SNAP: A LOCAL, ASYNCHRONOUS INTERACTION FRAMEWORK FOR PERVASIVE APPLICATIONS

BY VENKATESH PUTTY

A thesis submitted to the

Graduate School—New Brunswick

Rutgers, The State University of New Jersey

in partial fulfillment of the requirements

for the degree of

Master of Science

Graduate Program in Electrical and Computer Engineering

Written under the direction of

Professor Manish Parashar

and approved by

_____

_____

_____

New Brunswick, New Jersey

October, 2004

ABSTRACT OF THE THESIS

Snap: A Local, Asynchronous Interaction Framework For Pervasive Applications

by Venkatesh Putty

Thesis Director: Professor Manish Parashar

Evolution in our computing needs has seen evolution in all spheres of computing with advances in device technology and the software infrastructure driving these systems. The road ahead is through frameworks that aid us in comprehending, developing applications for and managing this wide-ranging evolution resulting in pervasive computing. But, pervasive applications differ significantly enough from traditional distributed computing, to warrant thought on suitable distributed computing interaction primitives.

This thesis presents an interaction framework for such applications that do not need transactional properties in interactions and can function with minimal guarantees. The framework offers an interaction protocol that is used for effecting asynchronous local actions and uses a decentralized consistency maintenance mechanism to reconcile system inconsistencies.This thesis presents scenarios where applications can use the interaction protocol and thus adapt better to pervasive environments.

Acknowledgements

This has been a journey, and for that I am grateful to many a people. To amma and appa for their persistence all these years. To Anand and Shoba for being them here. Their support, advice and shoulders have gotten me to this point.

I would like to thank Dr.Manish Parashar for his guidance and patience. I am grateful to Dr.Dipankar Raychaudhuri for giving me the opportunity and for his encouragement in times of need. Behind every result is an implementation; the suggestions Dr. Max Ott had offered early on in my research certainly eased its implementation and I am grateful to him for that! I would like to thank Dr. Yanyong Zhang for being on my thesis committee.

One of the great things about TASSL is the people. I'm thankful to Viraj, Sumir, Vincent, Xiaolin and and the rest of the gang for being there when it mattered; and to Xiaolin and Nanyang for tolerating my all-nighters, sleeping bag, musings and random music! Many thanks to the CAIP staff sitting a few rooms away, especially Bill, James and Steve, for their assistance with all things frea.

And finally, to God, in all his forms, for shining the torch in the tunnel- I would not be here without him.

Dedication

To,

My Parents

and

Anand & Shoba

Table of Contents

List of Tables

List of Figures

Chapter 1
Introduction

Our universe with all its quirks, is about as good as any system gets. Zillions of infinitesimal subsystems come together organically to create our perfect universe. Pervasive computing [34] seeks to provide a similar computing universe. The pieces of such a universe can be seen around us. Evolution in our computing needs has brought about simultaneous advances in all spheres of computing with advances in device technology and techniques in managing devices, methodologies of architecting systems of devices, and software infrastructure driving these systems. The road ahead is through frameworks that aid us in comprehending, exploiting ,managing and developing uses for this wide-ranging evolution resulting in pervasive environments.

In such emerging systems heterogeneity, availability, scalability, identity, roles and relationships, information quality, context sensitiveness and the coexistence of these issues makes such systems far more complex than traditional computing systems. On the flip side, emerging system applications have lower systems support expectations. Large and dense emerging systems offer redundancy in information, sub-systems and such. Exploiting these properties will allow a framework to be a feasible solution for developing emergent system applications.

Our research is a step in this direction. This dissertation discusses an interaction framework suited for Emerging Systems. The interaction framework provides a foundation to work with emerging systems and offers an interaction protocol to bring these systems to life. In fact, the real contribution is the interaction protocol at the core of the framework. The protocol offers a combination of asynchronous interactions,

consistency maintenance and a simple comprehension of participants.

1.1 Motivation

In non-trivial asynchronous interactions, the interaction participants have to be updated with the outcome of the interaction while ensuring that all participants have mutually consistent interaction states at the end of the interaction. This is normally achieved by using some form of synchronization between the participants. In traditional distributed computing, whenever the interaction outcome is available, all the participants and affected elements, are updated in the same step using appropriate synchronization mechanisms. Extending such an interaction pattern to emerging systems involves siginficant tradeoffs. Reaching out to elements system-wide synchronously in the same step would negatively affect the system behavior.

Emerging system applications typically have loose coordination between functional components, locality in functionality and can work with flexible contracts with the underlying infrastructure. In typical emerging applications, small clusters of elements are the ones that need to be actively consistent as far as an interaction that affects that part of the application. The other components need information about the interactions, but typically satisfied by delayed or summarized (percolated) updates. They don't need the correct or consistent result right away or at the same time for everyone.

For example, a video sensor monitoring a wildlife reserve, upon observing that that a new animal has entered its range, can come out of standby and start monitoring actively once its locality of sensors in the valley agrees on the intrusion. Here the monitoring application does not make as strict demands of the interactions as would any other traditional distributed application. If there were any issues in the information the application recieved then it can go back to standby and discard the data collected. But if the same had to be done after global consent in a system of a

million sensors, the additional data would never have been collected.

A feasible interaction framework for emergent system applications will need to exploit their characteristics and differences from existing systems. These leniencies in expectations are not exploited in traditional interaction models. Each of these is a significant advantage, if the model can be designed to use it.

1.2 Problem Statement

The thesis proposes an interaction protocol to enable asynchronous interactions between elements of an emerging system with "local" synchronization primitives. The protocol's synchronization primitives offer interaction consistency with characteristics that is sufficient for the application at hand. Put together they will allow for spontaneous interactions.

Overall this thesis will present the foundation for asynchronous interactions with consistency maintenance mechanisms. Such a framework would allow an interaction whenever one is possible, hence spontaneous, and will help achieve the objective of the application.

1.3 Contributions

The thesis offers the following ideas and protocols to support interactions in emerging systems:

- **Interaction framework for emerging systems:** The framework offers only asynchronous interactions. Resulting anomalies are handled by a consistency maintenance mechanism. This allows an application to setup and potentially complete them instantaneously allowing applications to take advantage of transient favorable conditions. Further, interactions are always with immediate peer elements (neighborhood) and never explicitly target a global audience. This

complements the locality that is found in emerging system applications. Still, the system is affected by these local interactions.

- **Consistency maintenance mechanism:** The consistency maintenance mechanism detects and repairs any inconsistencies caused due to asynchronous interactions. The mechanism leverages the locality of system components and a gossip style maintenance protocol to ensure that all elements of the system converge onto a single consistent state based on predetermined rules. Additionally, the consistency mechanism offers intermediate quasi-consistent phases before the final consistent state. These quasi-consistent phases guarantee the consistency of interaction state in a defined group of participants. Applications can leverage these phases to gain flexibility in functionality definition and spontaneity in functionality realization.

- **Opportunistic satisfaction of objectives:** Interactions need not be the rigid, well-defined task completion tools designed to complete a task with all interacting components working in lock-step in a predefined workflow. Tolerant applications can accept the compromise made when one is able to communicate whenever possible and with the best possible characteristics. Such an interaction means that an application's functional objectives need not be satisfied in one big shuffle; objectives can be met incrementally, possibly in random order. While the thesis does discuss this idea at length through later sections, the full potentially has not been realized in the presented work.

1.4 Scope

The thesis address only a few of the issues in developing applications in emerging environments, but a number of other critical issues are not tackled but acknowledged as existing and have to be address for a complete solution. The following issues have not been addressed in the thesis:

- **Transactional interactions**: The interactions being targeted cannot be used in applications that require ACID properties from the interactions. As would be discussed later on, the interactions presented are intended to be one of a suite of protocols at an applications disposal. Hence these interactions should be used only if transactional properties are not expected.

- **Security, Trust, Scalability** and **Availability** are important issues but have not been addressed in this interaction framework. At the node level, malicious nodes can cause havoc to a straight forward peer-to-peer algorithm. Lifetime of nodes in an emerging system can vary depending on node failures, the function and role of the node thus affecting availability.

- **Platform issues** are not realized. Emerging environments typically use a host of middleware technologies to create a complete system. But, this interaction framework has not been developed as part of a platform for emerging systems. All through the approach has been as a generic interaction protocol.

- **Node identity** is not dealt with. Consequently anonymous identities and such are not handled explicitly. Also, a simple mechanism to define and maintain roles is needed.

- **Membership maintenance** is not provided. Group definitions can change over time even if nodes doesn't explicitly vary membership - a node with varying location, roles or functionality.

- **Context sensitiveness** should be an important ability in a well-rounded interaction solution, but is not present in the interaction framework.

- **Constraint based application composition** is a much needed facility that can help define emerging systems more easily. Such a facility has not been fully explored in the interaction framework implementation.

1.5 Organization

The thesis is organized as follows:

Chapter 1 presents background on the thesis and defines its direction Chapter 2 discuses relevant research work and provides pointers to other research that target the issue at hand from different perspectives. Chapter 3 discuses the Interaction Model and the Opportunistic Interaction Protocol. The section also describes a test bed built to implement the protocol and an application described in the same section. The next section, Chapter 4 provides an experimentation analysis to show the characteristics of the consistency establishment mechanism in the Opportunistic Interaction Protocol. Finally, Chapter 5 summarizes the solutions and outstanding issues with the framework and discuses where it is headed.

