Washington University in St. Louis
Washington University Open Scholarship

All Computer Science and Engineering Research

Computer Science and Engineering

Report Number: WUCSE-2004-33

2004-06-18

Knowledge-driven Interactions With Services Across Ad Hoc Networks

Rohan Sen, Radu Handorean, Gruia-Catalin Roman, and Gregory Hackmann

Service oriented computing, with its aim of unhindered in-teroperability, is an appropriate paradigm for ad hoc net-works, which are characterized by physical mobility of het-erogenous hosts and by the absence of standardized application level protocols. The decoupled nature of computing in ad hoc networks can result in disconnections at inoppor-tune times during the client-service interaction process. We introduce the notion of a priori selection of services to reduce the likelihood of disconnection during service usage. A client may specify the times when it requires certain ser-vices. A knowledge base of the physical motion profiles of various service providers is... **Read complete abstract on page 2.**

Follow this and additional works at: https://openscholarship.wustl.edu/cse_research

Recommended Citation
Sen, Rohan; Handorean, Radu; Roman, Gruia-Catalin; and Hackmann, Gregory, "Knowledge-driven Interactions With Services Across Ad Hoc Networks" Report Number: WUCSE-2004-33 (2004). *All Computer Science and Engineering Research.*
https://openscholarship.wustl.edu/cse_research/1006

Department of Computer Science & Engineering - Washington University in St. Louis
Campus Box 1045 - St. Louis, MO - 63130 - ph: (314) 935-6160.

This technical report is available at Washington University Open Scholarship: https://openscholarship.wustl.edu/cse_research/1006

Knowledge-driven Interactions With Services Across Ad Hoc Networks

Rohan Sen, Radu Handorean, Gruia-Catalin Roman, and Gregory Hackmann

Complete Abstract:

Service oriented computing, with its aim of unhindered in-teroperability, is an appropriate paradigm for ad hoc net-works, which are characterized by physical mobility of het-erogenous hosts and by the absence of standardized application level protocols. The decoupled nature of computing in ad hoc networks can result in disconnections at inoppor-tune times during the client-service interaction process. We introduce the notion of a priori selection of services to reduce the likelihood of disconnection during service usage. A client may specify the times when it requires certain ser-vices. A knowledge base of the physical motion profiles of various service providers is used to select instances of a ser-vice that are co-located with the client at the required time and least likely to disconnect. A system for constructing the knowledge base is presented in this paper, along with the implementation details and the algorithm used to deter-mine the service usage pattern.

Department of Computer Science & Engineering

Washington University in St. Louis
School of Engineering & Applied Science

2004-33

Knowledge-driven Interactions With Services Across Ad Hoc Networks

Authors: Sen, Rohan; Handorean, Radu; Roman, Gruia-Catalin; Hackmann, Gregory

June 18, 2004

Abstract: Service oriented computing, with its aim of unhindered interoperability,
is an appropriate paradigm for ad hoc networks,
which are characterized by physical mobility of heterogenous
hosts and by the absence of standardized application
level protocols. The decoupled nature of computing
in ad hoc networks can result in disconnections at inopportune
times during the client-service interaction process. We
introduce the notion of a priori selection of services to reduce
the likelihood of disconnection during service usage.
A client may specify the times when it requires certain services.
A knowledge base of the physical motion profiles of
various service providers is used to select instances of a service
that are co-located with the client at the required time
and least likely to disconnect. A system for constructing
the knowledge base is presented in this paper, along with
the implementation details and the algorithm used to determine

Department of Computer Science & Engineering - Washington University in St. Louis
Campus Box 1045 - St. Louis, MO - 63130 - ph: (314) 935-6160

Knowledge-driven Interactions With Services Across Ad Hoc Networks

Rohan Sen, Radu Handorean, Gruia-Catalin Roman, and Gregory Hackmann
Department of Computer Science and Engineering
Washington University in St. Louis
Campus Box 1045, One Brookings Drive
St. Louis, MO 63130-4899, USA
{rohan.sen, radu.handorean, roman, ghackmann}@wustl.edu

ABSTRACT

Service oriented computing, with its aim of unhindered interoperability, is an appropriate paradigm for ad hoc networks, which are characterized by physical mobility of heterogenous hosts and by the absence of standardized application level protocols. The decoupled nature of computing in ad hoc networks can result in disconnections at inopportune times during the client-service interaction process. We introduce the notion of a priori selection of services to reduce the likelihood of disconnection during service usage. A client may specify the times when it requires certain services. A knowledge base of the physical motion profiles of various service providers is used to select instances of a service that are co-located with the client at the required time and least likely to disconnect. A system for constructing the knowledge base is presented in this paper, along with the implementation details and the algorithm used to determine the service usage pattern.

1. INTRODUCTION

The increasing ubiquity of mobile devices presents new and unique opportunities for electronic collaboration. A majority of such mobile devices are equipped with wireless LAN cards that enable them to communicate with other devices in proximity. A collection of wireless-enabled mobile devices within communication range can join together to form an ad-hoc network–a dynamic, peer-to-peer network whose infrastructure is supported by the hosts that comprise it. Ad hoc networks, characterized by the physical mobility of hosts, demand a decoupled style of interaction.

A key purpose of ad hoc networking is to facilitate opportunistic interactions among heterogenous hosts that encounter each other in both predictable and unpredictable ways. This requires the participants in an ad hoc network to be able to interact with each other in a uniform fashion. Service oriented computing (SOC) is a promising candidate for solving this problem because of its emphasis on flexible architectures and unhindered interoperability. However, most service oriented architectures, such as the Service Location Protocol (SLP) [9], Jini [17], Salutation [13], and those employed for Web Services [1, 11] are designed for infrastructure-rich wired networks where disconnections are rare. In ad hoc networks, where hosts are resource poor, and the likelihood of disconnection is high, the above mentioned architectures cannot function correctly [15]. The environment of an ad hoc network requires new approaches to designing SOC architectures that can withstand the dynamism of the ad hoc network. Our previous studies [5] focussed on the development of a SOC system that was able to withstand the rigors of an ad hoc network. For that system, we chose a proxy-based architecture, originally proposed in Jini. In proxy-based SOC systems, a *service provider* offers a *service*, which is a program running on the *provider host*. The service provider advertises a *proxy* which is retrieved by interested *clients* and used as a handle to the service. In most proxy-based systems (including Jini), the connectivity between the proxy and the service must be maintained for the duration of the interaction for the client to be able to effectively exploit the service.

