A Security Architecture for Agent Communication Languages

James Mayfield[1] and Tim Finin[2]

[1]The Johns Hopkins University Applied Physics Laboratory
Johns Hopkins Road
Laurel, MD 20723–6099 USA
James.Mayfield@jhuapl.edu
+1 (301) 953–6944

[2]Computer Science and Electrical Engineering Department
University of Maryland Baltimore County
1000 Hilltop Circle
Baltimore, MD 21250 USA
finin@cs.umbc.edu

Abstract. One of the essential features of a software agent is its ability to cooperate with other software agents. This cooperation requires, in general, that software agents be able to communicate in an appropriately rich *agent communication language* (ACL) and associated protocols. For an ACL to be effective in an open environment like the Internet, it must support security, privacy, the integrity of data, and authentication of agent identity. We discuss some basic and extended security requirements for software agents and an architecture to satisfy those requirements for KQML-speaking agents. Many of these security features will be provided by transport mechanisms which carry the ACL (*e.g.,* sockets, HTTP, SMTP). However, security properties must be part of and reflected in the ACL model and cannot simply be relegated to the lower levels of the communication protocol stack.

1 Introduction

One of the essential features of a software agent is its ability to cooperate with other software agents. This cooperation requires, in general, that software agents be able to communicate in an appropriately rich *agent communication language* (ACL) and associated protocols. For an ACL to be effective in an open environment like the Internet, it must support security, privacy, the integrity of data, and authentication of agent identity.

1.1 Why agent security?

Security, privacy and authentication are always desirable and often necessary properties of most general communication systems. Security features are being designed into the Internet communication standards (*e.g.,* sockets, email, corba, http, *etc.*) that software agents use to communicate among themselves. One could argue that we only need to design the security architecture into these basic communication substrates and agent communication systems will inherit them.

Although basic security features and properties will be provided by the underlying substrate, it is important that security be explicitly modeled at and reflected in the agent communication level for several reasons. Agents must, in general, be aware of the security aspects of their conversations with other agents. There may be costs associated with some of the security features and agents may want to be selective about when they are used. Moreover, not all of an agent's interlocutors will use every security mechanism; an agent might use this information to decide with whom to communicate, what to say, or whether secure protocols should be invoked.

For example, an agent might use a notion of *lazy authentication*, in which it accepts information from other agents without requiring a relatively expensive authentication protocol to establish identity. At some later time when a fact from an as yet unauthenticated source becomes relevant to an agent's reasoning, it might contact that source and engage in an authentication dialogue to verify the agent's identity and the fact's integrity.

Consider how human agents live and communicate in a world in which we have a range of communication media offering a wide range of security and privacy options. For some of us (more every day), security is important and we actively model and reason about the security of our communication. The same will be true of software agents that reside in a complex environment of communication options.

1.2 KQML

Knowledge Query and Manipulation Language (KQML) [1] is a communication language and protocol that enables autonomous and asynchronous software agents to share their knowledge and work towards cooperative problem solving. It was developed as a part of the *Knowledge Sharing Effort* [24, 22, 16]. The KQML language can be thought of as consisting of three layers: the content layer, the message layer, and the communication layer. The content layer bears the actual content of the message, in the program's own representation language. The communication level encodes a set of message features which describe the lower level communication parameters, such as the identity of the sender and recipient, and a unique identifier associated with the communication. The message layer forms the core of the KQML language, and determines the kinds of interactions one can have with a KQML–speaking agent. A primary function of the message layer is to identify the protocol to be used to deliver the message and to supply a speech act or performative which the sender attaches to the content (such as that it is an assertion, a query, a command, or any of a set of known performatives). In addition, since the content may be opaque to a KQML-speaking agent, this layer also includes optional features which describe the content language, the ontology it assumes, and some type of description of the content (such as a descriptor naming a topic within the ontology). These features make it possible for KQML implementations to analyze, route and properly deliver messages even though their content is inaccessible.