Chapter 2
Prior Work

"If I have seen further,it's by standing upon the shoulders of giants."-Isaac Newton

When developing applications for pervasive computing traditional distributed computing technologies are inadequate as the only solution. Still, technologies such as web services, p2p technologies [26], publish and subscribe models [6, 16], are suitable for a targeted role along with middleware technologies designed for pervasive systems [15]. P2p systems [29] are useful for their decentralized architecture.

In pervasive environments, applications, middleware and systems need to address radically different issues. Research in pervasive computing, autonomic computing and sensor networking offer solutions to target these issues from different perspectives. Sensor networking [24, 10] facilitates the realization of the physical systems composing pervasive environments by offering protocols and technologies to work with sensors, thus forming a real world system. Autonomic computing [2] enables systems to be open, self-defining, self-healing, self-configuring, self-protecting, self-optimizing, and contextually aware. Pervasive computing research [15, 27] works the middle ground by offering layered solutions to host the target environment. Project Oxygen[27] offers solutions to an entire spectrum of issues in pervasive computing, with solutions in device technologies, management, networking, user interface and middleware.

2.1 Building blocks

Much like in a Lego world, the interaction model builds on a number of pieces of existing and new technologies.

Any distributed system needs coordination and synchronization constructs. Consensus protocols offer some constructs that are useful in this context. Pure distributed consensus [30] has been proven to be impossible [12]. However, alternative approaches such as randomized consensus protocols [3] and charaterizations [23] have avoided this theorem's results and offer consensus in distributed systems[17]. The interaction model presented in the thesis offers a consistency maintenance mechanism that uses some ideas from consensus systems. The thesis proposes an interaction protocol that offers a consistency mechanism that uses concepts central to distributed consensus. However the mechanism does not handle the case of node failures.

A later section will show that the biggest advantage of emerging systems is their density, locality and their interconnectedness. Gossip [9] can leverage [4] these advantages to offer a form of global comprehension. An information element is passed along through the system using a probabilistic forwarding function thus spreading to the extremities of the system. This simple idea has diverse applications. Gossip has seen application in ad-hoc routing [20], large-scale system design [33, 18], database maintenance [32, 31], multicast [19, 5, 11], group membership maintenance [8, 14]. Significant research has been done in these and others to characterize gossip and offer solutions to issues in using gossip in distributed middleware [28]. The proposed interaction model uses some of these ideas to define a simple gossip based protocol to manage inconsistency in an asynchronous interaction. The interactions themselves do not use these ideas, but the consistency mechanism uses forwarding to spread a dominant view into a system. Such a usage is also seen in Astrolabe [32] and Bimodal Multicast [5].

Research in amorphous computing [1, 7, 25], cellular automata [21] and robot coordination [13] shows how local interactions have the power to effect global actions and hence system functionality. Using the interaction protocol presented in the thesis, an application can carry out actions in a small group of components with the knowledge that the system will get updated at some point in the system evolution.

Chapter 3
Design of the Interaction Framework

In this interaction framework each element's behavior is governed by its goals that guide the functioning of the element. Elements of the system mostly interact with their contextual neighbors. The system functionality emerges as one element interacts with its neighbors and which in turn with theirs, thus enveloping the entire system.

Elements interact with one other using an interaction protocol, proposed in this thesis. The interaction protocol assumes that a lightweight messaging layer exists below it and offers some basic services such as accepting messages from other elements, sending messages and forwarding messages as a conduit. At this level, the interaction protocol makes no attempts to ensure that interactions succeed. This simplicity is worked upon to provide all that is possible in adverse environments. The desirability of interacting with a participant is evaluated by predefined constraints on each element. Interactions in this protocol are asynchronous and offer soft guarantees with best effort semantics. In a non-trivial application asynchronous interactions without explicit synchronization are bound to create inconsistencies. The interaction protocol offers mechanism to restore consistency in the system. All these characteristics allow the framework to provide the applications that use it, the best possible mix of services in the best form that is possible from whatever is provided by the system. Such a framework can allow applications to take advantage of the environment in every possible way. Essentially this allows applications to be opportunistic as far as satisfying their functionality is concerned. Hence, system behavior evolves from "constrained" and "local" interactions between "available" and "desirable" element.

In this chapter, first the interaction framework is described, which talks about the big picture for emerging systems. The second is a part of this framework, the interaction protocol which is discussed later and implemented and analyzed. But first lets take a look at the target environment.

3.1 The Target Environment: Emerging Systems

This thesis focuses on Emerging Environments like Pervasive systems, Sensor Environments and Autonomic systems. The ecologies of devices and applications populating such environments have characteristics that are significantly different from traditional computing systems. Here on "elements" is used as a catch-all word for components of an application, devices in a system, applications in a system

Heterogeneity is the dominant theme in Emerging Systems with elements in domains spread across functional, geographical, technological, availability and mobility spheres. Information quality, identity, dynamic roles and relationships are a characteristic of interactions. Their complex interdependent nature requires a multi-pronged approach to providing a solution. For example, availability of specific elements with a given functionality might be uncertain, but since a complete failure is not likely, some random functionally relevant element would always be available, though not necessarily a specific one. If this "anonymity" can be tolerated then the availability is significantly higher though with significant changes in the definition of relationships. The size and density of emerging systems means that popular information could be replicated with a very high degree of probability, without an intentional replication. If the possible dirtiness of such replicated information can be tolerated then the biggest issue in large systems has just been made a strength.

Elements in an emerging system environment typically interact with other elements in their neighborhood. Because of this intrinsic locality in the system layout it is not trivial to do system wide actions. But, a real world application will have

some actions that, without relevant alternatives, require cooperation or information from most, if not all, elements in the system. This locality of existence means that interactions that affect entire system need not affect all at the same time, as long as they will affect eventually. This means, for example, in our target environment, applications don't need synchronization with all elements of the system right away and at the same time for everyone. As long as the groups relevant to the functionality quickly synchronize the rest of the system can stabilize slowly, potentially skipping a few intermediate stages. In fact, as will be seen later, a guarantee that the effect of synchronization will be uniform across all the system elements at some point of time is typically enough. These leniencies in expectations are not exploited in traditional interaction models. Locality means most interact with locality, so if locality is consistent in sync and interacts with environment in an uniform manner then all is fine. This coupled with relaxation of the immediacy of uniformity allows a lot of deviation from traditional synchronization based uniform ness.

Existing distributed computing models offer very good solutions to existing class of problems, because they were designed with such problems in mind. Similarly, exploiting the properties of Emerging systems will allow a framework to be a feasible solution for developing emergent system applications.

3.2 Functionality

Exponential complexity in pervasive systems, varying needs of the applications residing in these systems and the demanding real world environments require alternative approaches to composing pervasive applications. In the past complexity was accommodated by a number of approaches that were grouped into stacks (vertical) and components (horizontal) of varying coarseness. Aggregation along multiple dimensions offers increasingly sophisticated and application oriented set of services, though with an associated acceptable tradeoff.

But the characteristics of emerging environments and their evolving requirements imply defining the system goals a-priori is difficult and sometimes impossible. Our framework facilitates this by offering semantics that allow individual elements of a system to be autonomic and be associated to the larger system only by its goals. in our framework the system goals are defined in terms of individual elements and their functionality. The eventual convergence of individual element goals gives form to the system's goal and hence its functionality. In addition to reducing the complexity of the system, it allows a system to be built as a sum of parts, an idea that our framework aims to facilitate to the extreme.

Elements of the system define their functionality in terms of constraints that describe the elements goals and objectives. The constraints can evolve to shadow a dynamic environment. For an element to be able to function, all the constraints need not be satisfied. Depending on the definition of functionality, if satisfaction of a subset of constraints can allow acceptable behavior then the element's existing behavior would change to include this new behavior.

In defining an element using this sort of random and incremental yet, controlled, satisfaction, the element can take advantage of the available environment in the most suitable manner that will allow it to achieve some (or all) of its objectives. In other words, the constraints governing an element are more of a life-time "guide" to acceptable behavior; bounds on an element's operating range and hence bounds on the system's goals. Hence, system functionality originates from individual elements. When this locality (of elements) is used to define system objectives it follows that any explicit behavior- namely interactions- also originates in a similar manner.

In our framework a given element interacts only with elements in its immediate neighborhood and hence exposes its services and accepts services only from them. The neighborhood and its membership is application defined. Potentially, an application can use the available contextual information in conjunction with constraints to define membership in an element's neighborhood.

Further, an element can use constraints to define its interactions, and hence its behavior, with its neighbors. Since these constraints are loosely defined, evolving environments and neighbors can be accommodated. Participating elements' actions are a factor of environmental information, interaction state, local state, local constraints and decisions and indirectly all these of its neighbors and so on. Thus the entire system behavior evolves.

So, in our framework any given element's existence and behavior is defined by its constraints and its interactions with its immediate neighbors. This flexibility and loose definition of the neighborhood in turn allows fuzzy definitions of element relationships. Such fuzzy relationships are atypical of real world systems and allow systems to adapt to the demands of real world systems intrinsically.

3.3 Local interactions: System-wide effect of Local behavior

The framework exhibits a very strong preference for behavior in the neighborhood or what we call, local behavior. Almost all actions, and hence behavior, of the components of the system or elements are local in nature. The propagation of an action's effect(s) is decided by its importance as determined by the action's importance to the neighbors and possibly other indirect neighbors. A neighbor will show interest in propagating an action (or its effects) only if it is interested in it, or any related neighbor expressed interest in this action at any point of time.

