Expressing Quality of Service in Agent Communication¤

Lisa Cingiser DiPippo and Lekshmi Nair
Department of Computer Science
The University of Rhode Island
Kingston, RI USA 02881

Abstract

This paper presents extensions to a well-known agent communication language for the expression of quality of service. It describes the semantics of the extensions, while allowing quality of service to be interpreted as broadly as possible. The paper then describes the specific extensions to KQML through added performative parameters. A prototype implementation of the extended language is also discussed.

Keywords: agent, communication, quality of service, semantics

1. Introduction

An agent communication language (ACL) provides a mechanism for agents to express their desires and intentions to other agents in a content language independent manner. Agents can converse about what they know and what they want to know from other agents. This sharing of information allows multiple agents to work together to meet common goals, as well as individual goals. However, in some applications, it is not enough for one agent to let another agent know that it wants some information. A requesting agent must also be able to express something about how it wants the information to be delivered. For example, consider a system in which multiple agents communicate to provide stock market information to an end user. It is not enough for a *UserAgent* to request the price of Intel stock from a *QuotingAgent* because the price of stocks changes so rapidly. There must be a way for the *UserAgent* to express that it needs the price information within a certain amount of time, or with a specified degree of accuracy.

In general, in many applications, it is important for an agent to be able to express a desired quality of service (QoS) as part of a communication with another agent. Further, it is also necessary for agents to be able to express the level of quality that it can provide to other agents in the services that it can offer.

In this paper we present a methodology for expressing QoS in the capabilities of agents and in the requirements of agents. Section 2 defines the semantics of QoS in agent communication by extending the semantics of a well-known communication language (KQML). Section 3 presents extensions to KQML that allow for the expression of QoS in the language. Section 4 briefly describes a prototype that we have implemented to demonstrate the use of these language extensions. And Section 5 concludes with a summary and discussion of the applicability of our work.

2. Semantics of Quality of Service

The QoS provided or required by an agent should be an integral part of the communications among agents. This allows communicating agents to "know what they are getting". In this section, we describe the semantics of the Knowledge Query Manipulation Language (KQML) [1], a well-known ACL. We then describe what is meant by QoS in the context of agent communication. Finally, we present an extended semantics of KQML to allow for the expression of QoS capabilities and requirements.

¤ This work is partially supported by the U.S. Office of Naval Research grant N000140010060

2.1. KQML Semantics

KQML is an agent communication language in which agents communicate through the expression of performatives [1]. Each performative specifies the kind of communication the speaking agent wants to have with the receiving agent. For instance, the `tell` performative allows one agent to inform another agent about something it knows about.

The semantics of KQML is based on speech act theory [2]. Cognitive states of KQML-speaking agents are expressed using a meta-language of operators that specify propositional attitudes [2]. These operators express the *beliefs*, *knowledge*, *desires* and *intentions* of an agent.

The meta-language operators are used to describe the semantics of the performatives through *preconditions*, *postconditions* and *completion conditions*. For further details on the specifics of KQML semantics, see [2]. We more fully discuss the semantics in Section 2.3 through our extension of KQML.

2.2. Quality of Service

Quality of service is a broad term that can encompass many criteria within a multi-agent system. It can be used to express timing capabilties of an agent, or the accuracy of a response that an agent can provide. For example, if an agent can find the price of a requested stock within 10 seconds, this can be expressed as a QoS capability. Furthermore, if the same agent can find the price of the same stock more quickly, with slightly lower accuracy, this can be expressed through QoS as well. QoS can also express other criteria such as level of security and network bandwidth.

For the purpose of expressing agent communications with QoS, we do not distinguish among the different criteria that can be expressed. Rather, we use a general QoS parameter in the specification of agent communication semantics. Quality of service is treated as a general concept, and can be interpreted in any semantic context as "level of quality" provided by or required by an agent.

2.3. Extended Semantics

To express QoS as an integral part of agent communication, we have extended the semantics of KQML to include a "level of quality" parameter in the meta-language for expressing agents' states, and in the expression of pre-, post- and completion conditions for KQML performatives.