Maintaining connectivity between the proxy and the service for the duration of their interaction is challenging in ad hoc networks due to the unpredictable mobility patterns of hosts. This combined with the relatively modest range of current 802.11b wireless cards result in short and sporadic windows of communication between clients and service providers. However, premature termination of the interaction between the client and the service provider can cause undesirable behavior or abnormal program termination. This problem is further exacerbated when the service being used is part of a larger client application, since the failure of the proxy-service interaction can cause the entire client application behavior to become unstable and unpredictable. Current solutions to this problem involve eliminating the possibility of disconnection. For example, in a hoarding strategy, the code for the entire service is copied to the client machine and is used locally so that disconnections cease to be a factor. However this strategy cannot be employed in cases when the software footprint is very large or when the code is proprietary. Nomadic strategies have assured connectivity because they assume that mobile hosts will stay within communication range of access points that are attached to the wired infrastructure. Such strategies are limiting as they imposes bounds on the physical mobility of hosts in a manner which is most often neither practical nor desirable.

The approaches described illustrate that eliminating the possibility of disconnection is expensive, and usually impacts other aspects of program execution. Hence, our approach does not seek to eliminate the possibility of disconnection. Instead, we focus on trying to plan so that the disconnection does not occur at an inopportune time. In this paper, we introduce knowledge-driven interactions between the client and service provider as a strategy that leverages the temporal aspect of service requirement and availability to carefully select service providers that are likely to remain in communication range for the duration of the service requirement, thereby reducing the likelihood of a disconnection at a crucial juncture without compromising other aspects of program execution such as resource usage. The essential idea is to exploit knowledge about other hosts' physical motion to compute the time at which two hosts are likely to be within communication range for a reasonable interval of time. This information is then matched with the client application's *service requirement profile*, which is a list of services, and the instances in time that they are required. The result is a pro-actively planned *satisfying set*, a list of specific instances of services that are most likely to be co-located with the client at the times that they are required, and which will remain co-located for the projected duration of the need. More specifically, our contribution in this paper can be split in two distinct parts:

- An **algorithm** that computes the satisfying set for a service requirement profile given a *knowledge base* of the behavior patterns of other hosts in the ad hoc network.
- A **software architecture** that is responsible for building a knowledge base by gathering all the required information from other hosts in the ad hoc network.

We will also show how traditional SOC operations are affected by the introduction of knowledge-based interactions between clients and service providers.

The remainder of the paper is organized as follows. Section 2 provides background information on proxy-based services in ad hoc networks as well as related work. Section 3 describes a motivating example and formally defines the problem. The algorithm used to compute the satisfying set is described in Section 4 and our architecture to construct a knowledge base is described in Section 5. Section 6 describes our implementation of the proposed architecture. We present a discussion of our approach in Section 7 and conclude in Section 8.

2. BACKGROUND

In this section, we present the case for using proxy-based architectures for service oriented computing in ad hoc networks. We discuss the advantages of this approach along with some of our previous work that improves on the basic proxy concept. We also show the context in which information exploitation in proxy-based SOC can be used to give stronger guarantees and more desirable results. We follow this with a coverage of related work in the field.

2.1 Proxy-Based Services in Ad Hoc Networks

The idea of proxy-based service oriented architectures was first proposed in Jini. It has since been adapted for use in ad hoc networks as shown in [5]. Proxy-based architectures are especially effective in ad hoc networks for two reasons – a) They help reduce the amount of software required on the client side, thereby making thin clients possible and b) They abstract details of the protocol to be used between the client and the service provider. Since the proxy is a self contained piece of code that can communicate with the provider host, the client is not required to be aware of the communication protocol or possess any code to communicate with the provider. The client is required to only carry the code that allows it to browse for services and discover proxies. This results in a small footprint for the client software, which is useful when running such software on mobile devices. The fact that proxies abstract the communication protocol is especially useful in ad hoc networks, where standardized application level protocols are not prevalent. Hence, the abstraction of a heterogenous set of protocols by proxies allows a large set of hosts to communicate with multiple service providers without the overhead of needing to know the specific protocol for each provider.

Though proxies are a solution to certain problems associated with SOC in ad hoc networks, their usage does raise certain issues, some of which we attempted to solve in our previous work. In [6], we proposed an automatic code management system which transparently ships and installs proxy code on the client host (once the client has declared an interest in the service) as a solution to the problem of distributing the binary code required by clients to execute proxies. A proxy upgrade system described in [14] ensures that these proxies are upgraded transparently at run-time with very little impact on the client application so that the proxy software is kept consistent with upgrades on the software on the provider host without the client application having to explicitly handle this procedure. These mechanisms, which are portions of a larger system supporting SOC in ad hoc networks solve some of the issues associated with proxy-based SOC architectures. One of the remaining problems–that of ensuring that interactions between the proxy and the service are interrupted to the least possible extent–is the subject of this paper.

2.2 Related Work

Our work in this paper encompasses various topics such as meta information management, information gathering and dissemination, information semantics, and planned behavior. As such, we present a selection of related work from each of these topic areas.

Our overarching goal for introducing knowledge driven interactions to SOC in ad hoc networks was to establish a sense of order in a chaotic environment. In [16], the authors describe the Task Control Architecture (TCA), designed for autonomous agents that control robots. Among other things, the task control architecture uses the notion of perception of the environment to make decisions. The TCA uses information about the environment to ensure that the robot is not in any physical danger. We perform the same kind of knowledge aggregation, though it is used to protect us from the "danger" of unexpected disconnection.

All the knowledge aggregation and dissemination that must be done to support knowledge driven architectures happen at the meta-level, in the sense that all knowledge that is traded has little to do with the applications that are running on the individual hosts. Costa et al. propose a meta information management system in [2] that uses a centralized type repository that maintains the meta data related to a base object. However, the centralized repository creates a single point of failure. Because of this, we chose to have multiple decentralized repositories called knowledge bases on each host so that there would be a lesser chance of failure.

