A Security Architecture for Agent Communication Language

Tim Finin, James Mayfield and Chelliah Thirunavukkarasu
Computer Science and Electrical Engineering
University of Maryland Baltimore County
Baltimore MD 21250 USA

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 a appropriately rich *agent communication language* (ACL) and associated protocols. For an ACL to 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. The proposed architecture is based on cryptographic techniques and allow agents to verify the identity of other agents, detect message integrity violations, protect confidential data, ensure non-repudiation of message origin and take counter measures against cipher attacks. Many of these security features will be supported by and/or implemented by transport mechanisms (e.g., socket-communication, HTTP, SMTP) on which the ACL will be carried. However, we argue that such security properties must be part of and reflected in the ACL model and can not be totally 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 a appropriately rich *agent communication language* (ACL) and associated protocols. For an ACL to 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?

Several paragraphs saying why security must be modeled in the agent level in addition to the underlying transport levels. Main reason is that agents must be able to reason about the security properties of their agents. Lets us maybe do some useful things, like decide who to talk to and by what means and also to selectively ask for authentication when it is needed. Introduce notion of lazy authentication. TIe into a truth maintenance scenario – only ask an agent to authenticate when you *really* need to verify it is indeed the agent you think it is. etc...

1.2 KQML

Knowledge Query and Manipulation Language (KQML) [1] is a communication language and protocol which enables autonomous and asynchronous software agents to share their knowledge and or work towards cooperative problem solving. It was developed as a part of the *Knowledge Sharing Effort* [20; 19; 13]. 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 programs own representation language. KQML can carry expressions encoded in any representation language, including languages expressed as ASCII strings and those expressed using a binary notation. Some KQML-speaking agents (e.g., routers, very general brokers, etc.) may ignore the content portion of the message, except to determine where it ends.

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.

It is the message layer that is used to encode a message that one application would like to transmit to another. 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 upon the following requirements for a KQML security model based on the analysis of the security models for Privacy Enhanced Mail [4], CORBA [3] and DCE [5]. Interested readers are referred to [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 intensional 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 performative (i.e., an *ask-all* from an agent will entail a *tell* or *sorry* from the receiver). The security model should be general and flexible enough to support different models of agent interaction (e.g contract net, 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 that it should not have hard dependencies on any specific cryptographic algorithm. Similarly, we reject systems which assume a global synchronization of time – it difficult to achieve and leads to further security issues of its own [7].

2 Architecture Overview

The proposed security architecture is based on data encryption techniques [9]. In tune with the asynchronous nature of general ACLs like KQML, the model expects a secure message to be self authenticating and does not support any challenge/response mechanism to authenticate a message after it has been delivered. The architecture supports 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 from cipher attacks.

2.1 Cryptographic background

The following paragraphs define the cryptographic techniques used by this architecture and the new performative and the parameters that have been introduced to implement the architecture.

Encryption Keys. An agent that implements the proposed security architecture should have a master key, K_a, which it would use to communicate with other agents. This key can be based on a symmetric or asymmetric[1] 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 that it communicates with, for better privacy and stronger authentication. If more than two agents share a single master key, any of those agents can masquerade as the other or eavesdrop on all the conversations between the agents sharing the key. If a master key is shared by more than two agents, the strength of security is directly related to the degree of trust between the agents.

If an agent does not share a master key, $K_{a1,a2}$ with another agent, it can use its master key, K_a, or can use the services of a central authentication server to generate such a key. The agents may use different keys in either direction of message flow i.e., $K_{a1,a2}$ is created by *a1* and would be used when *a1* is sending a message to *a2* and $K_{a2,a1}$ is created by *a2* and would be used when *a2* is sending a message to *a1*.

If an asymmetric key mechanism is used, a unique key for each pair of agents is not necessary, as the agent can use the public key of its peer agent to encrypt the message and prevent eavesdropping. It can also use its private key to sign the message and prove its identity to its peer. We assume that the agents know the master key of the other agents. We also suggest a secure mechanism to do master key lookup.

