ICS 243F Paper Summaries for Week 3

Sushil K Bajracharya
sbajrach@ics.uci.edu

1 Summary of paper [Lam78]

The notion of a global time and global state is not easy to structure in Distributed Systems. [Lam78] presents a way to represent a global time in a distributed system using a total ordering of the set of events occurring in the system. The paper defines a *process* as a sequence of totally ordered *events* and a *distributed system* as a set of distinct spatially separated processes communicating by sending and receiving *messages*. An event can be a subprogram or a single machine instruction. *Receiving messages* and *sending messages* are two different events.

With these primitives the paper progresses by presenting these major concepts -

Happened Before relationship and partial ordering of events Given two events *a* and *b*, *a* is said to be **happened before** *b* given three conditions are fulfilled i) *a* and *b* belong to the same process and *a* comes before *b* ii) If *a* is a message sending event in one process then *b* is the receiving event in another process taking the same message from *a*. iii) If *a* happened before *b* and *b* happened before *c*, *a* happened before *c*.

a and *b* are concurrent if *a* did not happened before *b* and *b* did not happened before *a*. This makes the **happened before** relationship an irreflexive partial ordering on the set of all events in the system.

Logical Clocks Logical Clocks are not associated with any physical clocks but they define a way to assign a number to an event, the numbers representing the times at which events occur. A **Clock Condition** is defined that need to be satisfied in order for the system events to be ordered correctly. The Clock Condition says that the clock value assigned to an event *a* is smaller than the clock value assigned to event *b* if *a* **happened before** *b*.

Clock Condition can be satisfied by making sure two conditions are met i) there must be at least a clock tick between any two events in a process and ii) there must be at least a clock tick between the sending of a message by a process and its receipt by another process. These conditions can be implemented in system clocks by following -

- Each process increments its clock between any two consecutive events
- Each message *m* contains a timestamp T_m containing the current time of the sender
- For an event *a* sending message *m* from process P_i, $T_m = C_i < a >$ (Clock value of process P_i at event *a*)
- P_j sets C_j to the greatest value among its current clock value or the timestamp it receives from message m.

Total Ordering of Events Since the events are tagged with logical clock values these values are used to totally order them and any possible ties are broken using an arbitrary total ordering of the processes. This makes the possibility of the total ordering not being unique.

This total ordering of events is used to demonstrate solving a mutual exclusion problem and a possible anomaly due to the total ordering of events is discussed. The paper suggests manually describing the correct sequence of events (in a way the user would perceive them) or having stronger clock guarantees with synchronized physical clocks as two possible ways to deal with such anomaly.

Physical Clocks Finally the paper provides a method for synchronizing physical clocks using logical clocks. Two conditions are given for the validity of physical clocks i) Physical clock should run with only a small variation in the rate ii) The readings of all the clocks at a particular time only varies within a threshold amount.

In brief, [Lam78] defines the **Happened Before** relation that gives a partial ordering of events. The partial order can be made total with an arbitrary ordering in case there are ties among events. This can lead to anomaly as the ordering is not unique and can be solved with techniques like synchronized clocks.

Creating a linear order of events is somewhat losing independence of events and losing information and other alternatives to linear ordering had been proposed, such as using Vector or Matrix clocks.

2 Summary of paper [Cha85]

This paper [Cha85] addresses the second fundamental problem in distributed systems - simulating a global state. It presents a model of a distributed system and a global-state-detection algorithm based on that model that allows to capture a global state of the system by the running processes in background without interfering with the normal execution. The algorithm allows to record the states while the processes send and receive messages also considering the message in transit in the channel. The algorithm is useful to verify stable properties of a distributed system such as deadlock detection.

Model A distributed system comprises a finite set of processes and a finite set of channels. A process is defined by a set of states, an initial state and a set of events. An event e is defined by a 5-tuple $< p, s, s', M, c >$ where p = process in which the event occurs, s = state of p before the event, s' = state of p after the event, c = channel, M = message. c and M may not exist. e is an atomic action that may change the state of p and the state of at most one channel c. The state of c can be changed either by receipt or sending of message along c. A global state of a distributed system is a set of component process and a channel states.

Algorithm The global-state-detection algorithm works as processes send and receive a special message *marker* along the the channel according to the following rules.

Marker sending rule for process P_i

```
If (P_i has not yet recorded its state)
    records its process state now
    records the state of c as the empty set
    turns on recording of messages arriving over other channels
else
    P_i records the state of c as the set of messages received over c
    since it saved its state
```

Marker receiving rule for process P_i

```
After P_i has recorded its state,for each outgoing channel c
    P_i sends one marker message over c
    (before it sends any other message over c)
```

Usage The global state S^* recorded by the algorithm need not be identical to any of the global states the system had during the computation but S^* holds the following property that makes it useful -

Given that the algorithm was initiated at state S_l and terminated in state S_t :

- S^* is reachable from S_l
- S_t is reachable from S_*.

With such a property one can come with an algorithm for stability detection as follows -

```
begin
    record a global state S*
    definite := y(S*)
end
```

where *definite* is a boolean value and y is a stable property.

