Design Options for Subscription Managers*

Aloys Mbala, Lin Padgham, and Michael Winikoff

RMIT University
Melbourne, Australia
{aloys,linpa,winikoff}@cs.rmit.edu.au

Abstract. An important issue in open agent systems such as the Internet is the discovery of service providers by potential consumers (requesters). This paper is concerned with services that involve the ongoing provision of up-to-date information to requesters. We explore three separate issues: subscription to an information provider for ongoing provision of information; monitoring for new information providers; and maintaining awareness of when providers disappear from the system. We explore several models for how this functionality may best be provided, with emphasis on the ways in which certain choices affect the overall system; and provide an analysis of preferred design options for environments with different characteristics.

1 Introduction

An important issue in open agent systems such as the Internet is the discovery of service providers by potential consumers (requesters). There is a broad range of work in this area, including work on web service description languages, such as WSDL[1] and OWL-S [1], as well as work on distributed search algorithms and architectures such as peer-to-peer systems [2]. A common approach, even in peer-to-peer systems, is to have some specialised agents (or services) which assist providers and requesters to find one another. These are variously called *yellow page agents* [3], directory facilitators[2], brokers [4], and match-makers [5] with the term *middle-agent* being used to characterise these kinds of agents. UDDI (Universal Description, Discovery and Integration) directories[3] are one standard instantiation of such a facility while FIPA (Foundation for Intelligent Physical Agents) Directory Facilitators are another.

In many application areas a large number of the services that are required from other entities in the system are services that provide information. In many cases what is required is not just information at a given point in time, but rather ongoing updates of information as the situation changes. For example, in an intelligent alerting system that we are working on with the Australian Bureau of Meteorology [6], if the fire monitoring agent within the system discovers a new fire, it will then want to be informed of any weather events that may affect the fire, such as nearby storms. It is clearly preferable for

* We would like to acknowledge the support of the Australian Research Council, the Australian Bureau of Meteorology and Agent Oriented Software Pty. Ltd. under grant LP0347925.

[1] http://www.w3.org/TR/wsdl

[2] http://www.fipa.org/specs/fipa00023/SC00023K.html

[3] http://www.uddi.org

the relevant agent to set up *subscriptions* and to be notified immediately when relevant new information becomes available, rather than to make regular requests to determine whether new information is available. This notion of *subscription* is well known and it is supported by standard protocols[4].

However, an additional facility is needed. If the subscriptions are long-lived then it is quite likely that there will be changes in the available information providers. The subscribing agent may well need to be made aware of new information providers that join the system, and of any information providers that it has subscribed to that leave the system. Again, rather than have the subscribing agent make periodic requests, it is preferable for it to subscribe to this information. This subscription is to changes in the available (relevant) information providers rather than to information, and is made with the middle-agent. This requires the middle-agent to provide a *monitoring* capability, in addition to the more commonly discussed *matchmaking* (or *brokering*) functionality [7].

By providing information on changes in available information providers, we allow additional flexibility and intelligence in the requesters. For example, in the meteorology application two kinds of weather information sources are used in reasoning about whether there is an alertable situation with respect to a particular fire. If the storm observations from radar become unavailable, then storm likelihood forecasts from the atmospheric model are accessed instead. The provision of information on available relevant providers to requesters is a key difference between our work and event notification systems such as Siena [8] or NaradaBrokering [9], which do not provide requesters with information on changes to available providers[5].

In this paper we explore design options for "Subscription Manager" middle-agents which support subscriptions to changes in available relevant information providers. There are three issues that we concentrate on. Firstly, the mechanism that allows an information requester to be continually updated regarding new information sources. Secondly, the details of how subscriptions are created and cancelled. Thirdly, how the departure of agents from the system is detected and what is done in response to detecting a "dead" agent. With each of these issues we will explore what functionality can potentially reside with the middle-agent, and the costs and benefits of the alternative approaches.

The contribution of this paper is the detailed analysis of these three issues, identifying tradeoffs and leading to recommendations regarding design choices in Subscription Manager middle-agents. We believe that these recommendations would be useful to the designer of a system which is to use a Subscription Manager middle-agent. Since our concern is with key design decisions — such as whether subscriptions should be made by the middle-agent or by requester agents — we do not provide a complete design for the Subscription Manager.

The issues discussed in this paper are only a part of a complete solution. In order to implement a system one must also define a language for describing services and requests and a matching mechanism between these. However, these issues have been

[4] e.g. http://www.fipa.org/specs/fipa00035/

[5] What they provide corresponds to the design option where decision making is delegated to the middle-agent, i.e. what we call *subscribe-all* in section 4.

explored in previous work and a wide range of options exist for service/request description and matching including standards around web service, FIPA standards, KQML [10], and others such as LARKS [11] and Infosleuth [12].