1.3 Security Requirements

We arrived at the following requirements for a KQML security model based on an analysis of the security models for Privacy Enhanced Mail [4], CORBA [3] and DCE [5]. Interested readers are referred to Voydock and Kent [2], for a thorough treatment of security threats and mechanisms to counter them. The security capabilities that should be supported include:

- *Authentication of principals.* Agents should be capable of proving their identities to other agents and verifying the identity of other agents.
- *Preservation of message integrity.* Agents should be able to detect intentional or accidental corruption of messages.
- *Protection of privacy.* The security architecture should provide facilities for agents to exchange confidential data.
- *Detection of message duplication or replay.* A rogue agent may record a legitimate conversation and later play it back to disguise its identity. Agents should be able to detect and prevent such playback security attacks.
- *Non-repudiation of messages.* An agent should be accountable for the messages that they have sent or received, *i.e.,* they should not be able to deny having sent or received a message.
- *Prevention of message hijacking.* A rogue agent should not be able to extract the authentication information from an authenticated message and use it to masquerade as a legitimate agent.

We also consider several additional constraints or desiderata for the architecture. First, the security architecture should not depend on the semantics of KQML performatives. The security model should be general and flexible enough to support different models of agent interaction (*e.g.,* ContractNet, electronic commerce). Neither should the architecture depend on the features offered by any transport layer since we want to facilitate agents to communicate across heterogeneous transport mechanisms and to extend the security model to accommodate embedded KQML messages. Second, we desire a model which allows light-weight agents without cryptographic capabilities to authenticate the sender of a message using the services of trusted *authenticator* agents. Finally, we want to allow agents the flexibility to use different cryptographic algorithms so the security architecture should not have hard dependencies on any specific cryptographic algorithm. Similarly, we reject systems that assume a global synchronization of time; such synchronization is difficult to achieve and leads to further security issues of its own [7].

2 Message–Level Security

To accommodate the asynchronous nature of general ACLs like KQML, the model expects a secure message to be self authenticating; it does not support any challenge/response mechanism to authenticate a message after it has been delivered. The architecture provides two security models, basic and enhanced. The basic security model supports authentication of sender, message integrity and privacy of data. The enhanced security model additionally supports non-repudiation of origin (proof of sending) and protection from message replay attacks. The enhanced security model also supports frequent change of encryption keys to protect against cipher attacks.

2.1 Cryptographic background

This subsection summarizes the cryptographic techniques used by the architecture and the new performatives and parameters that have been introduced to implement the architecture. A fuller exposition may be found in Thirunavukkarasu [26].

Encryption Keys. An agent that implements the proposed security architecture should have a master key, K_a, which it will use to communicate with other agents. This key can be based on a symmetric key or an asymmetric key (*e.g.,* public key) cryptosystem. If a symmetric key mechanism is used, we suggest that the agent, in addition to the general master key, also use a specific master key, $K_{a1,a2}$ for each agent with which it communicates; doing so will provide better privacy and stronger authentication. If an agent does not share a master key, $K_{a1,a2}$ with another agent, it can use its master key, K_a, or it can use the services of a central authentication server to generate such a key.

Session key. In the enhanced model, the agents use an additional key, the session key, to ensure privacy, message integrity and proof of identity. Agents can use either the session key or master key for exchanging messages, and must inform the receiving agent of the key that was used for encryption to ensure proper decryption.

Message ID. The message ID is used in the enhanced security model to protect agents from message replay attacks. When the two agents establish a session key, they also exchange a message ID, which the sender uses in the next message. Each message from an agent carries a message ID and a new message ID for the next message. Each message ID is used only once to prevent replay and they are encrypted using the session or master key for security.

Message Digest. Each secure message generated using this architecture has a message digest or signature associated with it. The digest is calculated using a secure hash function like MD2, MD5 or SHS [9]. This hash function computes a digital fingerprint of the message (*i.e.,* it acts as a "checksum" for the message). The sender then encrypts this digest using the session or master key and attaches it to the message. This encrypted message digest forms the core of the security architecture. The receiver of a message uses the digest to verify the identity of the sender and the integrity of the message. The digest also protects the message ID field from being hijacked and used in a different message.

