

# Requirement Specification for the DECAF Matchmaker

Foster McGeary  
Computer and Information Sciences Department  
University of Delaware  
{mcgeary@cis.udel.edu}

June 6, 2000

## Abstract

This report states the functionality provided by the DECAF [7] Matchmaker agent. It updates requirement specification version 1.2 and is necessitated by recent extensive development of the DECAF software and concomitant changes to the Matchmaker agent. The creators of the original Matchmaker specification [9] graduated and did not participate in this report, which is a major overhaul of that earlier work. Extensive development of DECAF was done by John R. Graham with support from National Science Foundation Grant No. IIS-9812764. The present work was funded from National Science Foundation Grant Nos. IIS-9733004 and IIS-9812764.

## Document history

Ver.	Author(s)	Date	Comments
1.2	Laukkanen & Eskelinen	Through May 1, 1999	See original [9].
2.00	Foster McGeary	June 1999 - May 2000	Enhance & conform to DECAF2

## 1 Introduction

This document specifies the requirements for the Matchmaker included in the DECAF [7] Agent Framework. Matchmaker is an agent constructed according to DECAF specifications, and DECAF is a specific implementation of KQML [8]. The specifics of this Matchmaker apply to DECAF but may not apply to other KQML implementations.

Section 2 is an overview of Matchmaker and its function in DECAF. The matchmaking algorithm is presented in Section 3. Section 4 describes the various Matchmaker functions, with subsections for each of the KQML performatives implemented. Since DECAF uses plan files to assist in the specification of agent task and actions, segments of plan files for Matchmaker are included in this Section. Section 5 explains the structures of the two databases used by Matchmaker.

## 2 Matchmaker Description

Matchmaker is a well-known client-driven agent that exists to facilitate client agents finding the names of other agents with specified capabilities. To do this, Matchmaker stores information about the capabilities of agents. When an agent that performs a service for other agents begins operation, it sends an advertisement stating its capabilities to Matchmaker. Matchmaker stores this advertisement in its own database, deleting it only if an unadvertisement for a specific task(s) from the original advertising agent is received. After Matchmaker stores the advertisement, other agents can make queries to Matchmaker to find out which agent(s) have current

advertisements to provide specified services. Matchmaker responds to this kind of query by telling the name(s) of the matching agent(s) to the requestor. Excluding the ANS (Agent Name Server), Matchmaker is the only agent in the current DECAF architecture required to have a well known name. In general, agents learn the names of other agents by interrogating the Matchmaker.

In addition to receiving and storing advertisements, Matchmaker is to be able to receive and store subscriptions from other agents. When an agent sends a subscription to Matchmaker, the sending agent is requesting Matchmaker to inform it, the requesting agent, every time some specific service is advertised. Usually advertisements will be made whenever an agent with the subscribed-to capabilities begins operation and has advertised to Matchmaker.

## 2.1 Overview of the Interfaces

Matchmaker waits for incoming KQML messages from other agents. The diagram showing relationships between Matchmaker and other agents in the agent architecture is presented in Figure 1. Descriptions of interfaces between Matchmaker and all client agents, including sample KQML messages, are presented in the second subsequent section.