Research in amorphous computing [1, 7, 25], cellular automata [21] and robot coordination[13] shows how global behavior can be influenced with reasonable confidence by local interactions and behaviors. Local side effects can translate into global behavior with reasonable tolerance for loss in information percolation and noise. The system functionality emerges as one element interacts with its neighbors and which in turn with theirs, thus enveloping the entire system. Hence, system behavior evolves from localized interactions between desirable and local elements.

This loose grained definition of system composition (neighborhood and relationships) and behavior (interests, information/event flows, isolation, comprehension, coordination) can potentially translate into a highly resilient and scalable framework apt for the emerging systems.

3.4 Soft Guarantees

Traditional interaction semantics offer refined interactions with well defined characteristics and hard guarantees on the service offered. To provide these they have significant support needs from the system and offer only rigid interaction flows. However, in emerging systems, typical applications offer functionality that can be delivered just as well with less refined interaction semantics with minimal guarantees but one that allows flexible interaction flows. Typical applications need something that is equally unstructured and are not particular about interaction contracts and such.

Soft guarantees essentially is the additional effort that the framework makes to provide a consistent picture of the system in the event of any instability in the system. This may not necessarily be the correct view, though it would be in an ideal environment. But this is on a best effort basis and is not a binding guarantee. Thus an application can work in this environment if a consistent view is all that is needed - which may be relatively incorrect hence the softness in the guarantee.

3.5 Opportunistic Interaction Protocol

Increasing complexity in interaction modes coupled with varying needs of the applications and environments can be accommodated by providing the interaction middleware as a stack of services. Each level offers increasingly sophisticated and application oriented set of services. The interaction protocol builds itself from the ground up as a set of services feeding off the simpler (and lower) service offering to the user or the service above a richer service set. The interaction protocol is layered as shown

in Figure 3.1 with each protocol layer responsible for a minimal set of operations and behaviors. The lowermost layer offers the communication primitives that the

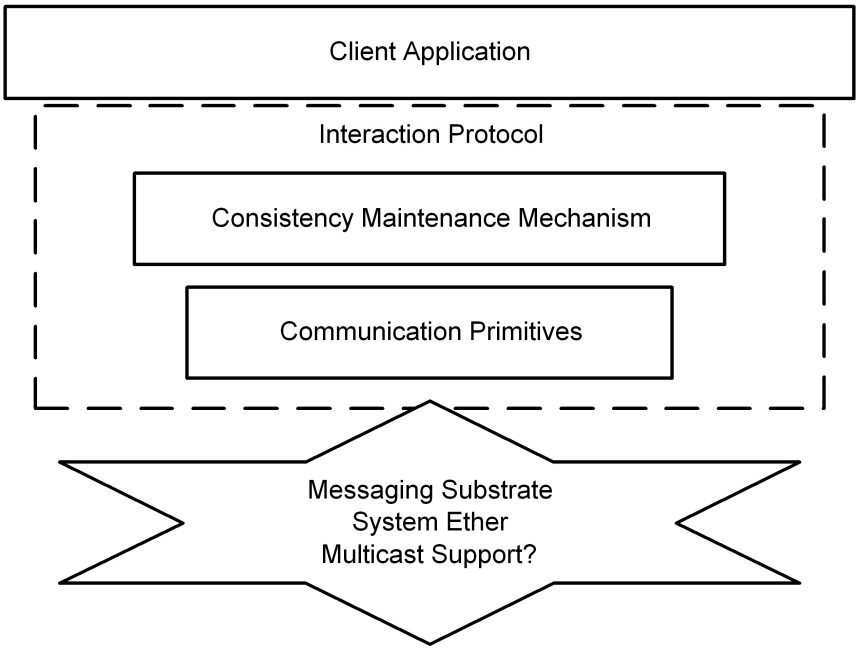

Figure 3.1: Interaction Protocol Stack

higher layers use to initiate and maintain interactions. Higher layers enforce various interaction behaviors and offer mechanisms to deliver the guarantees offered to the application.

3.5.1 Communication primitives

The protocol operates on top of a basic communication substrate offering a few basic services noted in Table 3.1.

These basic services are the ability to receive messages from, publish to and forward received messages to a specific group. Incoming messages are handed off using processing hooks from higher layers. The previous table shows the expected minimal service set of this substrate. The above service definitions indicate that the interactions use the notion of a group. For this the communication medium should offer a

Primitive	**Functionality**
Receive	Receives a message from it local group
Process	Handoff received message to the application
Listen	State to wait for a receipt of message.
Publish, Forward (derived)	Send a message to a selected group
DefineGroup	Create a group based on defined criteria

Table 3.1: Communication Primitives

basic multicast service where the group definition should be simple and allow contextual definition of groups. It should be possible to send messages only to immediate neighbors in the system.

3.5.2 Interaction Protocol Flow

The overall idea of an opportunistic interaction can be summarized as:

> "*Act, involve in group consistency, do what it takes to become consistent with the group, proceed with group view as interaction conclusion.*"

The initiator starts an interaction by publishing the initial view about an action the initiator completed. If this initial view of the action is different from the views held by the local group (neighborhood) then the group becomes inconsistent with one node of the group having a different view from the other members.

A phase of consistency establishment ensures that members of the local group and the system share the same view and thus the interaction completes with a stable conclusion. However, this phase mentioned might be composed of multiple intermediate consistency phases (discussed in the next section).

Once the system has consistency, and hence all members have agreed on result of the action, the initiator (and the system) can assume the action has had its effect and is completed. The Tables 3.2,3.3,3.4 show the interaction algorithms.

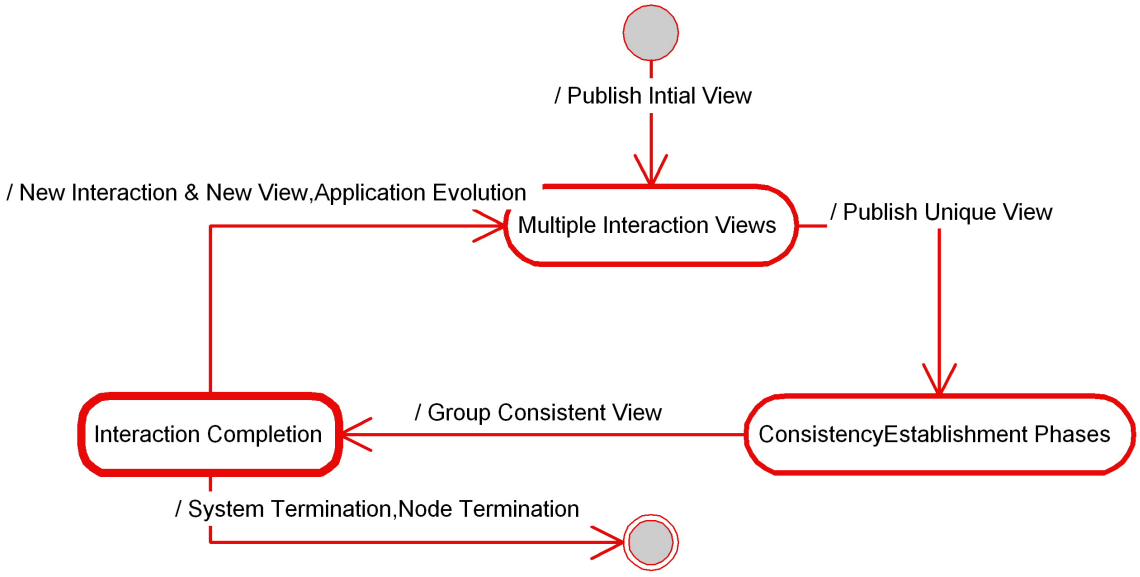

Figure 3.2: Protocol State Diagram

Of course other actions need not be executed in sequence. Applications are unhindered in any attempt to interleave a series of independent actions followed by consistency phases completing in no specific order. Though demonstrating this is a completely separate axis of analysis.

Presently **interaction terminates** when the group achieves local consistency. However, thrashing can occur. A newly consistent group could get a new view that is in fact just a continuation of the consistency establishment procedure, only this view bounced back from some extremity of the system. While it is easy to say that such a thrashing could occur primarily on account of a weak rule set, the protocol doesn't offer any mechanisms to guard against this. This is certainly one issue that needs future work.

Components	**Role**
Group	group of communicating nodes
Node	given node in system
View	application advertisement
Constraint	application ad processing logic

Table 3.2: Interaction Components

Variables	Description
Group::GLocal	Neighborhood of the node
View::Vself	Startup view of a node
View::Vcurr	Current view held by a node
View::Vrcvd	Received view held by a node
Constraint::Cview	Constraints operating on a view.

Table 3.3: Interaction Algorithm Variables

```
Interact
    Init(Node::Self, Glocal, Vself)
Publish(Glocal, Vself)
  While System is not Consistent
    Listen(Glocal, Vcurr)
    Process(Cview, Vrcvd)
    Publish(Glocal, Vcurr)
  Interaction Completion(Node::Self, Glocal, Vself)
Process(Cview, Vrcvd)
  EnsureConsistent(Cview, Vrcvd, Vcurr, Vhistorical)
    If GroupConsistent(Cview, Vcurr, Glocal)
            TransitionToGroupView(Vcurr, Vrcvd)
```

Table 3.4: Interaction Protocol Algorithm

3.5.3 Consistency Maintenance Mechanism

Defining Consistency

In any application it is possible to have distinct "views" of a given fact. For example, as show in Table 3.5, the shape of a ball is debatable based on how one wants to view it. It can be considered to be a circle, when drawn on paper, or a sphere when seen in the real world and as a cube, which would be plain incorrect. But, given the same motive in a system, only one of these views of the fact (shape of the ball) is valid and useful. This is the essential property of any system.

