Computing Systems*, vol. 7, no. 1, 1994, pp. 45-73.
2. B. Borcherding and M. Borcherding, "Efficient and Trustworthy Key Distribution in Webs of Trust," *Computers and Security*, vol. 17, no. 5, 1998, pp. 447-454.
3. A. McCullagh, "E-commerce — A Matter of TRUST," *Information Industry Outlook Conf.*, 1998; available at http://www.acs.org.au/president/1998/past/io98/etrust.htm.
4. S.P. Ketchpel and H. Garcia-Molina, "Making Trust Explicit in Distributed Commerce Transactions," *Proc. 16th Int'l Conf. Distributed Computing Systems*, IEEE CS Press, Los Alamitos, Calif., 1996, pp. 270-281.
5. V. Ahuja, "Building Trust in Electronic Commerce," *IT Professional*, vol. 2, no. 3, May 2000, pp. 61-63.
6. J. Su and D. Manchala, "Building Trust for Distributed Commerce Transactions," *17th Int'l Conf. Distributed Computing Systems*, IEEE CS Press, Los Alamitos, Calif., 1997, pp 322-329.

principals that are not in the path constraint E_P (step 2.3) and have not already been visited (step 2.5). The receiving TSP then checks whether any neighboring principal is in the target principal list, P (step 2.7). If so, that target principal is removed from P and appended to the constructed path, R. Once a TSP has located the last target principal yet to be found, it sends the message back to t_{sender}, from which it received the message (step 2.8.3).

Thus begins the message's backward traversal. If some target principals remain, the cost constraints are applied using the function *apply-cost-constraints* (step 2.9), and any violating principal is removed from the list of neighboring TSPs. Finally, the algorithm appends the current principal to R and sends forward messages to all its trusted principals that passed the constraints test (step 2.13).

Path-Building Example

Figure 3 (next page) illustrates how the trust-path building algorithm might work using the trust web in Figure 1. The number attached to each TSP node (t_1, t_2, t_3, and so on) indicates the trust service charge. Assume that cost-constraint list C states that the path should include no more than four TSPs and that total charges should not exceed 200. Customer A also wants to exclude principal t_5 from the path. For simplicity, omit the time stamp t_s and the session ID S from the message structure and trace only one message, issued from A and addressed to t_2. Table 2 (next page) provides a detailed legend for the link labels in Figure 3 that

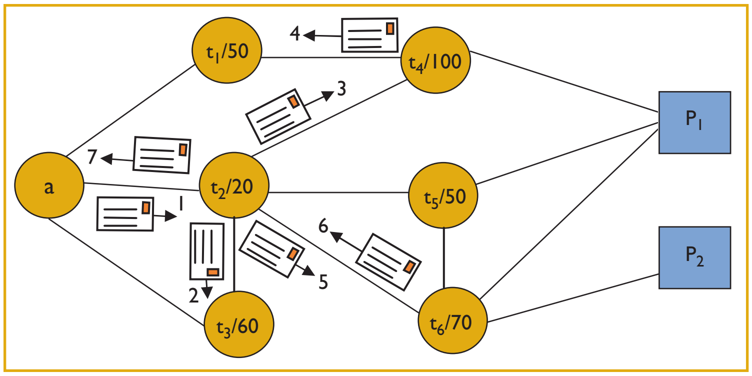

Figure 3. Trust-path building example. The cost-constraint list determines the message's path.

Table 2. Legend for Figure 3.

Step	Message	Type
1	*Customer A sends* to t_2 $\{[a],[p_1,p_2],[t_5],[4,200],[1,20]\}$	Forward
2	t_2 sends to t_3, $\{[a,t_2],[p_1,p_2],[t_5],[4,200],[2,80]\}$	Forward
3	t_2 sends to $t_4\{[a,t_2],[p_1,p_2],[t_5],[4,200],[2,120]\}$	Forward
4	t_4 sends to t_1:$\{[a,t_2,t_4],[p_2],[t_5],[4,200],[3,170]\}$	Forward
5	t_2 sends to t_6:$\{[a,t_2],[p_1,p_2],[t_5],[4,200],[2,90]\}$	Forward
6	t_6 sends to t_2:$\{[a,t_2,,t_6,,p_1,p_2],[],[t_5],[4,200],[2,90]\}$	Backward
7	t_2 sends to a:$\{[a,t_2,,t_6,,p_1,p_2],[],[t_5],[4,200],[2,90]\}$	Backward

represent the exchanged messages.

The algorithm's main overhead comes from the messages generated. In the worst-case situation — with no search constraints and a fully connected trust web in which each TSP trusts only one merchant — a single customer trust-path building request generates $k(n!)$ messages, where n is the total number of TSPs in the trust web and k is the number of TSPs the customer directly trusts. This network traffic rate is unlikely under normal conditions, however, because the real-life trust web is partially connected. Moreover, a TSP normally trusts many merchants; hence some messages do not travel through the whole network.