3 Comments

One of the subtle issues as both of these papers [Lam78] and [Cha85] have as they introduce such fundamental concepts of simulating a global time and state is that the theories and formalism are based upon many assumptions made to simplify tasks. Some of those being assumptions of reliable message delivery, perfect ordering of message transport and arbitrarily creating a total ordering. These factors certainly have to be considered while implementing a real system. This leaves many open issues and opportunities to improve upon the thesis presented.

One such work seems to be done in [Mat89] where the authors claim that the linearly ordered structure of time is not always appropriate for distributed systems and they present a partially ordered system of vectors forming a lattice structure as a natural representation of time in distributed system.

Above all such issues, these papers lay the foundation for reasoning about time and state in distributed system. The mathematical treatment given in these papers demonstrate a sound technique to reason about such issues.

References

[Cha85] Lamport L. Chandy, K.M. Distributed snapshots: Determining global states of distributed systems. In *ACM Transactions on Computer Systems*, volume 3, pages 63–75, 1985.

[Lam78] Leslie Lamport. Time, clocks and the ordering of events in a distributed system. In *Communications of the ACM*, volume 21, pages 558–565, 1978.

[Mat89] F. Mattern. Virtual time and global states of distributed systems. In Cosnard M. et al., editor, *Proc. Workshop on Parallel and Distributed Algorithms*, pages 215–226. North-Holland / Elsevier, 1989.

ICS 243F Paper Summaries for Week Four

Sushil K Bajracharya
sbajrach@ics.uci.edu

1 Summary of paper [BDF+97]

This paper outlines the goals and the principles the authors had for building a future generation of distributed operating systems in a project they call the *Millennium* project. They also present the technology trend as motivation and outline the desired features in such operating system.

The authors set their goals focusing on improved end user experience especially aiming - i) absolute transparency and scalability of the distribution of resources and computing tasks in the system to the users, ii) an autonomous fault tolerant system that can adapt with the changing environment handling failures and providing a self-tuning and self-configuring environment and iii) a non-centralized security model with flexible resource control. To build a system meeting such goals following principles are outlined - i) With ***Aggressive Abstraction*** the operating system will provide an interface to a virtual machine that would allow applications to be written for last moment execution. It will also abstract the security and portability details from the users and application developers. ii) ***Storage and Location irrelevance*** will enable a universal access of files and resources anywhere/anytime iii) ***Just-in-time Binding*** or 'binding-by-search' will enable the applications to be written without explicitly pointing to the distributed resources they use. iv) ***Introspection*** will provide an autonomous self-monitoring operating system that can provide dynamic and optimum configuration of resources and the system iteself.

These goals and principles were based upon the technology trends shaping the future of distributed computing. Some of such technology innovations being - advances in hardware technology, compilers and languages, distributed file systems and virtual machine abstractions like the Java platform. With these issues in mind the authors give a vision for future generation of Operating Systems that would enable dynamic plug-and-play of new machines in self-monitoring networks resistant to few hardware failures. Furthermore, they claim such environment would be an ideal platform for running Web Services and Distributed Programming.

2 Summary of paper [TKvRB1]

This paper is a brief status report of a distributed operating system - Amoeba. Amoeba is based on a microkernel architecture. The microkernel runs on every processor and is responsible for primitive kernel tasks like process management, low-level memory management, thread communications and I/O. Besides these taks the microkernel is responsible for thread communications where the threads participating in communication can be executing in different processors. RPC is the default mechanism for thread communiation. Further services like file and directory services are implemented as server processes. All the Servers in Amoeba are based on a basic concept of an *Object*.Objects are abstract representation of data a particular service might be using or serving. Objects are identified and controlled by a 128 long identifier (ticket) called *Capabilites* that describes where the object resides with what credentials. Clients and servers exchange these capabilities using RPC for distributed communication.

The basic motivation behind Amoeba was the fact that CPU chips being more affordable it was possible to have computer system with large number of processors within a Local Area Network (LAN). The role

of the operating system would be to make the numbers of processors or such distribution transparent to the end user giving him/her a consistent single interface to the system. The hardware Amoeba runs on basically is collection of processors with their own memory connected with ethernet. Users connect to the system through Workstations that basically are thin clients running some client interface like X-windows.

As an example of higher services that run outside the microkernel the authors present the examples of File Server, Directory Server and the Boot server. Possible domains where Amoeba can used are also listed. The authors claim that the lack of a broadcasting technique limits the architecture of Amoeba to be directly applied in Wide Area Networks, for this they suggest a gateway based interface to connect two separate LANs running Amoeba.

