Department of Computer Engineering and Information Technology
Bachelor of Engineering (Honours) in Computer Engineering

PROJECT REPORT

BEngCE-2001/02 - (CS/HWC) - (CS/HWC-201-BECE)

P2P for the Enterprise

Student Name: CHENG Wing Chuen
Student ID: 50186480
Supervisor: Dr. CHUN, Andy H W
Assessor: Dr. PAO, Derek C W

Content

Acknowledgements

By taking this opportunity, I want to thanks all the people who have given me support in my year long project.

I would like to express my sincere thanks to Dr. Andy Chun, my project supervisor, for giving a free learning atmosphere, valuable advices in both project ideas and presentation techniques. His encouraging attitudes toward my self proposed project make me more mature and self confident.

Again, I would also like to thanks Dr. Derek Pao, my project assessor, for his patient listening and teaching from the limiting time, and the various important guidance to the project.

Abstract

Traditionally, the computing systems are usually running by a small number of powerful servers but there is high tendency to implement that platform via the co-operation of a large number of less powerful computers. The reason for the move from centralized hosting to decentralized collaboration is that Peer-to-Peer (P2P) architecture can unlock the potential of unused processing power, provide unlimited storage and most importantly eliminate single point of failure.

The P2P technology is not yet a reality because there is no standard platform for peer-to-peer communication. The objective of my project is to build a P2P framework from a proposed Binding-oriented and Expandable Exchange (BEE) architecture, which mainly focuses on two areas: discovery and collaboration. The architecture gives up the use of complex lookup algorithms but comes out with a simpler solution - bind the peers as an efficient structure so that discovery and coordination among peers can be as simple as directories lookup in file system.

Introduction

Peer-to-peer Internet applications have recently been popularized through file sharing applications like Napster, Gnutella and FreeNet. While peer-to-peer systems have many technical challenges like decentralized control, self-organization, adaptation and scalability, it can be characterized as distributed systems in which all nodes have identical capabilities and responsibilities and all communication is symmetric.

One of the key problems in large scale peer-to-peer applications is to provide efficient mechanism for object location and function invocation within the network. A proposed Binding-oriented and Expandable Exchange (BEE) architecture is presented in this paper to overcome the mentioned challenges. The BEE network is a purely decentralized, self-organizing, fault resilient and scalable system. Whole system is maintained by the interactions of agents, which reside on every node of the network. In order to enable interoperability, the agents require to use SOAP as the communication message. The

asynchronous nature of messaging system provides a robust fundamental platform for all the interactions.

Since BEE architecture is an abstract idea, there requires a concrete implementation of this concept. The BEE framework is written using Java programming language. A set of interfaces is defined to represent all the basic operations that the developers can use. Reference implementation is also created to make the framework workable. The modular design makes the framework more flexible to use by simply replacing different version of implementation.

Finally, a logistics application is built to demonstrate the usage of that framework. The design steps are shown so that a complete view is provided.

Analysis of Current P2P Technology

Before delving into the discussion on searching and discovery in peer based network, it is helpful to understand why it is important.

Current centralized methods of discovery that are acceptable for dedicated servers hosting relatively static content, but it will break down when applied to large peer network. It is because searching the Internet and other large networks is currently a very centralized process. Most of the major search engines reply on very large databases and servers to process queries. These servers and storage systems are very expensive to build and maintain, and often have problems keeping information they containing up-to-date and relevant.

One of the benefits provided by peer based network is the fact that they allow access to all kinds of information and resources which were previously unavailable. This may be files and documents, or computing power for complex computational tasks. Another versatility of decentralized peer network is that more resources can be added by increasing the number of peers within the network. Thus, the value of the network will grow as its popularity increases.

Resource discovery in peer based networks is critical to the value of the network as a whole. Therefore, the ability to locate resources efficiently and effectively regardless of network size is critical to the utility of that network.

Below are some descriptions of the existing peer discovery methods. These are the methods in use today in a variety of designs and architectures and they have various strengths which make them attractive for different circumstances. However, none of them can be suitable for building a large and purely decentralized peer network. They are:

1. Aggressive flooding of queries

2. Selective routing of queries

3. Decentralized hash indexing

4. Centralized indexing

5. Distributed indexing

6. Relevance driven network

1. Aggressive flooding of queries

The original Gnutella implementation is a prime example of a flooding broadcast discovery mechanism. This type of method has the advantage of flexibility in the processing of queries. Each peer can determine how it will process the query and respond accordingly. Unfortunately, this type of method is efficient only of small network.

Due to the broadcast nature of each query, the bandwidth required for each query grows exponentially with a linear increase in the number of peers. Rising popularity will cause the network to quickly reaching bandwidth saturation. This causes fragmentation of

network into smaller number of groups because the method consumes large bandwidth while operating.

Segmentation of network reduces the number of peers visible and the quantity of resources available. Queries must be sent over and over again to try and compensate for the reduced range of queries in a highly segmented network. It may take a large amount of time for a suitable number of peers to be queried, which further reduces the effectiveness.

Problem

This type of discovery mechanism is very susceptible to attack. Malicious peers can send out large number of fault queries, which produce significant load on the network and reduce network efficiency. Furthermore, false replies to queries can stimulate extra traffic and lower the accuracy of searching.