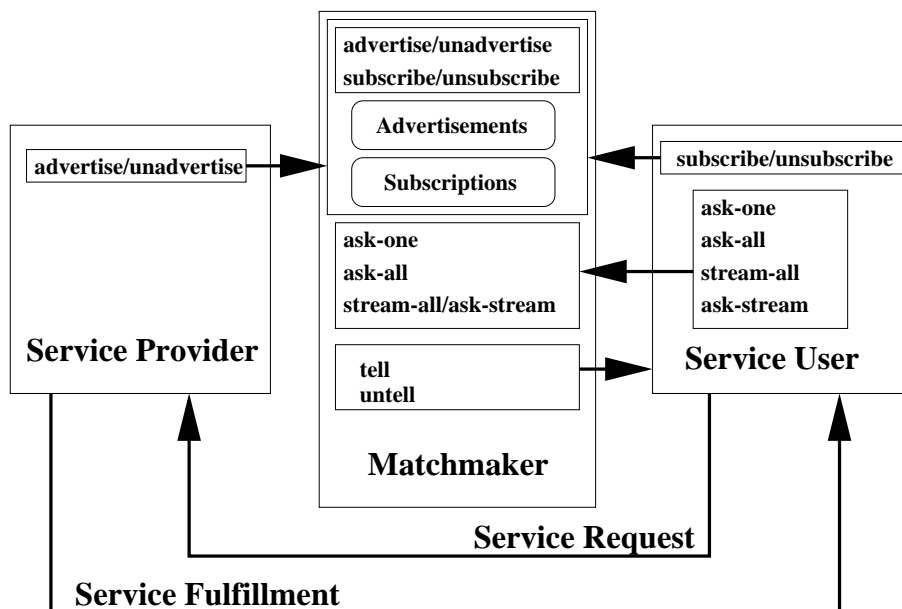


Figure 1: The agent architecture and interfaces between Matchmaker and other agents.

The figure is intended to show that databases for Advertisements and Subscriptions are maintained by Matchmaker using input provided by `advertise`, `unadvertise`, `subscribe`, and `unsubscribe` messages from other agents. Service Providers only advertise to Matchmaker. Service Users can subscribe or use `ask` messages to obtain information on available agents. Service Requests and Service Fulfillments do not involve the Matchmaker. This non-involvement of the Matchmaker in conversations among other agents is the principal reason Service Users and Service Providers need Matchmaker services.

Messages to and from the Matchmaker obey strict construction rules. All messages must conform to the form shown, with the client agent only able to control the content of fields within the angle braces on the KQML message specification.

### 3 The Matchmaking Algorithm

DECAF's Matchmaker selects matching agents by determining the advertised agents that best fit the requested service. The algorithm is quite simple: Matchmaker compares keywords from the query to the keywords in Matchmaker's database. The comparison may result in a perfect or a partial match. The algorithm returns all matches (sorted in descending order by the percentage of keywords matched) up to the number of responses that was specified in the query. If no partial matches that strictly exceed the minimum match percentage are found, then an empty list is returned.

The algorithm for the matchmaking is:

Input: Q = a list of keywords in the query,  
N = the amount of agents required,  
P = threshold percentage.  
Output: A list of tuples {agent\_name, match\_percentage}.

Calculate\_match:

1. L = {}, match\_count = 0, n = nbr. of keywords in the query.
2. FOR each entry e in the advertisement database DO:
  - 2.1. FOR each keyword k in Q DO:
    - 2.1.1. IF k is present in e THEN match\_count = match\_count + 1.
  - 2.2. L = L + {agent name in e, match\_count/n}.
  - 2.3. match\_count = 0.
3. SORT L in descending order BY match\_percentage.
4. IF the match\_percentage of the first element of L is > 0 AND N > 0 THEN RETURN N first elements of L.  
ELSE IF match\_percentage of the first element > P AND P > 0 THEN RETURN elements where match\_percentage > P  
ELSE RETURN empty list.

Here is an example query and the function of the algorithm:

Query:	auto	oil	change	
in DB:	auto	repair	oil	
	1	0	1	-> 2/3 = 66%
	auto	oil	change	24h
	1	1	1	0 -> 3/3 = 100%
	auto	oil	change	12h
	1	1	1	0 -> 3/3 = 100%
	auto			
	1			-> 1/3 = 33%

If two matches are required, then the second and the third agents are returned in this order. If threshold percentage is 50, then the second, third and first agents are returned.

### 4 Matchmaker Functions

This section describes the Matchmaker functions. The subsections describe and illustrate queries that Matchmaker must be able to handle.

## 4.1 Advertise

A KQML ADVERTISE performative is sent to Matchmaker by an agent that wants to advertise its capabilities to the other agents. The advertisement consists of two parts.

- **Keywords.** The keywords are used in the matching algorithm to determine if this is a desired service for the requestor.
- **Service Request form.** Once a desired service has been found, the requesting agent needs to know how to request the service from the service provider. The Service Request form is returned to the requestor when a match is made and tells how to request a service.

Note that if an agent wants to advertise multiple tasks, it must send multiple advertisements.

When Matchmaker receives an advertisement, the information is stored in the advertisement database. No tell message is sent back to the advertiser, as the advertisement is not acknowledged. Also upon receipt of an advertisement, Matchmaker performs matchmaking between the subscription database entries and the keywords of this received advertisement. For each match found, the corresponding agent from the subscription database is informed. A plan that handles advertisement is presented in Figure 2. The TELL performative is discussed in Section 4.5.