3 Comments

Both of these papers were published almost a decade ago especially [TKvRB1] is relatively older. The issues presented in these papers would be significantly revolutionary if we put ourselves in that time but looking over the major idea behind these papers today one would certainly find some concepts we need to rethink on. Here are few of them -

1. The architecture of Amoeba seems more of a Parallel System than a Distributed System in today's scenario as hardware has become even cheaper and what end-users have at their disposal are more powerful machines with big storage capacity so the idea of connecting users from a thin client having only a minimal client software no longer is valid.
2. [BDF+97] does not seem to make any significant contribution in terms of giving a new solution. It basically puts down all the ideal concepts we would like to see or have but realizing these concepts in an actual system is a totally different and more important issue that the authors seem to neglect in the paper.
3. Moreover there are systems that are very constrained in terms of resource like PDAs and embedded systems. Considering integration of these kinds of clients in a system adds a new dimension of pervasiveness. With current hardware and network capabilities Ubiquitous Computing is definitely the next or the current big thing and this issue is not touched upon directly in either of the papers.
4. There can be a good amount of debate in the issue that whether the layer of software that abstracts away the distributed computing aspects should be integrated in an underlying operating system or should be separately built as a second layer of software on top of it. With all integration there again comes the demand for standardization, will such a virtual machine abstraction and the format of the mobile code as demanded by just-in-time binding be based on a standard and vendor independent if such issues are integrated in the operating systems ? What will be the cost of interoperability ?
5. Reflective system surely provides more autonomous and adaptive features but what would be the cost of adding an extra layer of software needed for such reflective capabilities ?
6. Broadcasting in the WAN as seen as a bottleneck for scalability in Amoeba is no longer an impossible task. Certainly there have been progress in the area, especially with the advent of Internet scale notification services WAN broadcasting will be possible and will have a key effect in future design of Distributed Systems.
7. Targeting systems to be developed for End Users is always the viable concept. Eventually systems should be built to improve the users' computational experience.
8. Absolute Transparency is what we desire for, but how feasible is that when our infrastructure is build upon a network architecture that has been there for decades. Protocols and Services still have centralized base. For example how fully distributed is the DNS service ? What we are building as inherently distributed systems are autonomous systems on top of such infrastructure may be one day protocols will be replaced or enhanced but currently total transparency is still a trade off with full reliability.

9. Virtual Machines and Abstractions are the key to achieving ease of development and architectural separation of distribution aspects but again standardization is an issue. We have been experiencing the Java Virtual Machine as one of the dominant platform in this area but with an invent of competing technology like the .net CLR again there comes another issues of Interoperability.

References

[BDF+97] William J. Bolosky, Richard Draves, Robert P. Fitzgerald, Christopher W. Fraser, Michael B. Jones, Todd B. Knoblock, and Richard F. Rashid. Operating system directions for the next millennium. In *Workshop on Hot Topics in Operating Systems*, pages 106–110, 1997.

[TKvRB1] Andrew S. Tanenbaum, M. Frans Kaashoek, Robbert van Renesse, and Henri E. Bal. The amoeba distributed operating system: a status report. *Computer Communications*, 14:324–335, 1.

ICS 243F Paper Summaries for Week 5

Sushil K Bajracharya
sbajrach@ics.uci.edu

1 Summary of paper [Sch00]

[Sch00] is a survey of some popular CORBA implementation and moreover it describes a basic load balancing framework for CORBA based applications.

The CORBA application model constitutes four distinct entities - (i) client, (ii) naming/trading component, (iii) ORB and (iv) the server. For establishing a basic load balancing framework in such an application it is necessary to analyze how an application can be *partitioned* among these basic entities and how the application workload can be *assigned* to servers holding application objects.

Partitioning includes an appropriate functional decomposition of the application so that entities amenable for *replication* can be identified. Replication can be either *static* (during compile time, service instantiation time) or *dynamic* (during runtime). Assignment includes assigning a *request* to a particular servant or assigning a particular servant to an appropriate host (server). Requests can be assigned *on-demand* (per-invocation), as *trading* (per-session) or as *preemptive trading* (forceful redirection). Assignments of servants might be done at *instantiation time* and these servants might be *migrated* either on a preemptive or a non-preemptive basis.

Most of the load balancing methods can be related to and implemented based on the already described standard *design patterns* in the literature. Looking into the CORBA model, some *integration points* to realize the partitioning and assignment as mentioned above can be identified, such as - (i) in the ORB kernel itself, (ii) between ORB interfaces and the application code, (iii) in some of the basic object services in the model and (iv) in the application itself. *Monitoring*, *Strategies* and *Control* are the three basic components that makes up a load balancing method. A *load-index* defines a suitable integration point for monitoring a system. Possible load indices for CORBA can be - Load of the system resources, number of active threads, message queue length, operation and servant monitoring. A tradeoff between achieving a good load balance and the overhead involved with the execution of such load balancing methods can be balanced by introducing *thresholds*. Such thresholds should be defined and considered as load balancing strategies can be integrated into any part of a CORBA system.

Partitioning and assignment as mentioned earlier helps establishing the *control* or the implementation of the mechanism. For example, being object oriented, the CORBA model provides a good platform for a proper functional decomposition. Facilities like *communication filters* and *smart proxies* as provided by some CORBA implementations can be used to separate the load balancing functionality from the application specific code that can help implementing preemptive trading mechanisms in the application.