The need for subscription and monitoring services vary from application to application, but we would suggest that they are quite broadly applicable. For example in a travel and tourism services network it would be likely that there was a need to subscribe to information on schedule updates for planes, buses and trains. Similarly, a tourism operator in a particular region is likely to want to monitor for any new providers of services such as accommodation, tours and car rentals, in the region of interest. Similarly in an e-business domain, subscription to catalogues of items available from known providers may well make sense, and monitoring of providers of certain kinds of items is also motivated. Consequently we argue that subscription support, and monitoring for providers of certain kinds of services joining and leaving the system, are infrastructure facilities that are required in a dynamic and open domain of services. These capabilities should be provided by middle-agents. In the rest of this paper we explore several models for how this functionality may best be provided, with emphasis on the ways in which certain choices affect the overall system.

2 The Interaction Models

Service Discovery frameworks can be categorised in two groups. The first group includes peer-to-peer dissemination models where a peer propagates its requests through the network it belongs to and expects a list of relevant providers from its peers. A peer can act as a provider, a requester or simply be a kind of proxy that just redirects a given message to others. An alternative framework uses middle-agents where requesters and providers register to a middle-agent which provides some kind of connection service to assist the agents to find other relevant agents. Some systems propose a peer-to-peer structure amongst the middle-agents [13] in order to distribute the functionality of registering and servicing the client agents.

In this work we do not consider the structure of the middle-agents. Although we assume that in a large system this functionality would be distributed in some manner, this is left as future work, building on a range of existing work (e.g. [2, 8, 9, 13]). What we consider here is the relationship between the middle-agent (or network of middle-agents) and what we call the *end-agents*, namely the service requesters or service providers.[6]

Previous work [4, 5, 7, 14] has compared different styles of middle-agents and concluded that *Matchmakers* which provide a list of providers matching a request, are the most appropriate type of middle-agent for large open systems. Middle-agents such as broadcasters and blackboards which simply pass on all connections, un-filtered, result in unnecessarily large lists of agents being provided, and also require end-agents to have individual matchmaking capabilities. Brokers, which manage all interactions with a provider on behalf of a requester have the disadvantage that they are a bottleneck in

[6] A single agent can be both a provider and a requester, but for the purpose of this work we consider them separately.

large systems. In this work we assume a basic matchmaking capability, and then add to this a Subscription Management function, which we explore in further detail.

There are three different processes that we explore as part of this work. The first is the mechanism to allow an information requester to be continually updated regarding the existence of new information sources of a particular kind. The second is the basic subscription mechanism to support an information requester being able to subscribe to provider agents, and cancel subscriptions. The third is an ability to be aware of agents that disappear from the system. With each of these aspects we will explore what functionality can potentially reside with the middle-agent, and the costs and benefits of the alternative approaches.

2.1 Monitoring for new arrivals

As indicated previously, a common need in dynamic systems is for agents to be aware of new services arising in the system that may be of interest to them. One way to achieve this is to have middle-agents maintain information about requester needs, and update the requesters as new providers register. However this ability does not appear to be common in the various kinds of middle-agents that exist, or are discussed in detail in the literature. Retsina [5] mentions a monitoring capability, although very little detail is given[7]. The notion of facilitator defined by Finin et. al. [10] is broad and encompasses monitoring of both information and information providers, but little detail is given (for example, the issue of detecting "dead" agents is not discussed), and there is no exploration of the design options and associated tradeoffs.

Figure 1 indicates the type of mechanism we are suggesting. Providers and requesters send their profiles to the middle-agent which maintains information about both. When requesters request monitoring for a particular type of information, they are first sent an initial list of matches (message 3), and subsequently, if any new matching providers advertise with the middle-agent (message 4), the requester is sent an update (message 5).

However, this figure is incomplete as it focuses only on the monitoring capability. It does not consider aspects of the subscription life-cycle such as who sets up a subscription? Who cancels a subscription? Or, once a subscription has been established, who ensures that the agents involved in the subscription are still alive? These aspects are considered below. Of course, the monitoring capability must also include a mechanism for cancelling monitoring when it is no longer required, or cancelling an advertised profile.

2.2 Subscription Management

In order to handle subscriptions information providers need to be able to provide a subscription facility, sending information to their subscribers either at regular intervals, or when relevant changes occur. Hence there must be a mechanism to set up and cancel such subscriptions.

[7] The notion of "monitor" vs. "single shot" match-making is mentioned on page 42 of [5].

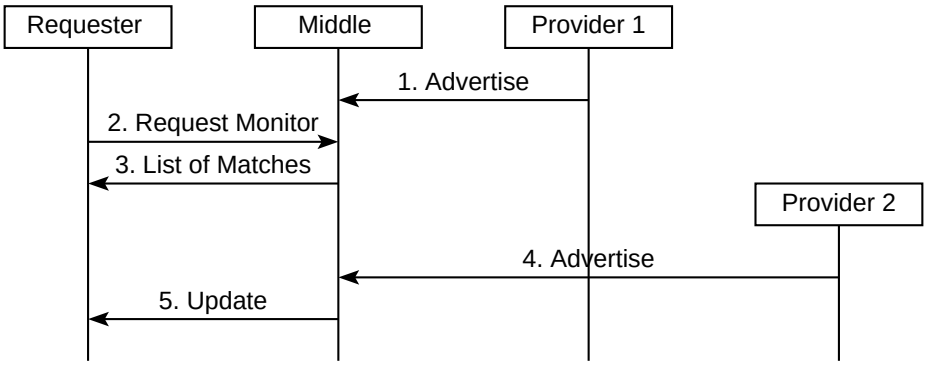

Fig. 1. The discovery mechanism

From the point of view of the information requester wishing to subscribe to a certain kind of information, they may wish to subscribe to all sources of information of a certain type, or a single source. The initial action would be a request to the middle-agent with a query describing the information need (attached to either a monitoring request, or a one-off request). At that point it would be possible either for the middle-agent to return a list of matching providers, as in figure 1, or for the middle-agent to simply set up the subscription(s). If the latter was done, presumably it would be necessary to have two forms of the request: one for subscribe to all, and one for subscribe to one[8].