2.3.1. QoS in Agent State

The cognitive state of an agent is expressed through beliefs, knowledge, desires and intentions. The following operators are extended from [2] to express QoS as a part of the agent's state.

- BEL(A,P) – Agent *A* believes *P* to be true.
- KNOW(A,S) – *A* has some knowledge about *S*, where *S* is a state description.
- WANT(A,S,Q) – A desires state (or action) *S* to occur in the future, with a level of quality *Q*.
- INT(A,S,Q) – A intends on doing *S* with a level of quality *Q*.

The belief (BEL) and knowledge (KNOW) operators refer to the current state of the agent's knowledge. They do not require any extension for expression of QoS. On the other hand, it is necessary for desires (WANT) and intentions (INT) to be able to express QoS. In the context of QoS expression, it is not enough to say that agent A wants to know something about S, or wants action S to occur. We must be able to express when or how A wants to know about S. Similarly, when expressing intentions, we must be able to express that agent A intends to do S within a specified level of quality.

2.3.2. QoS in Agent Performatives.

We now use the agent state operators described above to express the semantics of KQML performatives that can express QoS capabilities and requirements. In [2], the semantics of a performative are described with (1) a natural language description, (2) a fomalization of the description, (3) a set of preconditions, (4) a set of postconditions, and

(5) a completion condition. We use these same descriptors to characterize the extensions that we have made to the performative semantics. We present the semantics for three performatives: `ask-if`, `tell` and `advertise`. These semantics are representative of the extensions that we have made to all of the performatives in KQML for the expression of QoS.

We begin by presenting the semantics for `ask-if`, followed by an explanation of the specified conditions.

`ask-if(A,B,X,Q)`

1. *A* wants to know, with level of quality *Q*, what *B* believes about the truthfulness of *X*.
2. WANT(A,KNOW(A,Y),Q)
   Where *Y*=(BEL(B,X)) or *Y*=(~BEL(B,X))
3. Pre(A): WANT(A,KNOW(A,Y),Q) ^ KNOW(A,INT(B,PROC(B,M),Q))
   Where M = ask-if(A,B,X)
   Pre(B): INT(B,PROC(B,M),Q)
4. Post(A): INT(A,KNOW(A,Y),Q)
   Post(B): KNOW(B,WANT(A,KNOW(A,Y),Q)
5. Completion: KNOW(A,Y)

The above preconditions imply that before agent *A* sends an `ask-if` message, it wants to know something about *X* within a certain level of quality, and it knows that *B* can process the request within this level of quality. Also, *B* intends to process an ask-if message from *A* about *X* with level of quality *Q*. The postconditions indicate that after the ask-if message is sent, *A* intends to know something about *X* with the specified level of quality, and *B* knows that *A* wants to know something about about *X* with that level of quality. The completion condition, which specifies the result of the conversation in which this message exists, indicates that when the conversation is over, *A* will know something about *X*.

For example, if agent A asked agent B for the price if Intel's stock within 15 seconds, the above semantics imply that in A wants to know what B knows about the stock price of Intel within 15 seconds, and that B has expressed its intention to provide this stock price, perhaps through an advertisement.

The extended semantics for the `tell` and `advertise` performatives are shown below. The interpretation of the pre-, post- and completion conditions are similar to those for ask-if, so we do not explain the semantics further.

`tell(A,B,X,Q)`

1. *A* states to *B*, with a level of quality Q, that *X* is true.
2. BEL(A,X)
3. Pre(A): BEL(A,X) ^ KNOW(A,WANT(B,KNOW(B,Y),Q))
   Where Y=BEL(A,X) or Y=~BEL(A,X)
   Pre(B): INT(B,KNOW(B,Y),Q)
4. Post(A): KNOW(A,KNOW(B,BEL(A,X)))
   Post(B): KNOW(B,BEL(A,X))
5. Completion: KNOW(B,BEL(A,X))

To continue the example of above, if agent B tells agent A the stock price of Intel, it is expressing its belief about the stock price. The semantics also imply that B knows that A wants this information within 15 seconds.

