Technical Report WINLAB-TR-252

CONTENT-BASED MIDDLEWARE FOR DECOUPLED INTERACTIONS IN PERVASIVE ENVIRONMENTS

N. Jiang, C. Schmidt, V. Matossian and M. Parashar [1]

Rutgers University

April, 2004

RUTGERS WIRELESS INFORMATION

NETWORK LABORATORY

Rutgers - The State University of New Jersey

73 Brett Road

Piscataway, New Jersey 08854-8060

Phone: (732) 445-5954 FAX: (732) 445-3693

[1]The research presented in this paper is supported in part by NSF via grants numbers ACI 9984357 (CAREERS), EIA 0103674 (NGS), EIA-0120934 (ITR), ANI-0335244 (NRT), CNS-0305495 (NGS) and by DOE ASCI/ASAP (Caltech) via grant number 82-1052856.

Technical Report WINLAB-TR-252

CONTENT-BASED MIDDLEWARE FOR DECOUPLED INTERACTIONS IN PERVASIVE ENVIRONMENTS

N. Jiang, C. Schmidt, V. Matossian and M. Parashar [2]

CONTENT-BASED MIDDLEWARE FOR DECOUPLED INTERACTIONS IN PERVASIVE ENVIRONMENTS

N. Jiang, C. Schmidt, V. Matossian and M. Parashar [3]

RUTGERS WIRELESS INFORMATION

NETWORK LABORATORY

Rutgers - The State University of New Jersey

73 Brett Road

Piscataway, New Jersey 08854-8060

Phone: 732-445-5954 *FAX: 732-445-3693*

ABSTRACT

The growing ubiquity of sophisticated sensor/actuator devices with embedded computing and communications capabilities, and the emergence of pervasive information and computational Grids require a middleware infrastructure that: (a) is scalable and self-managing, (b) is based on content rather than names and/or addresses, (c) supports asynchronous and decoupled interactions rather than forcing synchronizations, and (d) provides some interaction guarantees. In this paper we propose *Associative Rendezvous (AR)* as a paradigm for content-based decoupled interactions for pervasive applications. In this paper we also present *Meteor*, a content-based middleware infrastructure to support AR interactions. The design, implementation, and evaluation of Meteor are presented.

Table of Contents

Keywords: content-based middleware, associative rendezvous messaging, decoupled interaction, content-based routing, DHT, pervasive computing.

1. Introduction

The growing ubiquity of sophisticated sensor/actuator devices with embedded computing and communications capabilities [1], and the emergence of pervasive information and computational Grids [2] are enabling new generations of applications based on seamless access, aggregation, and interactions [3]. For example, one can conceive of a fire management application where computational models use streaming information from sensors embedded in the building along with real time and predicted weather information (temperature, wind speed and direction, humidity) and archived history data to predict the spread of the fire and to guide firefighters, warning of potential threats (blowback if a door is opened) and indicating most effective options. This information can also be used to control actuators in the building to manage the fire and reduce damage.

Other examples include scientific/engineering applications that symbiotically and opportunistically combine computations, experiments, observations, and real-time data to manage and optimize its objectives (e.g. oil production, weather prediction) [3], pervasive applications that leverage the pervasive/ubiquitous information Grid to continuously manage, adapt, and optimize our living context (e.g. your clock estimates drive time to your next appointment based on current traffic/weather and warns you appropriately), crisis management applications that use pervasive conventional and unconventional information for crisis prevention and response, medical applications that use in-vivo and in-vitro sensors and actuators for patient management, ad hoc distributed control systems for automated highway systems, manufacturing system or unmanned airborne vehicles, and business applications that use anytime-anywhere information access to optimize profits.

The defining characteristic of these applications is their ability to leverage the pervasive Grid infrastructure to continuously manage, adapt, and optimize themselves to meet their objectives. However, these applications and the underlying pervasive environment are inherently large, distributed, complex, heterogeneous and dynamic. As a result, supporting these applications requires a middleware infrastructure that: (a) is scalable and self-managing, (b) is based on content rather than names and/or addresses, (c) supports asynchronous and decoupled interactions rather than forcing synchronizations, and (d) provides some interaction guarantees.

In this paper we propose *Associative Rendezvous (AR)* [1] as a paradigm for content-based decoupled interactions for pervasive applications. AR extends the conventional name/identifier-based rendezvous [5, 6] in two ways. First it uses flexible combinations of keywords (i.e. keywords, partial keywords, wildcards, ranges) from a semantic information space, instead of opaque identifiers that have to be globally synchronized. Second, it enables the reactive behaviors at rendezvous points to be embedded in the message or message request. AR differs from emerging publish/subscribe paradigms in that individual interests (subscriptions) are not used for routing and do not have to be synchronized - they can be locally modified at a rendezvous node at anytime. In this paper we also present *Meteor*, a content-based middleware infrastructure to support AR interactions. The design, implementation and evaluation of Meteor is presented.

The rest of this paper is organized as follows. Section 2 presents the Associative Rendezvous interaction paradigm and its semantics. Section 3 presents the design and implementation of Meteor. Section 4 presents an experimental evaluation of Meteor. Sections 5 presents related work. Section 6 concludes this paper.