The possible value in having the middle-agent set up the subscription would be that fewer messages are needed in the system as a whole. On receiving the request, the middle-agent could simply send the subscription message to the relevant information provider(s), and the requester would begin to receive information. Subscription cancellations could be sent either to the middle-agent, or directly to the information provider, if we assume that the identity of the provider(s) is known to the requester once information begins to arrive.

2.3 Monitoring for disappearances

If an agent has a subscription to an information source it is expecting that information will be sent whenever relevant. However, it is possible that the information provider disappears from the system, in which case it may be important for the information subscriber to know of this. This fact may affect reasoning done, or it may result in subscribing to other information sources.

For example in the meteorology application we are working with, two kinds of weather information sources are used in reasoning about whether there is an alertable situation with respect to a particular fire. If the storm observations from radar become unavailable, then storm likelihood forecasts from the atmospheric model are accessed instead.

[8] An additional form would be subscribe to N.

The only reliable way to be sure of knowing when an agent disappears is for some process to check liveness regularly. It is possible for this to be done by all interested subscribers. However, assuming there are likely to be multiple subscribers to any given information source, this is creating more message traffic than necessary. Another option would be for this to be done by the middle-agent, and for the information about a provider's disappearance to be passed on to the relevant agents.

3 Analysis

In this section we analyse the alternative design choices for a Subscription Manager middle-agent. The analysis makes certain simplifying assumptions, but is nonetheless valuable. The analysis focuses primarily on the message traffic, and looks specifically at the *number* of messages, the total *size* of the messages, and at *bottlenecks* in the system.

The number of messages circulating in the system is a natural and important parameter for the evaluation of service discovery frameworks since it is a reasonable approximate measure of the workload of the system, and an analysis of the message traffic received and sent by a given agent can be used to detect potential bottlenecks. However, using only the number of messages isn't sufficient, because it ignores the size of the messages, and therefore we also use the size of the messages to estimate the amount of network traffic.

The analysis in this section uses the terms below. Since the analysis is done at design-time, we do not need to concern ourselves with whether the terms can be measured at run-time in a real agent system: these terms are not used at run-time.

- R: the number of requester agents in the system.
- P: the number of provider agents in the system.
- α: the probability of a random capability and a random interest matching ($0 \leq \alpha \leq 1$). This is a measure of the matching precision, and can be expected to be well below 0.5.
- R_F: the (average) number of requesters whose interests match a given provider's capabilities $R_F = \alpha \times R$.
- P_F: the (average) number of providers whose capabilities match a given requester's interests $P_F = \alpha \times P$.
- S: the number of subscriptions in the system. If each requester agent subscribes to all relevant providers (P_F) then the number of subscriptions is $S = R \times P_F$. If each requester agent subscribes to P_S providers then $S = R \times P_S$.
- P_S: the (average) number of providers that a requester agent subscribes to. This can be all relevant providers (P_F), a single provider, or an arbitrary number ($1 \leq P_S \leq P_F$).
- R_S: the (average) number of requesters that are subscribed to a given provider ($0 \leq R_S \leq R_F$). The value of R_S depends on whether requesters subscribe to one provider, all providers, or P_S providers, and can be calculated by dividing the number of subscriptions in the system (S) by the number of providers. If each requester agent subscribes to all relevant providers (P_F) then $S = R \times P_F$ and $R_S = (R \times P_F) \div P = (R \times \alpha \times P) \div P = R \times \alpha = R_F$. If each requester agent

subscribes to P_S providers then $S = R \times P_S$ and $R_S = R \times P_S \div P$, which is just P_S if there are equal numbers of providers and requesters.

- P_D: the number of provider agents that have left the system since the last liveness monitoring check ($0 \leq P_D \leq P$).
- k: the size of a description of an agent's capabilities or interests relative to the size of its name ($k > 1$). This is used in computing the size of messages.
  We assign a message containing a simple request (e.g. a single name of another agent) a size of 1 and a message containing a description of the interests or capabilities of an agent a size of k. A message that contains a list has a size that is computed by multiplying the contents of the list by its length. For example, a message containing a list of P_F relevant provider names has size P_F, whereas a message containing a list of the capabilities of P_F relevant providers has size $k \times P_F$.