2. Selective routing of queries

Selective forwarding systems are much more scalable than flooding broadcast networks. Instead of sending a query to all peers, the query is selectively forwarded to specific peers who are considered more likely to locate the resource. This approach greatly reduces bandwidth limitation, but it still suffers from a number of shortcomings.

Problem

Due to the fact that a much smaller number of peers receive the query, it is important that all peers be reputable for this type of operation. A malicious peer can insert itself into the network at various points and misroute queries, or discard them completely. Degrading accuracy and relevance would be the results. Any system relies on an open, decentralized network will inevitably run into problems from misuse and malicious activity.

Since each peer must contains certain amount of information used to route or direct queries received, the overhead would be negligible for small network. However, in large network this overhead may grow to an unsupportable level.

3. Decentralized hash indexing

Decentralized hashing system can optimize the ability of locating a given piece of information. Every document or file stored within the system is given a unique hash code, which is used to identify and locate resource. The advantage is that the given key can be located very quickly despite network size.

Problem

This type of system does have several drawbacks which prohibit its use as a robust searching and discovery method. Since data is identified solely by ID, it is impossible to perform a fuzzy or related keyword search within the network. Everything must be retrieved or inserted using an ID.

The second drawback is that a malicious peer may misdirect queries, insert large amounts of garbage data or flood the network with queries to degrade performance.

4. Centralized indexing

Centralized indexes have proven to have good performance for resource discovery from the history. It also enables easy management of resources information. This approach is the most widely used discovery mechanism so far. For example, centralized naming and directory server (MSAD, NDS or NIS), UDDI service.

Problem

The most serious issue is single point of failure. No matter how many mirror backups the system are running, there would be a chance that they hang at the same time.

The inherited problem of mirroring is the cost. Bandwidth and hardware that required to support large network of peers are very expensive. Scaling this kind of network requires substantial capital investment.

5. Distributed indexing

Distributed indexes eliminate the need for expensive centralized servers by sharing the indexing task among peers in the network. One type of distributed indexing is called shared index system. This implies every peer is establishing a small portion of a large index. When designed correctly, this type of network can even out perform the centralized approach in both performance and scalability.

Problem

The most difficult problem with this type of indexing system is cache coherence among all the indexed data. Peer networks are much more volatile in terms of peers joining and leaving the network, as well as the resources contained within the index, than traditional networks. The overhead in keeping everything up to date and efficiently distributed is a major problem to scalability.

Various implementations of shared index systems are built to address the cache coherence problem. However, the kind of distributed indexes system still encounters the inherited problem once any of the situations below occurs:

Number of peers supporting the index network is large
Many peers join and depart the network maintaining the index
The amount of data to be indexed is significant
The meta data for the indexed data is very diverse
Malicious peers participate the shared index

To summarize that large peer number would make the distributed index system incredibly complicated while malicious activities are inevitable. Furthermore, supporting a wide range of meta data can also be difficult. XML can be used to contain this data, but tracking the meta data in addition to keys or names significantly increases indexing overhead. Since each peer must search its section of the index at given time, peer must be able to understand the meta data as it relates to the query that is processing. Therefore, diverse peers may not understand how to interpret the meta data.

6. Relevance driven network

Relevance driven network crawlers is a different approach to the resource discovery problem. Instead of performing a specific query based on peer request, they use a database of existing information to determine the resources that it encounters may be relevant or interesting to the peer.

Problem

Over time, a large amount of information is accumulated which makes the relevance search become too diverse. Furthermore, this type of system lacks the real time query function so that locating specific information is difficult. The support for a wide variety of resources is also missing, because the relevance engine expects only on certain kinds of data it can operate.

Finally, this type of discovery is too slow for most uses. The time required for the crawler to traverse a significant amount of content can be very long. Therefore, relevance driven network crawlers are not suitable for discovery in large network.

Idea of BEE

Binding-oriented and Expandable Exchange (BEE) architecture is a methodology of linking different peers to form a network. The nature of this peer network is purely decentralized and robust to peers failure. Most importantly, resources lookup is a simple operation by navigating across the peer tree.

Design concepts of BEE architecture are mainly classified as the sections below:

Topology Selection

The peer-to-peer trend has renewed interest in decentralized systems. The Internet itself is the largest decentralized computing network in the world. However, most applications developed so far are completely centralized. The growth of the Web means most systems

are single web servers running in expensive collocation facilities. Now with peer-to-peer, there is tremendous need for decentralized architecture.

The debate between centralized and decentralized systems is fundamentally about topology. In other words, how nodes are connected in the system. Topology can be considered at many different levels: physical, logical, connection or organizational.

The BEE architecture constructs the topology at the logical level. Physical level is not suitable because it is too elementary as the widely used Internet protocol - TCP is on top of physical level. Using connection level will cause wasting of network resources. Since the number of peers in the network is very large, connecting each of them with actual protocol connections is not possible due to the limitation from operating systems (OS) socket management application programming interface (API). The traditional client-server model does not require the establishment of a large number of connections to remote server. Therefore, building peer-to-peer framework on general OS requires another level of abstraction.