2.2 Proposed Changes to KQML

To implement this security architecture we propose several new KQML performatives, several new parameters and some modifications to a proposed standard ontology for agents.

Ontological assumptions. We assume that KQML-speaking agents use a basic agent ontology, which provides a small set of classes, attributes and relations helpful in talking about agents, their properties and the relationships and events in which they partake. Assuming this ontology, our security architecture introduces a new sub-class of agent named *authenticator* and a new relation, *key/5* which describes a key used by an agent:

```
(key <sending-agent> <receiving-agent>
   <master-key?> <key-type> <encrypted-key>)
```

An instance of this relation specifies a key that the sending agent will use in secure communication with the receiving agent. If the third argument is *true* then the key is a master key, else it is a session key. If the receiving agent is unspecified, then the key is used by the sending agents for communication with all agents. Note that this would typically be the case for asymmetric keys.

Several new KQML parameters are required to implement the security architecture:

:auth-digest (<digest-type> <encrypted-digest>). The *digest-type* specifies the hashing function used (*e.g.,* MD4, MD5, *etc.*) to compute the message digest. The *encrypted-digest* is the message digest encrypted using the key specified by the *:auth-key* parameter. This parameter should be present to prevent message hijack, and to provide for sender authentication and integrity assurance.

:auth-msg-id (<msg-id> <encrypted-msg-id>). This parameter is required only in the enhanced security model where it is used prevent message replay. The value is a list whose first element is the agreed upon random string, or NIL if this is the first message. The second element specifies the message ID for the next message and is encrypted using the key specified by the *:auth-key* parameter. For effective prevention of message replay, this parameter should be present in each message.

:auth-key (<bool> <key-type> <encrypted-key>). This parameter specifies the key being used to encrypt any *:auth-digest* and *:auth-msg* parameters present. If the first element of the triple is *true* then the master key is used, otherwise the session key is used.

The following new KQML performatives have been added to implement the security architecture:

auth-link. The sender wishes to authenticate itself to the receiver and set up a session key and message ID.

auth-challenge. The sender challenges the identity of the receiver in response to an *auth-link*. The sender encrypts a random string using the master key $K_{s,r}$ or K_s and sends it as *:content.*

auth-private. The sender is sending a confidential message to the receiver. The *:content* parameter contains the encrypted message and the *:auth-key* parameter specifies the encryption key. The *:auth-digest* parameter should be present to verify the identity of the sender and the *:auth-msg-id* and *:auth-key* parameters may be present if enhanced security model is used.

help. We introduce a new generic performative by which an agent can ask another for help in processing the the embedded performative given as the value of the *:content* parameter. The nature of the "help" is determined by the embedded performative and the value of the *:ontology* parameter. If the *:ontology* is *authentication*, then a crypto-unaware agent is enlisting the help of a trusted friend to process a performative it has received, which is included as the value of the :content parameter. This embedded message can be either an *auth-link* or a generic message to be authenticated. In the case of an *auth-link* (*i.e.,* a challenge initiation), the appropriate response is a reply with a random challenge string. In the case of a message to be authenticated, the response will be an error or a reply to forward.

2.3 Security Protocol Examples

When R2D2 sends a secure message to C3PO, it computes a message digest and encrypts it using the master key (as indicated by the value T for the *:auth-key* parameter).

```
<performative>
  :sender R2D2 :receiver C3PO :auth-key T
  :auth-digest (<digest-type><encrypted-digest>)
```

Alternatively, if R2D2 needs to send a confidential message to C3PO, it can encrypt the message and embed it in an *auth-private* performative.

```
auth-private
  :sender R2D2 :receiver C3PO :auth-key T
  :auth-digest (<digest-type> <encrypted-digest>)
  :content <encrypted-KQML-message>
```

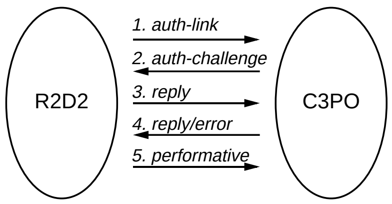
This model can be used when R2D2 does not know the recipient in advance, *e.g.,* for messages to be broadcast or routed by a facilitator agent, or if R2D2 and C3PO do not require prevention of message replay and can afford the cost of using the master key.