Our presentation of the analysis is structured according to the life-cycle of the system: we consider the metrics associated with adding an agent (requester or provider), with cancelling subscriptions, and with monitoring the liveness of provider agents. In order to help make the analysis more concrete we will include actual numbers, computed by assuming fairly arbitrary — but, we hope, reasonable — figures for the terms above. These assumed values are given in table 1, where brackets are used to indicate numbers which are derived from other values, for example, R_F is derived from R and α (since $R_F = \alpha \times R$). Two of these assumed values need explanation. Firstly, the value for P_S (and hence the value of R_S) depends on whether requester agents ask to be subscribed to one relevant provider, some constant number of relevant providers, or all relevant providers. This will obviously vary depending on the requirements of the requester agents. If we arbitrarily assume that half of the requester agents ask to be subscribed to one provider, a quarter of requester agents ask to be subscribed to 5 providers, and the remaining quarter of requester agents ask to be subscribed to all (in this case, on average, 10 providers) then we have that $P_S = (0.5 \times 1) + (0.25 \times 5) + (0.25 \times 10) = 4.25 \approx 4$. Secondly, the value for P_D assumes that over the course of a polling period 0.5% of provider agents will disappear, since we have 200 provider agents, this gives 1 agent that will disappear in a polling period, on average.

Table 1. Example values for terms

term:	R	P	α	R_F	P_F	S	P_S	R_S	P_D
value:	200	200	0.05	(10)	(10)	(800)	≈ 4	(≈ 4)	1

3.1 Adding an Agent

Adding a Requester Agent: The sequence of messages associated with adding a requester agent depends on whether subscription is done by the middle-agent or the requester.

If subscription is done by the middle-agent then the sequence of messages is: (1) the requester registers with the middle-agent its interests, (2) the middle-agent sends messages to all relevant providers asking them to subscribe the requester, (3) the middle-agent optionally sends a message informing the requester of its subscriptions. The number of messages involved is $1 + P_F$ if the third (optional) notification message isn't sent and $2 + P_F$ if it is sent.

If we assume that each requester wants to subscribe to P_S relevant providers, and that the decision of which providers can be made on its behalf by the middle-agent, then the number of messages is $1 + P_S$.

If subscription is done by the requester then the sequence of messages is: (1) the requester registers with the middle-agent its interests, (2) the middle-agent responds with a list of relevant providers, (3) the requester selects some (P_S) or all (P_F) of the providers in the list and sends each of the selected providers a subscription request. If the requester selects a subset of the available relevant providers and the middle-agent needs to track subscriptions then it must be notified by the requester of its choice of providers, unless it is assumed that requesters always subscribe to all relevant available providers or to some easily predicted subset such as only the first provider in the list. The number of messages involved is $2 + P_S$ (if the middle-agent needs to be informed then the number of messages goes up by one).

We now consider the message *size* and begin with the first case where subscription is done by the middle-agent. If we assume for the moment that requesters subscribe to all relevant providers (P_F), then the size of the three messages is respectively k for the first step, 1 for each of the messages involved in the second step , and (optionally) P_F for the third step giving a total size of $k + P_F$ (or $k + 2P_F$ if requesters are informed of their subscriptions). If we assume that each requester subscribes to P_S providers, then the total size if $k + P_S$ (or $k + 2P_S$ if requesters are informed of their subscriptions).

Consider now the second case, where subscription is done by the requester. If we assume for the moment that requesters subscribe to all relevant providers, then the size of the three messages is respectively k, P_F, and 1 for each of the P_F messages from requester to providers, giving a total of $k+2P_F$ (and $k+3P_F$ if the middle-agent needs to be informed). If we assume that requesters will only subscribe to P_S providers, then the message to the requester containing the list of relevant providers will need to contain the provider's capabilities, as well as their names (so that the requester can decide which providers to subscribe to). Therefore, the size of the messages is $k + kP_F + P_S$ (or $k + kP_F + 2P_S$ if the middle-agent needs to be informed).

These cases are summarised in table 2. In all cases informing the other agent takes a single additional message of size equal to the number of desired providers. The numbers in the table give the actual number of messages, computed using the assumed values in table 1.

In summary, having the middle-agent subscribe saves a single (potentially large) message, and if the middle-agent needs to track subscriptions, then a second message is also saved (assuming that requesters don't need to be notified of their subscriptions). However, having the middle-agent subscribe prevents a requester from being able to directly select its provider(s), and if requesters need to subscribe to something other

Table 2. Adding a requester (message size analysis is in brackets)

	Middle Subscribes	Requester Subscribes
All providers	$1 + P_F$ ($k + P_F$) 11	$2 + P_F$ ($k + 2P_F$) 12
P_S providers	$1 + P_S$ ($k + P_S$) 5	$2 + P_S$ ($k + kP_F + P_S$) 6

than all providers then there is additional complexity in specifying how many providers are desired (e.g. one, all, or some constant number P_S).

Adding a Provider Agent: The sequence of messages associated with adding a provider agent depends on whether subscription is done by the middle-agent or the requester.