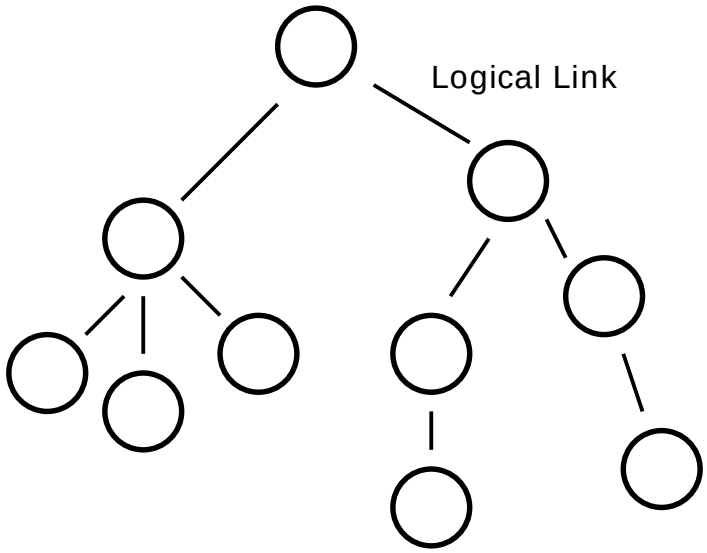
Logical binding is most suitable for BEE environment because it exhibits tight linkage at logical level while maintaining loosely bind at connection level. The connection problem can then be overcome easily.
The shape of the topology is also another major concern. They can mainly divided to ring, grid or tree model. For ring model, a cluster of computers is arranged into a ring shape and coordinates by sharing state information. The resulting group of nodes provides identical function and load balancing capabilities. The ring topology is implicitly assuming the computers are nearby on the network since its scalability is not good. The non-symmetric structure of grid topology makes it exceptionally difficult to manage the coordination among peers. A centralized coordinator is usually used to solve that problem but another serious problem comes out - single point of failure.

The topology that the BEE architecture using is the Spanning Tree topology. The reason is that tree topology is a hierarchical structure, which has proven to be a scalable solution in the long Internet history. The best known hierarchical system on the Internet is the Domain Name Service (DNS), where information flows from the root name servers to all

the member servers seamlessly. Most importantly, the member nodes in Spanning Tree topology are generally few links away from the root, which makes that topology extremely scalable.

The actual topology used in BEE network is a Spanning Tree topology with logical level of binding.

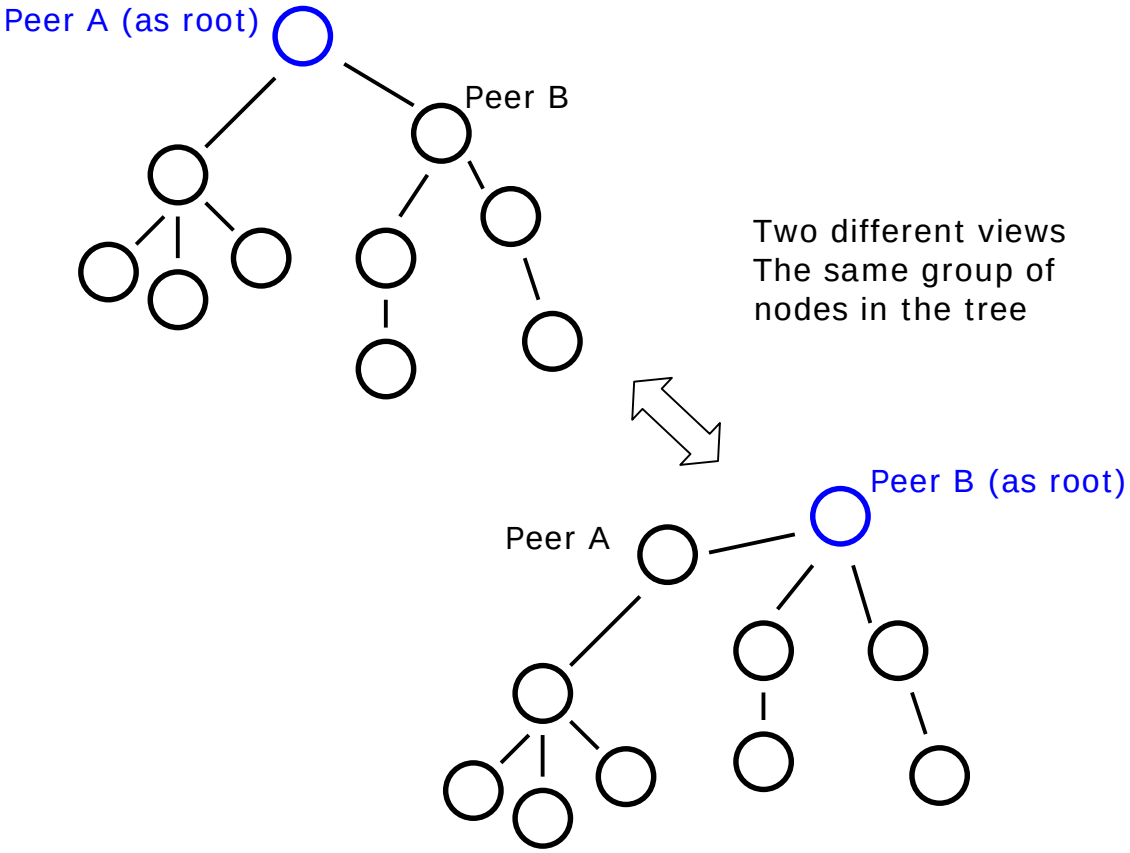
Figure 1. Spanning Tree



Further Discussion on Topology

Within the Spanning Tree, each node can be the root except the view of that root node is changed a little bit.