`advertise(A,B,M,Q)`

1. *A* states to *B* that it can (and will) process the message *M* from *B* with level of quality *Q*.
2. INT(A,PROC(A,M),Q)
   Where M is a performative
3. Pre(A): INT(A,PROC(A,M),Q)
   Pre(B): none
4. Post(A): KNOW(A,KNOW(B, INT(A,PROC(A,M),Q)))
   Post(B): KNOW(B,INT(A,PROC(A,M),Q))

The advertise performative can be used to allow an agent to inform another agent about its capabilities. It can also be used to allow an

agent to register its capabilities with a facilitator agent that helps match agent requests with servicing agents.

3. Extending KQML to Express QoS

The semantics expressed in the previous section provide a foundation for extending KQML performatives to express QoS capabilities and requirements. In this section we briefly describe the agent model on which our work is based. We then explain how we have extended KQML performatives with a QoS parameter. We show examples of the the extension for several specific performatives.

3.1. QoS Agent Model

Our QoS extensions to KQML are based upon a model of a real-time multi-agent system (RTMAS) that we have developed [3]. The model is based on the assumption that agents may be able to perform their tasks in multiple ways. It is made up of a set of real-time agents (*RTAgent*) and a set of communications among the real-time agents (*Message*). Figure 1 displays the elements of the model.

$RTAgent = \{S_1, S_2, \ldots, S_n\}$

$S_i = < O, ES>$

$ES = \{es_1, es_2, \ldots, es_f\}$

$es_i = <ex, a, tv>$

$$tv_i = \frac{(a_i - a_{i+1}) / a_i}{ex_i - ex_{i+1}}$$

$Message = <A, V, Q>$

$Q = <I, D, H>$

Figure 1 - RTMAS Model Elements

3.1.1. RTAgent

Each *RTAgent* is comprised of a set of *solvables*, $\{S_1, S_2, \ldots, S_n\}$ where a solvable is a problem that the agent is designed to solve. Each solvable within the agent is represented by an optimal result (*O*) and a set of execution strategies (*ES*). The optimal result is a system-specific definition of what is considered to be the best result for this problem. For instance, in a system in which agents buy, sell and recommend stocks, a *BuySellAgent* may have a solvable, *BuyStock*, to purchase a specified stock. The optimal solution in this scenario might be to buy the stock at the current price with no fee.

The *ES* component of a solvable is the set of execution strategies that can be used to produce a result for a solvable. For example, the solvable *BuyStock* may have an execution strategy, BS_1, that uses a discount broker with a low fee. This execution strategy may come close to the no fee requirement of the optimal result, but if the discount broker typically has a longer turn around time, then the deadline of the *BuyStock* request may be violated and the price of the stock may have changed. On the other hand, an execution strategy, BS_2, that uses a more expensive broker may be able to handle the request more quickly.

Each execution strategy of a solvable is comprised of three elements. This model uses criteria for expressing real-time constraints. However, it can easily be modified to handle other measures of QoS. The execution time, *ex*, represents the amount of time it takes a strategy to run. The level of accuracy, *a*, is a rating of the result of an execution strategy. Accuracy is calculated as a percentage of the optimal result (*a = strategy result / optimal result*). In the example above, we quantify the optimal result of the *BuyStock* solvable by specifying zero fee for the transaction. While this optimal result may be impossible to achieve, it provides a metric by which to measure the results of the actual execution strategies. The accuracy of a result of a particular execution strategy may be known a priori. In other cases the accuracy can be based upon a statistical average of returned results.

The last component of an execution strategy is the tradeoff value (*tv*). This parameter provides a measure of how much value would be lost by choosing one execution strategy of a solvable over another one. That is, it measures the percent change in accuracy per change in execution strategy execution time.

3.1.2. Message

Communication among agents in this model is performed through messages sent between agents. The formal specification for a message is displayed in Figure 1. *A* represents the name of the real-time agent to which the message is directed. *V* is the name of the solvable that the requesting agent wants to be performed, and *Q* is the QoS requirement of the message. While *Q* can represent any number of QoS constraints, we specifically represent three parameters that model real-time agent behavior. *I* is the level of importance of the request. This value is based on some system-wide scale of importance agreed-upon by all agents. *D* represents the deadline by which the request must be completed. *H* specifies the accuracy threshold for the request. If the servicing agent cannot provide at least this accuracy, then the requesting agent may choose to abort the request.