The paper also gives a concise summary of these issues in a tabular form at last.

2 Summary of paper [CCY99]

This paper is a survey of different dynamic load balancing strategies for distributed web servers. It compares how different techniques stand against scalability and availability. Four basic approaches for dynamic load balancing are compared and further different possibilities under each category are analyzed and compared.

The basic four techniques are -

Client-based approach Not very feasible, this approach is simple and requires the client or the browser to maintain information about multiple available servers that can be contacted. Redundancy and random selection is the key but this solution absolutely is not attractive in today's scenario.

DNS-based approach *cluster* DNS servers are used to resolve the client requests to alternate or lightly loaded web servers. Caching of URLs in intermediate name servers and even in the client becomes an issue. This can be

controlled to some degree by changing the *TTL* values so that the client requests get to the *cluster* DNS server more frequently. But the solution still does not take into account the capabilities and state of servers in terms of load they are experiencing. Scheduling algorithms based on *server state*, *client-state* or combination of both that can take in to account the dynamic status of client/servers and change the *TTL* value accordingly is an improvement.

Dispatcher based approach A *packet dispatcher* can be used that rewrites the packet and modifies the address to dispatch and reroute the client requests to an appropriate server. Instead of doing at the lower level, HTTP protocol itself can be used to have a dispatcher that works as a HTTP redirector that redirects the clients requests appropriately.

Server-based approach The redirection can be done at the server itself rather than be done by a separate dispatcher and this can be combined with previous approaches like *DNS based approach* for better efficiency.

Almost all these techniques have strengths and weaknesses, and often they are suitable in different situations like in a LAN based environment of a WAN based environment. Finally the paper gives a brief quantitative and qualitative analysis of these techniques.

3 Analysis

Both of the papers were interesting as they touch upon fundamental issues in load-balancing techniques but at different levels. [Sch00] describes the load balancing issues in application and middleware layer whereas [CCY99] describes load balancing from a network and protocols viewpoint.

The organization of [Sch00] was good as it gradually presented the required information in succint way. The introduction of the CORBA application model was benefitial as even those not familiar with the innards of CORBA would understand the fundamental issue discussed in the paper. The partition and assignment model discussed seems to fit many of the application domains besides CORBA and definitely provides a basis for any similar load-balancing implementation. However I did not find the paper covered information from enough of the CORBA implementations besides few. So the generalized summary would have been more full proof if based on more case studies. Also the paper notes many of the issues to be missing from then current CORBA model. Some the difficult and performance critical aspects like monitoring and dynamic adaptation can be implemented in a more controlled fashion if not in an easy way by adopting techniques like computational reflection or aspect oriented programming. If the paper had further discussed the trade offs and costs involved in adopting such techniques then it would have been an additional advantage in understanding subtle load-balancing issues.

[CCY99] on the other hand surely has included more case studies and examples from the literature. But one thing that I felt lacking in the paper was that it somewhat seems obselete when web servers are treated barely as a machines serving static HTML pages. A large faction of the web server family today go far beyond this simple information publishing model. Most of the web servers are used as a front end to serve the client interfaces of web applications the content of which might be dynamically generated by some application server or the web server itself on demand. In this scenario where the server has to do deal with client sessions and stateful data, it becomes more difficult to handle scalability and availability. I could not see how the techniques described in [CCY99] directly or indirectly addresses this issue.

References

[CCY99] Valeria Cardellini, Michele Colajanni, and Philip S. Yu. Dynamic load balancing on web-server systems. *IEEE Internet Computing*, 3(3):28–39, 1999.

[Sch00] Thomas Schnekenburger. Load balancing in corba: A survey of concepts, patterns, and techniques. *J. Supercomput.*, 15(2):141–161, 2000.

ICS 243F Paper Summaries for Week 6

Sushil K Bajracharya
sbajrach@ics.uci.edu

1 Summary of paper [AWB+94]

[AWB+94] describes how traditional object oriented (OO) techniques lack features to describe the rich semantics of message passing that makes it difficult to represent abstract inter-object interactions in a flexible manner and at a higher level. This serves as a bottleneck when purely traditional object oriented techniques are used in developing complex systems, like those having layered communication architectures. As a solution [AWB+94] presents *Abstract Communication Types* (ACTs) to augment object orientation with higher-level language semantics.

ACT is based on an improved technique for object oriented interaction called the *Composition Filter* model and can be applied using the concurrent programming language *Sina*. ACT mainly tries to address the three limitations seen in the traditional OO models - (i) Lack of support for meta-levels and reflection, (ii) Complexity and Lack of Reusability to separate out interaction patterns among objects and (iii) Distribution or scattering of invariant behavior over a number of objects. The composition filter model enchances the object-oriented technology by introducing the notion of Input and Output *Filters* that are controlled by a set of *conditions*. These filters are first-class entities that can be reused and composed as other OO classes. Filters are the basic entities that are used to define ACTs.