Figure 2



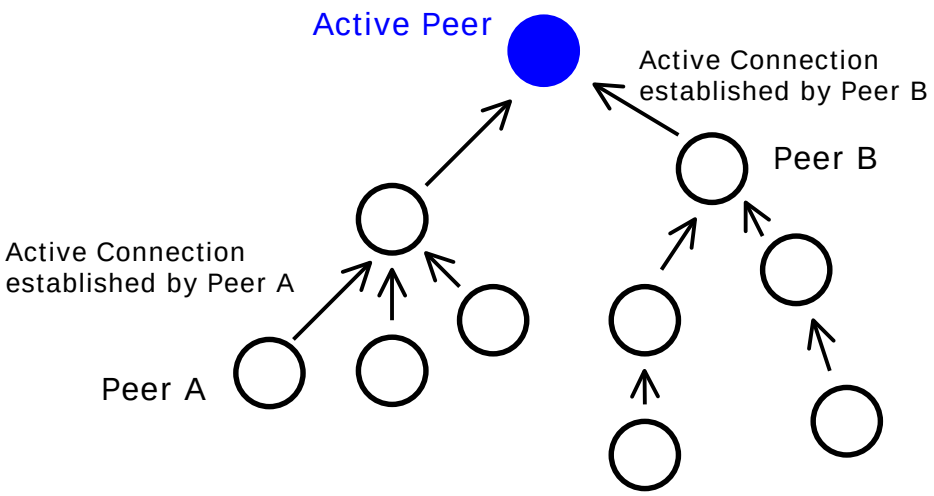
The figure above shows that each peer in the tree can have the same privilege and responsibility, which suit the concept of peer-to-peer computing.

Since each peer in the BEE network responses for the topology construction, the rule of BEE is that every peer must establish one active logical connection to form tree structure while number of backup connections is unlimited. Each peer is responsible for maintaining the availability of its own active connection only. It can neglect the stability of the attached connections that are being initiated by other peers.

It is easy to observe that both trees above (figure 2) have 10 peers and only 9 connections. This implies one peer in each of the above tree cannot establish a logical connection or a closed loop would form which in turn ruin the tree structure.

In BEE environment, the peer that cannot establish connection is called Active Peer because it is always looking for a new peer group to bind. Without the Active Peer, the whole tree of peers can be say "dying". It is because this peer has to actively searching for a new tree of peers for it to bind rather than itself.

Figure 3. Binding View



Linking all peers in the network as a Spanning Tree is just too simple to suit the complex peer-to-peer interaction model. Therefore, a more complicated binding should be introduced.

The 10 bound peers in figure 3 are said to have the 1st level binding. The precondition for them to link together is that they all provide one common subset of service. Furthermore, this tree acts as one entity in BEE environment.

For example, when there are five entities exist in BEE network. There are actually five Spanning Trees of peers. Each peer can participate in more than one tree if it deploy more than one kind of services.

The figure below shows five entities, which are made up from a number of peers:

Entity A made up from 18 Peers

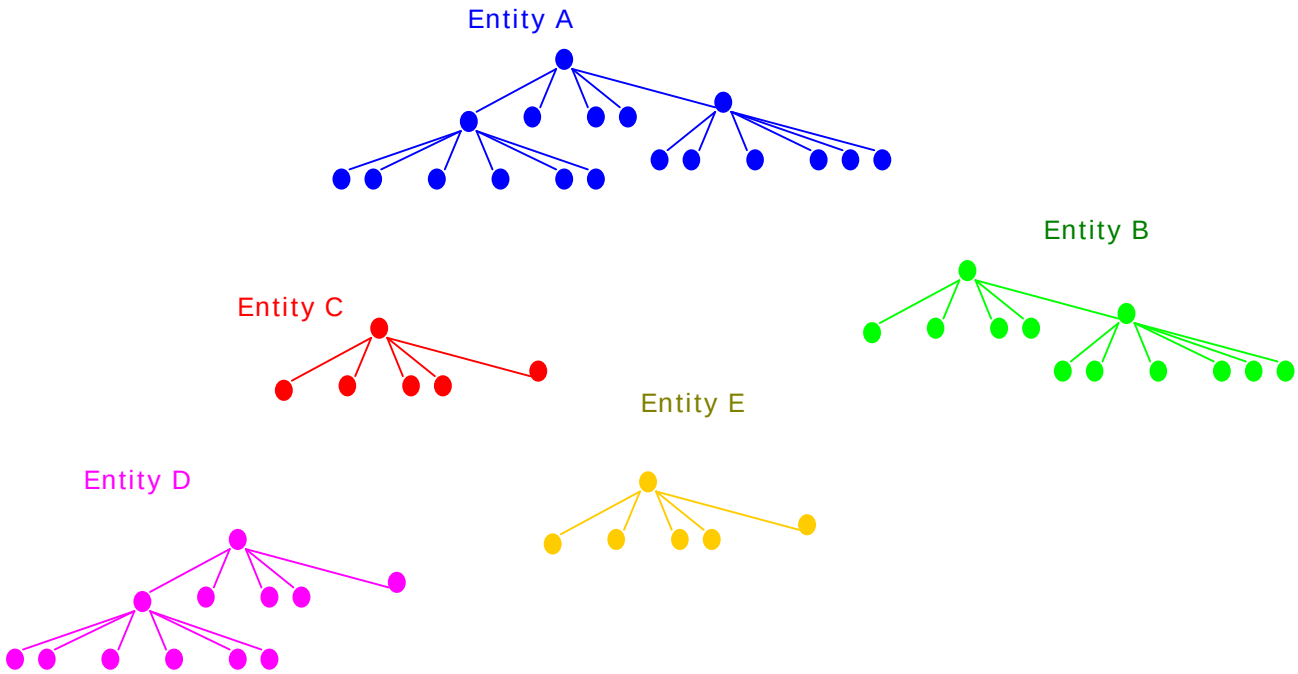
Entity B - 12 Peers

Entity C - 6 Peers

Entity D - 12 Peers

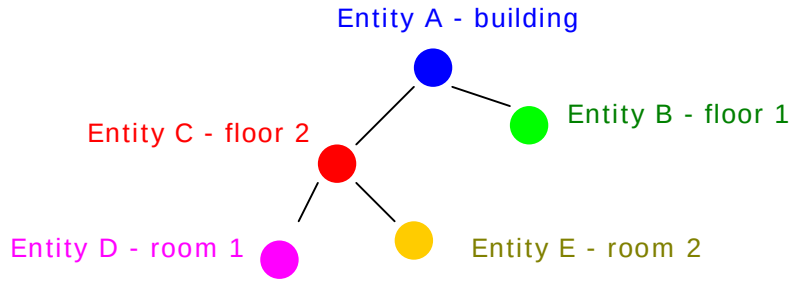
Entity E - 6 Peers

Figure 4. Five Entities Providing Different Service



The above five entities will have 2nd level binding so that a tree of services is formed rather than viewing the entities in plain level. The tree is show in figure 5.

Figure 5. Service Tree



A property management system is used here to demonstrate the usage of Service Tree, which is constructed based on BEE architecture.

From figure 5, it can be assumed that Entity A is a building with two floors. Entity B is floor 1 with no room. Entity C is floor 2 with two rooms. Entity D is room 1. Entity E is room 2. Each entity can be represented by a number of PCs or embedded systems inside the building. They collaborate to form one decentralized system without the need for central coordinator.

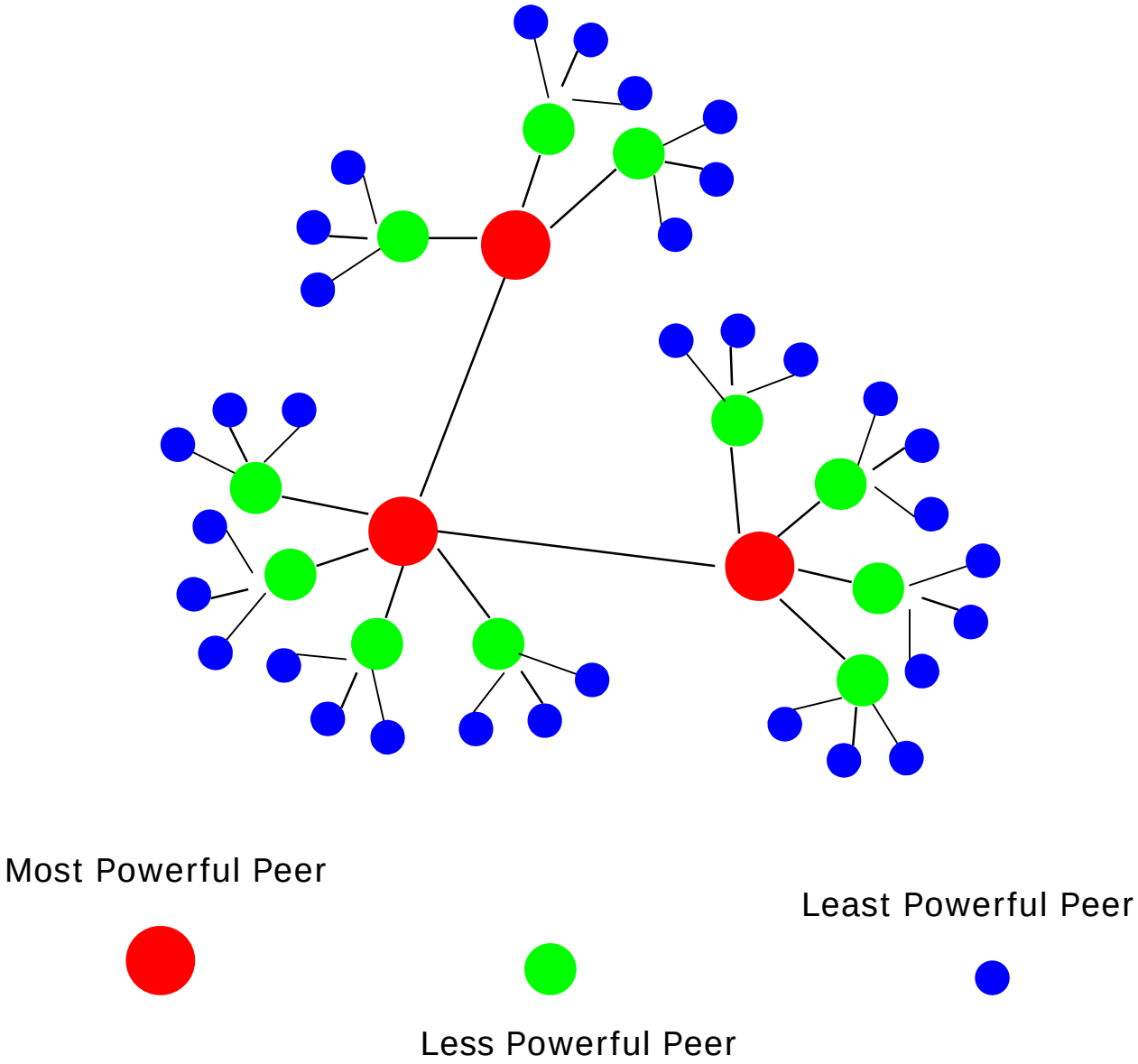
The analysis of topology so far come in a high level point of view. The sections below "Adding Intelligence to the Hierarchical Network" and "Self Organizing Network" will give in depth descriptions of the lower level design concepts of BEE architecture.