Architecture and Implementation

To make the trust web and the trust-path building process a reality, the implementation must meet certain requirements that are common to all open system standards.[2]

- *Interoperability.* Internet-based applications must be able to work together through well-defined interfaces.
- *Portability.* Applications must be decoupled from any particular computing environment.
- *Integration.* Applications should require minimal effort to be incorporated with existing systems.

I have chosen to use CORBA[3] to implement the trust web model because it satisfies all these requirements.

TSP Architecture

Exploring the TSP architecture in depth shows how to integrate trust services in existing e-commerce systems. Successful e-commerce systems follow an object-oriented approach based on two major principles: *Modularity* divides the system into self-contained modules or objects, and *abstraction* separates these objects' descriptions from their actual implementation. Architectures based on these design principles prove to be scalable and flexible, allowing easy integration of new services. Figure 4 shows how such an e-commerce system can be fine-tuned to embody trust services and thus become a TSP.

The architecture shown in Figure 4 divides an e-commerce system into three distinct layers:

- The *interface* layer presents the services to external entities (customers or other businesses). This layer identifies service types. Trust services augment the already advertised group of services.
- The *service implementation* layer contains a processing methodology for the advertised services. It includes the business-aware rules and control functions embedded in objects; each service implementation might require a set of such objects.
- The *service-related data* layer incorporates the data management functionality implemented using database servers (possibly from multiple vendors). ODBC and Java database connectivity (JDBC) drivers allow access to these heterogeneous database systems, as all major databases have dedicated ODBC/JDBC drivers.

This open architecture allows us to easily integrate new services — including trust services — into existing systems by defining additional interfaces and creating the implementing objects and required data sources for the new services. For an e-commerce system to integrate trust services and become a TSP, an additional interface that includes the signature of the `FindPath` method augments the interface layer. This interface at the service implementation layer corresponds to the trust-path finding algorithm proposed in Figure 2.

Each TSP stores information about its trusted neighbors (IDs, charges, and so on) in its *trust database*, which is integrated at the data level of the TSP architecture. Table 3 shows the database's structure. The algorithm searches this table to find a registered trusted merchant or check for cost constraints.

Trust Web Architecture

TSPs act as both servers and clients in the trust web. As servers, TSPs expose a remote object through its interface. Trusted clients can invoke the exposed-object methods remotely. TSPs become clients when they invoke other TSPs' methods. As discussed earlier, the interface is a purely declarative component that hides the implementation details. This deliberate strategy facilitates interoperability and software integration.

Figure 5 (next page) depicts the trust web's basic system architecture. Implementing the trust web as middleware allows remotely located TSP objects to communicate. A TSP joins the trust web by advertising its services interface through an ORB (such as CORBA). TSPs can thus locate each other automatically on the trust web using their individual IDs (stored in their individual trust databases).

A TSP also runs a Web server so that clients (customers, merchants, or other trusted TSPs) can remotely invoke its trust-building API, FindPath. This process includes a public-key authentication mechanism to ensure that only *TSP registered clients* who have previously subscribed to the services can access a host's TSP server. The client specifies the target principals and path and cost constraints and then remotely invokes the FindPath method supplied by the trust manager object in each TSP's service implementation layer. When a host contacts the trust managers of its neighboring trusted intermediaries, it becomes their client. This process continues until the ORB locates targeted merchants and constructs trust paths.

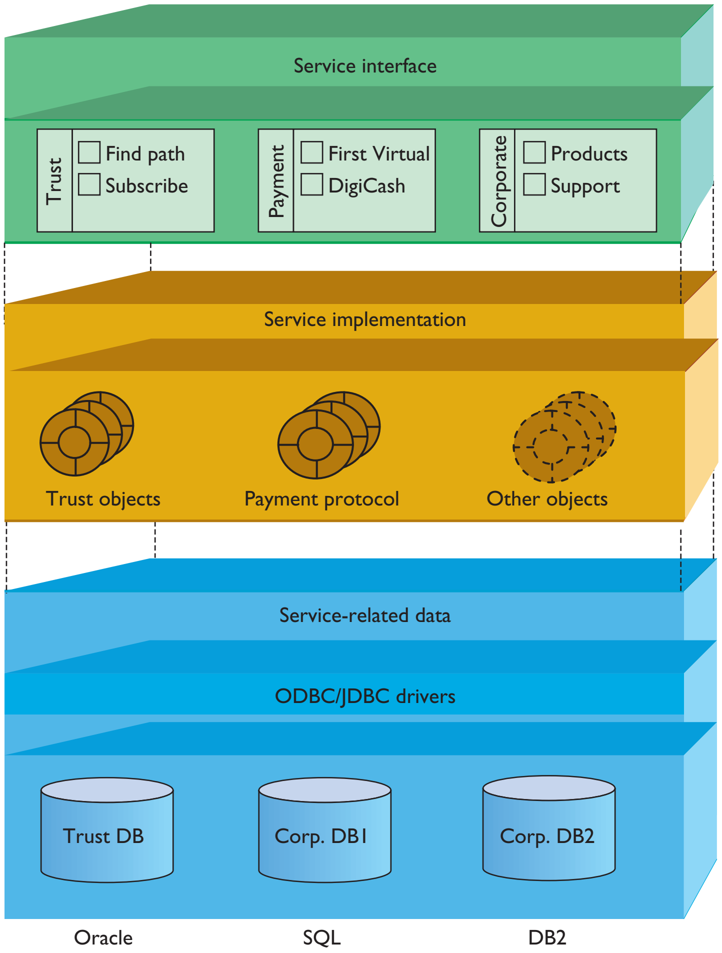

Figure 4. TSP system architecture. Trust services can be integrated into the three primary layers of most modular e-commerce systems.