Another issue that relates to our work is information dissemination. Essentially, to support knowledge driven interactions, each host must disseminate knowledge about themselves so that others may react to it. In [8], the authors propose three schemes for information dissemination that are especially tailored to mobile ad hoc networks. The strategy described focusses on rapid dissemination of information without duplication. Three strategies - select then eliminate (STE), eliminate then select (ETS), and a hybrid of the two are described as ways to ensure that only the relevant hosts get the information. While this is a relevant concern, we chose to design a less complex system in the interest of saving computational resources and also because the tuple space paradigm we adopt (described in detail later) ensures that information is distributed to all connected hosts, which is exactly the set of hosts that we want to disseminate our knowledge to. Perry et al. [12] describe the usage of ontologies to extract semantic information from the information that is being distributed. However, ontologies are cumbersome in that every host in the network is required to be aware of the ontology, which is unreasonable in an ad hoc network, with no centralized repository that can distribute such an ontology. Hence, we adopted alternate ways to give semantics to knowledge that we gather and distribute.

Finally, in [7], the authors describe planned scheduling for the DECAF architecture. Essentially, the scheduler is able to take in functions that enable it to do planning. Two strategies are suggested. The first is contingency planning, where every possibility is evaluated and the best one chosen, and if that fails for some reason, the next best option is chosen. The second strategy is to give the scheduler a utility function, which it can then use as a metric to choose among options.

3. FORMAL PROBLEM SPECIFICATION

In this section, we use a motivating example to show how knowledge management can improve the quality of interactions among clients and service providers in an ad hoc network. We follow this with a formal definition of the problem.

3.1 Motivating Example

We consider a scenario involving cars travelling on a highway. The occupants of the cars carry PDA's which offer various kinds of services. In one of the cars, Robert is currently reading a book, but would like to listen to some music in 30 minutes time, when he would have finished reading the book. His PDA does not have the resources to play music, hence it must discover this functionality at run time.

Robert instructs his PDA to have an MP3 music service discovered and ready to use at 4:30 PM, which is 30 minutes from now. The PDA queries PDA's in other cars that are surrounding the car that Robert is in for an MP3 music service. When the replies come back, it turns out that there are three cars that offer the service. This is shown pictorially in Figure 1. Robert has also indicated that he would like to listen to the music for an hour and a half, so the service must be available for that duration. From the figure it is clear that the service offered by Car A will not remain connected for the required duration. Car B will remain connected for the duration, but it is farther away from Robert's car than Car C, which could result in signal degradation. Hence, Robert's PDA chooses the service on Car C as the service that it will use. It stores this information until it is time to invoke the service.

The determination of the correct candidate is done by exploiting knowledge about the other hosts in the network such as current location, velocity, and intended *motion profile*, which gives the host's projected location as a function of time. These parameters are collected by the client host.

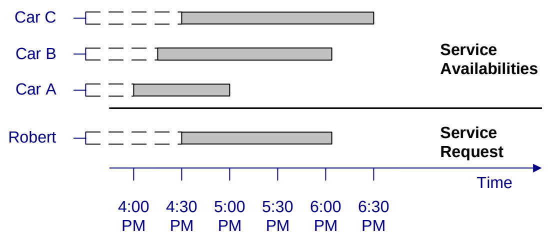

Figure 1: A client host obtaining information from two hosts with the purpose of planning interaction patterns

The client feeds the parameters it collects to an algorithm that helps it choose the *satisfying set* of services, which in this case has just one element since there is only one host that offers the service that is currently connected. On a high level, the computation of the satisfying set can be described as follows:

- The **motion profile** of each of the service providers is evaluated to check if the service provider is going to be within communication range of the client at the time that the service is required.
- The **location** parameter is used to determine how much a host has deviated from its motion profile by comparing the current reported location with the location computed by the motion profile. This figure is used to adjust the estimated location at the time of requirement of the service.
- The **velocity** of the service provider's host is similarly used to check whether the duration of time that the provider will remain has deviated from what is

promised by the motion profile. This is to ensure that the connection is available for as long as it is required.

Once the algorithm returns the satisfying set, the client connects to the service that is in the satisfying set at the time that it needs it. The service chosen is the one that is likely to be co-located for the longest interval of time (based on the criteria of the above algorithm). It should be noted that in this example, the client had only one service requirement, hence the service set had only one service in it. We expect a client application to have more than one requirement during normal operation, in which case the satisfying set would have one service corresponding to each requirement.

3.2 Problem Definition

The problem we solve is to match the set of needs for a client application with a set of available services such that the services chosen satisfy the needs of the client and are unlikely to disconnect in the interval that they are required. In this subsection, we formally describe representations for service description, service request and knowledge gathering and maintenance. We follow this with a formal problem definition. However, we start by introducing some fundamental definitions and notation.

- Every host in the ad hoc network is characterized by a **motion profile** function $\lambda(\tau)$ which gives the location of the host as a function of time.
- A **service** is intuitively described by the capabilities it offers. We consider the external interface exhibited by a service to be representative of its capabilities. In addition to its capabilities, a service is characterized by a motion profile (aiding knowledge driven interactions), which is the motion profile of the host that it resides on. Thus a service σ having an external interface ψ and resident on some host θ inherits the host's motion profile $\lambda_\theta(\tau)$ and can be written as $\sigma = (\psi, \lambda_\theta(\tau))$.
- A **service request** is traditionally a description of capabilities desired in the service. We specify our needs by providing a desired interface, letting the inheritance mechanism of the programming language decide whether one interface can be satisfied by another. Also, in addition to specifying the interface, we introduce a temporal aspect to service requests as part of our knowledge driven interaction scheme. A service request thus includes an interval of time during which the service is desired. Thus a service request ρ for a service implementing interface ψ from time τ^s to time τ^e can be written as $\rho = (\psi, \tau^s, \tau^e)$
- The **Service Requirement Profile** denoted by **P** is a list of service requests, e.g., P $= (\rho_1, \rho_2, \rho_3, ..., \rho_n)$ Figure 2 shows a service requirement profile.
- A **Satisfying Set** denoted by **S** is a set of services whose external interfaces are a superset of the interfaces specified in service requests that comprise a service requirement profile, e.g., S $= (\sigma_1, \sigma_2, \sigma_3, ..., \sigma_n)$
- The **current location function** Loc(H) gives the current location of host H. τ^c is used to indicate **current time**.
- The $\simeq$ operator is used to denote that an interface is the same as or a subset of another interface. For example, if $\psi_i \simeq \psi_j$, then ψ_i is an interface that is the same as or a subset of interface ψ_j
- The $\sim$ operator is used to indicate that two locations are within communication range. For example $\lambda_i(\tau) \sim \lambda_j(\tau)$ means that $\lambda_i(\tau)$ is in communication range of $\lambda_j(\tau)$
- Finally, the $\cdot$ operator is used to denote a property of a service or service request, e.g., $\rho \cdot \psi$ is used to indicate the interface ψ of the service that is desired as part of the service requirement $\rho = (\psi, \tau^s, \tau^e)$.