Adding Intelligence to the Hierarchical Network

The use of hierarchy to differentiate dedicated peers from client peers. Only the server peers are required to form hierarchical structure and answer to discovery queries. This method can effectively separate peers into two groups. The client peers groups' traffic can be reduced because fewer query flowing among them. For the server peers, since they are divided into different domains within the hierarchy. The discovery queries of the same type of resource are restricted to flow inside that logical domain. Therefore, the traffic of the network is optimized in BEE environment by placing a large number of peers into different namespaces.

The intelligence of the network can further enhanced by classifying peers according to their bandwidth and processing power. Peers with more resources can act as super peers, which in turn act as the hub to allow less powerful computers to attach.

Figure 6. Adaptive Binding



Connection profiles are implemented to favor higher bandwidth connections over slower modem connections so that slow users are pushed to the outer edges of the network. Performance bottleneck to network communication is prevented.

Self Organizing Network

For a large and pure decentralized network, self-configuring and adaptation features must be present in order to make it scalable.

As mentioned in section "Topology selection", there requires a logical connection between each peer inside one BEE entity. All peers can directly control which neighbour peers they communicating with, how bandwidth is consumed and the quality of service level required. This interaction model provides powerful abilities to resist abuse of network resources. Allocating bandwidth according to users' preferences which allows optimizations of the discovery process.

Since each logical connection is long lived than typical direct connection, frequent polling of the connection is necessary to ensure the availability and stability of remote peer. These connections can be re-established when peer joining and leaving the BEE environment. They persist as long as the peers agree to communicate.

The long live of logical connections allows peers to maintain a history of their interaction with each member peers which in turn used for reputation management and optimization of discovery processes. Peers can detect malicious activities and immediately terminate the connection so that response to any type of attack can be in real time. The time of response to any query to remote peers can be recorded. The discovery mechanism can be optimized by first sending message to the peer that gives faster responses in the past.

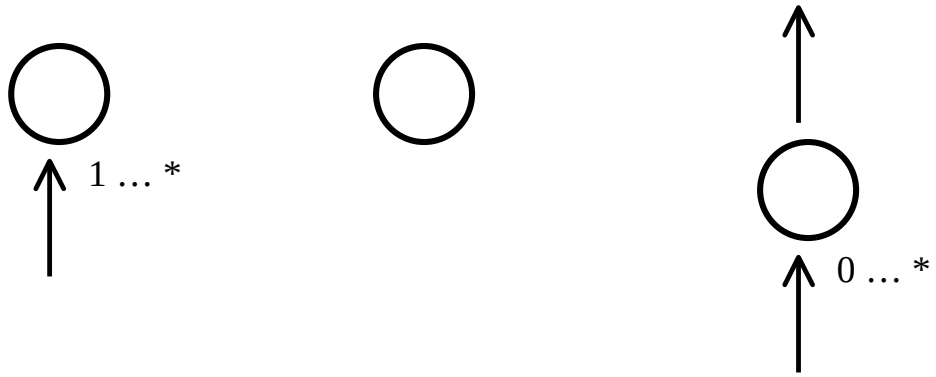
Interoperability within a large network is critical to its success. It is because there are generally more than one protocol present and they are usually made interoperable by using middleware. Self-organizing peer-to-peer network would not be possible if middleware or proxy is inserted between communication. Simple Object Access Protocol (SOAP) is the

standard and low overhead XML messages used to form the foundation of peer communication platform.

The collaboration of objects inside BEE network is done by using the concept of agent interactions. Since each peer can expose its services to different namespace, individual handling for each service object is necessary to make them self-organizational. The most elementary operation that those agents do is to maintain the integrity of the Spanning Tree topology.

The intelligence of the agents is specified by the following rules:

(1) Root Node State (2) Active State (3) Binding State



The three states here are used to represent the states that the agent would have during its operation.

Rule 1: Agent in all the three states above can be bound by any number of agents, but it can initiate one logical bind to other agent only when it is in state (1) and (2).

Rule 2: Agent in both state (1) and (2) tends to be state (3). While agent in state (3), it would try to maintain at its current state.

Rule 3: Each agent is responsible for maintaining the stability of the logical connection that initiated by itself. The condition of the bound connections that initiated by the others need not be considered. Those bound connections would be discarded automatically if they are lost at runtime.

Rule 4: When there is lost in connection availability, the agent will loop infinitely until it can bind to suitable place.