For the moment let us assume that requesters subscribe to all relevant providers. If subscription is done by the middle-agent then the sequence of messages is: (1) the provider registers with the middle-agent its capabilities, (2) the middle-agent sends a message back to the provider with all relevant requesters that it should subscribe (possible none), and (3) the requesters are (optionally) informed of their new subscriptions. The number of messages involved is 2 if the third (optional) notification message isn't sent and $2 + R_F$ if it is. The messages informing the requesters (step 3) could be sent by either the middle-agent or the provider. In the interests of trying to avoid overloading the middle-agent it is preferable to have the provider inform the requesters.

If subscription is done by requesters then the sequence is: (1) the provider registers with the middle-agent, (2) the middle-agent sends a message to each relevant requester with the identity of the provider, (3) each requester sends a subscription request message to the new provider. The number of messages involved is $1 + 2R_F$. Note that there is a bottleneck issue here: the provider will, during a short time period, be sent messages from a number of requesters, potentially overloading it.

Considering the size of the messages, in the first case, where subscription is done by the middle-agent, the size of the three messages is respectively k, R_F and (optionally) 1 for each of the R_F messages giving a total size of $k + R_F$ (or $k + 2R_F$ if requesters are informed of their subscriptions). Considering the second case, where subscription is done by the requester, the size of the three messages is respectively k for the first message, 1 for each of the R_F messages, and 1 for each of the R_F messages from requesters to the provider, giving a total of $k + 2R_F$.

These cases are summarised in the top row of table 3. Informing the requester (if the Subscription Manager subscribes) takes an additional R_F messages of size 1. The numbers in the table give the actual number of messages, computed using the assumed values in table 1.

The bottom two rows of table 3 assume that requesters only want to be subscribed to a fixed number of providers. In this case when a provider joins an existing multi-agent system most or all requesters will already have the desired number of subscriptions.

Table 3. Adding a provider (message size analysis is in brackets)

	Middle Subscribes	Requester Subscribes
All providers	2 $(k + R_F)$	$1 + 2R_F$ $(k + 2R_F)$ 21
typical P_S providers	1 (k)	1 (k)
max. P_S providers	2 $(k + R_S)$	$1 + R_F + R_S$ $(k + R_F + R_S)$ 15

This is because requesters subscribe when they join the system and departing providers are detected and replaced, therefore the only situation where a requester will not have its desired number of subscriptions is where there are not enough relevant providers in the system. In this case the typical number of messages generated by a new provider joining an existing system is one (of size k), but it is possible for this to be higher: up to the (unlikely) maximum shown in the third row of table 3. Informing the other agent takes an additional R_S messages of size 1.

In summary, if requesters subscribe to all relevant providers then having the middle-agent subscribe saves a significant number of messages and also has a saving in terms of the size of messages. Additionally, if the requesters subscribe then there are potential bottleneck issues. If requesters subscribe to a fixed number of providers then the saving is much smaller.

3.2 Cancelling Subscriptions

Cancelling a subscription can be done directly, by having the requester send a message to the provider (or vice versa if the provider is the one cancelling the subscription). Alternatively, cancelling a subscription can be done via the middle-agent. In the first case, cancelling a subscription involves a single message, with an optional second message informing the middle-agent. Both messages have size 1. In the second case, cancelling a subscription involves two messages each with size 1. Thus the difference in terms of messages involved between direct and indirect cancellation of subscriptions is minor, and is non-existent if the middle-agent needs to be informed of the cancellation.

If a provider wishes to cancel *all* of its subscriptions then there are a number of cases: (1) If requesters don't need to be kept informed of their subscriptions then a single message (of size 1) to the middle-agent is all that is required. (2) If requesters need to be told, but the middle-agent doesn't need to be told then there are R_S messages from the provider to the requesters that are subscribed to it. (3) If both middle-agent and requester agents need to be informed then there is one message from the provider to the middle-agent, and R_S messages from the provider to the requesters. Although it is possible to have the middle-agent inform the requesters, this increases the load on the middle-agent, requires that the provider specify explicitly the list of subscribed

requesters (unless the middle-agent has a record of subscriptions), and doesn't give any benefit.

Thus if a provider wishes to cancel all of its subscriptions then it is most efficient to not inform the requesters, but only inform the middle-agent. However, if the requesters do need to be informed then the cost of also informing the middle-agent is low.

The analysis for a requester cancelling all of its subscriptions is similar. If the requester agent does not know who it is subscribed to then it needs to first obtain the list from the middle-agent (which also has the side effect of informing the middle-agent of the cancelled subscriptions). In this case cancelling all subscriptions requires $2 + P_S$ messages with total size $1 + 2P_S$. If the requester agent does know who it is subscribed to then informing the providers takes P_S messages of size 1, and informing the middle-agent is a single additional (size 1) message.

3.3 Monitoring Liveness

Providers need to be monitored, so that a provider disappearing is detected and appropriate action taken. Monitoring liveness of requesters by providers doesn't seem to make sense: if the providers have information to send, then that transmission acts as a ping[9]. If they don't have information to send, then they don't really care about the requester being alive! If monitoring of requesters is desired, then it makes sense to have the middle-agent do this.

