1 Transformational level brokering

The key elements for understanding the operation of the broker are the service description and the broker algorithm.

1.1 Service description language

A competence is advertised to the broker as a *Service_advertisement* message consisting of the following terms:

- *senderURI*: the address of this agent (also used for communications: currently just a fully resolved machine name plus an open port for receiving messages);
- *service-name*: the name of this particular service, which should be uniquely identify the service for this agent – a given agent may advertise many services, a particular service is identified by the combination of URI and name.
- *service*: the service offered itself; this is described as a *Knowledge_transformation* construct, and is described using the following elements:
    - *transformation-type*: a list of one or more terms describing the nature of this transformation; the terms are drawn from a standard ontology.
    - *input-knowledge*: a list of one or more *body of knowledge* descriptions (see below), each representing some structured information supplied as input to this transformation.
    - *output-knowledge*: a *body of knowledge* description representing the structured information that is the result of this transformation.

  Each body of knowledge is described using the following terms:

    - *name*: an identifier for this body of knowledge;
    - *abstraction-level*: the level of abstraction of this body of knowledge, described using a term from the given ontology.
    - *representation-format*: the representational paradigm (in AI-terms) used to structure the body of knowledge, once again a term from the given ontology.

From this it can be seen that a transformation is here considered to be some sort of standard alteration of one or more input bodies of knowledge into a single output body of knowledge. It should be noted that the above competence description makes no mention of preconditions (other than the required *input-knowledge*) on service provision or external competences that are required during execution of the services: services are assumed to be ‘stand-alone’ components, available regardless of the current state of the system or environment. (Such a view would seem to be more in keeping with the developing notions of web services.)

It is expected that a service advertisement description will be *complete*, inasmuch as values will be supplied for all the above fields, and, moreover, where ontological terms are specified, these will be at a (reasonably) low level of abstraction (since, generally speaking, it is expected that services will effect concrete rather than abstract transformations).

A query description has the same basic structure as a service description (albeit a construct of type *Service_query* rather than of *Service_advertisement*), but it may contain more abstract ontological terms (if for example, one of a class of transformations is desired, but no one particular member of that class), and it may be incomplete in two particular ways:

- it may lack a *transformation-type* description, if this is implied by the descriptions of the *input-knowledge* and *output-knowledge* (both of which, in this case, must be supplied);
- it may lack the description of an *output-knowledge* if this is implied by the *transformation-type* and the *input-knowledge* (both of which, in this case, must be supplied).

In practice, the communication of both service advertisements and queries is made according to an RDF specification of the above terms;[1] the following section gives an example of an advertisement in this form.

1.1.1 Service description: example

This is an example representing the advertisement of ‘id3’, a machine learning service. This particular service effects a transformation of type *Generalisation_synthesis* on a body of knowledge at abstraction level *Instance_abstraction_level* and having representation type *Symbolic-value_pairs*, resulting in an output body of knowledge at abstraction level *Exclusive_domain_abstraction_level* and representation type *Rule-based*. (These terms, and all the constructions are all defined against the namespace *kb*: the corresponding RDFS declaration of the terms used is not presented here, but it is hoped that the semantics of these will be reasonably obvious.)

---

[1] The change to using RDF for the descriptions was made partly to exploit the facilities of RDFS for defining and using ontological terms, and partly to encourage greater interoperability with non-Prolog-based agents.

```
<?xml version='1.0' encoding='ISO-8859-1'?>
<!DOCTYPE rdf:RDF
       [<!ENTITY rdf 'http://www.w3.org/1999/02/22-rdf-syntax-ns#'>
       <!ENTITY kb 'http://protege.stanford.edu/kb#'>
       <!ENTITY rdfs 'http://www.w3.org/TR/1999/PR-rdf-schema-19990303#'>]>

<rdf:RDF xmlns:rdf="&rdf;"xmlns:kb="&kb;"xmlns:rdfs="&rdfs;">

       <kb:Service_advertisement rdf:about="&kb;KnowTransComms2_0"
              kb:senderURI="http://amedee.inf.ed.ac.uk:12003"
              kb:service-name="id3" rdfs:label="&kb;KnowTransComms2_0">
              <kb:service rdf:resource="&kb;KnowTrans_0"/>
       </kb:Service_advertisement>

       <kb:Knowledge_transformation rdf:about="&kb;KnowTrans_0"
              kb:name="id3" rdfs:label="id3">
              <kb:transformation-type
                     rdf:resource="&kb;Generalisation_synthesis"/>
              <kb:input-knowledge rdf:resource="&kb;KnowTrans_1"/>
              <kb:output-knowledge rdf:resource="&kb;KnowTrans_2"/>
       </kb:Knowledge_transformation>

       <kb:Body_of_knowledge rdf:about="&kb;KnowTrans_1"
              kb:name="input-examples" rdfs:label="input-examples">
              <kb:representation-format
                     rdf:resource="&kb;Symbolic-value_pairs"/>
              <kb:abstraction-level
                     rdf:resource="&kb;Instance_abstraction_level"/>
       </kb:Body_of_knowledge>

       <kb:Body_of_knowledge rdf:about="&kb;KnowTrans_2"
              kb:name="output-rules" rdfs:label="output-rules">
              <kb:representation-format rdf:resource="&kb;Rule-based"/>
              <kb:abstraction-level
                     rdf:resource="&kb;Exclusive_domain_abstraction_level"/>
       </kb:Body_of_knowledge>

</rdf:RDF>
```