Four basic requirements are identified for defining effective communication abstractions with ACTs - (i) ability to treat communication as *First-class property*, (ii) rich semantics to express *large scale synchronization mechanisms*, (iii) reflection upon message for monitoring, logging etc, and (iv) uniform integration of communication semantics. Two important classes in the ACT framework are the *Meta-Filter* class that can reify messages that pass through them and the class *Message* that is the first-class representation of the message being passed between objects. In particular the use of the composition filters in realizing ACTs provides much of the base needed to achieve the four before mentioned requirements for ACTs.

2 Summary of paper [Ven]

[Ven] describes CompOSE|Q (Composable Open Software Environment with QoS), a Qos enabled customizable middleware framework for distributed computing based on the TLAM model. CompOSE|Q allows the concurrent execution of multiple resource management policies in a distributed system in a safe and correct manner.

Since based on TLAM, CompOSE|Q consists of actors (base and meta) that are distributed over network, run parallel to each other and can communicate with each other via asynchronous message passing. It is an actor based framework that permits customization of resource management mechanisms such as placement, scheduling and synchronization.

The architecture of CompOSE|Q is made up of three components - (i) Modules implementing the three core services - remote creation, distributed snapshot and directory services with interaction constraints that ensure their concurrent execution with each other and other meta level services, (ii) Common services built upon the core services such as actor migration, replication of services and data, actor scheduling, distributed garbage collection, name services etc with their own definitions and interaction constraints, and (iii) QoS specification and enforcement mechanisms.

CompOSE|Q includes a set of *meta level resource management services* to implement sophisticated policies and mechanisms for QoS management. It also includes a *reflective communication service architecture* which basically extends TLAM with a *composable reflective communication framework* (CRCF). This is the core that provides correct composition of communication services to QoS-based applications in a transparent and scalable fashion while ensuring correctness of basic middleware services. The *CompOSE|Q Runtime Architecture* is implemented in Java and is made up of three basic components - i) A *NodeManager* that manages and coordinates various components on a node, ii) A

NodeInfoManager that manages information needed by the local actors and interfaces with the directory service and iii) A *communication sub-system* that handles messaging between actors.

In brief, CompOSE|Q is a dynamic adaptive middleware platform with a reflective messaging core and provides a safe composability of services it provides.

3 Analysis

It is interesting to see how both [AWB+94] and [Ven] focus on *Computation Reflection* as the foundation upon which both of the systems mentioned in the papers are built. But, reflection comes at a price and these papers do not seem to explicitly address the issue that what extra computational overhead goes on with the use of reflection and what strategies are there to compromise upon such issues. [AWB+94] does mention about semi-reflective systems that are not purely reflective but this is not explained in further depth. Certainly there are newer concepts like *Aspect Oriented Programming* that extend the semantics of an object oriented language thus providing similar features as Composition Filters. Comparing all these related techniques might be worthwhile in seeing which approach best fits a particular problem domain.

[AWB+94] has a neat presentation of topics and almost clearly explains all the issues that are relevant to understand the issue being addressed in paper. The core issues CompOSE|Q addresses as such was understood after further referring to papers related to the project ([Ven02] [WEB] [VT01] [ea]) and it seems it indeed addresses the subtle issue of *composability* of services that becomes more of a problem especially when the system is more flexible and components can be composed dynamically. A formal basis for such compositions is surely a viable solution. As a framework CompOSE|Q [Ven] is more focused on distributed middleware. Composition Filters or ACTs [AWB+94], on the other hand seems more general and applicable to various domains. But with this generalization again, ACTs do not directly seem to provide support to reason about composing the components in the systems, for instance the already implemented set of reusable *filters*. [AWB+94] claims these filters are orthogonal and can be freely used independent of each other but it is not clear was there any sound technique to come up with such orthogonality and independence between filters. [AWB+94] does give a semantic description of the message passing in the appendix but it is not clear where such formalism can be applied or was used.

It seems both papers do share a common point and certainly there are concepts in the systems described in both papers that complement each other.

References

[AWB+94] Mehmet Aksit, Ken Wakita, Jan Bosch, Lodewijk Bergmans, and Akinori Yonezawa. Abstracting object interactions using composition filters. In *Proceedings of the Workshop on Object-Based Distributed Programming*, pages 152–184. Springer-Verlag, 1994.

[ea] Nalini Venkatasubramanian et al. Design and implementation of a composable reflective middleware framework. *The 21st International Conference on Distributed Computing Systems*.

[Ven] Nalini Venkatasubramanian. CompOSE|Q - A QoS-enabled customizable middleware framework for distributed computing.

[Ven02] Nalini Venkatasubramanian. Safe 'composability' of middleware services. *Commun. ACM*, 45(6):49–52, 2002.

[VT01] Nalini Venkatasubramanian and Carolyn L. Talcott. A semantic framework for modeling and reasoning about reflective middleware. *IEEE Distributed Systems Online*, 2(6), 2001.

[WEB] Web Site: ComPOSE|Q (http://http://www.ics.uci.edu/d̃sm/compose/compose_design.html).