[1] The term associative to describe content-based interactions was introduced in [4]

2. Associative Rendezvous

Associative Rendezvous (AR) is a paradigm for content-based decoupled interactions with programmable reactive behaviors. Rendezvous-based interactions [5] provide a mechanism for decoupling senders and receivers. Senders send messages to a *rendezvous point* without knowledge of which or where the receivers are. Similarly, receivers receive messages from a *rendezvous point* without knowledge of which or where the senders are. Note that senders and receivers may be decoupled in both space and time [5]. Such decoupled asynchronous interactions are naturally suited for large, distributed and highly dynamic systems such as pervasive Grid environments.

In conventional rendezvous interactions, *rendezvous points* are defined by opaque identifiers [6] that have to be globally synchronized before they can be used. This limits both, the scalability as well as the dynamism of the system. Associative interactions [4, 7] use semantic content-based resolution, used by the naming service, to enable interactions. In associative interactions participating clients locally maintain and export "profiles" consisting of attributes specifying credentials, context, state, interests, capabilities, etc. Messages are similarly enhanced to include "semantic-selectors". The semantic-selector is a prepositional expression over all possible attributes and specifies the profile(s) that are to receive the message. Thus the notion of a static client or client group name used by conventional interactions is subsumed by the selector which descriptively names dynamic sets of clients of arbitrary cardinality. Associative interactions only require the existence of globally known information spaces (ontologies), and eliminates the need for expensive synchronization and complex tracking protocols in pervasive Grid environments.

AR extends the conventional name/identifier-based rendezvous in two ways. First, it uses flexible combinations of keywords (i.e, keyword, partial keyword, wildcards, ranges) from a semantic information space, instead of opaque identifiers (names, addresses) that have to be globally known. Interactions are based on content described by keywords, such as the type of data a sensor produces (temperature or humidity) and/or its location, the type of functionality a service provides and/or its QoS guarantees, and the capability and/or the cost of a resource. Second, it enables the reactive behaviors at the rendezvous points to be encapsulated within messages increasing flexibility and expressibility, and enabling multiple interaction semantics

(e.g. broadcast, multicast, notification, publisher/subscriber, mobility, etc.).

2.1 The Semantics of Associative Rendezvous Interactions

The AR interaction model consists of three elements: *Messages*, *Associative Selection*, and *Reactive Behaviors*.

2.1.1 AR Messages.

An AR message is defined as the triplet: (*header*, *action*, *data*). The data field may be empty or may contain the message payload. The header includes a semantic *profile* in addition to the credentials of the sender, a message context and the TTL (time-to-live) of the message. The profile is a set of attributes and/or attribute-value pairs, and defines the recipients of the message. The attribute fields must be keywords from a defined information space while the values field may be a keyword, partial keyword, wildcard, or range from the same space. At the rendezvous point, a profile is classified as a *data profile* or an *interest profile* depending on the action field of the message. A sample data profile used by a sensor to publish data is shown in Figure 2.1 (a), and a matching interest profile is shown in Figure 2.1 (b). Note that the number or order of the attribute/attribute-value pairs in a profile are not restricted. However our current prototype requires that the maximum possible attribute/attribute-value pairs must be predefined. The *action* field of the AR message defines the reactive behavior at the rendezvous point and is described in Section 2.1.3.

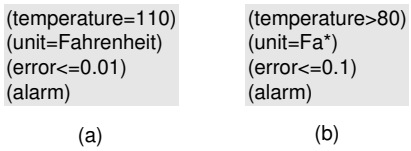

Figure 2.1: Sample message profiles: (a) a data profile for a sensor; (b) an interest profile for a client.

The AR interaction model defines a single symmetric *post* primitive. To send a message, the sender composes a message by appropriately defining the header, action and data fields, and invokes the post primitive. The post primitives resolve the profile of the message and deliver the message to relevant rendezvous points. The profile resolution guarantees that all the rendezvous points that match the profile will be identified. However, the actual delivery relies on existing transport protocols. A receive operation is similar except that the action field is defined appropriately and the data field is empty.

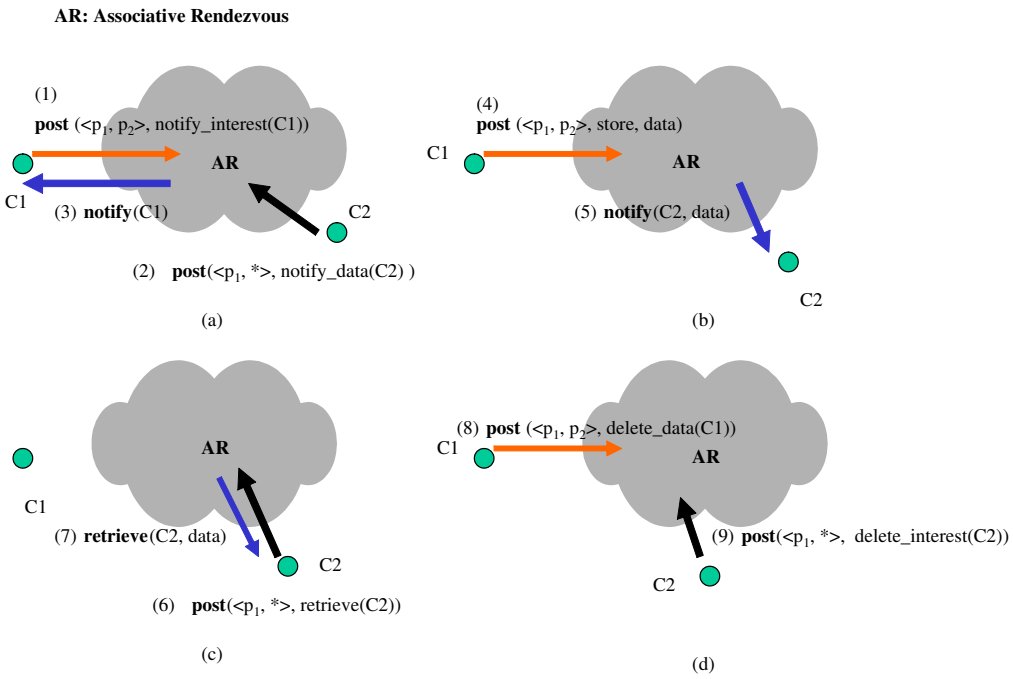

Figure 2.2: An illustrative example.