Rule 4.1: Finding the similar objects in the network by using a discovery interface. The concrete mechanism of this interface depends on the implementation of framework.

Rule 4.2: Filter the found objects such that binding to them would not cause closed loop in the tree topology. This is done by forwarding message to the root node to check if there is any loop back before hitting the root.

Rule 4.3: After gaining a set of suitable destinations, a number of binding selection criteria are applied to them. For example, connection bandwidth and processing power from the remote peer's profile. Or number of hops to root, which is detected at runtime.

Rule 4.4: Bind to the desired object. The challenge here is that the binding action must be atomic as problem will occur if two objects try to bind to each other simultaneously. Since two distributed objects can not synchronize their operation by executing instruction in CPU level, a remote locking interface is used to solve that problem. The concrete mechanism is also depending on the framework. After acquire lock on the remote object, the binding action can be done successfully.

Rule 4.5: After the completion of binding, the agent changes to state (3). i.e. the Binding State.

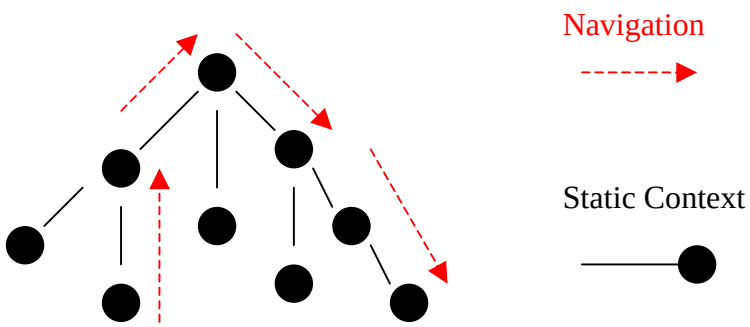
In conclude that the interactions among peers in the network map closely to the actual interaction that occurs between people in the society. Groups of peers - Entity providing similar services will bind close to each other as they would in physical world. Continuous interaction among peers as long as they are joining the BEE environment. By taking advantage of this style of interaction, the quality, performance and flexibility required for decentralized resources management in large peer network can be implemented successfully.

Sharing of Responsibilities among Peers

One of the most important design concepts of BEE architecture is special consideration of Separation of Concern (SOC). The BEE focuses on topology structure, how intelligent and self-configurable it is. Therefore, it promises for simple discovery mechanisms. This architecture tries to facilitate the resources locating processes, but the question now is how does the actual mechanism come out?

Binding-oriented and Expandable Exchange (BEE) is indeed a static environment such that it does not provide any routing or analysis to the incoming queries. The static nature of BEE network makes it robust when facing heavy network load.

The dynamic behavior is residing on the user peers' side. The discovery mechanism is done by navigating along the BEE network. How the methods do peers using to traverse along the topology leaves to the user peers' logic. This kind of Sharing of Responsibilities make the whole peer-to-peer network more scalable as users are quietly contributing their power to the network.



The static context can be thought as the road, highway, tunnel or overpass in our society while the dynamic action can be thought as the cars. The cars can travel everywhere, but the path is selected by the decision of driver. Since all the decision makings are done by driver, there would not be any overhead to the network even the deciding process is very complex as it is done locally by the driver.

Hierarchies of services are supported by distinct types of peers. In many cases, user peer will use various types of resources on the network. While some peers with high processing power are good for providing calculation intensive services, it may be average in providing storage area. For this reason, hierarchical structure can be used to differentiate their responsibilities efficiently. If peers are dedicated for specific jobs and they are linked together with respect to certain structure, the whole system can then be optimized for highest performance.

Conclusion of BEE Design

The BEE network architecture is a proposed solution to the future computing that requires free utilization of resources in network. The interactions of the agents inside the network map closely to the actual interaction occurred in the real world.

Peers groups providing similar services are bound together so that a more powerful entity is constructed. A more complex structure can be formed by binding the entities to form a Service Tree (figure 5). The system is based on a self-organizing infrastructure so that continuous interaction among peers is required. Furthermore, the intelligence of the network can be enhanced by programming the behavior of agents, which can further resist the malicious attack and network usage efficiency.

Peer-to-Peer and Web Services

Developers are always looking for a programming environment that is homogeneous, reliable, secure and de-centrally managed. However, current distributed computing is concerned with collaboration, data sharing and new modes of interaction that involve heterogeneous resources. The result of interconnecting systems both within or across enterprises does add complexity to the existing infrastructure. Companies are now realizing that they can achieve significant cost savings by standardizing on one computing standard.

Peer-to-peer technology supports the sharing and coordinating of diverse resources in network, which are geographically distributed, so that they can be integrated to deliver desired performance.

Integration of peer-to-peer concept and Web services standards is the natural fit for future network infrastructure.

Web Services

Web services describes an important emerging distributed computing standard that differs from other approaches such as DCE, CORBA, DCOM and Java RMI in its focus on Internet based standard XML to address heterogeneous computing. Web services define a technique for describing software components, methods for accessing them and discovery methods that enable the identification of service providers. It is programming language and operating system neutral.

The reasons for Web services to be the heart of the next generation of distributed systems are:

Interoperability: Any Web services enabled applications can interact with each other. This new standard protocol is supported by all major vendors. Conversion between CORBA, DCOM is available freely. Because Web services can be written by any programming

language, developers do not need to change their development environments in order to produce or consume it.