Consistency can be considered as the property of a system wherein all member elements hold the same view on a given fact. Conversely, a system with multiple views about a given fact can be considered to be inconsistent and unstable.

Fact to converge on	Shape of a ball in the system.
Consistent view	The ball is a cube.
Consistent and correct view	The ball is a sphere.

Table 3.5: A Consistency Perspective: A system may be consistent and still not hold the correct view.

The underlying principle

In an asynchronous interaction, some of the participants are bound to have multiple distinct views of the same factoid. A process is needed to reconcile these multiple views of the same fact and arrive at a unique and accepted view that can be used by all system elements. In traditional systems this would have been done by some mechanism that would have reached out to participants of the interaction and those affected by it. But in our target systems this could mean reaching out to the entire system, which as stressed earlier is counter-productive. As discussed earlier, given the nature of emerging systems, this consistency need not be achieved with the same urgency for all system elements. Consistency can be provided by leveraging the density, interconnectedness and locality of the system.

Before diving further into consistency, let us dwell on an important property of emerging systems. An application in such a system derives its definition from a functional aggregation of element groups and the individual elements. Conversely, a view of a fact agreed to by a given neighborhood would be the view the system will mostly converge on. So, neighborhood consistency on a given view of a fact can translate to system consistency. The interesting part is how this local consistency translates into system consistency.

To achieve consistency in a fractured system with many views, the framework actively ensures that this neighborhood has a consistent view of the system. By extension, the entire system will then have a uniform view if each neighborhood is consistent in its perception. Hence any such fact which has neighborhood consensus will effectively be the current valid truth which if used by any member of the system will not cause instability in the system.

The goal driving the consistency process in the framework is that other elements in the contextual relative neighborhood should have similar views of the world. The element(s) discovering an inconsistency initiates a mechanism to ensure that the neighborhood arrives at a common view of the system. Each neighbor offers a combined view based on its consensus and those of its neighbors (and the remaining system it is in contact with) about the view of the system that appears to be the most consistent. The combined consensus will decide the consensus that the neighbors will agree to. This process will cascade across the system ensuring that all those with divergent views agree on one uniform view. An application can work in this environment if a consistent view is all that is needed.

Consistency Mechanism Details

The consistency establishment mechanism ensures that the system has consistent views at distinct points in its evolution.

When a node in a group receives a message with a new version of a view it holds, then the node triggers the consistency mechanism. The process of achieving system-wide consistency starts in the immediate neighborhood of the initiator (for now, let's work with a single initiator). The driving goal is to ensure that in a neighborhood group, all members have a consistent view. Any inconsistency in a group is resolved by view exchange and evaluation based on defined rules on the acceptability of views. At the end of this local phase, the group becomes consistent. As the state chart in Figure 3.3 shows, after local consistency is established the protocol initiates consistency

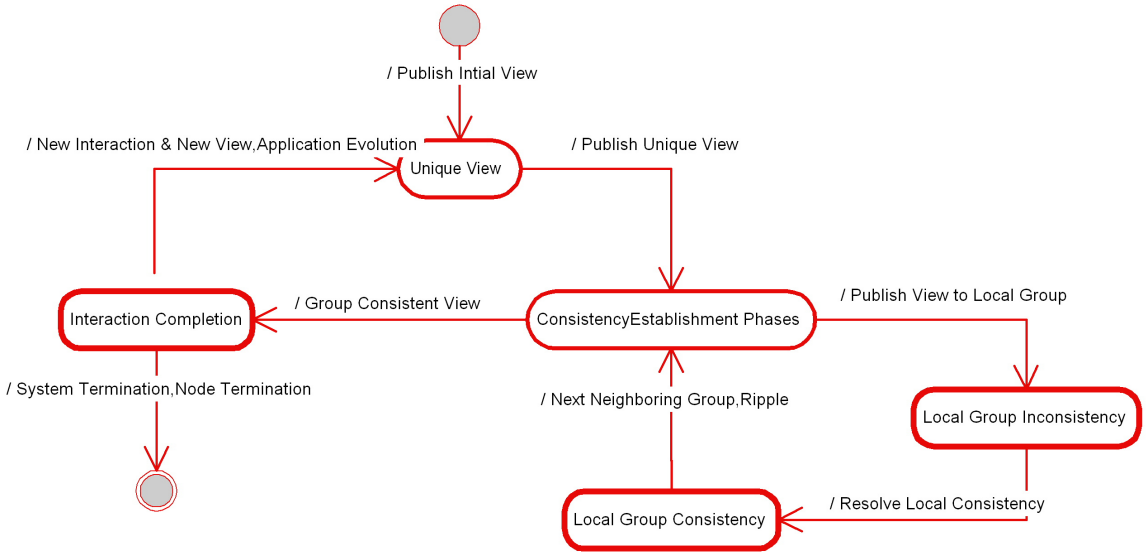

Figure 3.3: State Diagram With Consistency Establishment Mechanism

in the neighboring groups. Members of a consistent group propagate a consistent view to their neighbors. These neighbors in turn initiate consistency establishment in their local group. Eventually this ripple effect spreads through the entire system making the system view consistent. Thus, the consistency establishment phase might be composed of multiple intermediate consistency phases with each improving the overall system consistency. This node to node/group consistency establishment means that consistency gets established a group at a time and by association translates to a system level consistency.

What is interesting in all this is the quasi-consistent states that the mechanism goes through before converging on the consistent states. A tolerant application can use this quasi-consistent state to get its functionality going.

For example, a video sensor monitoring a wildlife reserve spread across a few hills and plains, upon observing that that a new animal has entered its valley (and possibly, the reserve), can come out of standby and start monitoring actively (collect additional data) once its locality of sensors in the valley agrees on the intrusion. Here the monitoring application is able to tolerate and exploit the quasi-consistent intermediate

state. If the consistency mechanism comes back with a system disagreement that the animal had only strayed into range from another part of the sanctuary, then it can go back to standby (and discard the data). But if the same had to be done after global consent in a system of a million sensors, the additional data would never have been collected. Smarts in the operating rules could potentially be wary the next time and demand a better consistency from the mechanism. Say a quasi-consistent state that spans a few groups. However, such a request system is not provided in the protocol.

Known Issues in the Consistency Mechanism

The protocol allowed asynchronous interactions to be consistent with some guarantees. The upfront guarantees ensured that this consistency would be achieved in waves of neighborhoods. While this suggests that the entire system will become consistent at some point, there are no deterministic bounds associated with this system wide consistency. What is needed is a probabilistic quantification or an empirical bounds on of the consistency achieved at any given stage. Additionally, the present mechanism doesn't yet handle the entire range of dynamics of view updates in real systems. Scenarios exist where a view change can get subdued by a more "powerful" view. Thrashing can lead to a system that swings between views. Also, the consistency mechanism needs to accommodate issues such as multiple conflicting initiators, updates frequency exceeding the consistency attainment delay. Though node availability has be out of scope, it needs to be worked into this mechanism in a real world system. While conjecture suggests that availability may not be a showstopper considering the ad-hoc nature of the mechanism, experiments need to be conducted to characterize availability's effect.

3.6 Test bed

The focus of the test bed is to provide a platform to implement the protocol and to help analyze consistency establishment in the interaction protocol.

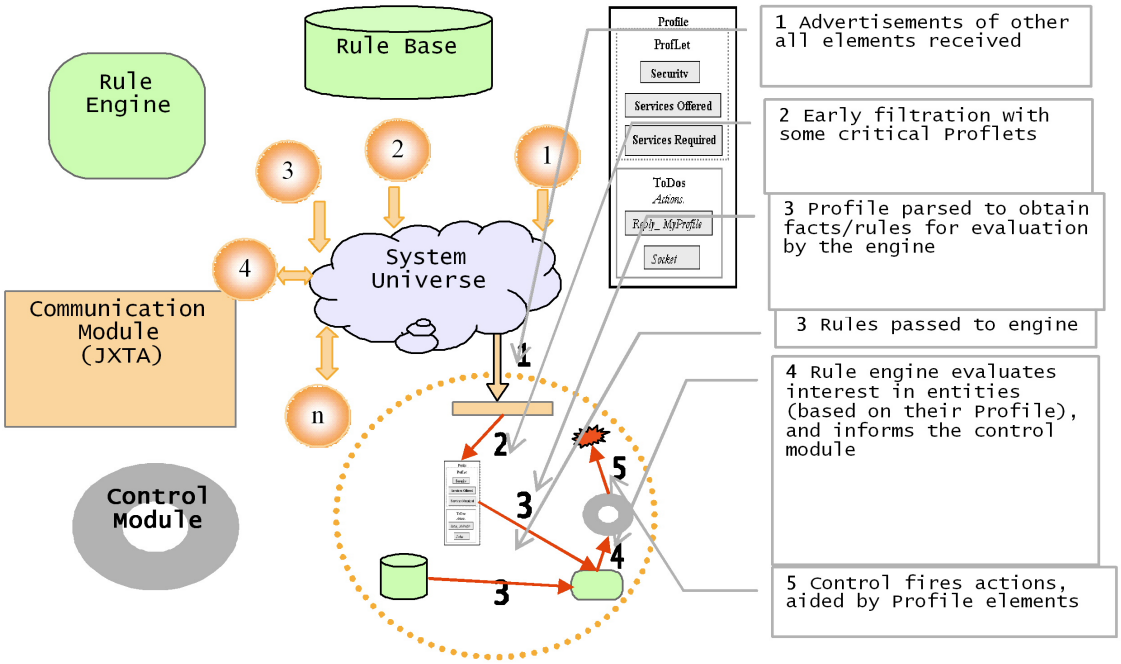

Figure 3.4: Event flow in an interaction participant using the TestBed. Each node in the cluster hosts a single JXTA peer and uses JXTA Discovery protocol for its communication. Each node has a Rule Engine ,JESS , executing rules in a Rule Base that evaluate a recieved view.