2.1.2 Associative Selection

Profiles are represented using a hierarchical schema that can be efficiently stored and evaluated by the selection engine [8, 4]. A profile p represents a path in the hierarchical schema, $[e_0 \triangle ... e_k]$, where $\triangle$ can be a parent-child ("/") operator (i.e. at adjacent levels) or an ancestor-descendant ("//") operators (i.e. separated by more than one level). Within a level, the profile defines a propositional expression where $\triangle$ represents propositional operators, such as $\wedge$ and $\vee$, between elements at the same level. Note that the propositional expression at a level must evaluate to TRUE for the evaluation to continue to the next level. The elements in the profile can be an attribute, $e_i(a_i)$, or an attribute-value pair $e_i(k_i, v_i)$, where e_i is a keyword and v_i may be a keyword, partial keyword, wildcard or range. The singleton attribute a_i evaluates to true if and only if p contains the simple attribute a_i. The attribute-pair (a_i, u_i) evaluates to true with respect to a profile p, if and only if p contains an attribute a_i and the corresponding value v_i satisfy u_i. For example, the profile (a) is associatively selected by the profile (b) in Figure 2.1, since (1) both have matched singleton attribute *temperature*, (2) for attribute *degree*, $100 > 80$, which satisfies the binary relation, (3) for attribute *unit*, *Fahrenheit* matches wildcard *Fa∗*, (4) *error<0.01* satisfies the request *error<0.1*.

A key characteristic of the selection process is that it does not differentiate between interest and data profiles. This allows all messages to be symmetric where data profiles can trigger

the reactive behaviors associated with interest messages and vice versa. The matching system combines selective information dissemination with reactive behaviors. Further, both data and interest message are persistent with their persistence defined by the TTL field.

Actions	Semantics
store	store data profile and data; match message profile with existing interest profile; execute action if match.
retrieve	match message profile with existing data profiles; send data corresponding to each matching data profile to the sender.
notify_data notify_interest	match message profile with existing data/interest profiles; notify sender if there is at least one match.
delete_data delete_interest	match message profile with existing data/interest profiles; remove all matching data profiles and data from the system in case of delete_data; remove all matching interest profiles from the system in case of delete_interest.

Table 2.1: Basic reactive behaviors.

2.1.3 Reactive Behaviors.

The *action* field of the message defines the reactive behavior at the rendezvous point. Basic reactive behaviors currently defined include *store*, *retrieve*, *notify*, and *delete* as shown in Table 2.1. The *notify* and *delete* actions are explicitly invoked on a data or an interest profile. The *store* action stores data and data profile at the rendezvous point. It also causes the message profile to be matched against existing interest profiles and associated actions to be executed in case of a positive match. The *retrieve* action retrieves data corresponding to each matching data profile. The *notify* action matches the message profile against existing interest/data profile, and notifies the sender if there is at least one positive match. Finally, the *delete* action deletes all matching interest/data profiles. Note that the actions will only be executed if the message header contains an appropriate credential. Also note that each message is stored at the rendezvous for a period corresponding to the TTL defined in its header. In case of multiple matches, the profiles matching are processed in random order.

2.2 Illustrative Examples

The operation of the model is illustrated in Figure 2.2. In Figure 2.2(a), client C1 first requests notification of interest with profile $< p1, p2 >$. Client C2 then requests notification of data corresponding to its interest profile $< p1, * >$. This causes a notification to be sent to C1. C1

now posts data with data profile $< p1, p2 >$ (Figure 2.2(b)). Since this data profile matches C2's interest profile, a notification is sent to C2. C2 now requests data with interest profile $< p1, * >$ (Figure 2.2(c)). This matches C1's data profile and the corresponding data is sent to C2. The example assumes that the TTL for data and interest profiles have not expired. C1 and C2 now delete the data and interest respectively (Figure 2.2(d)).

Note the symmetric behavior of the *post* operator. As seen in Figure 2.2(a), a client can subscribe to both interests and data. This is particularly useful for sensor networks where an energy constrained sensor may want to produce data when there is an interest for its data, allowing it to use power and bandwidth more effectively.

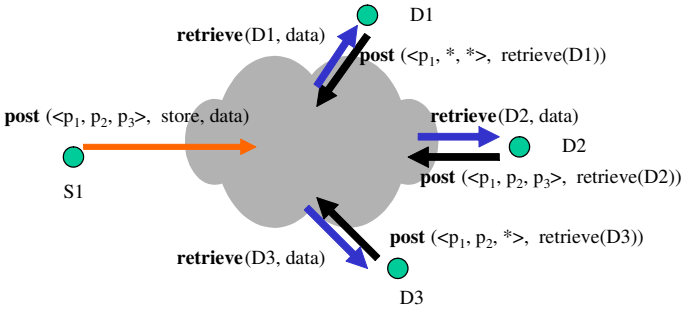

Figure 2.3: One-to-many interactions using Associative Rendezvous.