In the above message, the *:auth-digest* parameter can be used to verify the integrity of the message, authenticate the sender and ensure non-repudiation of origin (if the master key is asymmetric). If the message has been corrupted, the message digest will not agree with the value of the *:auth-digest* parameter. Since the message digest is encrypted with the master key of the *:sender*, only the *:sender* or the agents with which the *:sender* shares the encryption key could have generated the message. If the master key is an asymmetric key, only the *:sender* could have generated the message, as only the *:sender* knows the private key that has been used for encryption. Note that we can only verify the identity of the generator (*i.e.,* that the message was encrypted by the *:sender* agent) of the message. This message could be a replay of a legitimate message previously sent by the generator.

The enhanced security model adds prevention of message replay, and stronger non-repudiation of message origin (if asymmetric keys are used). Even though non-repudiation can be achieved in the basic security model, we can only be sure that the message was generated at some point by the sender; a rogue agent can replay a message without detection under the basic model.

In Thirunavukkarasu [26] we demonstrate how the new KQML performatives and parameters can be used to communicate securely, and describe the role of authenticator agents for key registration and management. In the remainder of this section we give an example of the use of the enhanced security model for self authentication. The next section shows how the protocols can be captured in Protolingua, a conversation specification language.

Suppose that agent R2D2 has cryptographic capabilities and would like to prove its identity to agent C3PO. The agents would follow the following handshake protocol to achieve it.



First, R2D2 sends an *auth-link* performative to C3PO.

```
auth-link                (1)
  :sender R2D2 :receiver C3PO
  :reply-with <expression>
```

If C3PO will not authenticate senders, it can respond with an *error*, otherwise it sends an *auth-challenge* with a random string encrypted using the master key. A random string is used to prevent message replay.

```
auth-challenge           (2)
  :sender C3PO :receiver R2D2
  :in-reply-to <expression>
  :reply-with <expression>
  :content <encrypted-random-string>
```

R2D2 responds with a *reply* performative whose *:auth-digest*, *:auth-msg-id* and new session key (if present) are encrypted using the master key. The value of *:content* and *:auth-msg-id* is the decrypted random string. The session key parameter is optional.

```
reply                    (3)
  :sender R2D2 :receiver C3PO
  :in-reply-to <expression>
  :reply-with <expression>
```

```
    :auth-digest (<digest-type> <encrypted-digest>)
    :auth-msg-id (<msg-id> <encrypted-msg-id>)
    :auth-key (T <key-type> <encrypted-key>)
    :content <random-string>
```

Now, C3PO can verify if the sender is R2D2 by inspecting the random string. Only R2D2 (or in the case of symmetric key, one of the other agents that shares the same key) could have decrypted the random string as it was encrypted using the master key. The message digest can be used for non-repudiation if asymmetric keys are used.

C3PO responds with a *reply* or an *error* depending on the success of authentication (3).

Now, R2D2 can send an authenticated message to C3PO by using the session key or master key to encrypt the message digest and a non replayable message by using the *:auth-msg-id* parameters.

```
  <performative>         (4a)
    :sender R2D2 :receiver C3PO
    :auth-digest (<digest-type> <encrypted-digest>)
    :auth-msg-id (<msg-id> <encrypted-msg-id>)
    :auth-key (<bool> <key-type> <encrypted-key>)
```

Or if R2D2 needs to send a confidential message to C3PO, it can encrypt the message and embed it in an *auth-private* performative.

```
  auth-private           (4b)
    :sender R2D2 :receiver C3PO
    :auth-digest (<digest-type> <encrypted-digest>)
    :auth-msg-id (<msg-id> <encrypted-msg-id>)
    :auth-key (<bool> <key-type> <encrypted-key>)
    :content <encrypted-KQML-message>
```

Further examples of the enhanced security model can be found in Thirunavukkarasu [26].