Session key. In the enhanced model, the agents use an additional key, the session key, to ensure privacy, message integrity and proof of identity. The session key can be symmetric or asymmetric and can be generated with the help of the authentication server. The session keys are set up by using a protocol explained later which requires the use of a master key to ensure security.

[1] Public key cryptosystems are a very familiar example of an asymmetric key system.

The agents can use either the session or master key for exchanging messages and must inform the other agent of the key that was used for encryption to ensure proper decryption. When agents exchange keys, they encrypt them using the current session or master key. Keys are never exchanged in clear text form. We recommend the use of the enhanced security model [2] with an expensive master key and a cheap session key which is changed frequently.

Message Id. The message ID is used in the enhanced security model to protect agents from attacks by message replay. When the two agents establish a session key, they also exchange a message ID which the sender would use in the next message. Every message from an agent would carry 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., 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 this 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 Changes to KQML

In order 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, this 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>)
```

[2]This may not always be possible. The enhanced security model cannot be used if the sender of a message does not know who the intended recipient is; i.e in the case of facilitation performatives the facilitator determines the intended recipient and not the message originator.

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 a variable (e.g., $?$), 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. We assume that agent addresses are represented in this ontology with the *address/3* relation:

```
(address <agent>
         <transport>
         <transport-address>)
```

Instances of this relation specify transport addresses for the agent given in the first argument, as in:

```
(address r2d2 smtp r2d2@umbc.edu)
(address r2d2 tcpip (cujo.cs.umbc.edu 8088))
```

These addresses are known to special agents, such as *agent name servers* and *authenticator agents*.

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 (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 is $true$ then the master key is used[3], 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. [4]

[3]An agent would use the master key for encryption if it does not share a session key with the receiving agent or if it does not know the receiver in advance. Under these circumstances, it could use this parameter to help the receiver in choosing the proper decryption key.

[4]We could eliminate the need for this performative if we are willing to send an *achieve* with an embedded *auth-challenge* but this is simply a mater of protocol detail design.

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*.[5]

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 a auth-link (i.e., a challenge), 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.

3 Security model

An implementation should support the following protocol to conform with the basic security model. This model supports authentication, integrity and privacy of data. If asymmetric keys are used for session and master keys, this model also supports non-repudiation of origin.

When R2D2 sends a secure message to C3PO, it would compute a message digest and encrypt 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 in nature). 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., the message was encrypted by the *:sender* agent) of the message. This message can be a replay of a legitimate message previously sent by the generator.

3.1 Enhanced security model

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 by the sender, as a rogue agent can replay a message and we will not be able to detect it.

In [**?**] we demonstrate how the new KQML performatives and parameters can be used to converse/communicate securely, the role of authenticator agents for key registration and management. In the remainder of this section we give two small examples and show how the protocols can be captured in Protolingua.

Self authentication

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.

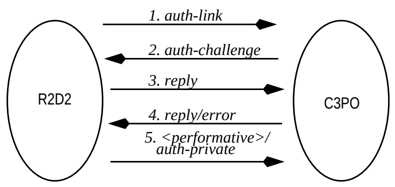

Figure 1: **The self authentication protocol is initiated by an agent who wants to establish its identity before beginning a dialog.**

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

[5] Again, if we were motivated to decrease the number of performatives at the expense of putting more in the authentication ontology, we could represent an *auth-challenge* performative as an *ask-one* with *:content* of *(encrypt ?s ES)* where ES is the encrypted string and the encrypt/2 relation holds between the string with respect to the current key.

```
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 a *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 with the *:auth-digest*, *:auth-msg-id* and new session key (if present) encrypted in 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 which 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>
```

Crypto un-aware agents

Agent Leia may not have crypto capabilities. But it trusts its friend R2D2 and R2D2 is prepared to authenticate messages on behalf of Leia. Since Leia does not have crypto capabilities, it will not accept *auth-private* performative. The agents would follow the handshake protocol given below to verify SkyWalker's identity.

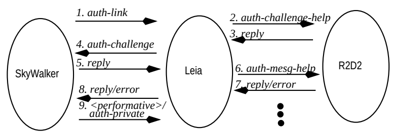

Figure 2: **Using the trusted friend protocol, agent leia asks agent R2D2 for help in authenticating agent Skywalker.**

Agent SkyWalker begins by sending Agent Leia an *auth-link* message to initiate the process of proving its identity to Leia.