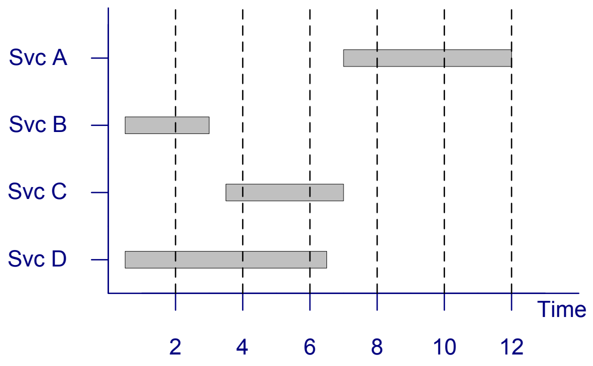

Figure 2: A given client's service requirement profile with the shaded regions showing each service and the time interval for which that service is required

The problem of finding a set of services that match a service requirement profile given a knowledge base can be defined as follows: Given a client application A resident on some host H, characterized by a service requirement profile P, find a service set S such that S *satisfies* P. A service set is said to satisfy P if $\forall \rho_i \in P$, $\exists \sigma_j \in$ S offered by some host θ such that

- $\rho_i \cdot \psi \simeq \sigma_j \cdot \psi_j$ and
- $\forall \tau \in [\rho_i \cdot \tau^s, \rho_i \cdot \tau^e]$, $\lambda_\theta(\tau) \sim \lambda_H(\tau)$

In this paper we will show how gathering knowledge about the physical mobility of hosts over a period of time can facilitate the process of computing a satisfying set that meets the service requirement profile for a given client application.

4. COMPUTING THE SATISFYING SET

Having defined the problem, we now present our algorithm for computing the satisfying set, shown in Figure 3. Rather than focus on a detailed explanation of the algorithm, we show how the algorithm uses the knowledge base to calculate the satisfying set.

The implementation of the algorithm is available locally as a library that can be accessed by all processes on a host. The algorithm is invoked via a programmatic call, and the client

```
SatisfyingSet S = NULL
$\forall \rho_i \in$ ServiceRequirementProfile P
of client application A on Host C {
  Service candidateService = NULL
  Double closestService = $\infty$
  $\forall$ Hosts $H_i$ in communication range {
    $\forall \sigma_i$ offered on $H_i$ {
      if($\rho_i . \psi$ matches $\sigma_i . \psi$) {
        distanceToService = checkForLocation()
        if(distanceToService >= 0) {
          if(distanceToService < closestService) {
            candidateService = $\sigma_i$
          }
        }
      }
    }
  }
  Add candidate to S
}
Return S

checkForLocation() {
  clientLocation = $\lambda_C(\rho_i . \tau^s)$
  serviceLocation = $\lambda_H(\rho_i . \tau^s)$

  clientDelta = loc(C) - $\lambda_C(\tau^c)$
  serviceDelta = loc(H) - $\lambda_H(\tau^c)$

  totalDistance = 0

  for($\tau = \rho_i . \tau^s, \tau < \rho_i . \tau^e, \tau$++) {
    if(EuclidianDistanceBetween
              ($\lambda_C(\rho_i . \tau)$ + clientDelta,
              $\lambda_H(\rho_i . \tau)$ + serviceDelta) > THRESHOLD) {
      totalDistance = -1
      break
    }
    totalDistance = totalDistance +
              EuclidianDistanceBetween
                        ($\lambda_C(\rho_i . \tau)$ + clientDelta,
                        $\lambda_H(\rho_i . \tau)$ + serviceDelta)
  }
  averageDistance = totalDistance / (($\rho_i . \tau^e$) - ($\rho_i . \tau^s$))
  return averageDistance;
}
```

Figure 3: Algorithm for Computing the Satisfying Set

application's service requirement profile is passed in as a parameter. For each service request in the service requirement profile, the class implementing the algorithm searches the list of available services (maintained by the traditional SOC middleware) for services that match the service request under consideration. For each of the services that match the client's request, a request is made for all available knowledge about the host that offers the service. This results in the current location, current velocity, and reported motion profile being returned for the host that offers the service. The current reported location of the host is cross checked with the location for the current time using the motion profile function to check for any deviations. If any deviations exist, they are factored into the algorithm. The values from each host are then used to determine if the host is going to be within communication range at the times that a service is required. This is repeated for each service that matches the client's request. Eventually, the service that is offered by the host that is the smallest distance away from the client host on average during the duration of the requirement is chosen as the service that is to be added to the satisfying set. The process continues until each service request in the client's service requirement profile has a corresponding service in the satisfying set. If a client request was not satisfied, an exception is raised indicating the failure to find a matching service. If no exception is raised, the satisfying set is returned to the calling entity.

Observe that for the algorithm to function as intended, it requires knowledge about the hosts that offer services. This knowledge is stored in a knowledge base that is maintained on each host in the ad hoc network. The next section shows how such a knowledge base is constructed and maintained.

5. BUILDING A KNOWLEDGE BASE

To build a knowledge base, a host must gather knowledge about the motion of other hosts in the ad hoc network. In addition to gathering knowledge about other hosts, it must also distribute knowledge about its own motion to other hosts on demand so that they may construct their own knowledge bases. Both these activities must be carried out in the demanding and unstable environment of an ad hoc network. We present our architecture for knowledge exchange among hosts followed by a brief description of the effort to make it robust for ad hoc networks.

5.1 Defining Knowledge

We define knowledge as any parameter or characteristic about a host that gives an insight into the current or intended behavior of the host. Knowledge can encompass parameters like location, security credentials, power availability, motion profile among others. Knowledge can be divided into two subclasses– *static knowledge* and *dynamic knowledge.* Static knowledge is knowledge that does not change over time. For example, the motion profile of a train travelling between two stations is static knowledge since it travels the same route at the same time everyday, and hence the motion can be represented by a static equation. Dynamic knowledge is knowledge that changes over time, for example, the location of the train as it travels between stations is considered dynamic knowledge.