So, a query matching to this service might, say, request the transformation *Generalisation_synthesis* of a body-of-knowledge described at the *Instance_abstraction_level* and in terms of *Attribute-value_pairs* (assuming this representational paradigm is defined as a superclass of *Symbolic-value_pairs*). Hence, in this environment, all agents are constrained to describe their services and pose their queries in terms of a shared knowledge transformation ontology, and moreover, it is assumed that this ontology will be applicable across a range of domains. The introduction and use of this more general ontology for describing services is an attempt to address the problem noted above of having to ‘know’ the terms in which services are advertised in order to be able to formulate queries requesting them.[2]

1.2 Broker algorithm

When it receives a new transformational query, the task of the broker is to determine whether any advertised service – or sequence of services – would result in the desired transformation. The algorithm to do this is as follows:

1. Let the solutions set, $S=\{\}$. Let the possibles set, $P=\{\}$. Search through the advertised services to find possible *terminal services* (these are services that *may* constitute the final element in a sequence of services to achieve the goal query):
    - If the query contains both *transformation-type* and *output-knowledge* descriptions, then any service which matches[3] both these represents a possible

---

[2] This is not to say that this representation is either correct or universally appropriate (for instance, the language currently permits only descriptions of domain-independent services: there is no facility for describing the domain that a service operates in, and to rectify this would seem likely to introduce some of the problems noted above with the functional level description). Ultimately, the appropriateness of this representation depends on its pragmatic value, and testing is required to determine this.

[3] To determine whether a query *transformation-type*, T_q, consisting of a list of ontological terms, matches a service *transformation-type*, T_s, the following algorithm is applied:

1. Let $Q = T_q$ and $R = T_s$.
2. If $Q = \{\}$ or $R = \{\}$ then stop: T_q and T_s are deemed to match.
3. Otherwise, taking the next term q of Q, compare it to each of the terms r in R in turn to determine the ‘best’ matching term among these, r_B. The terms q and r match if and only if q is equal to or subsumes r, and the best match is that which involves the least ‘ontological distance’ between the q and r – if the terms are equal, their ontological distance is 0, if q is an immediate superclass of r, their distance is 1 and so forth. If there is more than one term in R that fulfils the criteria of the best match, an arbitrary choice is made amongst these as to the best, r_B. However, if no term in R is found to match q then the process stops - T_q and T_s do not match.
4. The two best matching terms q and r_B are removed from Q and from R respectively, and the matching process continues from 2 above.

terminal service, and is added to *P* as the first element of a potential service sequence (a service sequence is a list of services whose application, in order, would result in the satisfaction of the query).

   - Else, if the query contains no *output-knowledge*, then any service that matches the desired *transformation-type* represents a possible terminal service, and is added to *P* as the first element of a potential service sequence.
   - Else, if the query contains no *transformation-type*, then any service that matches the desired *output-knowledge* represents a possible terminal service, and is added to *P* as the first element of a potential service sequence.

2. If $P = \{\}$ then stop. Return *S* as the set of plausible complete sequences for satisfying the current query.
3. Consider each element $p \in P$ in turn to determine whether *p* constitutes a 'complete' service sequence for the current query: if the *input-knowledge* of the query and the *input-knowledge* of the first element (a service) of *p* match then this is a complete sequence, and *p* is removed from *P* and added to *S*.
4. For each incomplete plausible sequence $p \in P$ search the current services for any other known service *s* that could plausibly precede this in a service sequence: *s* can plausibly precede *p* if the *output-knowledge* of *s* matches the *input-knowledge* of the first element of *p* (in other words, if the output of *s* could plausibly represent the input to the sequence *p*).[4] For each *s* that meets this criterion, a new plausible service

---

A query *output-knowledge* description, O_q (a *body of knowledge*), is considered to match a service *output-knowledge* description, O_s, if and only if:

- The *abstraction-level* of O_q is equal to or subsumes the *abstraction-level* of O_s, and;
- The *representation-format* of O_q is equal to or subsumes the *representation-format* of O_s.

To determine if a query *input-knowledge* description, I_q (a list of *bodies of knowledge*), matches a service *input-knowledge* description, I_s, a similar algorithm is applied as for the *transformation-type* matching described above but, within this, matching the individual (*body of knowledge*) elements using the same approach as for the *output-knowledge* elements.

[4] Given particular combinations of particular service descriptions, it will be noted that with this step there is the potential of propagating non-terminating circular sequences. Currently this is avoided by checking that *s* does not already exist in *p*, but this simple approach could result in the exclusion of certain valid sequences.

sequence consisting of *s* appended to the front of *p* is added to P. Finally, remove *p* from *P*.

5. Repeat from step 2 above.