ICS 243F Paper Summaries for Week 7

Sushil K Bajracharya
sbajrach@ics.uci.edu

1 Summary of paper [Mic]

[Mic] gives an overview of the architecture of Jini, an extension to the Java application environment to build systems in dynamic distributed environments. The idea behind Jini is to leverage the strong typing in the Java language and built in secure computing model in the JVM (Java Virtual Machine) to extend the JVM to a network of machines where services can be registered, dicovered and used as required. A Jini system is a collection of a set of *components*, providing the infrastructure and a *programming model* for building reliable distributed services. *Services* offer a particular functionality to other members of the system. A Jini based system comprises a federation of all kinds of computers and computing devices ranging from normal workstations to embedded devices, all having a JVM running in them.

Jini provides a loosely coupled, service oriented network environment where devices can *lookup* for the required *services*, service providers can *publish* their services and, services consumers and providers need not be explicitly known to each other before the interaction between them actually happens. *Java RMI* is the default infrastructure to support communication between service providers and consumers. Access to the services is *lease* based and there are provisions to wrap services to implement functionalities like two-phase-commit *transactions*. Furthermore, objects can communicate using distributed *events*.

The infrastructure that enables Jini is an extension to the existing Java platform in three different ways -

Infrastructure extension A *distributed security system* integrated into RMI, the *discovery/join protocol* - a service protocol enabling hardware and software services to be discovered as well as advertised and the *lookup service* which serves as a repository of services.

Programming Model Extension Jini adds on a set of interfaces to the base Java programming model such as the *leasing* interface, the *event and notification* interface (extension to the JavaBeans components to the distributed environments) and the *transaction* interfaces.

Services Services are offered as programmable objects written in the Java programming language. These services have well defined interfaces that define their capabilities and these can be discovered and used by other services. A *lookup service* can be used to maintain a registry of available services in a network. A typical interaction between clients and service providers includes a service request by client to the lookup service and the lookup service providing the client with the appropriate reference for the service it requires. The lookup service has this information as the service providers register their capabilities with it. Once clients get the appropriate handle then they can directly interact with the service provider. It might even be a case that a service might be a small self-contained code that the client can download from the lookup service provider and go on using.

2 Summary of paper [Gon]

JXTA is a set of protocols providing a platform to build inherently distributed, P2P (peer-to-peer) applications. JXTA initially started out as a research initiative at Sun Microsystems and now is an open source project. It provides a set of language and platform independent protocols for developing truly P2P applications, a reference implementation in Java is available along with some demo applications from its official site `http://www.jxta.org`.

A quick and short description of P2P computing as [Gon] outlines, is a network based computing style that neither excludes nor inherently depends on centralized control points. The goal behind distribution being to achieve maximum utilization of resources and improved performance in terms of information discovery, reliability etc. On top of these *interoperability*, *platform independence* and *ubiquity* are stated as the three high level goals for JXTA.

JXTA defines a layered P2P software architecture made up of three layers sitting on the top of lowermost layer that is *a peer on the extended web* (the host on which a JXTA system is installed and can be any device or a computing machine connected to a network on which JXTA operates). The three basic layers in JXTA are -

JXTA Core This layer deals with lower level issues like *peer establishment* and *communication management*. It again consists of four basic elements - (i) *Security*, (ii) *Peer Groups*, (iii) *Peer Pipes* and (iv) *Peer Monitoring*.

JXTA services Built on top of the core this layer handles higher level issues like indexing, searching and file sharing.

JXTA Applications Finally on the top are the application running in JXTA.

JXTA relies on but is not tied to an underlying transport protocol like TCP/IP, HTTP or any other protocol. In essence JXTA is an abstract model of writing distributed application on top of such existing network infrastructure. In this regard the following protocols that JXTA defines are of importance - (i) *Peer Discovery Protocol*, (ii) *Peer Resolver Protocol*, (iii) *Peer Information Protocol*, (iv) *Rendezvous Protocol*, (v) *Pipe Binding Protocol* and (vi) *Endpoint Routing Protocol*. These protocols work together to allow a peer to *advertise* its existence and services in the network, which can be cached by special hosts known as *rendezvous* nodes. When peers discover each other they can communicate using *pipes*. Pipes can be created and destroyed in demand and can be reused. Peers can also have provision to form *groups* and implement special authentication and authorization policies.

3 Analysis

Both Jini and JXTA has been there for a while, both of these technologies surely address the basic issues that exist in distributed software systems. Yet, it is a matter to think why these technologies have not got a wide-spread adoption even with the solutions they claim to have with them. Especially Jini is not new, it tries to present the same service oriented model to develop distributed systems that Web Services these days are trying to address. Yet, why did Jini could not come out of the labs and why we do not really have seen a 'killer' application based on Jini, or JXTA ?

One possible issue with Jini might be that it is vendor specific. Adopting Jini means the need to have JVM and Java in all the devices. Java is an elegant technology and solves lot of problem, makes life lot easier but it is not the only solution out there. Especially with the semi-open model of Java platform where some of the control still come from Sun, a global adoption of such a single technology is not feasible. JXTA on the other hand is not tied to JAVA alone but it assumes following a standard set of protocol for all the systems participating, which again does not seem to be an easily attainable goal.