For the purposes of this paper, we consider only three parameters from each host as knowledge: current location, current velocity, and motion profile. Each host maintains a knowledge base, which is a list of all other hosts that are connected to it, and the three parameters listed above that are associated with each host. The knowledge is stored in the knowledge manager, the details of which are presented later in the section.

5.2 Exchanging Knowledge Among Hosts

To support knowledge-driven interactions between client and service, each host in the ad hoc network must support two functionalities– a) Aggregation of knowledge from other hosts in the network to build a local knowledge base and b) Dissemination of local data to other hosts in the network so that they may build their own knowledge base. These functionalities are encapsulated by the knowledge manager, shown in Figure 4, which sits between a coordination layer that handles communication in the ad hoc network and a traditional SOC layer that offers SOC primitives and operations such as advertisement, discovery, invocation, composition, etc.

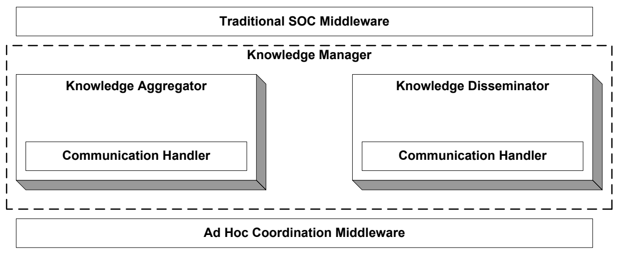

Figure 4: The Knowledge Manager

The knowledge manager is the component that represents the knowledge base on a host. There is one knowledge manager per host. All knowledge about other hosts collected by the knowledge manager is available to processes on the host via a common API. Though our emphasis in this paper is on gathering spatiotemporal knowledge about other hosts, we designed the knowledge manager to be generic enough to support the aggregation and dissemination of other kinds of knowledge. As such, the knowledge manager is highly customizable. Customization features include being able to specify which knowledge parameters are to be gathered or disseminated and the hosts from which they are to be gathered. The customization can be done at startup via command line flags or at any point during the execution of an application via a programmatic call. It should be noted that the customization can be mutated at any point in time via simple programmatic calls to the knowledge manager.

We now show how the knowledge manager performs its two activities at a high level, followed by a more detailed discussion of each of the activities.

- **Knowledge Aggregation** is the gathering of knowledge from other hosts that comprise the ad hoc network. The knowledge manager takes requests to collect knowledge related to certain parameters on certain hosts in the ad hoc network. It channels each request to a subcomponent called the *aggregator* which actually services the request.
- **Knowledge Dissemination** is the distribution of knowledge related to the local host to other hosts in the ad hoc network The knowledge manager takes requests from both local applications as well as other hosts in the network to disseminate knowledge about the host it is resident on. It delegates this request to another subcomponent, the *disseminator*, which services such requests.

5.2.1 The Knowledge Aggregator

The knowledge aggregator performs the aggregation functions for the knowledge manager. It aggregates the knowledge disseminated by knowledge disseminators on other hosts and adds them to the local knowledge base. It can also update knowledge that exists in the local knowledge base with more recently gathered knowledge. In the case that a host is not disseminating some knowledge that is required by the local host, the knowledge aggregator can send a request to the relevant host to begin disseminating that knowledge. The main components of the aggregator are – a) The communication handler, b) An array of host level aggregators, and c) The aggregation manager. This is shown in Figure 5. The communication handler reads data that is sent to it from disseminators on other hosts via the ad hoc network. Depending on the host that the data came from, the communication handler sends the data to the host level aggregator assigned to that host (If none exists, then one is created.)

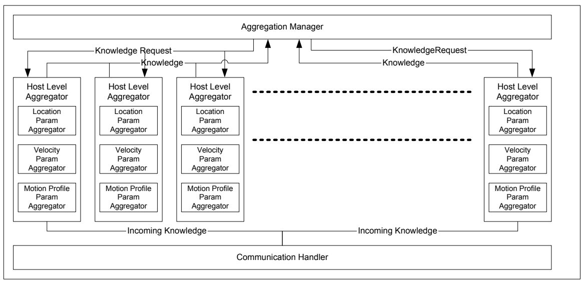

Figure 5: The Knowledge Aggregator

Within the main knowledge aggregator, there is one host level aggregator for every host in the ad hoc network that the local host is connected to. Each host level aggregator contains three parameter aggregators–for current host location, current host velocity, and intended motion profile. Additional parameter aggregators may be added to support the aggregation of parameters other than the three specified, if required but we focus on just three for the purposes of this paper. The location and the velocity parameter aggregator are updated periodically to track the latest position and velocity of the host associated with their parent host level aggregator. The motion profile parameter aggregator is usually static (i.e., not updated regularly) unless the host it is tracking changes its motion profile. The host level aggregators combined with the parameter aggregators that they encapsulate form the knowledge base on a given host. The values inside the parameter aggregators can be accessed by external entities by making programmatic requests to the knowledge manager, which delegates the request to the aggregation manager. The aggregation manager oversees the correct functioning of the knowledge aggregator by ensuring the correct parameter aggregator is updated when an updated value is reported to the host. It also services local requests made to the knowledge base. Once it is determined (by inspecting the motion profile), that a host is going to disconnect permanently, the knowledge aggregator removes the host level aggregator corresponding to the host. This ensures that unnecessary knowledge is not maintained on any host.

5.2.2 The Knowledge Disseminator

The dissemination of knowledge related to a host is performed by the knowledge disseminator, which is split into three main components – a) The dissemination manager, b) An array of parameter disseminators, and c) The communication handler. This is illustrated in Figure 6. The dissemination manager acts as a controller for the disseminator, dispatching data to other hosts as required. To perform the data dissemination, the knowledge disseminator employs three parameter disseminators for location, velocity and motion profile (though like parameter aggregators, there can be more than just three).