Monitoring of providers can be done either by the middle-agent or by the requesters. Consider the first possibility, in this case the cost for checking each provider for liveness can be worked out as follows[10]. Firstly, there are P messages to the providers. Secondly, there are P_D responses, one for each departed agent[11], where P_D is the number of departed agents found in this check (we assume that live agents do not respond). If subscriptions are done by the requester agents then the middle-agent will need to inform the requesters ($P_D \times R_F$ messages[12]), otherwise informing the requester agents is optional.

Consider now the second possibility, where monitoring the providers is done by the requester agents. This is considerably less efficient because each provider will be monitored (redundantly!) by each requester agent that is subscribed to it. More precisely, each provider will be monitored by R_S agents. Thus $P \times R_S$ messages are sent, and $P_D \times R_S$ responses received. If the middle-agent needs to be informed, then it will (eventually) receive messages from each of the R_S requester agents that are monitoring the departed provider (an additional $R_S \times P_D$ messages).

An alternative is for the first requester agent that detects a departed provider to inform the other requester agents that are subscribed to that provider, rather than allowing

[9] That is, we assume that the provider will detect a departed requester when it attempts to send the requester information.

[10] Note that a reasonable design decision is to spread this monitoring over a time period by gradually traversing a list of providers.

[11] The responses are sent by either the relevant agent platform (saying that the agent is unknown), or from the middle-ware (saying that the agent platform is unknown).

[12] If the middle-agent has an up-to-date record of the subscriptions then this can be tightened to $P_D \times R_S$

them to independently realise that the provider is departed. This involves the following sequence of messages: (1) a message from a requester to the departed provider, (2) a message from the departed provider's platform to the requester, (3) a message from the requester to the middle-agent, and (4) $R_S - 1$ messages from the middle-agent to the other requesters. The total number of messages for pinging a single departed provider then is $3 + (R_S - 1) = 2 + R_S$ and the message size is also $2 + R_S$. The total number of messages for pinging *all* providers is this multiplied by the number of departed providers, plus R_S messages to each live provider, i.e. $(P - P_D) \times R_S + P_D \times (2 + R_S) = P \times R_S + 2P_D$.

Note that this slightly more efficient, but more complex, approach requires that the middle-agent have a record of subscriptions (otherwise it is more expensive: replace R_S by R_F). This approach also avoids a bottleneck issue: the middle-agent is only informed of a departed provider agent once, rather than R_S times.

A much more significant potential saving in having liveness monitoring be done by requesters is that it becomes possible to exploit "implicit" pings: if a provider sends data to a requester then this is evidence that the provider is alive and it can be assumed to have been pinged. If a provider agent is sending data frequently enough, then it will never need to be explicitly pinged as long as it is alive. If this is the case, and assuming that the optimisation described above is not used, then the number of ping messages that are sent goes down from $P \times R_S$ to $P_D \times R_S$, giving $2 \times P_D \times R_S$ messages overall and $3 \times P_D \times R_S$ if the middle-agent needs to be informed. If the optimisation described above is included then the effect of implicit pings is, in the best case, to eliminate the pinging of live agents, i.e. the term $(P - P_D) \times R_S$, leaving $P_D \times (2 + R_S) = 2P_D + P_D R_S$ messages. However, it is not clear that this best case will hold, so the significant reductions promised by exploiting 'implicit' pings is perhaps exaggerated by the numbers in table 4.

This analysis is summarised in table 4. The bracketed formulae include informing the requesters (if the middle-agent pings) or middle-agent (if requesters ping). The third row ("Improved") is when requesters ping, but includes informing both the middle-agent and other (relevant) requester agents of a departed provider. The numbers in the table give the actual number of messages, computed using the assumed values in table 1; the numbers in brackets include informing the requesters.

Table 4. Monitoring provider liveness (bracketed formulae include informing)

Who pings?	Number of messages	+ Implicit pings
Middle agent	$P + P_D$ $(P + P_D + P_D R_S)$ 201 (205)	N/A
Requester agents	$PR_S + P_D R_S$ $(PR_S + 2P_D R_S)$ 804 (808)	$2P_D R_S$ $(3P_D R_S)$ 8 (12)
Improved	$PR_S + 2P_D$ 802	$2P_D + P_D R_S$ 6

The analysis above only considers monitoring and detecting departed agents. What is done in response to detecting a departed agent depends on the subscription policy of the requester agents that were subscribed to the departed agent. If a requester is subscribed to all relevant providers then there is nothing further to be done – there are no other relevant providers that could be added, because the requester is already subscribed to them. However, this doesn't mean that monitoring liveness is not important – for instance, there is a difference between receiving no information because there is no information, and receiving no information because there is no available source for the information. On the other hand, if a requester is subscribed to one provider (or, more generally, P_S providers), then a replacement provider needs to be found. How this is done, and the number of messages involved, depends on whether subscriptions are done by the requester or by the middle-agent. The analysis is similar to that presented in section 3.1.