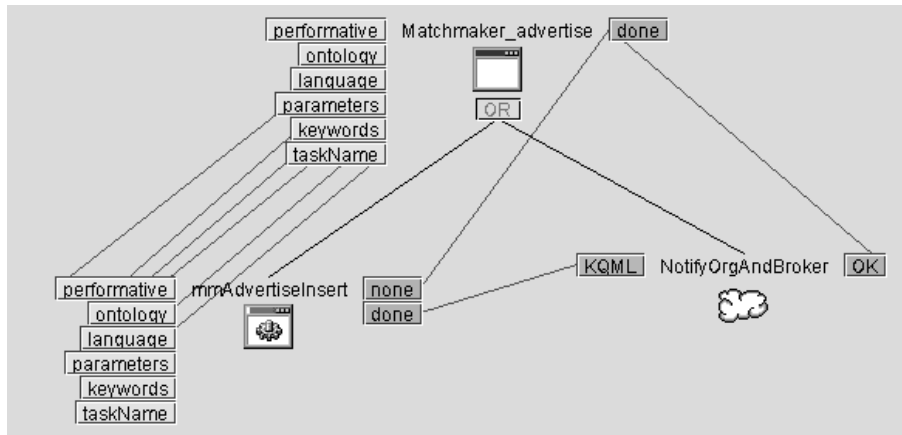


Figure 2: An advertise plan.

The KQML message for the ADVERTISE performative is:

```
(advertise
  :sender      <sending_agent>
  :receiver    Matchmaker
  :ontology    Matchmaker
  :language    DECAF
  :content     (:performative <performative>
                :ontology <ontology>
                :language DECAF
                :taskName <task_name>
                :parameters param_1 param_2 ... param_n
                :keywords word_1 word_2 ... word_n
              )
)
```

## 4.2 Unadvertise and Untell

The KQML UNADVERTISE performative is used to inform Matchmaker that the agent wants to remove its advertisement. An agent can unadvertise one task at a time. When Matchmaker receives an unadvertisement from an agent, the corresponding entry is deleted from the advertisement database. Unadvertisements are not acknowledged by Matchmaker. Subscribing agents are informed of the removal of the advertisement by Matchmaker sending an untell message to each subscription to which a comparable tell message would have been sent when the advertisement was made. Agents that learned of the unadvertising agent through ask operations are not informed of the unadvertisement, as no record is kept of such messages. A plan that handles unadvertisement is presented in Figure 3. No acknowledgement is sent for successful unadvertisements.

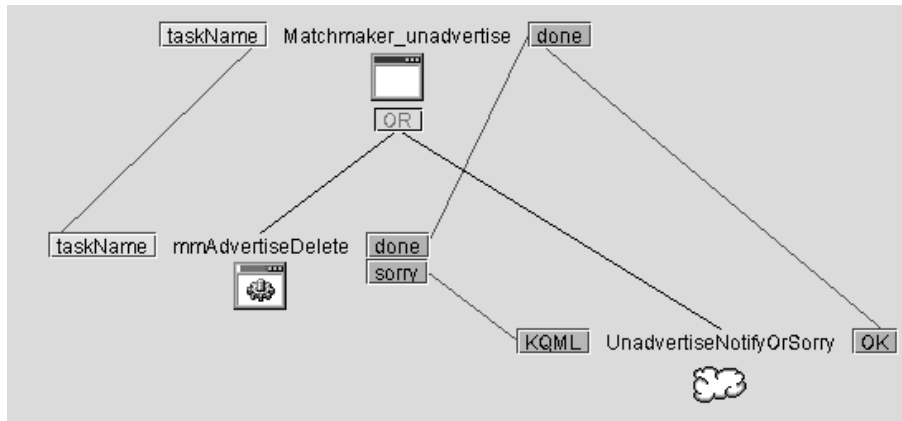


Figure 3: An unadvertise plan.