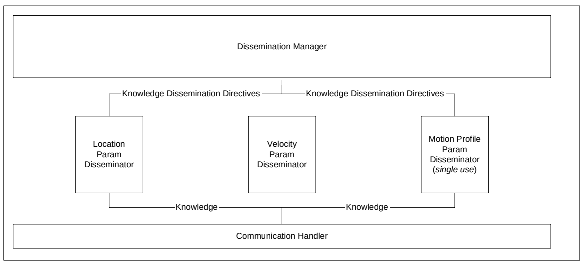
Dissemination Manager
Knowledge Dissemination Directives
Knowledge Dissemination Directives
Location Param Disseminator
Velocity Param Disseminator
Motion Profile Param Disseminator (single use)
Knowledge
Knowledge
Communication Handler

Figure 6: The Knowledge Disseminator

A parameter disseminator is a process that gathers a specific type of knowledge, e.g., GPS location from a *knowledge generator*, which may be a hardware entity, e.g., a GPS device, a software process, e.g., a random number generator, or simply some knowledge stored locally. The parameter disseminator stores the knowledge obtained as a member variable and is responsible for ensuring the value it stores is kept up to date. The dissemination manager disseminates knowledge periodically (by default at a rate of once per second) by sending the values in each of the parameter disseminators to every host in the ad hoc network. The only exception to this rule is if a parameter disseminator is tagged as *single use*, in which case it sends the data in that parameter disseminator only once per host. For example, the motion profile parameter disseminator may be a single use one if the motion profile is never expected to change. The actual transmission of the values is the responsibility of the communication handler, which packages the values in a manner appropriate for transmission, and hands it off to the coordination model which is responsible for getting it to its destination knowledge aggregator.

5.2.3 Robustness for Ad Hoc Networks

Both the knowledge aggregator and the knowledge disseminator are heavily dependent on communication across the ad hoc network to distribute or gather knowledge. As such, they must be designed to work effectively in ad hoc networks given the constraints described in earlier sections of this paper. The repeated, periodic dissemination of knowledge ensures that even though the topology of the network changes at a rapid rate, all hosts have up to date knowledge available. Correspondingly, the aggregation manager ensures that values in the parameter aggregators are replaced by more current knowledge as it is received. The aggregator supports fine granularity during the data gathering process through the use of separate host-level aggregators to ensure that no unnecessary data is gathered, thereby saving precious computational resources. Finally, to protect against disconnection, the dissemination and aggregation managers use coordination models targeted to ad hoc settings, allowing them to communicate as effectively as possible in the ad hoc network, allowing efficient dissemination and aggregation of knowledge.

6. IMPLEMENTATION

Our knowledge manager is used in conjunction with a traditional SOC architecture for ad hoc networks that we developed as part of our previous work. The SOC architecture is implemented on top of LIME [10], a coordination model for physical and logical mobility. We present a brief overview of LIME and follow that with implementation details of the knowledge manager.

6.1 LIME & SOC Implementation Overview

LIME is a Java implementation of the Linda [3]coordination model and is tailored for use in ad hoc networks. LIME masks details associated with coordination and communication from the application programmer. A host offering LIME runs a `LimeServer` which supports one or more LIME agents, analogous to services or client applications.

Coordination in LIME occurs via transiently shared tuple spaces. Every tuple space in LIME is identified by a name. Tuple spaces having the same name can be merged to form a federated tuple space when their hosts are within communication range. The service directory is modelled as a tuple space. Tuple spaces are containers for tuples. Tuples are ordered sequences of Java objects which have a type and a value. An agent places a tuple in the tuple space, making it available to all other agents that are sharing the same tuple space. To read a tuple from the tuple space, an agent needs to provide a template, which is a pattern describing the tuple that the agent is interested in. A template is a sequence of fields, each of which can contain a formal (wildcard) representing the required type for that field or an actual value that identifies the type and value of the corresponding field. A template is said to match a tuple if all the corresponding fields match pairwise. Service advertisements are in the form of tuples that contain a description of the service's capabilities while service requests are in the form of templates.

An agent can access the tuple space via standard Linda operations (`rd` (read a tuple), `in` (remove a tuple), `out` (write a tuple)). The `in` and `rd` operations take a template as a parameter and return a tuple as the result or block until a match is found (the operations are synchronous). To provide asynchronous interactions, LIME offers a reaction mechanism. An agent can declare interest in a tuple by registering a reaction on a tuple space using an appropriate template and by providing a callback function to be called when a matching tuple becomes available. If multiple candidate tuples exist for a given reaction template, one is chosen non-deterministically from the set.

We wrapped the LIME tuple space class with a `ServiceDirectory` class which represents the service directory. It provides standard SOC operations such as advertise, request, and invoke. A service is advertised by placing a tuple in the tuple space using the `out` operation. A request is implemented as a `rd` operation, with the interface of the desired service being passed as the template. Invocation is done by doing a targeted `out` operation where the tuple is stamped to indicate its destination host.

6.2 Implementation of the Knowledge Manager

We implemented the knowledge manager in Java, and integrated it into our existing Java-based middleware for SOC in ad hoc networks. On startup, the knowledge manager starts a Lime agent which we call the *knowledge agent.* This agent is the interface between the knowledge manager and Lime, which it uses to disseminate and aggregate knowledge. When the knowledge manager wants to disseminate some knowledge, it instructs the knowledge agent to place a tuple into the federated tuple space. For example, to disseminate location knowledge it places a tuple of the form <"Knowledge", "Location", [19.32N, 23.45W], Host1> in the tuple space. The first field indicates that the tuple contains knowledge, the second the type of knowledge, the third the actual knowledge, which in this case is a latitude-longitude pair, and the fourth indicates the host that placed the tuple in the tuple space.

To aggregate knowledge, the knowledge manager registers reactions on the tuple space (via the knowledge agent since no class other than a Lime agent may perform tuple space operations). The reactions are registered for each type of knowledge that the knowledge manager wants to aggregate. For example, to aggregate location information, the knowledge manager registers a reaction with a template of the form <"Knowledge", "Location", Location.class, Host.class>. The first two fields indicate that we are interested in knowledge tuples and more specifically location knowledge tuples. The third field is a formal for a class that encapsulates location information, while the fourth is a class that encapsulates the Host ID. Formals are used since we want any location value that corresponds to any host.