4 Subscription Manager Specification

Based on the analysis in the previous section we now specify a Subscription Manager middle-agent. The most difficult issue is regarding whether or not the Subscription manager should actually set up subscriptions on behalf of a requester. On the one hand there is a reasonable savings in doing this and it assists with bottleneck issues at the provider. On the other hand it removes flexibility from the requester, which may need or prefer to make its own choices. If requesters subscribe to all providers, then there is no issue with flexibility, and the savings are significant, so in this case it makes sense to have the Subscription Manager subscribe. On the other hand, if requesters subscribe to a fixed number of providers (and especially if this fixed number is low) then the savings are lower, and allowing the requester to select its providers becomes more important. In this case it may make more sense to have requesters subscribe themselves. Consequently we recommend that the Subscription Manager allow *both* options.

In addition to supporting subscription being done by either requesters or the Subscription Manager, there is also a need to allow for both one-off and ongoing matching, as well as subscription to one or subscription to all[13]. This requires that the interface allows four[14] kind of requests: *single-match* (requester subscribes), *ongoing-match* (requester subscribes), *subscribe-one* (Subscription Manager subscribes requester, and replaces if provider disappears), and *subscribe-all* (Subscription Manager subscribes requester, and subscribes to new providers as they arrive). Additionally, the Subscription Manager's interface needs to allow for a requester to cancel the ongoing-match, subscribe-one or subscribe-all, and for a provider to cancel its registration.

It is slightly more efficient for end-agents to manage cancellations directly, if the Subscription Manager does not need to be updated. If the Subscription Manager is updated the overhead is little. Consequently we recommend that cancellations be done directly between end-agents, since it relieves the Subscription Manager of a centralised responsibility that carries no real benefit. Requesters with an ongoing *subscribe-one*

[13] We assume that subscription to some other number must be handled by the requester.

[14] If the requester subscribes then it doesn't make sense to distinguish between subscribe-to-one and subscribe-to-all. If the middle-agent subscribes then an ongoing match is assumed.

request, will need to notify the Subscription Manager of the cancellation so that they can be subscribed to a new provider.

Monitoring of provider liveness can be done by either requesters or by the Subscription Manager. If we use the improved version of requester monitoring, and assume that "implicit" pings completely eliminate pinging of live agents, then requester-based liveness monitoring requires fewer messages ($2P_D + P_D R_S$ compared with $P + P_D + P_D R_S$, given the assumptions of Table 1 these are respectively 6 and 205). However, this requires a more complex mechanism, shifts the responsibility for a crucial infrastructure task onto the requesters (which is not practical in an open system), and assumes that implicit pings completely eliminate pinging of live agents and that requester agents need to be informed of departed providers[15]. Therefore, we recommend that monitoring of provider liveness be done by the Subscription Manager.

Figure 2 shows an example interaction. In this example a Requester has asked to be subscribed to all relevant providers (*SubscribeAll*). Provider 1 is relevant, and so the requester is subscribed (by the Subscription Manager) to Provider 1. The Provider then begins providing the requester with regular information (*Data*). The Subscription Manager also periodically checks that the provider is still available by sending *Ping* messages. A little later a second provider joins the system, and since it is also relevant to the requester, the requester is subscribed to this provider as well. The first provider then changes its service specification (*RemoveProfile* followed by a new *AdvertiseProfile*). The requester is notified that provider 1 is no longer relevant. Finally in this example, provider 2 disappears, and the Subscription Manager realises this when it attempts to Ping the provider, at which point the requester is notified that provider 2 is no longer available.

5 Conclusion

We presented a new type of middle-agent, the *Subscription Manager*, and motivated its use in systems that involve ongoing information provision to requesters. An analysis of different design options for the Subscription Manager was presented, leading to recommendations for the design of Subscription Managers. To summarise, the key recommendations are:

- That the Subscription Manager provide support for setting up subscriptions to be done either by itself, or by requester agents.
- That the Subscription Manager provider a number of ways of requesting information:
  1. *single-match* which returns a list of matching providers at the current time, but will not inform the requester of additional (relevant) providers that subsequently join the system.
  2. *ongoing-match* which returns a list of matching providers, and also asks the Subscription Manager to inform the requester should new relevant providers become available.

[15] If requesters are not required to be informed of departed providers, then having middle-agents monitor providers requires $P + P_D$ messages, and in this case having requesters monitor is more efficient if $P_D(1 + R_S) < P$.