AR can be used to realize different interaction semantics such as one-to-many, one-to-some, one-to-all while appropriately setting data and interest profiles. Further, as these profiles are defined locally by a client, no synchronization is required to achieve these interaction semantics. Figure 2.3 illustrates a one-to-many (e.g. multicast) interaction using AR.

3. Meteor: A Content-based Middleware Infrastructure for Decoupled Interactions

Meteor is a content-based middleware infrastructure for decoupled interactions in pervasive Grid environments based on the Associative Redendezvous model. It is essentially a dynamic P2P network of Rendezvous Peers (RP). To use Meteor, applications must have access to at least one RP and can post messages to this RP. A schematic overview of the Meteor stack is presented in Figure 3.1. It consists of 3 key components: (1) a self-organizing overlay, (2) a content-based routing infrastructure (Squid), and (3) the Associative Rendezvous Messaging Substrate (ARMS). These components are described below.

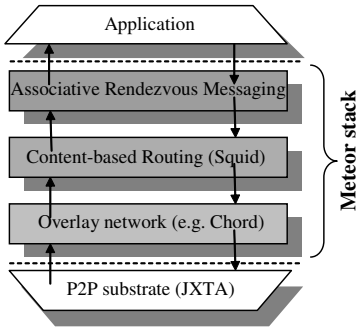

Figure 3.1: A schematic overview of the Meteor stack.

3.1 The Overlay Network Layer - Chord

The Meteor overlay network is composed of RP nodes, which may be access points or message forwarding nodes in ad-hoc sensor networks and servers or end-user computers in wired networks. RP nodes can join or leave the network at any time.

The current Meteor overlay network uses Chord [9]. Peer nodes in the Chord overlay form a ring topology. Every node in the Chord overlay is assigned a unique identifier ranging from 0 to 2^m-1 using consistent hashing [10]. The identifiers are arranged as a circle modulo 2^m. Each node maintains information about (at most) m neighbors, called *fingers*, in a *finger table*. The finger table is used for efficient routing and enables data lookup with O(log N) cost [9], where N is the number of nodes in the system. The finger table is constructed when a node joins

the overlay, and it is updated any time a node joins or leaves the system. The cost of a node join/leave is $O(\log^2 N)$. An example of an overlay network with 5 nodes and an identifier space from 0 to 2^4-1 is shown in Figure 3.2.

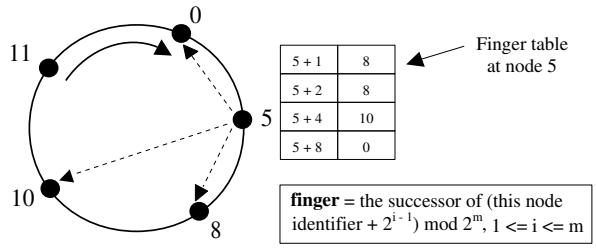

Figure 3.2: Example of the overlay network. Each node stores the keys that map to the segment of the curve between itself and the predecessor node.

Advantages of Chord include its guaranteed performance, logarithmic in number of messages, and its ease of implementation. Drawbacks include the cost of node join and leave operations (i.e. key reallocation) and the fact that constant periodic messages are required to maintain the ring (i.e. update propagation).

The overlay network layer of the Meteor stack provides a simple abstraction to the layers above, consisting of a single operation: **lookup**(*identifier*). Given an identifier, this operation locates the node that is responsible for it, i.e, the node with an identifier that is the closest identifier greater than or equal to the queried identifier. Application names can be mapped to identifiers using hashing mechanisms, and then mapped to nodes in the overlay network.

3.2 Content-based Routing with Squid

Squid builds on top of the Chord overlay to enable flexible content-based routing. As mentioned above, the **lookup** operator provided by the Chord overlay requires an exact identifier. Squid effectively maps complex queries consisting of keyword tuples (multiple keywords, partial keywords, wildcards, and ranges) onto clusters of identifiers, and guarantees that all peers responsible for identifiers in these clusters will be found with bounded costs in terms of number of messages and the number of intermediate RP nodes involved.

Keywords can be common words or values of globally defined attributes, depending on the nature of the application that uses Squid. These keywords form a multidimensional keyword space; keyword tuples represent points in this space and the keywords are the coordinates. In Figure 3.3 (a) the keyword tuple (2, 1) is a point in a 2-dimensional space. A *keyword tuple* is defined as a list of d keywords, wildcards and/or ranges, where d is the dimensionality of the keyword space. For example (computer, network), (computer, net*) and (comp*, *) are

all valid keyword tuples in a 2-dimensional space. A keyword tuple containing ranges can be (256 - 512MB, *, 10Mbps - *) and specifies a computational resource with memory between 256 and 512 MB, any CPU frequency and at least 10Mbps base bandwidth. If the keyword tuple contains only complete keywords, it is called *simple*, and if it contains partial keywords, wildcards and/or ranges it is called *complex*.

The key innovation of Squid is the use of a locality preserving and dimension reducing indexing scheme, based on the Hilbert Space Filling Curve (SFC), which effectively maps the multidimensional information space to the peer identifier space. An SFC [11, 12, 13] is a continuous mapping from a *d*-dimensional space to a 1-dimensional space, generated recursively. Figure 3.3 (b) shows an example of Hilbert SFC, in a 2-dimensional space. Additional details about the use of Hilbert SFC in Squid can be found in [14].