The KQML message for the unadvertisement is:

```
(unadvertise
  :sender      <sending_agent>
  :receiver    Matchmaker
  :ontology    Matchmaker
  :language    DECAF
  :content     (:taskName <task_name>
               )
)
```

The KQML message for the untell is:

```
(untell
  :sender      Matchmaker
  :receiver    <subscribing_agent>
  :ontology    Matchmaker
  :language    DECAF
  :in-reply-to <subscription_message_id_number>
  :content     (
                :matches 1
                :agent0  (:agentName <agent_name1>
                          :taskName <root_task_name>
                        )
                )
)
```

### 4.3 Subscribe

Subscribing means that an agent requests to be informed if some specific service comes available on the agent environment. The subscription is presented as a list of keywords as in the normal ask query. The subscriber determines the quality of matches by giving a threshold percentage that is used in the matchmaking process.

The incoming information is stored into the subscription database. After this, the advertisement database is scanned through and checked if there is already an agent that matches the subscription. If there is, the subscriber is informed right a way by using the TELL performative (tell is explained in Section 4.4).

A plan that handles subscription is presented in Figure 4.

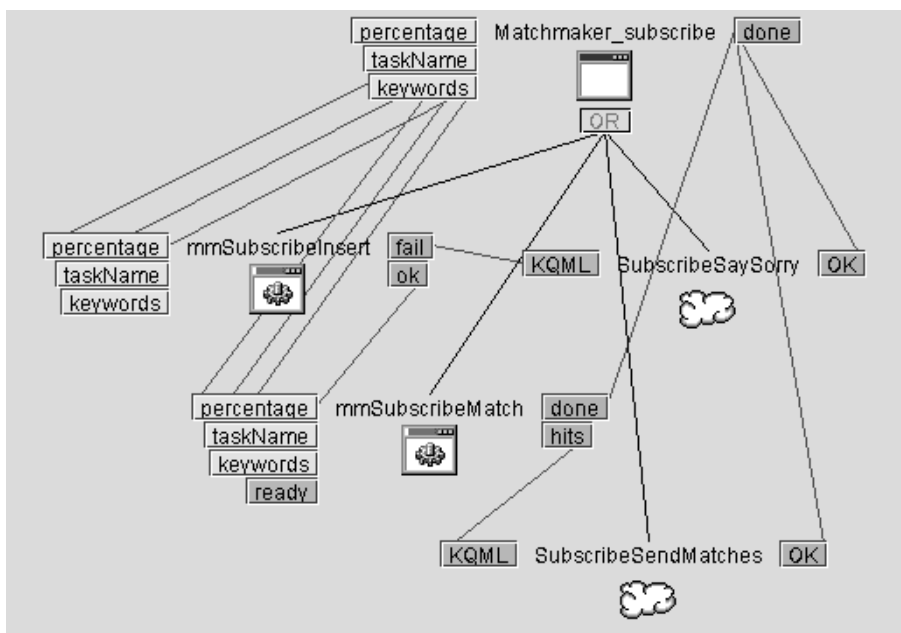


Figure 4: A subscribe plan.

The KQML message for the subscription is:

```
(subscribe
  :sender      <sending_agent>
  :receiver    Matchmaker
  :ontology    Matchmaker
  :language    DECAF
  :reply-with  <id_number>
  :content     (:taskName <name of subscribing task>
                :keywords word_1 word_2 ... word_n
                :percentage <float>
              )
)
```

## 4.4 Unsubscribe

The unsubscription is used to inform Matchmaker that an agent no longer needs to be informed for the subscribed service. Note that UNSUBSCRIBE is not a standard KQML performative. When Matchmaker receives an unsubscription, the corresponding agent's entry is deleted from the subscription database. The agent is informed of the deletion by an untell message, which is presented after the unsubscribe message.

A plan that handles unsubscription is presented in Figure 5.