Within the knowledge manager, whenever a reaction fires, indicating that new knowledge of the type we are interested in has been placed in the tuple space, a *reactive program* runs to retrieve a copy of the tuple from the tuple space of the form <"Knowledge", "Location", [19.32N, 23.45W], Host1>. Once it is retrieved the third field in the tuple is used to update the value in the parameter aggregator that stores the type of knowledge indicated in the second field within the appropriate host aggregator, the host name being given by the fourth field in the tuple. Note that a copy is retrieved so that other hosts in the network may also avail of the same knowledge. It is the responsibility of the host that disseminates the knowledge to remove any old tuples using the `in` operation before outing a tuple with updated knowledge.

When a client wants to discover a service, it makes a request to a new `KnowledgeManagedServiceDirectory` (KMSD) class, which wraps the traditional `ServiceDirectory` class. This class, instead of channelling the request directly to the Lime tuple space as in the traditional architecture, channels the request to the knowledge manager. The request is of the form <DesiredInterface, StartTime, EndTime>. The knowledge manager then searches the `Lime` tuple space for all services that match the required interface using the `rdg` operation of Lime, which is similar to the `rd` operation except that it can return more than one match. The template used is of the form <DesiredInterface, Host.class>. Once a list of services has been returned, the knowledge manager executes the algorithm described in Section 4 to determine which service is the best, and returns a handle for that service to the client.

6.3 The Influence of Knowledge on Traditional SOC Operations

The use of knowledge to pro-actively compute the satisfying set of services influences the way standard SOC operations are implemented. In this subsection, we show which operations are affected and the manner in which they are affected.

6.3.1 Service Discovery

As described above, we implemented service directories as tuple spaces [3], a data structure that comprises of local tuple spaces on hosts that logically combine to form a federated tuple space when hosts are co-located. A host that wishes to advertise a service places an advertisement in its local tuple space. Since the local tuple space is part of a federated tuple space, this advertisement is viewable by all hosts that are within communication range of the advertising host. This also ensures consistency of the directory, i.e., there are no orphan advertisements. Thus at any given point in time the list of advertisements mirrors the list of available services exactly. Interested clients browse the directory for the kinds of services that they require, and when the first match is found, begin using that service immediately without consideration to other options.

In the knowledge-driven architecture, we retain the use of tuple spaces as service directories. Service providers place their service advertisements in these directories as in traditional SOC architectures. However, clients are not permitted to access these directories directly. Instead, they provide their requirements to the KMSD. The KMSD takes the request for some service, and searches the service directory for candidate services. Once a list of such candidate services is returned, it uses the algorithm mentioned earlier in the section to choose the one instance of the service that is most suitable for the requirement. It then returns this service to the client that requested it. It should be noted that requirements may consist of more than one service, at different points in time. In this case, the knowledge manager returns a set of services rather than a single service.

6.3.2 Service Invocation

The traditional manner in which clients have interacted with services is via a client-side service proxy. The service proxy communicates with the service on the provider host via any available communication mechanism. We use federated tuple spaces as a means of communication between proxies and service providers. Proxies can place tuples containing data in tuple spaces which are then retrieved by the provider. The provider uses a similar procedure to communicate with the proxy. The traditional invocation procedure does not crash the system in face of unanticipated disconnections but may cause deadlock.

In our architecture, when the client interacts with the service (via a service proxy), the duration of time for which the client host is likely to remain connected with the provider's host is known a priori because the knowledge manager can provide such information in the form of *coordination knowledge*. This helps solve two critical problems. The first is the orphan call problem, where the client sends a message to the service, which disconnects before it can reply. The client is then potentially blocked waiting for the reply. In our architecture, this is averted because if the time to disconnection is less than the time that it would take to service the call and get a reply, the call is not allowed to go through and an exception is propagated which allows the client to handle it and continue processing. It should be noted that we assume a static value for the time taken to process a call and send a reply, and that if the actual time taken is greater than the static value, the call is broken into two or more smaller calls. The second problem occurs when the proxy places a tuple destined for the provider in the tuple space, but the provider is disconnected. In our architecture, knowledge about the behavior of hosts can be used to determine if the disconnection is permanent or not. If the disconnection is not permanent, then the request can be held in until the provider reconnects, else an exception is raised on the client to indicate that the provider is no longer connected.

6.3.3 Service Composition

Service composition refers to building a single composite service from several smaller independent services. Ignoring the interoperability issues involved to carry out such a task, service composition in ad hoc networks remains a challenge. This is because in a dynamic ad hoc network, a composed service has points of failure equal in number to the number of smaller services that it is made up of. In other words, if a composed service is made up of N different smaller services, and if any one of those services get disconnected, then the composed service as a whole fails. Hence it is very difficult to maintain a composed service for extended periods of time.

In our architecture, some of the problems associated with maintaining a composed service can be alleviated. The knowledge manager can provide coordination knowledge about the behavior of each host that contributes a service towards the composed service. If it is determined that a host is going to get disconnected at some point in the future, a discovery protocol can be launched to find a substitute for that service well before the disconnection actually occurs. The service that is about to get disconnected can then be replaced with the newly discovered service, allowing for the continuous operation of the composed service. (The continuous operation can be achieved due to previous work on the topic, described in [14].)

The knowledge managed architecture supporting the three operations described can be seen in Figure 7.

7. DISCUSSION

Our vision of SOC in ad hoc networks is similar to, yet different from, that of SOC for wired networks. It is similar in the sense that we too have unhindered interoperability of heterogenous entities as a core goal and we employ architectures that are closely related to the SOC paradigm. The difference lies in the kinds of applications and users we target. We see SOC as a means of letting people collaborate with each other in the course of their daily lives. In other words we focus on *personalized services* involving interactions among small mobile devices rather than large, enterprise-scale B2B type interactions that are being envisaged for wired networks. Our current work makes the assumption that users of our SOC software for ad hoc networks will have patterns of behavior that can be represented by relatively simple mathematical equations. Our future work involves studying whether this is indeed a reasonable assumption; if not, then we plan to devote effort to making our knowledge managed architecture capable of handling random and complex motion.