Jini and JXTA state all the attractive goals to be fulfilled, but in what extreme does the implementation of these technologies address much fundamental problems in distributed systems is yet to be researched and definitely is not and, cannot be expected to be known from reading industrial whitepapers or technical guides. Jini and JXTA do put together issues that are current challenges and demands to develop inherently distributed systems but it is yet to see how much do these technologies actually contribute to the research community by providing significantly new ideas. If they just put together things that already have been there and package them in a vendor specific solution, they are not as much as consequential as they seem to be.

References

[Gon] Li Gong. Project JXTA: A technology overview.

[Mic] Sun Microsystems. Jini architectural overview. *Technical White Paper*.

ICS 243F Paper Summaries for Week 8

Sushil K Bajracharya
sbajrach@ics.uci.edu

1 Summary of paper [RAV99]

Egida (Extensible Toolkit for Low-overhead Fault-Tolerance) is an object-oriented toolkit for specification and implementation of fault tolerant protocols based on *log based rollback recovery*. Log based recovery is one of the many techniques available for implementing fault tolerant systems. It sure has limitations but owing to its simplicity it is suitable for non-mission critical application domains such as parallel scientific applications running on supercomputers. This is the motivating factor behind the design and implementation of Egida.

As a toolkit Egida is made up of constituents ranging from grammar for protocol specification to object libraries. The design approach Egida takes is interesting as it generalizes three categories of log-based rollback recovery protocols - (i) optimistic, (ii) pessimistic and (iii) causal protocols to an event based structure. The idea behind this generalization is to be able to define these recovery protocols as event driven programs. Five types of events are identified that are relevant to all the protocols, the events being - (i) Non-deterministic, (ii) Dependency-generating, (iii) Output-commit, (iv) Checkpointing and (v) failure detection events. A protocol is defined in terms of actions it takes in response to one of these events. Having defined these events, a general grammar for a simple language is presented that contains several event variables which when instantiated can be used to define already known protocols or to create new ones.

Besides these formal structures Ediga also has a set of *module definitions* and *interfaces*. These extensible module definitions are bound together after synthesizing the protocols, once they are defined by an appropriate grammar.

The implementation of Egida comes integrated with MPICH (an implementation of the MPI standard) and has been used to study the performance of rollback algorithms with a set of NAS benchmarks.

2 Summary of paper [CRS+98]

[CRS+98] describes AQuA (Adaptive Quality of Service Availability), a framework consisting of several software components to provide an architecture for *dependable distributed objects* via adaptability and fault tolerance. AQuA is an middleware approach towards fault tolerance especially suited for systems composed of *off-the-shelf* hardware. Being a framework for adaptive fault tolerant system, AQuA is designed to stand against two kinds of changes, changes due to *faults in the environment* and changes in *application's availability requirements*. One of the features of AQuA is that it provides a layer of abstraction where QoS requirements can be specified at the application level.

AQuA is made up of the following software components -

QuO (Quality Objects) Runtime Provides an environment for a programmer to specify *contract states* (QoS requirements) for the distributed application and its objects.

Proteus Dependability Manager This allows the system to be configured in response to faults and availability requests. This is responsible for providing the fault tolerance by dynamic management of replicated distributed objects. It is also responsible for implementing a particular fault tolerant scheme.

Ensemble Group Communication System Provides the group communication infrastructure for the reliable communication between groups processes.

CORBA Gateway Translates between CORBA object invocations generated by application objects and the QuO runtime and messages that are multicast by Ensemble.

Other important issues that AQuA defines are the *fault models* and the *group categories*. AQuA has support to handle three categories of faults, fault by *crashes*, *value* and *time*. And it classifies four categories of groups namely *Replication*,*Connection*,*Proteus Communication Service* and *Point-to-Point* groups.

3 Analysis

The formal approach Egida takes towards protocol composition and synthesis seems insightful. However it is not clear from [RAV99] that whether they had actually developed the compiler for the grammar mentioned or if the grammar is just there as a formal reasoning tool. The section that talks about synthesizing protocol implementations from grammar does not mention whether the process was manual or automated. There was further confusion regarding this as the figures that show the specification of existing protocols, that were supposed to come from the grammar, was in a 'not-so-clear' psuedocode form which seemed to be manually derived instead of automatically generated.

The AQuA framework, especially the AQuA gateway seems a heavyweight software component (filter) in itself that needs to do lot of things including converting all the messages to IIOP format and again deal with issues like voting, monitoring and buffering. Put together with all other overheads from the AQuA runtime and the messaging layers it seems that the whole fault tolerant framework itself would be quite resource hungry. Considering the fact that fault tolerance is an upmost issues in mission critical domains like real time applications, the paper should have justified the overhead incurred would not stand in the way to meet special QoS needs of such applications. Also the system itself seems to consider CORBA applications as the only one that can leverage the functionalities provided by it. It was not quite clear whether AQuA supports distributed application models other than CORBA.