Content-based routing in Squid is achieved as follows: SFCs are used to generate the 1-dimensional index space from the multi-dimensional keyword space. Applying the Hilbert mapping to this multi-dimensional space, each profile consisting of a simple keyword tuple can be mapped to a point on the SFC. Further, any complex keyword tuple can be mapped to regions in the keyword space and the corresponding clusters (segments of the curve) in the SFC (see Figure 3.4 (a)). The 1-dimensional index space generated corresponds to the 1-dimensional identifier space used by the Chord overlay. Thus, using this mapping RP nodes corresponding to any simple or complex keyword tuple can be located. The Squid layer of the Meteor stack provides a simple abstraction to the layer above consisting of a single operation: **deliver**(*keyword tuple, data*), where *data* is the message payload provided by the messaging layer above. The routing process is described below.

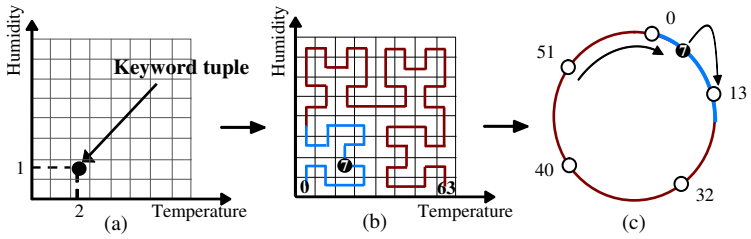

Figure 3.3: Routing using a simple keyword tuple in Squid: (a) the simple keyword tuple (2, 1) is viewed as a point in a multidimensional space; (b) the keyword tuple is mapped to the index 7, using Hilbert SFC; (c) the data will be routed in the overlay (an overlay with 5 RP nodes and an identifier space from 0 to 2^6-1) at RP node 13, the successor of the index 7.

Routing using simple keyword tuples.

The routing process for a simple keyword tuple is illustrated in Figure 3.3. It consists of two steps: first, the SFC-mapping is used to construct the index of the destination RP node from the simple keyword tuple, and then, the overlay network lookup mechanism is used to route to the appropriate RP node in the overlay.

Routing using complex keyword tuples.

The complex keyword tuple identifies a region in the keyword space, which in turn corresponds to clusters of points in the index space. For example, in Figure 3.4 (a), the complex keyword tuple (2-3, 1-5) representing data read by a sensor (temperature between 2 and 3 units and humidity between 1 and 5 units) identifies 2 clusters with 6 and 4 points respectively.

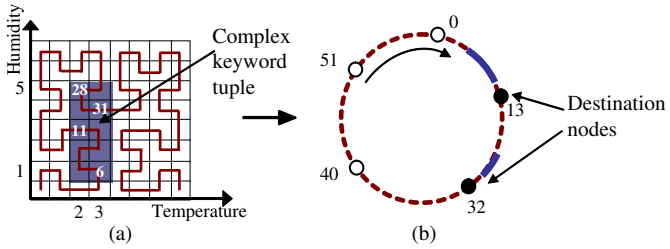

Figure 3.4: Routing using complex keyword tuple (2-3, 1-5): (a) the keyword tuple defines a rectangular region in the 2-dimensional keyword space consisting of 2 clusters (2 segments on the SFC curve); (b) the clusters (the solid part of the circle) correspond to destination RP nodes 13 and 32, which are routed to.

Thus a complex keyword tuple is mapped by the Hilbert SFC to clusters of SFC indices and correspondingly, multiple destination identifiers. Once the clusters associated with the complex keyword tuple are identified, a straightforward approach consists of using the overlay lookup mechanism to route individually to each RP node. Figure 3.4 illustrates this process. However, as the originating RP node cannot know how the clusters are distributed in the overlay network, the above approach can lead to inefficiencies and redundant messages, especially when there are a large number of clusters. The routing process can be optimized by using the recursive nature of the SFC to distribute the list of clusters to be routed. This optimization is presented in detail in [14].

3.3 Associative Rendezvous Messaging Substrate

The matching engine component is essentially responsible for matching profiles. An incoming message profiles is matched against existing interest and/or data profiles depending on the

```
<message id="xxxxxx">
   <header>
        <credentials FireFighterId = "xxxxxx"/>
        <profile maxNumAttributes="8">
            <temperature operator=">=", value="80">
                <unit value="Fa*"/>
                <error operator="<=" value="0.1"/>
            </temperature>
            <wind>
                <degree value="*"/>
                <direction value="*"/>
            </wind>
            <alarm/>
        </profile>
        <ttl>600000</ttl>
   <header>

   <action>
        retrieve(123. 34. 56. 78 : 9999)
   </action>
</message>
```

Figure 3.5: Sample XML message with interest profile.

desired reactive behavior. If the result of the match is positive, then the action field of the incoming message is executed first and then the action field of the matched profile is evaluated.

```
<message id="xxxxxx">
   <header>
        <credentials sensorID = "xxxxxx"/>
        <profile maxNumAttributes="8">
            <temperature  value="110">
                <unit value="Fahrenheit"/>
                <error operator="<=" value="0.01"/>
             <wind>
                <speed value="20MPH'/>
                <direction value="northwest"/>
            </wind>
            <alarm/>
        </profile>
        <ttl>200000</ttl>
   <header>

   <action>store</action>

   <payload>
        <location>
            Rutgers.CoreBuilding.Room832
        </location>
        <alarm>
           <status>caution</status>
        </alarm>
   </payload>
</message>
```

Figure 3.6: Sample XML message with data profile.