```
auth-link           (1)
  :sender SkyWalker
  :receiver Leia
  :reply-with <expression>
```

When Leia receives an *auth-link* message from SkyWalker, Leia requests a challenge string from its trusted friend, R2D2.

```
help  (2)
  :sender Leia
  :receiver R2D2
  :reply-with <expression>
  : ontology authentication
  :content (auth-link
            :sender SkyWalker
            :receiver Leia
            :reply-with <expression>)
```

R2D2 will generate a random string on behalf of Leia, encrypt it using the master key (shared by Leia and SkyWalker or Leia's master key, which R2D2 knows) and will forward it to Leia.

```
reply               (3)
  :sender R2D2
  :receiver Leia
  :in-reply-to <expression>
  :content (SkyWalker <encrypted-random-string>)
```

Leia will construct an *auth-challenge* performative and send it to SkyWalker. Subsequent performative from SkyWalker with an *:auth-digest* will be forwarded to R2D2.

```
auth-challenge      (4)
  :sender Leia
  :receiver SkyWalker
  :reply-with <expression>
  :in-reply-to <expression>
  :content <encrypted-random-string>
```

SkyWalker will respond with a secure *reply*.

```
reply               (5)
  :sender SkyWalker
  :receiver Leia
```

```
    :reply-with <expression>
    :in-reply-to <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
```

Leia will wrap the response in an *help* and send it to R2D2.

```
help           (6)
  :sender Leia
  :receiver R2D2
  :reply-with <expression>
  :ontology authentication
  :content (reply
             :sender SkyWalker
             :receiver Leia
             ... message (5) ...)
```

R2D2 will respond with a *reply* or an *error* (7). Leia would forward the R2D2's reply to SkyWalker (8). The handshake is now complete and SkyWalker can send secure performative to Leia, which Leia would verify with the help of R2D2 (9).

4 Conclusion

The proposed security model addresses privacy, authentication and non-repudiation (if asymmetric key mechanism is used for the master and session keys) in agent communication.

The limitations of our agent security model limitations are discussed in detail in [CHELIAH95] and briefly touched on here. The model does not provide a mechanism to exchange credentials nor support the non-repudiation of message receipt. Message replay detection requires that recipients is known in advance cuasing problems in an agent architecture whichh makes use of facilitator class agents to automatically rout messages whose intended recipients are described 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, this security model depends on the strength of the crypto algorithm, message digest function and the random number generator used by the agent for its effectiveness.

References

[1] Draft specification of the KQML agent communication language, Tim Finin, Jay Weber et al, Jun 15 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, Jun 83

[3] OSTF RFP3 Submission, Corba Security, OMG Document Number 95-3-3, Mar 8 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, Oct 02 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, Jan 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] 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.

[12] 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.

[13] Michael Genesereth and Richard Fikes. Knowledge interchange format, version 3.0 reference manual. Technical report, Computer Science Department, Stanford University, June 1992.

[14] Mike Genesereth. An agent–based approach to software interoperability. Technical Report Logic–91–6, Logic Group, CSD, Stanford University, February 1993.

[15] Michael R. Genesereth and Steven P. Katchpel. Software Agents. newblock Communications of the ACM, v37, n7, pp 48–53, 147, 1994.

[16] 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.

[17] 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. Available as `http://www.cs.umbc.edu/kqml/papers/-kqml-semantics.ps`.

[18] 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.

[19] 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.

[20] 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.

[21] M.Tenenbaum, J. Weber, and T. Gruber. Enterprise integration: Lessons from shade and pact. In C. Petrie, editor, *Enterprise Integration Modeling*. MIT Press, 1993.

[22] 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.

[23] KQML Agent Technology Software. UMBC technical report. 1995.