3.6.1 Testbed infrastructure

The experiment runs on the Frea-Tolkein research cluster with each node in the experiment mapping to a box in the cluster.

The nodes use the Jxta p2p toolkit for a simple communication medium. The Jxta discovery service is used for communication between the nodes in the system. The simulation components forming the simulation run as Jxta peers carrying out a certain task. Each component running on a node has a rule engine, JESS[22] (an expert system shell), which compares views recieved based on rules in a Rule Base.

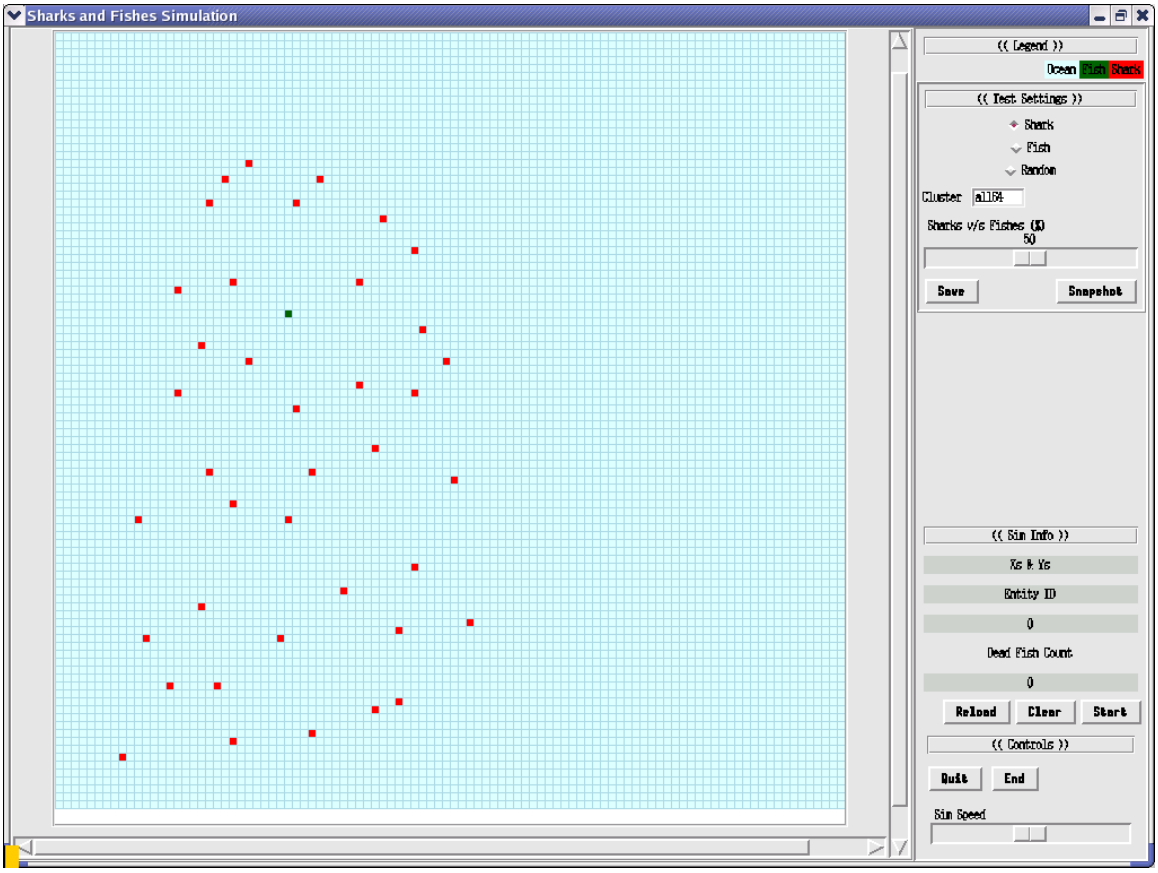

Figure 3.5: SimViewer: The testbed offers a simulation viewer that can display the topology configuration for a given simulation. It is shown here displaying a system with 40 nodes (red and green dots). The green dot represents a fish and the red ones the sharks. SimViewer was written in Python.

The topology seen in the Figure 3.5 is for one of the simulation runs discussed in a later section. Most of the applications using the testbed are based on cluster nodes mapped to a logical 2d space, an ocean 100x100 units large, with a layout as shown below. For example, in the Game of Life application discussed in the next section the green spot (the fishes) represents the initiator of the task ("alcarin") with other nodes represented as red spots (the sharks) with a light blue Pacific. The task and its relevance to the experiment are explained in the next section.

3.6.2 Assumptions

For this testbed, message delivery in the communication substrate is assumed to be reliable and ordered. Consistency establishment is by nature a group based activity. An interesting part of any group based activity is behavior under unpredictable group membership. This is best seen in a system where nodes do fail at various points in the activity lifecycle. This behavior is not our current focus, and hence in our experiment all the nodes are reliable and don't fail.

The comparability of available views based on defined rules is central to the establishment of consistency. The experiment assumes that view change eligibility can be expressed as a rule that can evaluate available views to yield decision. Moreover, it is assumed that a trivial rule can adequately approximate a larger system where nodes utilize rules to analyze the world they observe. This is actually the subject of an entire body of research comprising of constraint systems. Also a single view change by a single initiator is assumed.

There are a few limitations imposed on the experiments on account of implementation choices. Group messaging is achieved by using a thin filtering layer over a broadcast to simulate a notion of multicast. Also, The nodes in the experiment have unique names are based on the network hostnames.

3.7 Applications

The framework is suited for applications with loose coordination between functional components, locality in functionality and can work with flexible contracts with the underlying infrastructure. This section a few motivating applications are described that show how the various components of the framework can be exploited in different real world examples.

3.7.1 Concept Illustration Using Automata

Consider a system containing a and b in the contextual neighborhood with c being a distant element. The distant element is "indirectly" a neighbor of a and b by the neighborhood principle. Any "information event" generated by c can get communicated to a and b through the intermediate elements in varying coarseness and availability (may get sent by itself or as a part of a bigger "information event"). This in a free flowing system translates into every element having a fair view of any event in the system by way of interactions and the information it receives over time. If it doesn't, there is a very high possibility of its neighbors having such information.Hence if at any point in an element's existence it realizes that it has a divergent view of the system, then it can ask its neighbors for consensus on what is their perception of the view it has and what is the consistent view of the system.

3.7.2 Protocol implementation: A game of life

This section describes how the protocol can be used in an application scenario.

Assumptions

The protocol implementation builds on the previous experiment with consistency and adds to it the notion of an application. Specifically, messages (earlier used solely to build consistency) are suitably loaded with additional information which coupled with application logic at each node delivers the application functionality being described in this section. So, assumptions made in the previous experiment are also needed here.

Details of the Game

The interaction protocol provides mechanisms (for asynchronous interactions) that offer system consistency at distinct intervals. This section describes an implementation of this protocol to implement a simple sharks and fishes game of life. The objectives of the game are trivial but the game offers interesting opportunities to observe the protocol "in action".

The game itself revolves around a school of sharks hunting a single fish and attempting to eat it uniquely - only one shark gets to munch its lunch- without any explicit coordination while attempting to eat the fish. To elaborate on the meaning of "eat" in this example, eating can be considered as a series of steps which starts with an attempt to eat. The eating completes only when the system is consistent. The ultimate arbiter of succesful eating is, the object of attention, the fish.

The ocean with the sharks and fishes are depicted in Figure 3.5. It shows a game layout with 39 sharks and 1 fish. Further, each entity in this ocean can only listen to views from so far away. Given that the ocean is 100 by 100 units wide, each entity can listen to views published by another entity only 15 units away. This rule informally defines the neighborhood group for a given node. A final twist in the game is that the fish doesnt move throughout the game, while a shark can move and pounce on the fish the moment it recieves the advertisment. Although incredulous, this rule serves an important purpose: it doesnt bring in mobility into the game and extracts only the group activity from the classic game of life.

Using the protocol the sharks publish their attempt at eating the fish. Consistency establishment mechanisms ensure that there is only one successful shark. Simple rules evaluate which shark actually ate the fish. This decision is used by various groups of sharks to arrive at a single conclusion and a single successful shark. The decision propagates through the various groups via the protocol gossip. Additionally each group uses group consistency to arrive at local consistency regarding the successful shark.

Results

The two graphs in this section represent datasets for various runs of the application with the interaction protocol.