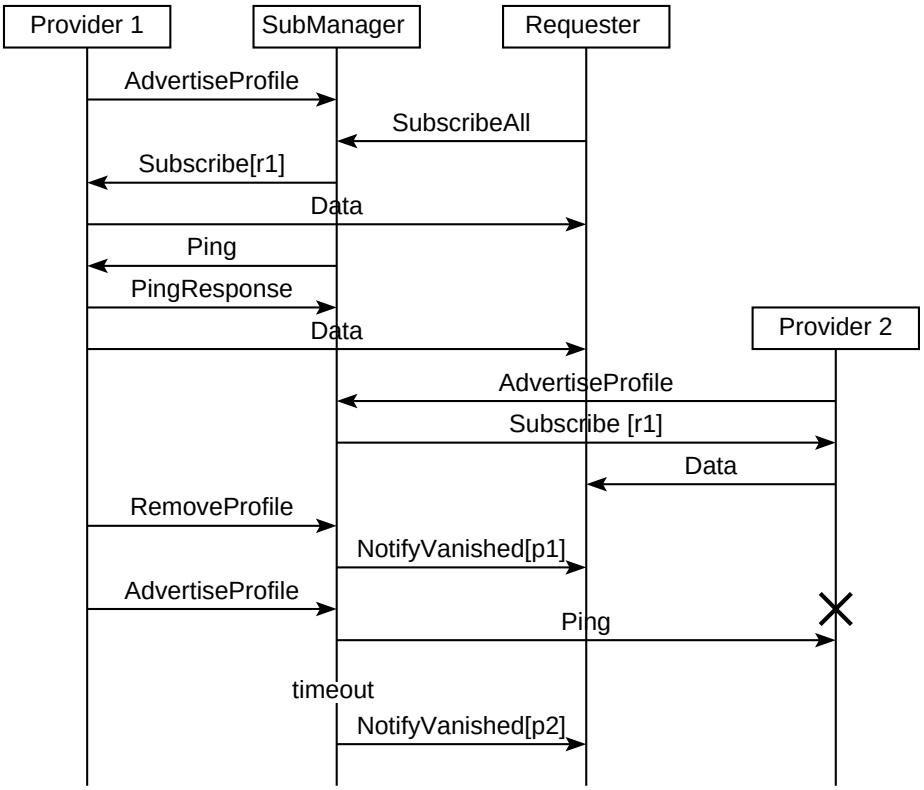

Fig. 2. An example interaction

3. *subscribe-one* which asks the Subscription Manager to maintain a subscription by the requester to exactly one relevant provider (which is selected by the Subscription Manager).
4. *subscribe-all* which asks the Subscription Manager to maintain subscriptions by the requester to *all* relevant providers.

– That cancellations of subscriptions be done directly between end-agents.
– That monitoring of provider liveness be done by the Subscription Manager agent.

For a given application some of the flexibility recommended may not be needed. For example, in a domain where requester agents always subscribe to all available information sources there is no need for the Subscription Manager to support subscription to a single provider.

Areas for future work include investigating ways of structuring a *network* of middle-agents, carrying out experimental evaluation of the analysis presented, and looking at how often agents should be 'pinged' given a particular rate of agent departure.

References

1. Paolucci, M., Soudry, J., Srinivasan, N., Sycara, K.: A broker for OWL-S web services. In: First International Semantic Web Services Symposium. (2004)
2. Schmidt, C., Parashar, M.: A peer-to-peer approach to web service discovery. World Wide Web Journal **7**(2) (2004) 211–229
3. Alonso, G., Casati, F., Kuno, H., Machiraju, V.: Web Services: Concepts, Architectures and Applications. Springer-Verlag, Berlin, Germany (2004)
4. Decker, K., Sycara, K., Williamson, M.: Middle-agents for the internet. In: Fifteenth International Joint Conference on Artificial Intelligence, Morgan Kaufmann (1997) 578–583
5. Sycara, K.: Multi-agent infrastructure, agent discovery, middle agents for web services and interoperation. In: Multi-Agent Systems and Applications, LNAI 2086, Springer-Verlag (2001) 17–49
6. Mathieson, I., Dance, S., Padgham, L., Gorman, M., Winikoff, M.: An open meteorological alerting system: Issues and solutions. In Estivill-Castro, V., ed.: Proceedings of the 27th Australasian Computer Science Conference, Dunedin, New Zealand (2004) 351–358
7. Decker, K., Williamson, M., Sycara, K.: Matchmaking and brokering. In: 2nd International Conference on Multi-Agent Systems (ICMAS 1996), MIT Press (1996)
8. Carzaniga, A., Rosenblum, D.S., Wolf, A.L.: Design of a scalable event notification service: Interface and architecture. Technical Report CU-CS-863-98, University of Colorado, Department of Computer Science (1998)
9. Fox, G., Pallickara, S.: The Narada event brokering system: Overview and extensions. In: Proceedings of the 2002 International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA'02). (2002) 353–359
10. Finin, T., Fritzson, R., McKay, D., McEntire, R.: KQML as an agent communication language. In: CIKM '94: Proceedings of the third international conference on Information and knowledge management, ACM Press (1994) 456–463
11. Sycara, K., Widoff, S., Klusch, M., Lu, J.: Larks: Dynamic matchmaking among heterogeneous software agents in cyberspace. Autonomous Agents and Multi-Agent Systems **5**(2) (2002) 173–203
12. Cassandra, A., Chandrasekara, D., Nodine, M.: Capability-based agent matchmaking. In: AGENTS '00: Proceedings of the fourth international conference on Autonomous agents, ACM Press (2000) 201–202
13. Gibbins, N., Hall, W.: Scalability issues for query routing service discovery. In: Proceedings of the 2nd International Workshop on Infrastructure for Agents, MAS, and Scalable MAS. (2001) 209–217
14. Wong, H.C., Sycara, K.: A taxonomy of middle-agents for the internet. In: 4th International Conference on Multi-Agent Systems (ICMAS 2000), IEEE Press (2000) 465–466