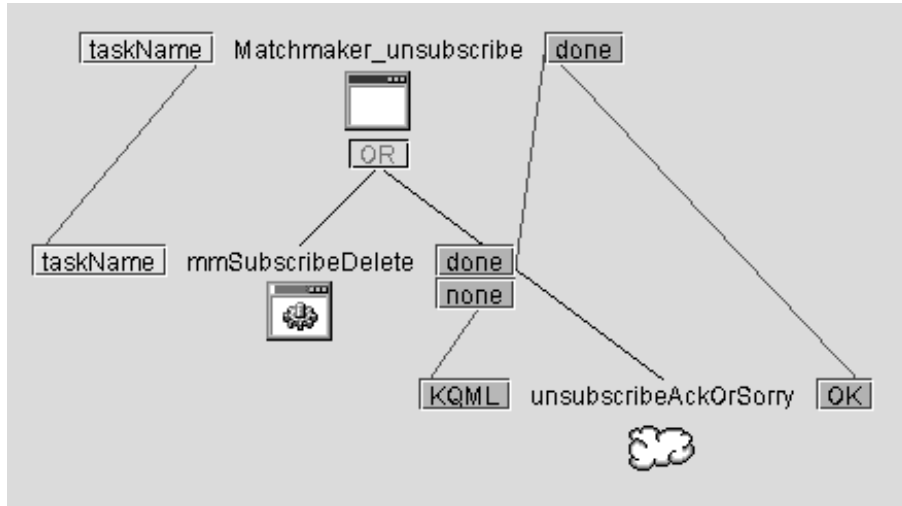


Figure 5: An unsubscribe plan.

The KQML message for the unsubscription is:

```
(unsubscribe
  :sender      <sending_agent>
  :receiver    Matchmaker
  :ontology    Matchmaker
  :language    DECAF
  :reply-with  <id_number>
  :content     (:taskName <from Subscription>)
)
```

The entry corresponding to the agent name (taken from the :SENDER field) is deleted.

There is no KQML message for the confirmation of unsubscription.

## 4.5 Ask, Tell, and Sorry

An agent wanting to locate service providers sends an ASK query to Matchmaker. The requester can ask for just one match by sending the ASK-ONE message, which still requires the threshold percentage for the match. If the requester wants multiple matches, ASK-ALL and STREAM-ALL messages can be used. The difference between ASK-ALL and STREAM-ALL is that in the former, KQML messages are sent in one message, and in the latter multiple KQML messages are sent. Using any of these two performatives, the requester can ask for certain amount of providers or specify a certain threshold percentage corresponding to the quality of the match.

Matchmaker receives a list of keywords and checks for matching keywords in the advertisement database. If matches are found, the requester receives a list (based on the performative, threshold and the quantity) of

agents with advertisements matching the query. If the performative is ASK-ONE or ASK-ALL, agent names and the information about how to request their services are included in one KQML message. If the performative is STREAM-ALL, then the information is sent in multiple KQML messages, one match per message. However, if there are no matching agents, a SORRY message is sent back to the requester. In any event, a response is received by the requesting agent.

A plan that handles the ASK-ONE performative is presented in Figure 6.

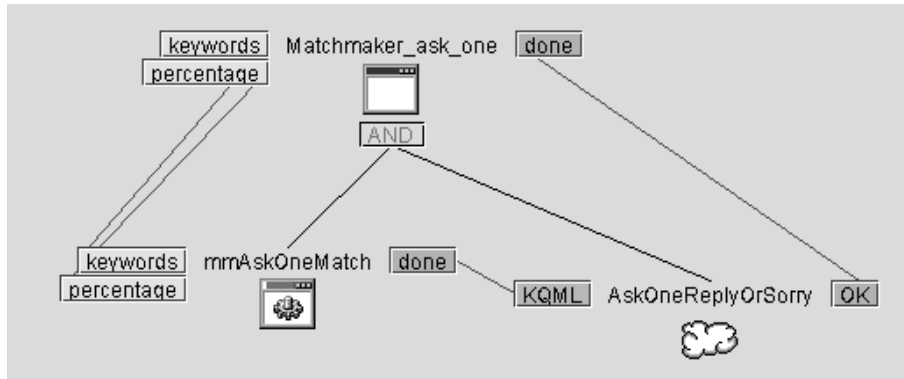


Figure 6: An ask-one plan.

The KQML message for the ASK-ONE query is:

```
(ask-one
  :sender      <sending_agent>
  :receiver    Matchmaker
  :ontology    Matchmaker
  :language    DECAF
  :reply-with  <id_number>
  :content     (:percentage <float>
                :keywords word_1 word_2 ... word_n
               )
)

(ask-all
  :sender      <sending_agent>
  :receiver    Matchmaker
  :ontology    Matchmaker
  :language    DECAF
  :reply-with  <id_number>
  :content     (:percentage <float>
                :keywords word_1 word_2 ... word_n
                :agentNbr <integer>
               )
)
```

The STREAM-ALL performative differs from ASK-ALL only in the name of the performative.