Table 3. Trust database layout.

Host ID	Charges	Type
B1	0	Merchant
T1	100	TSP
T2	200	TSP
T4	50	TSP

Parallel Web Search

The search for targeted merchants is distributed and parallelized across the trust web. Initially, the customer may approach several TSPs concurrently, which then relay in parallel the trust-path building request to their trusted neighbors. This implementation (see http://faculty.uaeu.ac.ae/~atif/research/ecommerce/ec.html) uses multithreading to invoke the FindPath method many times simultaneously. CORBA's multithreading support allows a TSP to simultaneously receive and process multiple FindPath method invocations. For each invocation session, the TSP server spawns a new trust manager thread that supervises the incoming trust request. Each thread has its own state-related information with a serialized shared section-part to avoid any critical section violation problem.

Multithreading also lets a host send parallel requests to neighboring trusted intermediaries, which helps cut down the search time and com-

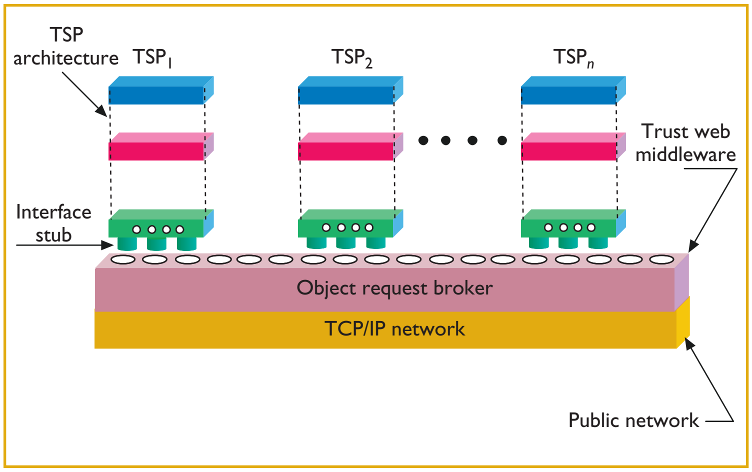

Figure 5. Trust web architecture. A TSP advertises its trust services and joins the trust web by plugging its interface to the ORB.

plexity. My Java-based TSP implementation uses Java's synchronization mechanism on the concurrent threads. A client TSP sends forward messages to all the TSPs in its trust database that pass the path and cost-constraint tests. Client threads spawned to "bind" to a neighboring host's trust server object then remotely call its FindPath method. Each thread then waits for a response from the server, which might invoke further FindPath instances on its neighboring nodes until it locates the target principals or encounters a dead end. In either case, the resulting response is channeled back to the originator.

Conclusion

Building confidence in e-commerce requires more than robust processing systems. The human perception of trust is a core ingredient in any online transaction,[4] and future e-commerce systems must support trust services to gain loyalty at both the consumer and provider ends.

Trust might seem an intangible feature to incorporate into computing systems, but the automatic trust-building algorithm and system architecture proposed here achieve that by negotiating customers' requests to identify a sequence of trustworthy intermediaries in order to complete a transaction with a merchant. However, this model is based only on Boolean relationships. That is, any two entities can share either a complete trust or a complete distrust relationship. Future work will consider a trust model based on fuzzy logic for determining the best-suited trust-path.[5]

References

1. H. El-Rewini and T.G. Lewis, *Distributed and Parallel Computing*, Manning Publications, Greenwich, Conn., 1997.
2. A. Umar, *Application (Re)Engineering: Building Web-Based Applications and Dealing With Legacies*, Prentice Hall, Upper Saddle River, N.J., 1997.
3. Z. Tari and O. Bukhres, *Fundamentals of Distributed Object Systems: The CORBA Perspective*, John Wiley & Sons, New York, 2001.
4. D.W. Manchala, "E-Commerce Trust Metrics and Models," *IEEE Internet Computing*, vol. 4, no. 2, 2000, pp. 36-44.
5. H. Hsiung, S. Sheurich, and F. Ferrante, "Bridging E-Business and Added Trust: Keys to E-Business Growth," *IT Professional*, vol. 3, no. 2, Mar. 2001, pp. 41-45.

Yacine Atif is an assistant professor of computer science at the Information Technology College of United Arab Emirates University. He received a PhD from the Hong Kong University of Science and Technology. His primary research area is distributed computing techniques to support e-commerce applications.

Readers can contact the author at Yacine.Atif@uaeu.ac.ae.