References

[CRS+98] Michel Cukier, Jennifer Ren, Chetan Sabnis, David Henke, Jessica Pistole, William H. Sanders, David E. Bakken, Mark E. Berman, David A. Karr, and Richard E. Schantz. AQuA: An adaptive architecture that provides dependable distributed objects. In *Proceedings of the The 17th IEEE Symposium on Reliable Distributed Systems*, page 245. IEEE Computer Society, 1998.

[RAV99] Sriram Rao, Lorenzo Alvisi, and Harrick M. Vin. Egida: An extensible toolkit for low-overhead fault-tolerance. In *Proceedings of the Twenty-Ninth Annual International Symposium on Fault-Tolerant Computing*, page 48. IEEE Computer Society, 1999.

ICS 243F Paper Summaries for Week 9

Sushil K Bajracharya
sbajrach@ics.uci.edu

1 Summary of paper [CEF]

[CEF] draws some general requirements for modern adaptive mobile applications and presents an architecture that enables efficient adaptation of mobile applications under pervasively changing environments. The commonly practiced approaches of the having the adaptation layer either in application or in the system providing services to applcations are considered inefficient. Even a combined approach of having a middleware deal with adaptation issues is claimed to be inappropriate. The primary reasons behind these inefficiencies being the inability to compose conflicting adaptation policies and absence of proper *control* flows between the application and the system along with the data flows. One example for such a scenario might be making multiple applications with auto save feature aware of each other so that they could cluster their disk access which would give the system enough idle disk time during which the disk can be spinned down saving system power.

To implement such system-wide adaptation policies the architecture [CEF] proposes is divided into two layers - (i) Application(A) and (ii) Middleware(M). There would be four distinct flows of information and control between these two layers -

Requirements flow from A to M set by the application concerning resources or attributes of the system.

Control of Middleware from A to M representing the ability of the application to control the functionality of the system.

Notification from M to A as an adaptation trigger.

Control of Application from M to A representing the ability of the underlying platform to control the application operations.

With these message flows the architecture separates the mechanisms from policies with a common *context space* for handling the adaptation attributes. Other components that make up the architecture are *device monitors, applications and mechanisms*, and *adaptation control and policies*.

2 Summary of paper [ICPW]

[ICPW] describes the overall architecture of a system built as a client-server framework for an adaptive multimedia video application to be used in a dynamically changing mobile environment. The system is based on a layered architecture with four distinct layers -

User This layer allows *presentation specification* that determines what kind of attribute can be scaled in the multimedia application.

Application This layer handles the *scalable content* and deals with the implementation of scaling the content according to appropriate attributes.

Dynamic Library Provides *software feedback* that governs the amount by which the attributes of the multimedia contents can be scaled.

Operating System Handles the network and *device indications* used to monitor the add/removal of hardware and change in network properties.

Lying in the core of the architecture is the methodology based on *quasi-invariants*, *guards* and *intelligent adapatation*. The system implemented is a traditional client-server system with a multimedia video player and a video server with customized operating system and network settings. The player itself is adaptive and includes a *controller* that can control the content being displayed based on *mobility indications* and *user actions*. The client and the server work in a dynamically reconfigurable network. The system works on a modified version of FreeBSD system. One of the key issue is the support for device availability through the *pmid* interface that supports *Physical Media Independence*. Network awareness is achieved using *pulse* processes and enhancements are made to routing policies for more adaptive routing.

The paper furhter discusses experiments and different metrics used in evaluating those experiments.

3 Comments

The two papers summarized above have exactly opposite qualities. Both claim to describe the *architecture* of some adaptive application. [ICPW] describes an actual implementation of the system and is more inclined in describing the implementation details and how the system was set up and tested. It is too specific to a system and one particular domain. From an architectural viewpoint it doesnot describes a general model very well. Considering the paper is rather old the issues it addresses might have been very interesting back then. But features like hot-swapping of hardware devices are common these days. It however touches one intereting issue, that the protocols upon which most of the current networks work might not be suitable to meet the special demands of newer requirements like adaptability and need to be enhanced in order to meet such demands. [CEF] on the other hand gives a very high level generic description for the architecture. As much as the concepts go, it put some fundamental requirements together but the presentation is rather superficial. Had it presented the architecture with some implementation specific details or more coarse grained description for the architecture itself it would have been more contributing. Overall the basic idea behind the papers seem to be these facts -

- For adaptive applications there need to be two way flow of control and data between the application client and the service provider.
- There needs to be a more precise and formal way to reason about the specification and composition of different application parameters that govern adaptabilty to meet efficient QoS (Quality of Service) in adaptive environments.

References

[CEF] Nigel Davies Christos Efstratiou, Keith Cheverst and Adrian Friday. Architectural requirements for the effective support of adaptive mobile applications.

[ICPW] J. INOUYE, S. CEN, C. PU, and J. WALPOLE. System support for mobile multimedia applications. In *Proceedings of the 7th International Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV 97) (St. Louis, Missouri, May 1997)*.