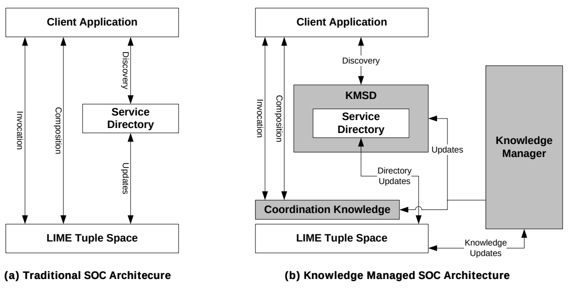

Figure 7: Contrasting traditional and knowledge managed SOC architectures

Turning to the work presented in this paper, we mentioned at various points in time that clients may specify requirements, which can be combined with the knowledge base to compute a satisfying set a priori. This seems to eliminate the on-the-fly immediate nature of service discovery and usage. However, this is not true since clients can specify a service requirement with the time for the requirement being the present time. In this case, any service that is currently available is returned as the satisfying set. A related issue is a mismatch in the time that the client specified for the requirement of the service and the actual time of requirement. In the case that the actual time was earlier, the system performs a search for immediately available results as described above. In the case that it is later, the system monitors the service that was to be used, and performs evaluations to check if it is still the best instance, replacing it if it is not.

Another issue is that of proactive searching. If a client needs some service at a point in the future, how can we discover its existence even though it is not in communication range? A solution to this problem can be found by using disconnected message delivery [4] which uses disconnected routes to inform a client about the existence of a service that it is not within communication range of. Alternatively, ad hoc routing can be used. Both these approaches are outside the scope of this paper, but are part of our future work. For now, we assume that services go in and out of communication range. Thus, when a service is first in communication range, it leaves its motion profile and advertisement which can be used to deduce whether it will be in range at the required time.

The final issue we discuss is determining the length of an interaction between client and service. In this paper, we have claimed that knowledge can be used to determine if the connectivity is going to last longer than the duration of the interaction. For this version of our software, we assumed a static bound on how long an interaction would take. In the case that an interaction were to take longer than the static length of time, we assumed that it would be split into two. We intend to investigate whether one can calculate the duration of an interaction dynamically. For example, one might consider the number of bytes to be sent across, the bandwidth and protocol level details to obtain an average case estimation. If we are successful, we intend to modify this version of the software to be compatible with dynamic calculation of the length of interaction. We expect to add some software that does the calculation.

8. CONCLUSIONS

Using knowledge about other hosts in ad hoc networks to plan interactions between clients and service providers can yield benefits in terms of predictability and stability of applications that expand their capability via the opportunistic use of external services. In this paper, we have described a software architecture for building a knowledge base which aggregates knowledge as well as an algorithm that uses the knowledge base to facilitate proactive planning of interactions. We have also described how traditional SOC operations and their semantics change in an knowledge driven SOC architecture. We have shown that such an architecture is more reliable that traditional SOC architectures in ad hoc networks. This is a first step towards introducing a new concept, which we believe can play a role in achieving wired network-like quality of service in ad hoc networks. Much work and analysis remains to be done towards optimizing such a system.

Acknowledgements: This research was supported by the Office of Naval Research under MURI research contract N00014-02-1-0715. Any opinions, findings, and conclusions or recommendations expressed in this paper are those of the authors and do not necessarily represent the views of the research sponsors.

9. REFERENCES

[1] A. Ankolekar, M. Burstein, J. Hobbs, O. Lassila, D. Martin, D. McDermott, S. McIlraith, S. Narayanan, M. Paolucci, T. Payne, and K. Sycara. Daml-s: Web service description for the semantic web. In *Proceedings of the 1st International Semantic WebConference*, 2002.

[2] F. M. Costa and G. S. Blair. The role of meta-information management in reflective middleware. In *Proceedings of ECOOP'2000 Workshop on Reflection and Meta-level Architectures*, 2000.

[3] D. Gelernter. Generative communication in Linda. *ACM Transactions on Programming Languages and Systems*, 7(1):80–112, January 1985.

[4] R. Handorean, C. Gill, and G.-C. Roman. Accommodating transient connectivity in ad hoc and mobile settings. In *Proceedings of the Second International Conference on Pervasive Computing (Pervasive 04)*. Springer-Verlag.

[5] R. Handorean and G.-C. Roman. Secure service provision in ad hoc networks. In *Proceedings of The First International Conference on Service Oriented Computing (ICSOC 03)*, number 2910 in Lecture Notes in Computer Science, pages 367–383, 2003.

[6] R. Handorean, R. Sen, G. Hackmann, and G.-C. Roman. Automated code management for service oriented computing in ad hoc networks. Technical Report WU-CSE-2004-17, Washington University Department of Computer Science, 2004.

[7] T. Harvey and K. Decker. Planning ahead to provide scheduler choice. In *Proceedings of Autonomous Agents Infrastructure Workshop*, 2001.

[8] G. Karumanchi, S. Muralidharan, and R. Prakash. Information dissemination in partitionable mobile ad hoc networks. In *Proceedings of the 18th IEEE Symposium on Reliable Distributed Systems*, pages 4–14. IEEE Computer Society, 1999.

[9] J. Kempf and P. S. Pierre. *Service Location Protocol For Enterpriser Networks: Implementing and Deploying a Dynamic Service Resource Finder*. John Wiley and Sons, 1999.

[10] A. Murphy, G. Picco, and G.-C. Roman. Lime: A middleware for physical and logical mobility. In *Proceedings of the 21^{st} International Conference on Distributed Computing Systems*, pages 524–533, 2001.

[11] M. Paolucci, T. Kawmura, T. Payne, and K. Sycara. Semantic matching of web services capabilities. In *Proceedings of the 1st International Semantic Web Conference*, 2002.

[12] B. Perry, M. Taylor, and A. Unruh. Information aggregation and agent interaction patterns in infosleuth^{TM}. In *Proceedings of the Fourth IECIS International Conference on Cooperative Information Systems*, 1999.

[13] Salutation Consortium. Salutation web page. http://www.salutation.org.

[14] R. Sen, R. Handorean, G. Hackmann, and G.-C. Roman. An Architecture Supporting Run-Time Upgrade of Proxy-Based Services in Ad Hoc Networks. In *To appear in the Proceedings of the International Conference on Pervasive Computing and Communications PCC-04*, 2004.

[15] R. Sen, R. Handorean, G.-C. Roman, and C. Gill. *Service Oriented Software Engineering: Challenges and Practices*. Idea Group Publishing, To appear in 2004.

[16] R. Simmons. Towards reliable autonomous agents. In *Lessons Learned from Implemented Software Architectures for Physical Agents*, pages 196–203, 1995.

[17] J. Waldo. The Jini Architecture for Network-Centric Computing. *Communications of the ACM*, 42(7):76–82, 1999.