The first graph shows how inconsistency arises in the system as a results of the asynchronous eating (gobble curves). Each point on the curve represents a node gobbling alcarin on receipt of the alcarin ad. One of the gobbles is by the fish acknowledging the successful shark. Note that the gobble curves track pretty closely the consistency curves from the previous section (note: the 30 and 40 consistency curves are based on the gobbles of the implementation). On closer analysis it makes sense. For the consistency establishment experiment, a node receiving an "alcarin" ad transitions to a consistent state. Whereas in the Game of Life implementation such a node would "eat alcarin" i.e. a gobble.

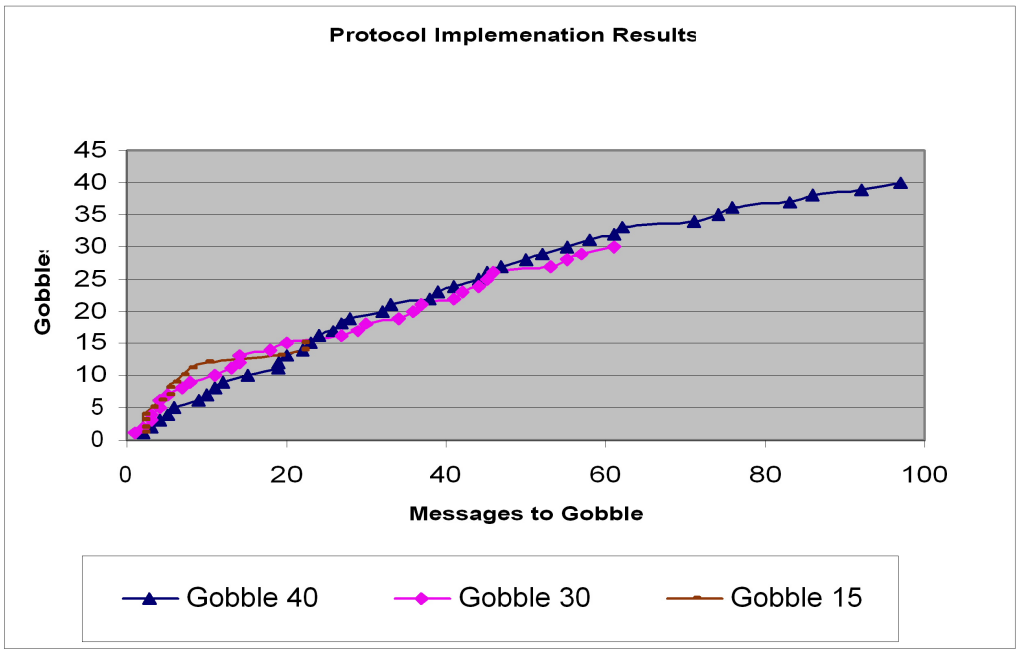

Figure 3.6: Inconsistency caused by Asynchronous interactions. Plots the rise in inconsistency for systems with 15, 30 and 40 nodes. Each includes only one fish with the remaining as sharks. Each gobble is treated as an inconsistency since only one shark can legally gobble the fish.

The next chart shows the consistency repairs by each node that wasn't the successful shark. This curve shows the consistency establishment phase for resolving the inconsistency caused by the asynchronous gobbles by the sharks. The consistency is achieved by each group agreeing on the most consistent and correct view, the successful shark, and passing this on to the neighboring group. The charts shows only those cases where this local consistency results in the "correct" successful shark being selected for local consistency (since that would be the desired consistency). This consistency also uses the fish's (alcarin's) acknowledgement. Considering these two points and the gobble curve it is interesting to note that the consistency phase took more messages than the gobble phase. This could be in part due to the simplicity of message propagation and the gossip approach itself.

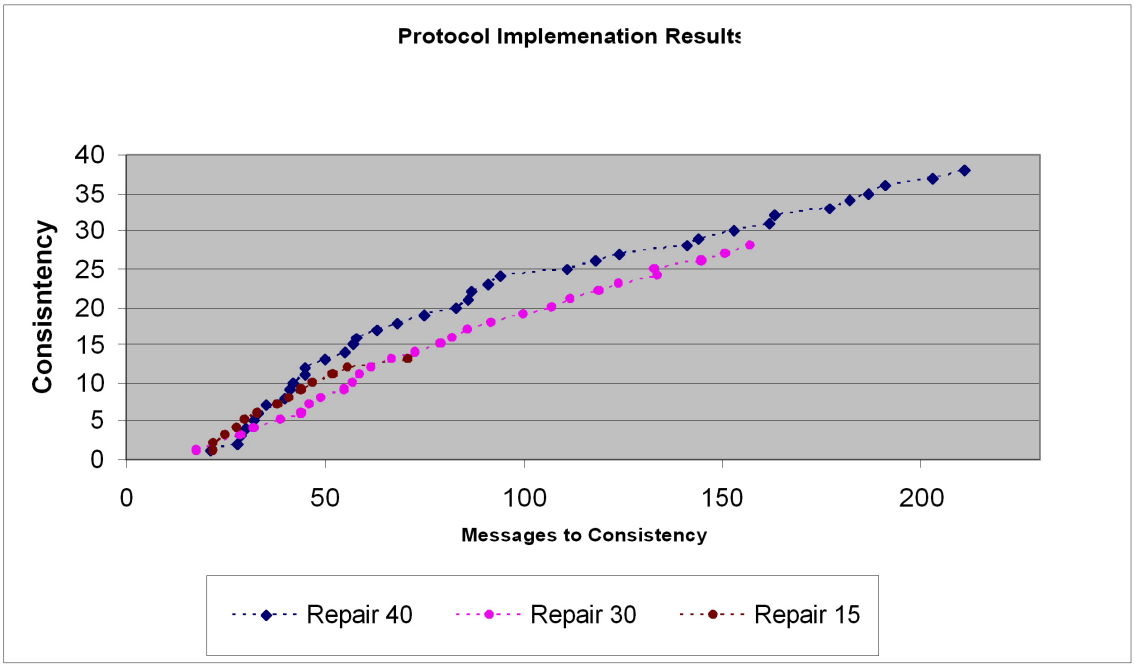

Figure 3.7: Consistency Establishment:Plots the repair of inconsistency for systems with 15, 30 and 40 nodes. Each includes only one fish with the remaining as sharks. Each message implies that a shark realised that it wasn't the legal gobbler of the fish, and hence converged toward the consistent view.

Oddly enough, early in the consistency establishment, the run with 40 nodes does establish consistency a few messages earlier than the run with 30 nodes, though it is

not significantly faster. This is most probably due to the implementation's approach to group messaging using broadcast and vagaries induced by it, but does encourage an interesting possibility. Existing research on gossip systems show that large systems can leverage gossip based systems better- so will such an interaction protocol converge faster in systems of sizes greater than 100 or even 1000 nodes?

3.7.3 YACE: Yet Another Campus Example

Consider a campus visitor tracking application that tries to track campus visitors opting to wear RFID badges and provides them maximum assistance with few student volunteers. Such and application could tell Open house volunteers approximate campus locations of visitors beamed in from sensors spread through the campus in addition to "beacon balloons" floating around the campus, campus busses tracking onboard visitors, cafeterias and so on.

3.7.4 Opportunistic interactions in Coffee Production

Coffee is indispensable as a drink, and is an important traded commodity. Information about coffee is useful at all levels from the plantations all the way to the commodity trading desks.

Maturity of beans sensed by sensors, changes in weather, insect populations, new results on how to better regulate drainage of plantations, cost of labor during harvest, shipping and packaging news, changing commodity prices of coffee amongst others can affect decisions regarding irrigation, harvest timing, shipping, supply, commodity trading prices and even regulatory measures to protect commodity prices from catastrophic events. It is important to realize that decision systems driving each part of the chain can benefit from, but will survive without, information from other pieces. But since the number of sources and their quality and interpretation can differ a rigid system to tie in all these interactions to set up decisions cannot be orchestrated at

system design.Harvesting has to be done when the beans are almost ripe. But inclement weather or approaching infestations might prompt an earlier harvest. If the decisions are driven primarily by meteorological data from satellites then information quality might be accurate but might be offset by a couple of hours. If any information from weather sensors from nearby plantations, HELIOS (Nasa) crafts or plain sight indicates otherwise emergency harvest can be kicked in to save what can be saved.

This example depends on a free form information sharing subspace that can be composed out of any available information spaces. Here the interaction framework exists as a information sharing and correlation tool.

Chapter 4
Evaluation

Consistency in our model evolves through a number of intermediate phases before finally achieving complete consistency in the system. This section explores consistency establishment in a simple experiment running on a cluster-based custom test bed.

4.1 Experimental setup

The task is to make the system agree on a desired view evaluated using a simple rule. The system is composed of interlocking groups of nodes. The view spreads by gossiping, and hence consistency follows this by changing consistencies in groups of nodes at a time. A stable system will have a single "eventual system view" - the view that all nodes in the system will hold after a change in the view of the system.

A view in this application is based on the node names which in turn are derived from the network hostnames of the cluster nodes. The desirability rule is trivially expressed as lexicographical precedence of current node's name versus the received view's propagator's name. So, At the start of the experiment the nodes are seeded with their node names as the views. Nodes transition to the most useful view in their neighborhood and eventually converge on the most useful view in the system. Consistency maintenance depends on these transitions.

Note that the experiment's view change rule has been designed such that a node could potentially change a other intermediate views before finalizing on the system view; a view change need not be a one shot job.