3 Conversation–Level Security

The security mechanisms presented here demand specific sequences of message exchange. Such sequences form a good match with our natural tendency to view communication among agents as conversation, in the same way we view communication among people as conversation. Unfortunately, KQML is a message-oriented language. As such, it lacks a mechanism for collecting messages into conversations, or for describing conversations in the abstract. Thus, an additional formalism is needed to allow the specification of, engagement in, and reasoning about KQML conversations.

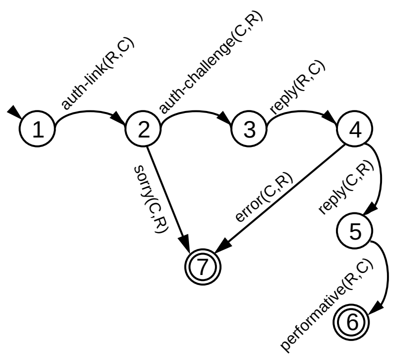


Fig. 1. A deterministic finite–state automaton representing a conversation for authenticating the transmission of a KQML performative

3.1 Conversation Specification

A number of conversation formalisms have been presented in the literature [12, 23]. We believe that a conversation formalism must exhibit two properties to be generally useful for KQML-speaking agents:

1. Conversation specifications should be easy to express in ways that are useful to both people and machines.
2. All agents that participate in a conversation should be able to share a single description of that conversation.

We have developed a conversation specification mechanism that meets these criteria, called *Protolingua.* In Protolingua, conversations are specified as deterministic finite-state automata (DFAs). For example, the self authentication protocol used in Section 2.3 can be expressed graphically, as shown in Figure 1 The linearized form of this DFA, suitable for machine processing, is:

```
start 1;
succeed 6;
fail 7;
1->2: auth_link(R,C);
2->3: auth_challenge(C,R);
2->7: sorry(C,R);
3->4: reply(R,C);
4->5: reply(C, R);
4->7: error(C, R);
5->6: performative(R, C);
```

Note that the DFA does not indicate how messages are generated; it simply dictates what sequences of messages count as engaging in a specified conversation. It is this independence of form from action that allows the same conversation specification to be used by the agents at both ends of the conversation.

We have fully implemented such KQML conversations in Java. A single conversation specification is shared among agents, who can use it to engage in either side of the conversation. Furthermore, conversation specifications can be placed in a hierarchy, with detailed conversations extending simple ones. This system forms the basis for the CIIMPLEX factory floor automation project [13].

3.2 Integrating Security into Conversations

The DFA shown in Figure 1 represents a conversation whose purpose is to authenticate transmission of a single KQML performative. In general, conversations between agents will not be *about* security; rather, they will be about a domain of mutual interest to the conversing agents. For example, a conversation designed to support price negotiation will express offers, counteroffers, and acceptance or rejection of a final price. To integrate authentication into such a conversation, one could simply replace each transition with the entire authentication DFA depicted in Figure 1. Unfortunately, such a brute force approach misses at least three important points about the way that security ought to be incorporated into an agent communication language. First, it obscures the close relationship between the authenticated and unauthenticated conversations. Second (or perhaps as a corollary to the first point), it misses the subservient nature of those portions of the conversation that are devoted to authentication. Third, it is unnecessarily expensive. For example, in a price negotiation, the final agreement on price will likely need to be authenticated. However, offers and counteroffers along the way will not benefit from authentication (at least in an environment where uncompleted sales are not deprecated).

For these reasons, we recommend an approach in which individual arcs of a DFA representing a conversation can be tagged with privacy and authentication requirements. Such requirements are simply references to standard conversations, such as the one depicted in Figure 1. If we view a conversation as an augmented transition network [11], the addition of a security requirement to an edge converts that edge to a push arc.

This approach maintains the close relationship between a basic conversation and its variants that incorporate privacy and authentication, in a way that clearly leaves the security mechanisms subservient to the rest of the conversation. Furthermore, security can be applied selectively; only arcs that are tagged will invoke the appropriate security mechanisms.