As an example of a real-time agent message, consider a *UserAgent* in the stock trading example that sends a message to a *QuotingAgent* to find the price of Intel stock (*GetPrice*). The deadline that the *UserAgent* specifies on this message may be based on the amount of time available before a decision must be made. The *UserAgent* may specify an accuracy threshold that allows for a quarter of a point difference from the actual stock price in order to meet its deadline. The importance of the request depends upon the overall transaction that the *UserAgent* is attempting to perform. If the transaction involves spending a few hundred dollars, then the importance may be low. But if it involves thousands of dollars, the importance may be higher.

3.2. QoS in Performatives

The expression of QoS in agent communication is in the form of an agent making known its capabilities, and of an agent specifying its requirements to another agent. We have extended KQML by adding two optional parameters to each performative [4]. We refer to the extended language as KQML-Q. The first parameter is `QoS_requirement`, which allows an agent to specify the level of quality that it requires from the other agent to whom it is sending a message. This parameter is meant to be used with performatives that specify a request for information from another agent. The other parameter that we have added is `QoS_capabilities`. This parameter is designed to be used with an `advertise` performative to allow an agent to let other agents know what levels of quality it can provide for a particular request.

In our examples, and in our implementation, we have considered timing information and accuracy as our QoS measures. However, we have designed the KQML extension in such a way that other measures can be added easily.

3.2.1. Agent Message

In our RTMAS model, there are three kinds of QoS constraints: deadline, importance, and accuracy threshold. The `QoS_requirement` parameter includes these constraint specifications. Consider an example in the stock trading system in which a *UserAgent* requests information from another agent (*TrendWatcher*) that keeps track of trends in a particular segment of the stock market. The following example shows how the *UserAgent* would ask the *TrendWatcher* to report on current trends in internet stocks within 15 seconds, providing at least 75% accuracy.

```
(ask-one
  :sender          UserAgent
  :receiver        TrendWatcher
  :content         Watch(internet)
  :QoS_requirement (dl 15,imp 4,acc 75))
```

Note that in our examples, we leave out some performative parameters for brevity.

The *TrendWatcher* could respond to this request with a the following message:

```
(tell
  :sender          TrendWatchingAgent
  :receiver        UserAgent
  :content         ReportTrend(35)
  :QoS_requirement (dl 5,imp 4,acc 75))
```

This `tell` message expresses a QoS parameter because agent communication is asynchronous, and therefore all messages must be sent explicitly. All KQML-Q performatives

may express QoS constraints in the form of the `QoS_requirement` parameter, so that they can be scheduled to meet their constraints.

3.2.2. Advertisement

Communication between agents, and from agents to facilitators is extended to allow for expression of QoS capabilities. Agents that provide some services to other agents may advertise to other agents, or to a facilitator. For example, if an agent that can buy and sell stocks (*BuyerSeller*) wants to advertise to a facilitator that it can buy a stock with two explicit execution strategies, one with execution time 5 seconds and 85% accuracy, the other with execution time 2 seconds and 65% accuracy, the `advertise` performative would be as follows:

```
(advertise
  :sender      BuyerSeller
  :receiver    Facilitator
  :content     BuyStock(A)
  :QoS_capabilities(
              (ex 5, acc 85)
              (ex 2, acc 65)))
```

The facilitator can use this information to match requests to buy a stock given specific QoS specifications with this *BuyerSeller* agent.

4. Implementation

This section describes the implementation of a prototype RTMAS that allows agents to communicate through KQML-Q. The implementation is based on the KCobalt system [5] that maps KQML messages to CORBA Interface Definition Language (IDL) [6].