The ARMS layer implements the Associative Rendezvous interaction model. At each RP, ARMS consists of two components: the *profile manager* and the *matching engine*. The profile manager manages locally stored profiles. Profiles are implemented as XML files. Sample XML messages with interest and data profiles are illustrated in Figure 3.5 and Figure 3.6 respectively. The managers monitors message credentials and contexts and ensures that related constraints are satisfied. For example, a client cannot retrieve data that it is not authorized to. Similarly,

a client cannot delete a profile it is not authorized to. The profile manager is also responsible for garbage collection. It maintains a local timer and purges interest and data profiles when their TTL fields have expired. Finally, the profile manager executes the action corresponding to a positive match.

3.4 Implementation Overview

The current implementation of Meteor builds on Project JXTA (http://www.jxta.org), a general-purpose peer-to-peer framework. JXTA defines concepts, protocols, and a network architecture. JXTA concepts include peers, peergroups, advertisements, modules, pipes, rendezvous and security. JXTA defines protocols for : (1) discovering peers (Peer Discovery Protocol, PDP), (2) binding virtual end-to-end communication channels between peers (Pipe Binding Protocol, PBP), (3) resolving queries (Peer Resolver Protocol, PRP), (4) obtaining information on a particular peer, such as its available memory or CPU load (Peer Information Protocol, PIP) (5) propagating messages in a peergroup (Rendezvous protocol, RVP), (6) determining and routing from a source to a destination using available transmission protocols (Endpoint Routing Protocol, ERP). The JXTA architecture builds on three layers, a core layer, for essential common functionalities, a service layer, for additional pluggable/unpluggable behaviours, and an application layer for end-to-end high-level control.

The overlay network, Squid and the ARMS layers of the Meteor stack are implemented as event-driven JXTA services. Each layer registers itself as a listener for specific messages, and gets notified when a corresponding event is raised.

Since Meteor is designed as an overlay network of redezvous peers, it is incrementally deployable. A joining RP uses the Chord overlay protocol and becomes responsible for an interval in the identifier space. In this way, the addition of a new rendezvous node is transparent to the end-hosts.

The overall operation of the Meteor overlay consists of two phases: bootstrap and running. During the bootstrap phase (or join phase) messages are exchanged between a joining RP and the rest of the group. During this phase, the RP attempts to discover an already existing RP in the system and construct its routing table. The joining RP sends a discovery message to the group. If the message unanswered after a set duration (in the order of seconds), the RP assumes that it is the first in the system. If a RP responds to the message, the joining RP queries this bootstrapping RP according to the Chord join protocol and updates routing tables to reflect the join.

The running phase consists of a stabilization and a user mode. In stabilization mode, a RP responds to queries issued by other RPs in the system. The purpose of the stabilization mode is to ensure routing tables are up to date, and to verify that other RPs in the system have not failed or left the system. In user mode, RPs paritcipates in interactions as part of the Squid and ARMS layers. The ARMS matching engine at each RP is based on MySQL (http://www.mysql.com), a lightweight SQL database. Finally, message credentials are not currently implemented.

4. Experimental Evaluation

Meteor was experimentally evaluated using a prototype deployment on a Linux cluster consisting of 64 1.6 GHz Pentium IV machines and an 100 Mbps Ethernet interconnect. Each of the 64 nodes acted as a RP and ran the complete Meteor stack. Profiles at each RP were locally stored in a MySQL database. The overheads at each layer of the stack were measured. Further, we used simulations of up to 5000 RPs and 10^6 unique profiles to evaluate the scalability of the content-based routine layer. The experiments are presented below.

4.1 Overlay Network Layer

This experiment measured the latency for peer lookup in the overlay network as a function of the size of the system. To get an accurate measurement of the latencies, a single peer was run on each node of the cluster and each peer sent messages to a randomly selected destination peer. Each message required an overlay lookup operation. Average times are plotted in Figure 4.1 (a). The graph shows that the average elapsed time is not affected by the linear growth of the size of the system, validating the scalability of the overlay network lookup operation and the Chord routing algorithm.

4.2 Content-based Routing Layer

This experiment measured the overhead of routing using complex keyword tuples. Three sets of keyword tuples were used, the first containing wildcards, the second containing ranges, and the third containing both wildcards and ranges. The routing overheads at the Squid layer were measured at each RP and averaged. The results are plotted in Figure 4.1 (b). The measured overhead includes times for cluster refinments and subcluster lookup. As Figure 4.1 (b) shows, the overhead grows slowly and at a much smaller rate than the system size. This demonstrates that Squid can effectively scale to large numbers of peers while maintaining acceptable routing times. As expected, the routing times are high for queries with wildcards as they involve a larger number of clusters and correspondingly larger number of peers.

To evaluate the scalability of the content-based routing layer we simulated Squid with up

to 5000 peers where each peer is an RP, and up to 10^6 unique profiles. In the simulations, the number of profiles stored in the system increases with the number of RP peers. We used complex keyword tuples to test the routing scalability: keyword tuples with partial keywords and wildcards, and keyword tuples containing ranges. Figure 4.2 shows results for a 3D keyword space. The number of nodes that process the query is a small fraction of the total nodes, and it increases at a slower rate than the size of the system. For wildcard queries, the average number of processing nodes is below 11%, and the number of nodes that found matches is below 6%. These percentages decrease as the system size increases, demonstrating the scalability of the system. As Figure 4.2 shows, range queries are more efficient than wildcard queries, which is expected, as query optimization and pruning are more effective for range queries. Additional simulation results can be found in [14].