To make sense of all this, consider the popular sensor application where sensors

are tracking, albeit discreetly, animals in a sanctuary, in the interests of conservation-Each node aims to track the largest animal in sanctuary, and hence the most important view. So, a t-rex entering a forest teeming with springboks would be of interest to the sensors, and possibly, the puzzled sanctuary intern manning the sensor network.

The experiment starts with each node believing in similar views (that each is the leader of the system). One node believes differently, say "alcarin" for a given group, and claims leadership (view) by alphabetical superiority(rules). For the system to be consistent, all nodes have to agree on a single view. So, any node receiving another's claim has to accept the superior one of the two as the leader. In a consistent system all nodes will end up picking up "alcarin" as the leader since that's the lexicographically superior view in the system.

4.2 Progressive consistency

This experiment shows the behavior of consistency establishment after a single view change initiated by a single node. Such a view change interaction is one of the basic interactions that can be used to compose complex interactions that form the functionality of a system.

The Figure 4.1 shows messages taken to converge on a particular "eventual system view" by individual nodes after one of the nodes, the initiator, publishes a view change. Hence, a bump up in the chart, say at x messages received, implies that a node in the system converged after receiving x messages. The nodes converged at the initiator's view represent the consistency of the system; in other words the chart presents consistency curves for various system sizes.

These consistency curves show the part of the system that has already converged (cumulative). The graph plots each converging node by itself and each curve saturates at the system population.

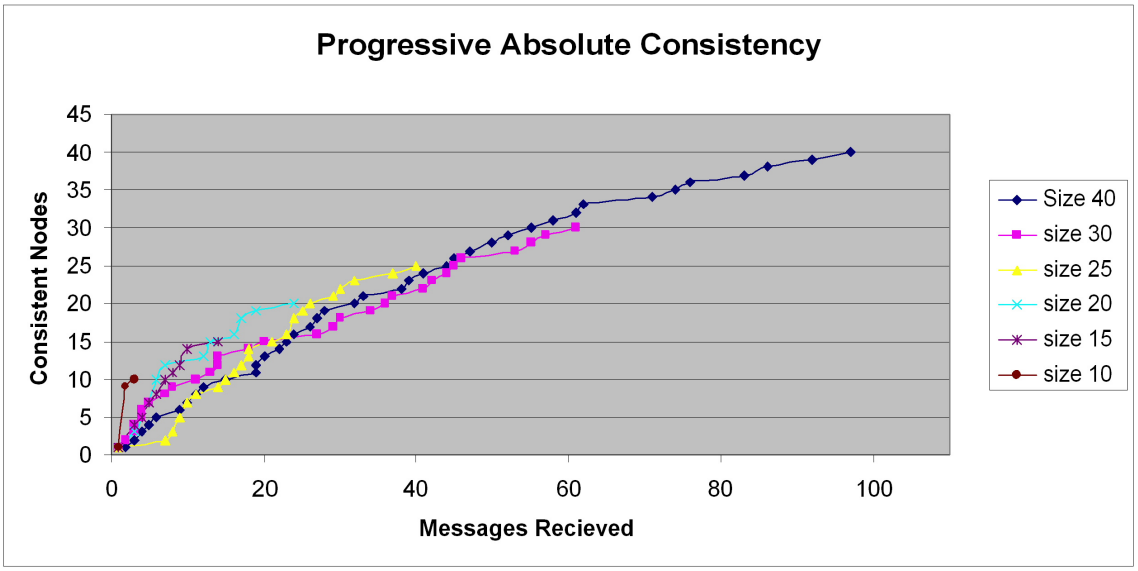

Figure 4.1: Progressive consistency establishment:Results for systems with 10,15,20,25, 30 and 40 nodes. Each includes only one valid view.

4.3 Role of Gossip

Need to understand the pattern of alternating regions of growth and temporary saturation in consistency. Also, given that gossip is an important part of our model, it would be interesting to note its role in consistency establishment.

The Figure 4.2 shows two data sets that complement each other and lend support to the observations in the earlier section.

The curves in broken lines plot nodes undergoing direct transitions. Direct transitions are by the immediate neighbors of the initiator of the view change, who received the view change directly from the initiator. These, as expected, are the front of the view changes in the system. Their effect is almost immediate, and to some extent, can improve the initial indirect transitions. The solid curves plot nodes undergoing indirect transitions. Indirect transitions are the view changes of nodes not in direct contact with the initiator. These transitions are based on the gossiped information.

Indirect transitions follow the pattern noticed in the previous section - they have a healthy initial growth, implying that a lot of nodes transitioned within a few messages.

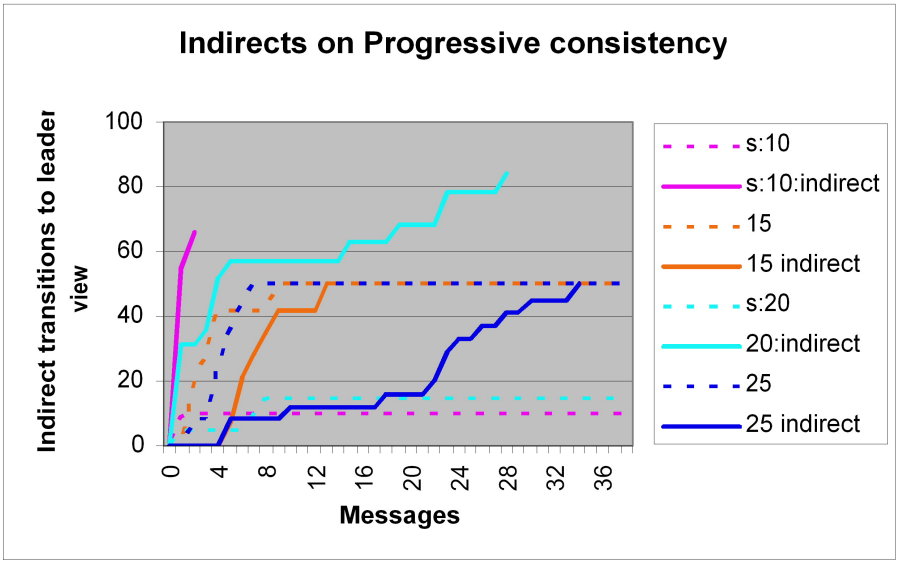

Figure 4.2: Effect of indirect transitions on Consistency Establishment, show here for systems with 10,15,20 and 25 nodes. The solid line shows the effect of transitions due to indirectly recieved views while the direct ones show the transitions based on observations of the neighbours.

This initial growth is followed by alternating regions of saturation and growth.

4.4 Inferences: Behavior of Consistency Establishment

The first experiment starts with a change of system view initiated by a single node. Each new entity that converges to the eventual system view increases the cumulative consistency of the system. Our interest is in the manner in which this cumulative consistency improves.

The consistency curve rises sharply for the first few messages and then temporarily saturates and additional messages do not cause an improvement to the system consistency. This temporary saturation is followed by another increase in consistency, although not as steep as the first one. This increase is staggered over a larger number of messages. If the system hasn't converged yet, a temporary saturation follows. This pattern gets repeated till the system converges.

The initial spike followed by gentle improvements in consistency, points to a phenomenon existing only at the start. This coupled with the pattern of increases interspersed with temporary saturation point to some local phenomenon. The second experiment shows that the pattern noticed in the earlier section is exhibited solely by the indirect transitions. The initial increase can be attributed to the nodes in the immediate neighborhood. This increase is almost never due to gossip. The gossip effect can be seen beyond this initial spike. These indirect transitions are based on gossiped view changes. While this doesn't explain the pattern by itself, it does bring to the front the role of gossip and its effect on achievement of consistency in our model.

Further, a rapid increase in transitions implies a number of nodes got the desired "trigger" within a few messages of each other. An obvious reason seems to be proximity to the initiator. Nodes would tend to take additional messages to get a specific view update as the number of intermediate nodes increases. This could mean that nodes receiving updates with a few messages of each other are at about the same level or hops from the initiator. Which implies that nodes could be in equidistant groups that may or may not talk within the group.

The alternating regions of temporary saturation followed by the groups of nodes converging within a few messages of each other represents the messages that are processed by the remaining group members before processing the trigger message that brings about the next wave of transitions. This pattern is repeated till the system converges to a single consistent view. Hence, nodes in a group transition quickly and completely in a span of few messages of each other. This means that a node receiving a trigger message can safely transition after seeing some percentage of its group transition. The confidence stems from the knowledge that the gossip would infect the remaining members.

Chapter 5
Discussion and Conclusions

5.1 Discussion

The interaction protocol and the model presented is a first step towards a solution that can support applications in the emerging systems described earlier. Understanding what this step accomplished will help define what lies beyond it.

The protocol allowed asynchronous interactions to be consistent with some guarantees. The upfront guarantees ensured that this consistency would be achieved in waves of neighborhoods. But, the present mechanism doesn't yet handle the dynamics of view updates in real systems. Scenarios exist where a view change can get subdued by a more "powerful" view. Thrashing can lead to a system that swings between views. Also, the consistency mechanism needs to accommodate issues such as multiple conflicting initiators, updates frequency exceeding the consistency attainment delay. Though node availability has be out of scope, it needs to be worked into this mechanism in a real world system. While conjecture suggests that availability may not be a showstopper considering the ad-hoc nature of the mechanism, experiments need to be conducted to characterize availability's effect. Additionally classic protocol issues like medium induced delays, out of sequence message delivery and such need to be consisdered. These need not necessarily be handled by the consistency mechanism, as is the case in the present model. But their effect has to be characterised and the cost of being protected from these issues by the communication substrate needs to be quantified.