All agents in our implementation are represented as CORBA [6] objects whose interface contains a method for each KQML-Q performative. The IDL interface for an agent object in our implementation extends the IDL of KCobalt, and includes the following specifications:

```
interface CoreS {
   void askOne ( in string sender,
      in string receiver,
      ...
      in string qos_Info);
...  }
```

This segment of IDL code shows the ask-one method of an agent object. Each other KQML-Q performative is represented as a method as well, with a string parameter for each performative parameter. We have extended the specification of each KCobalt performative interface to provide a parameter for expression of QoS constraints. Figure 2 displays the flow of control in our implementation. When a real-time agent object expresses a KQML-Q string (1), a parser object parses the string, determines what performative is being requested and forwards the information to a dispatcher object (2). Given the QoS specification in the KQML-Q message, the dispatcher calls a scheduling object (3) that provides real-time scheduling parameters to the system. The scheduling object also provides information that will be used by the servicing agent to determine which execution strategy will meet the QoS specifications of the requesting agent (4). Finally, the dispatcher calls the method corresponding to the requested performative on the servicing agent, with the QoS information provided by the scheduler. Further details on scheduling our real-time agents can be found in [6,7].

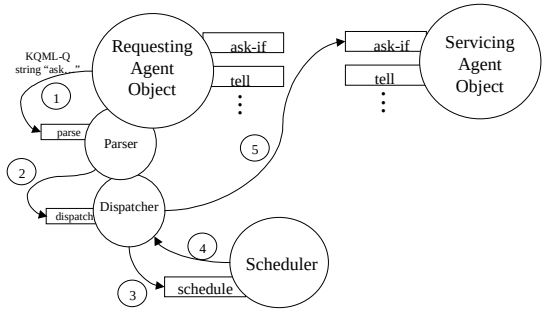

Figure 2 - Implementation

5. Conclusion

In this paper we have presented a way of expressing quality of service in agent communication. By making clear the semantics of this expression, we were able to easily show how the KQML language could be extended. The semantics of QoS that we have presented are independent of the kind of quality that is being expressed. While we have specifically focused on real-time characteristics to define quality, the QoS extension to KQML is flexible

enough to be easily modified to handle any expression of quality that is appropriate for a particular application.

We have chosen to apply our QoS extensions to KQML because it is a widely accepted agent communication language that has many current implementations. We are confident that the spirit of this work would apply equally well to the only other well-known agent communication lanaguage, FIPA's ACL [8]. While the semantics of FIPA-ACL are somewhat different from KQML semantics, we feel that the expression of QoS requirements and capabilities is sufficiently straightforward to apply to FIPA-ACL semantics as well.

References

[1] Y. Labrou, T. Finin, Y. Peng. Agent Communication Languages: The Current Landscape. *IEEE Intelligent Systems*. March/April 1999, pp. 45-52.

[2] Y. Labrou T. Finin. Semantics for an Agent Communication Language, *Proceedings of the Fourth International Workshop on Agent Theories, Architectures and Languages*, Providence, RI, July 1997.

[3] L.C. DiPippo, V. F.-Wolfe, L. Nair, E. Hodys and O. Uvarov. A Real-Time Multi-Agent System Architecture for E-Commerce Applications, *Proceedings of the The Fifth International Symposium on Autonomous Decentralized Systems*, March 2001.

[4] L. Nair, Extending ACL to Support Communication in a Real-Time Multi-Agent System. *University of Rhode Island Technical Report TR00-279*, Dec. 2000.

[5] D. Benech, T. Desprats and Y. Raynaud. A KQML-CORBA based Architecture for Intelligent Agents Communication in Cooperative Service and Network Management. *Proceedings of the IFIP/IEEE MMNS'97*, Montréal, Canada, July 97.

[6] OMG. *Common Object Request Broker Architecture – Version 2.2*. OMG, Inc., 1998.

[7] E. Hodys. A Scheduling Algorithm for a Real-Time Multi-Agent System. *University of Rhode Island Technical Report TR00-275*, Aug. 2000.

[8] FIPA. FIPA Agent Management Specification [FIPA00023]. *Foundation for Intelligent Physical Agents*, 2000. http://www.fipa.org/specs/fipa00023/.