4.3 Associative Rendezvous Messaging Substrate

This experiment measured the matching overhead at each RP node. The reaction action type for all messages in this experiment was 'notification'. Only the overhead of querying the database and of constructing the notification message was considered. The notification delivery was done outside of the Meteor stack and was not measured. The experiment was conducted for profiles containing sets of complex keywords tuples containing wildcards and/or ranges. The number of profiles in the database was varied. The results are plotted in Figure 4.1 (c). The results were grouped based on the results of the matching expressed as a percentage of the total number of stored profiles, and averaged. As seen in Figure 4.1 (c), for a moderate-size database (up to 10^4 profiles) the overhead incurred is very low. The overhead increases substantially when 10^5 profiles are stored locally, as could have been expected given the memory and data access times required by such a large number of items. However, we believe that even 10^4 profiles seems greater than needed for an RP.

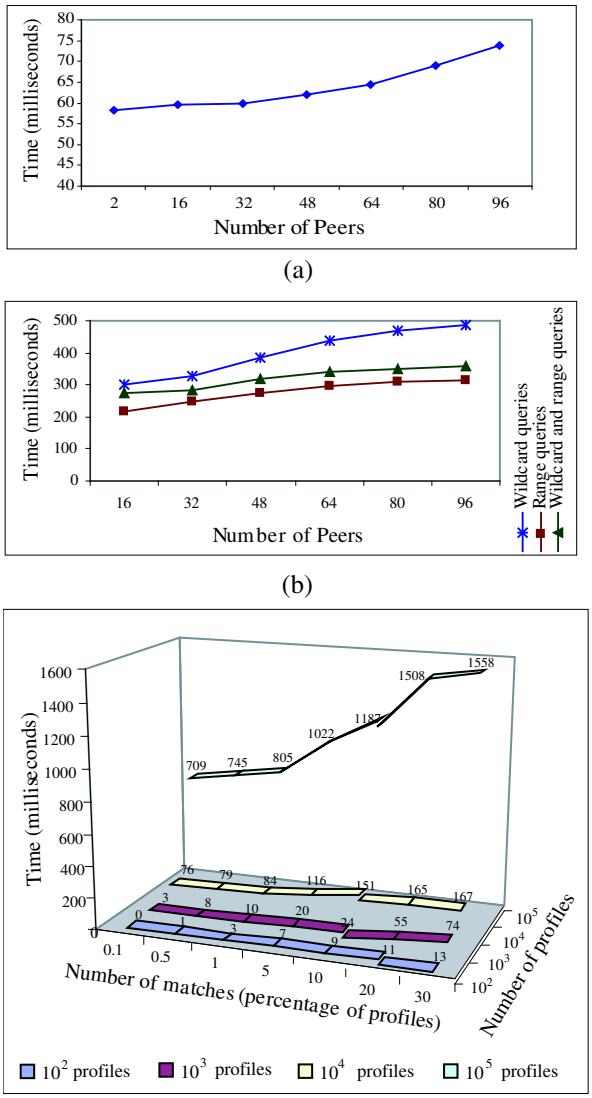

Figure 4.1: (a) Overlay network lookup overhead (Chord); (b) Content-based routing overhead (Squid); (c) Matching overhead at a single RP (ARMS).

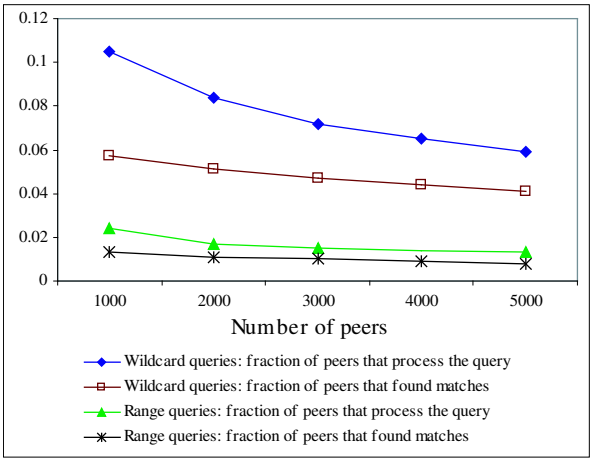

Figure 4.2: Simulated results for a P2P system with up to 5000 RPs and up to 10^6 interest profiles. The results show the percentage of peers that process a query (on average) and the percentage of peers that found matches, for wildcard queries and for range queries respectively.

5. Related Work

Content based decoupled interactions has been addressed by publish-subscribe-notify (PSN) models [5]. PSN based systems include Sienna [15] and Gryphon [16]. The associative rendezvous model differs from PSN systems in that individual interests (subscriptions) are not used for routing and do not have to be synchronized - they can be locally modified at a rendezvous node at anytime. While PSN systems provide flexible matching capabilities, scalability remains a concern in these systems.

i3 [6] provides a similar rendezvous-based abstraction and has influenced this work. However, an i3 identifier is opaque and must be globally known. Associative rendezvous uses semantic identifiers that are more expressive and only require the existence of agreed on information spaces (ontologies). Besides, its dynamic binding semantics enables profiles to be added, deleted or changed on-the-fly.