The consistency maintenance mechanism depends on plain vanilla multicast. Such

propagation is subject to noise in the form of local system disruptions, intermediate local interests downgrading an information item's importance and hence its spread by intermediate elements. This however is not without a solution. Gossip middleware has long recognized and offered various solutions to this problem. Only, the consistency mechanism needs to use an appropriate gossiping scheme. Further, the effects of gossiping on the consistency process needs to be better understood.

Application evolution as a result of maximum probability interactions is not addressed.

5.2 Conclusions

This thesis offers the interaction model as an approach to develop applications in emerging systems. In this model, each element's behavior is governed by local goals. Each element mostly interacts within a defined neighborhood and these cascade into the system functionality. Elements interact with one other using the proposed interaction protocol.

5.3 Future Work

Let us consider what lies beyond the basic protocol and model presented.

At its core, the interaction model uses a plain vanilla multicast scheme, simple functionality specifications and a consistency mechanism to build its services. Each of these can use improvements. A well chosen gossiping scheme can offer useful characteristics to the protocol. The existing simple functionality specification needs to make way for a more powerful and rich functionality composition model. A number of open issues exist in the consistency model that have been mentioned in the previous section. These have to be addressed for a real world consistency mechanism. Further, the characteristics of consistency establishments need to be better understood. For example, as long as the groups relevant to the functionality quickly synchronize, can

the rest of the system stabilize slowly, potentially skipping a few intermediate stages? Such observations can help fine tune the consistency mechanism and make it more suited for the real world.

Beyond these foundational enhancements, the interaction model can be expanded with several nice capabilities. Neighborhood has an important role in the model and in the protocol. Application functionality is expected to leverage neighborhood and the consistency mechanism uses it to establish a notion of consistency after any instability. Bringing in context sensitiveness makes all this exciting. A flexible definition of neighborhood based on the contextual environment can potentially accommodate dynamic populations (in size, relationships or roles). This is a very powerful concept leveraging the advantages of neighborhood interactions and available contextual information that can describe other elements and the system.

As discussed earlier, complex systems are better defined in terms of the individual component goals and their interdependencies. With neighborhood interactions and a sufficiently diverse mix of elements, these individual pieces of functionality will eventually converge to create the desired system wide behavior. This flexibility in emergence of system behavior allows an application to rapidly adapt to changing environments and fulfil its objectives. This purposeful relaxation of the desired system wide behavior and the paths taken to achieve the goals has been one of the corner stones of the model. Taking it forward, the goals can potentially be "infinite" goals- they will never be completely fulfilled; this gap in the fulfillment of the goals defines the system lifecycle. Mutual goal satisfactions incrementally drive the system forward only to be brought to some state in a given element's lifecycle which causes it to be reinitialized (restarted) causing the system to remain unsatisfied, but useful to the users. But this approach means that goals should be amenable to reasoning and need to be subject to sufficient validation safeguards both at definition and at composition to achieve higher system goals. Theoretically, such systems can organically scale to real world dynamics.

In emerging systems the interpretation of guarantees itself is different from traditional systems. In many situations, qualified or probabilistic guarantees are sufficient. Adjusting to such guarantees allows an application to offer functionality in situations where none would have been possible. Though the model proposes such a mechanism, the real protocol itself doesn't offer a refined mechanism to query and argue about the guarantees available with an interaction.

References

[1] H. Abelson, D. Allen, D. Coore, C. Hanson, G. Homsy, T. F. Knight, Jr., R. Nagpal, E. Rauch, G. J. Sussman, and R. Weiss. Amorphous computing. *Communications of ACM*, 43(5):74–82, 2000.

[2] M. Agarwal, V. Bhat, Z. Li, H. Liu, B. Khargharia, V. Matossian, V. Putty, C. Schmidt, G. Zhang, S. Hariri, and M. Parashar. Automate: Enabling autonomic applications on the grid. In *Proceedings of the Autonomic Computing Workshop, 5th Annual International Active Middleware Services Workshop (AMS2003)*, pages 48–57, Seattle, WA, USA, June 2003. IEEE Computer Society Press.

[3] J. Aspnes. Randomized protocols for asynchronous consensus. *Distrib. Comput.*, 16(2-3):165–175, 2003.

[4] K. P. Birman. The surprising power of epidemic communication. In *Workshop on Future Directions in Distributed Computing*, Bertinoro, Italy, June 2002. Spinger-Verlag.

[5] K. P. Birman, M. Hayden, O. Ozkasap, Z. Xiao, M. Budiu, and Y. Minsky. Bimodal multicast. *ACM Transactions on Computer Systems*, 17(2):41–88, 1999.

[6] A. Carzaniga, D. S. Rosenblum, and A. L. Wolf. Design and evaluation of a wide-area event notification service. *ACM Transactions on Computer Systems*, 19(3):332–383, August 2001.

[7] D. Coore, R. Nagpal, and R. Weiss. Paradigms for structure in an amorphous computer. Technical report, 1997.

[8] A. Das, I. Gupta, and A. Motivala. Swim: Scalable weakly-consistent infection-style process group membership protocol. In *Proceedings of the 2002 International Conference on Dependable Systems and Networks*, pages 303–312. IEEE Computer Society, 2002.

[9] A. Demers, D. Greene, C. Houser, W. Irish, and J. Larson. Epidemic algorithms for replicated database maintenance. *SIGOPS Oper. Syst. Rev.*, 22(1):8–32, 1988.

[10] D. Estrin, R. Govindan, J. Heidemann, and S. Kumar. 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*, August 1999.

[11] P. Eugster and R. Guerraoui. Probabilistic multicast, 2002.

[12] M. J. Fischer, N. A. Lynch, and M. S. Paterson. Impossibility of distributed consensus with one faulty process. In *Proceedings of the 2nd ACM SIGACT-SIGMOD symposium on Principles of database systems*, pages 1–7. ACM Press, 1983.

[13] J. Fredslund and M. J. Mataric. Robots in formation using local information. In *The 7th International Conference on Intelligent Autonomous Systems (IAS-7)*, March 2002.

[14] R. A. Golding and K. Taylor. Group membership in the epidemic style. Technical report, 1992.

[15] R. Grimm, J. Davis, B. Hendrickson, E. Lemar, A. MacBeth, S. Swanson, T. Anderson, B. Bershad, G. Borriello, and S. Gribble. Systems directions for pervasive computing. pages "147–151", "2001".

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

[17] R. Guerraoui and A. Schiper. The generic consensus service. *IEEE Transactions on Software Engineering*, 27(1):29–41, 2001.

[18] I. Gupta, K. P. Birman, and R. V. Renesse. Fighting fire with fire: Using randomized gossip to combat stochastic scalability limits. *Quality and Reliability Engineering*, 18:165–184, 2002.

[19] I. Gupta, A.-M. Kermarrec, and A. J. Ganesh. Efficient Epidemic-style Protocols for Reliable and Scalable Multicast. In *21st Symposium on Reliable Distributed Systems (SRDS 2002)*, pages 180–189, October 2002.

[20] Z. J. Haas, J. Y. Halpern, and L. Li. Gossip-Based Ad Hoc Routing. In *IEEE INFOCOM 2002*, New York, NY, June 23–27 2002.

[21] W. Hordijk, J. P. Crutchfield, and M. Mitchell. Embedded-particle computation in evolved cellular automata. Working Papers 96-09-073, Santa Fe Institute, Sep 1996. available at http://ideas.repec.org/p/wop/safiwp/96-09-073.html.

[22] Jess. Internet: http://herzberg.ca.sandia.gov/jess.

[23] L. Lamport. Lower bounds for asynchronous consensus. In *Future Directions in Distributed Computing*, volume 2584 of *Lecture Notes in Computer Science*, pages 22–23. Springer, 2003.

[24] Muse. Internet: http://www.winlab.rutgers.edu/pub/docs/focus/MUSE2.html.

[25] R. Nagpal, G. J. Sussman, and H. Abelson. *Programmable self-assembly: constructing global shape using biologically-inspired local interactions and origami mathematics*. PhD thesis, 2001.

[26] Project JXTA. Internet: http://www.jxta.org.

[27] Project Oxygen. Internet: http://oxygen.lcs.mit.edu.

[28] F. B. Schneider and Y. Minsky. Tolerating malicious gossip. *The Distributed Computing Journal*, 16(1):49–68, February 2003.

[29] SETI. Seti@home. Internet: http://setiathome.ssl.berkeley.edu.

[30] J. Turek and D. Shasha. The many faces of consensus in distributed systems. *Computer*, 25(6):8–17, 1992.

[31] R. van Renesse and K. Birman. Scalable Management and Data Mining Using Astrolabe. In *Proceedings of the 1st International Workshop on Peer-to-Peer Systems*, 2002.

[32] R. Van Renesse, K. P. Birman, and W. Vogels. Astrolabe: A robust and scalable technology for distributed system monitoring, management, and data mining. *ACM Trans. Comput. Syst.*, 21(2):164–206, 2003.

[33] W. Vogels, R. van Renesse, and K. Birman. The power of epidemics: robust communication for large-scale distributed systems. *SIGCOMM Comput. Commun. Rev.*, 33(1):131–135, 2003.

[34] M. Weiser. The computer for the 21st century. *Scientific American*, 256(30):94–104, 1991.