A further advantage of this approach is the ease with which it can be embedded in a mediated architecture. If agents are to remain light–weight, they will need to rely on other agents to provide basic services. For example, an authentication agent might serve as an expert in authentication across a wide variety of underlying security substrates. Assuming that authentication of the authentication agent itself was not an issue (perhaps because of judicious use of firewalls), such an agent could be relied upon for authentication of conversations with third parties. Our approach allows such mediation without complicating the conversation specifications shared by the two negotiating agents.

4 Conclusion

The proposed message-level security model addresses privacy, authentication and non–repudiation (if asymmetric key mechanism is used for the master and session keys) in agent communication, at a level that is appropriate for reasoning agents. The limitations of the model, discussed in detail in Thirunavukkarasu [26], were briefly touched on here. The model does not provide a mechanism to exchange credentials, nor does it support non-repudiation of message receipt. Message replay detection requires that recipients are known in advance; this may cause problems in an agent architecture that uses facilitator agents to automatically route messages whose intended recipients are described only in general terms by the sending agent. The security architecture requires that agents maintain state information, *e.g.,* next message ID and next session key, to prevent message replay attack and cipher attack.

Ultimately, the effectiveness of this security model depends on the strength of the underlying cryptologic algorithms and functions. Yet, by expressing the details of the security architecture in a way that agents can reason about, these dependencies need not hamper the high-level development of intelligent agent systems.

Acknowledgments.

This work has been the result of fruitful collaborations with a number of colleagues with whom we have worked on KQML and other aspects of the Knowledge Sharing Effort. Foremost, Chelliah Thirunavukkarasu was the prime developer of the message–level security architecture. We also wish to specifically thank and acknowledge Don McKay, Robin McEntire, Richard Fritzson, Charles Nicholas, Yannis Labrou, R. Scott Cost, and Anupama Potlori. Arulnambi Kaliappan suggested the agent names and and Senthil Periaswamy of the University of South Carolina provided stimulating discussions on possible security threats and attacks. This work was supported in part by the Air Force Office of Scientific Research under contract F49620–92–J–0174, and the Advanced Research Projects Agency monitored under USAF contracts F30602–93–C–0177 and F30602–93–C–0028 by Rome Laboratory.

References

1. Draft specification of the KQML agent communication language, Tim Finin, Jay Weber *et al.,* June 1993, <http://www.cs.umbc.edu/kqml/kqmlspec/spec.html>
2. Security Mechanisms in High-Level Network Protocols, Victor L.Voydock, Stephen T. Kent, ACM Computing Surveys, Vol.15, No. 2, 135-171, June 1983
3. OSTF RFP3 Submission, Corba Security, OMG Document Number 95-3-3, March 1995, <http://www.omg.org/docs/95-3-3.ps>
4. Privacy Enhancement for Internet Electronic Mail: Part I: Message Encryption and Authentication Procedures, J. Linn, October 1993, <http://ds.internic.net-/rfc/rfc1421.txt>