The KQML message for the TELL response is:

```
(tell
  :sender      Matchmaker
  :receiver    <requesting_agent>
  :ontology    Matchmaker
  :language    DECAF
  :in-reply-to <id_number>
  :content     (
    :agentN (:percentage <float>
             :agentName <agent_nameN>
             :performative <performative>
             :ontology <ontology>
             :language DECAF
             :taskName <root_task_name>
             :parameters param_1 param_2 ... param_n
            )
    ...
    :agent1 (:percentage <float>
             :agentName <agent_name2>
             :performative <performative>
             :ontology <ontology>
             :language DECAF
             :taskName <root_task_name>
             :parameters param_1 param_2 ... param_n
            )
    :agent0 (:percentage <float>
             :agentName <agent_name1>
             :performative <performative>
             :ontology <ontology>
             :language DECAF
             :taskName <root_task_name>
             :parameters param_1 param_2 ... param_n
            )
    :matches <integer>
  )
)
```

The KQML message for the sorry response to an ASK performative is:

```
(sorry
  :sender      Matchmaker
  :receiver    <requesting_agent>
  :ontology    Matchmaker
  :language    DECAF
  :in-reply-to <id_number>
  :content     (:task sorry
               :matches 0
              )
)
```

Asking agents can test for either the SORRY performative or the number of matches being zero. In both cases, no matches were found.

## 5 Databases

Matchmaker uses two databases. The advertisement database contains information about agent names and their advertisements. Advertisements are represented as lists of keywords. In addition to the agent names and keywords, the advertisement database contains fields in the outgoing KQML message, and are needed to respond to queries about advertised capability.

The subscription database contains agent names and their subscriptions. A subscription is represented as a list of keywords and a threshold percentage. There is also a task name, so that requesters can dynamically format messages.

**Database for the Advertisements:** Entries in the advertisement database (ADVERTISEMENTDB.TXT) contain the name of the agent, its advertisement represented as a list of keywords, performative, ontology, language, root task name and parameters. The database is an ASCII file, where a line corresponds to an entry. Fields are separated by semi-colons, and list elements by a space. All the other fields are represented as strings without spaces. An example of a database entry for agent WillyLoman who seeks work as a salesman or a broker in a Sales ontology is:

```
WillyLoman;salesman broker;achieve;Sales;DECAF;Worker;hireas
```

**Database for the Subscriptions:** The entries in the subscription database (SUBSCRIPTIONDB.TXT) contains the name of the agent and its subscriptions represented as a list of keywords and a threshold percentage. The database is an ASCII file, where a line corresponds to an entry. Semi-colons separate the agent name, keyword list, percentage, task name, and response message. Blanks separate elements of keyword lists. Agent names are represented as a string without spaces. An example of the database entry for the agent BigDave who wants to know about restaurants is:

```
BigDave;restaurant;1;Diner;%*&SubscribeOn_msg_2%*&SubscribeRestaurant_msg_2_0
```

## References

- [1] K. Sycara, J. Lu, M. Klusch, S. Widoff. Matchmaking among Heterogeneous Agents on the Internet
- [2] K. Decker, K. Sycara. Intelligent Adaptive Information Agents. *Journal of Intelligent Information Systems*, 9, pp. 239-260, 1997
- [3] K. Decker, K. Sycara, M. Williamson. Modeling Information Agents: Advertisements, Organizational Roles, and Dynamic Behavior
- [4] K. Decker, A. Pannu, K. Sycara, M. Williamson. Designing Behaviors for Information Agents
- [5] K. Decker, K. Sycara, M. Williamson. Middle-Agents for the Internet.
- [6] Leonard N. Foner. A Multi-Agent Referral System for Matchmaking
- [7] John Graham and Keith Decker, Towards Distributed, Environment Centered Agent Framework, Appearing in "Intelligent Agents IV, Agent Theories, Architectures, and Languages," Springer-Verlag, 2000, Nicholas Jennings, Yves Lesperance, Editors
- [8] Y. Labrou and T. Finin, "A Proposal for a new KQML Specification", University of Maryland Baltimore County, CSEE Technical Report, August, 1997, TR CS-97-03
- [9] Mikko Laukkanen and Jukka Eskelinen, Requirement Specification for the DECAF-Matchmaker, Computer and Information Sciences Dept., University of Delaware, May 1999