The associative broadcast [4] paradigm has also influenced this effort. The key difference between this model and associative rendezvous is that the binding of profiles takes place at intermediate nodes instead of the broadcast medium. As a result associative broadcast only supports transient interactions. Further, its scalability over wide areas is a concern.

Unlike other rendezvous-based models [17], associative rendezvous enables programmable reactive behaviors at rendezvous points using the action field within a message. Further, interactions in the associative rendezvous model are symmetric allowing participants to simultaneously be producers and consumers. Finally, Grid messaging systems such as the Narada Broker [18] focus on persistence and reliable message delivery rather than content-based interactions.

6. Summary and Conclusions

As the scale, complexity, heterogeneity and dynamism of pervasive Grid environments increase, interaction paradigms based on static names (addresses, identifiers) and on synchronous or strictly coupled interactions are quickly becoming insufficient. This has lead researchers to consider alternative paradigms that are decoupled and content based. In this paper, we presented Associative Rendezvous, a content-based decoupled interaction abstraction model for pervasive Grid environments. AR extends conventional name/identifier-based rendezvous in two ways. First, it uses flexible combinations of keywords (i.e. keywords, partial keywords, wildcards, ranges) from a semantic information space instead of opaque identifiers that have to be globally known. Second, it enables the reactive behavior at rendezvous points to be defined by the message. Messages and interactions are symmetric allowing participants to simultaneously be producers and consumers. For example, a sensor may produce data only when there is an interest for its data, allowing it to conserve energy and bandwidth.

In this paper we also presented the design, implementation and evaluation of Meteor, a content based interaction infrastructure based on AR. Initial evaluation results demonstrate its scalability and effectiveness as a paradigm for pervasive Grid environments. We are currently working on a wider deployment of Meteor. We are also investigating alternate overlay network topologies.

References

[1] Estrin, D., Govindan, R., Heidemann, J., Kumar, S.: Next century challenges: Scalable coordination in sensor networks. In Proceedings of the Fifth Annual International Conference on Mobile Computing and Networks (MobiCOM '99), Seattle, Washington (1999)

[2] Gibbons, P.B., Karp, B., Ke, Y., Nath, S., Seshan, S.: Irisnet: An architecture for a worldwide sensor web. IEEE Pervasive Computing (2003) 22–33

[3] Matossian, V., Parashar, M.: Autonomic optimization of an oil reservoir using decentralized services. Proceedings of the 1st International Workshop on Heterogeneous and Adaptive Computing– Challenges for Large Applications in Distributed Environments (CLADE 2003), Seattle, WA, USA (2003) 2–9

[4] Bayerdorffer, B.: Distributed programming with associative broadcast. Proceedings of the 27th Annual Hawaii International Conference on System Sciences, Volume 2: Software Technology (HICSS94-2), Wailea, HW, USA (1994) 353–362

[5] Eugster, P.T., Felber, P.A., Guerraoui, R., Kermarrec, A.M.: The many faces of publish/subscribe. ACM Computing Surveys **35** (2003) 114–131

[6] Stoica, I., Adkins, D., Zhuang, S., Shenker, S., Surana, S.: Internet indirection infrastructure. Proceedings of ACM SIGCOMM'02, Pittsburgh, PA (2002) 73–86

[7] Bhandarkar, P., Parashar, M.: Semantic communication for distributed information coordination. In Proceedings of the IEEE Conference on Information Technology, Syracuse, NY (1998) 149–152

[8] Diao, Y., Altinel, M., Franklin, M.J., Zhang, H., Fischer, P.: Path sharing and predicate evaluation for high-performance xml filtering. TODS (2003)

[9] Stoica, I., Morris, R., Karger, D., Kaashoek, M.F., Balakrishnan, H.: Chord: A scalable peer-to-peer lookup service for internet applications. In: Proceedings of the ACM SICOMM'01 Conference, San Diego, California (2001) 149–160

[10] Karger, D., Lehman, E., Leighton, T., Levine, M., Lewin, D., Panigrahy, R.: Consistent hashing and random trees: Distributed caching protocols for relieving hot spots on the world wide web (1997)

[11] Bially, T.: A Class of Dimension Changing Mapping and its Application to Bandwidth Compression. PhD thesis, Polytechnic Institute of Brooklyn (1967)

[12] Butz, A.R.: Alternative algorithm for hilbert's space-filling curve. IEEE Transactions on Computers **20** (1971) 424–426

[13] Sagan, H.: Space-Filling Curve. Springer Verlag (1995)

[14] Schmidt, C., Parashar, M.: Flexible information discovery in decentralized distributed systems. In: Proceedings of the 12th High Performance Distributed Computing (HPDC), IEEE Press (2003) 226–235

[15] Carzaniga, A., Wolf, A.L.: Content-based networking: A new communication infrastructure. NSF Workshop on an Infrastructure for Mobile and Wireless Systems (2001)

[16] IBM: Gryphon: publish/subscribe over public networks. (Internet: http://www.research.ibm.com/gryphon/papers/Gryphon-Overview.pdf)

[17] Gao, J., Steenkiste, P.: Rendezvous points-based scalable content discovery with load balancing. In Proceedings of the Fourth International Workshop on Networked Group Communication (NGC'02), Boston, MA (2002) 71–78

[18] Fox, G., Pallickara, S., Rao, X.: A Scaleable Event Infrastructure for Peer to Peer Grids. In: Proceedings of the 2002 joint ACM-ISCOPE conference on Java Grande, Seattle, Washington, USA, ACM Press (2002) 66–75