5. Security in a Distributed Computing Environment, OSF-O-WP11-1090-3, <http://www.osf.org/comm/lit/OSF-O-WP11-1090-3.ps>
6. Project Athena Technical Plan, Section E.2.1, Kerberos Authentication and Authorization System, S.P.Miller, B.C.Neuman, J.I.Schiller and J.H.Saltzer, Oct 27 1988, <ftp://athena-dist.mit.edu/pub/kerberos/doc/techplan.PS>
7. Limitations of the Kerberos Authentication System, S.M. Bellovin, M. Merritt, Proceedings of the Winter 1991 Usenix Conference, January 1991, <ftp://research.att.com/dist/internet_security/kerblimit.usenix.ps>
8. Security Service API: Cryptographic API Recommendation, NSA Cross Organization, CAPI Team, Jun 12 1995, <http://www.omg.org/docs/95-6-6.ps>
9. RSA Labs' frequently asked questions (FAQ), <http://www.rsa.com/rsalabs/faq>
10. Software Design Document for KQML, Revision 3.0, Mar 1995, LORAL Corporation, Paoli PA, USA
11. James Allen. *Natural Language Understanding.* Benjamin/Cummings, 1995. Second edition.
12. Mihai Barbuceanu and Mark S. Fox. COOL: A language for describing coordination in multi agent systems. In *Proceedings of the First International Conference on Multi-Agent Systems* (ICMAS '95). 17–24. 1995.
13. B. Chu, B. W. Tolone, R. Wilhelm, M. Hegedus, J. Fesko, T. Finin, Y. Peng, C. Jones, J. Long, M. Matthews, J. Mayfield, J. Shimp, S. Su. Integrating manufacturing softwares for intelligent planning–execution: A CIIMPLEX perspective. In *Plug and Play Software for Agile Manufacturing, SPIE International Symposium of Intelligent Systems and Advanced Manufacturing.* Boston, November 1996.
14. Tim Finin, Don McKay, Rich Fritzson, and Robin McEntire. KQML: an information and knowledge exchange protocol. In Kazuhiro Fuchi and Toshio Yokoi, editors, *Knowledge Building and Knowledge Sharing.* Ohmsha and IOS Press, 1994.
15. Tim Finin, Don McKay, Rich Fritzson, and Robin McEntire. The KQML information and knowledge exchange protocol. In *Third International Conference on Information and Knowledge Management*, November 1994.
16. Michael Genesereth and Richard Fikes. Knowledge interchange format, version 3.0 reference manual. Technical report, Computer Science Department, Stanford University, June 1992.
17. Mike Genesereth. An agent–based approach to software interoperability. Technical Report Logic–91–6, Logic Group, CSD, Stanford University, February 1993.
18. Michael R. Genesereth and Steven P. Katchpel. Software Agents. Communications of the ACM 37(7):48–53, 1994.
19. Daniel R. Kuokka, James G. McGuire, Jay C. Weber, Jay M. Tenenbaum, Thomas R. Gruber, and Gregory R. Olsen. Shade: Technology for knowledge–based collaborative. In *AAAI Workshop on AI in Collaborative Design*, 1993.
20. Yannis Labrou and Tim Finin. A semantics approach for KQML – a general purpose communication language for software agents. In *Third International Conference on Information and Knowledge Management*, November 1994. <http://www.cs.umbc.edu/kqml/papers/kqml-semantics.ps>.
21. James G. McGuire, Daniel R. Kuokka, Jay C. Weber, Jay M. Tenenbaum, Thomas R. Gruber, and Gregory R. Olsen. Shade: Technology for knowledge–based collaborative engineering. *Journal of Concurrent Engineering: Applications and Research (CERA)*, 1(2), September 1993.
22. R. Neches, R. Fikes, T. Finin, T. Gruber, R. Patil, T. Senator, and W. Swartout. Enabling technology for knowledge sharing. *AI Magazine*, 12(3):36–56, Fall 1991.

23. H. Van Dyke Parunak. Visualizing agent conversations: Using enhanced Dooley graphs for agent design and analysis. In *Proceedings of the Second International Conference on Multi-Agent Systems* (ICMAS '96). 1996.
24. Ramesh Patil, Richard Fikes, Peter Patel-Schneider, Donald McKay, Tim Finin, Thomas Gruber, and Robert Neches. The DARPA knowledge sharing effort: Progress report. In B. Nebel, C. Rich, and W. Swartout, editors, *Principles of Knowledge Representation and Reasoning: Proc. of the Third International Conference (KR'92)*, San Mateo, CA, November 1992. Morgan Kaufmann.
25. M.Tenenbaum, J. Weber, and T. Gruber. Enterprise integration: Lessons from shade and pact. In C. Petrie, editor, *Enterprise Integration Modeling*. MIT Press, 1993.
26. Chelliah Thirunavukkarasu. A Security Architecture for KQML. Technical Report MS-EECS-95-nn, Computer Science and Electrical Engineering Department, University of Maryland Baltimore County. August, 1995.

This article was processed using the LaTeX macro package with LLNCS style