Ubiquity: There is no restriction on the communication protocol used for Web services. Therefore, any device that connecting to the Internet can both host and access Web services.

Easy to Entry: Developers can adopt that technology conveniently because many free toolkits from vendors like SUN, IBM and Microsoft allow them to quickly create and deploy Web services. Wrapper toolkits are also present to allow the existing COM or JavaBeans to be exposed as Web services.

Web services standards are being defined within the W3C and major industry initiatives. This is one of the standards that is particularly concerned:

Simple Object Access Protocol (SOAP)

It provides a means of messaging between a service provider and a service requestor. SOAP is a simple enveloping mechanism for XML payloads that defines remote procedure call (RPC) or messaging convention. Furthermore, SOAP is independent of the underlying transport protocol. SOAP payloads can be carried on by HTTP, FTP, Java Messaging Service (JMS) or even Jini. SOAP is just one means of formatting a Web service invocation.

SOAP is an extension of XML. It is generally accepted standard for Web services. The reason for SOAP's widespread adoption is its simplicity. SOAP is lightweight that involves relatively small amount of code and easy to understand. The basic item of transmission in SOAP is the SOAP message, which consists of mandatory SOAP envelope, an optional SOAP header and mandatory SOAP body.

SOAP Envelope: The envelope is the top element of the message. It specifies two elements, which are XML namespace and encoding style. The XML namespace is a collection of names that can be used in XML element

types and attribute names. In other words, it is an XML schema. Encoding style identifies the data types recognized by SOAP message and specifies rules for how these data types are serialized for transport across the Internet.

SOAP Header: As mentioned before, header is optional. The header element extends the SOAP message in a modular way. It is important to understand that a SOAP message travels from a client application to the final destination would potentially passing through a set of intermediate nodes along the message path. Each node is an application that can receive and forward SOAP messages. The SOAP header can be used to indicate some additional processing at a node that is the analysis independent of the processing done at the final destination.

SOAP Body: It contains the main part of the SOAP message. The service provider at the destination needs to understand the request and take appropriate action. The value type of this body message can be defined by using WSDL. The return message body's format must conform to the agreed type.

Web Services Description Language (WSDL)

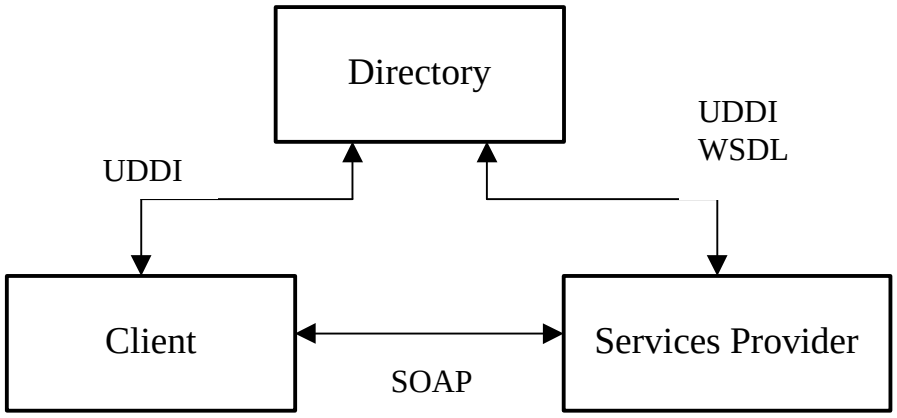
WSDL is an XML document for describing Web services as a set of endpoints operating on messages containing either messaging or RPC payloads.

Universal Description, Discovery and Integration (UDDI)

The UDDI specifications define how to publish and discover information about Web services in UDDI conforming directory.

Combining the use of SOAP, WSDL and UDDI, a standard Web services application can be built.

Architecture of Web services

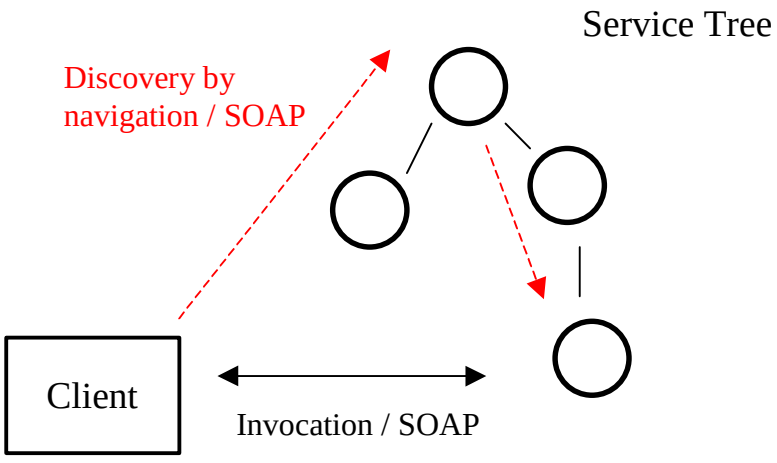


Integration of P2P & WS

The enterprise IT infrastructures are increasingly concerned with the creation, deployment, management and application of dynamic assembled resources and services. With SOAP as the communication protocol, this can set the base for all the future developments. In addition to the use of peer-to-peer concept as a means of collaboration platform, the potential of the network can be revealed.

The figure below shows the concept of integrating BEE and SOAP:

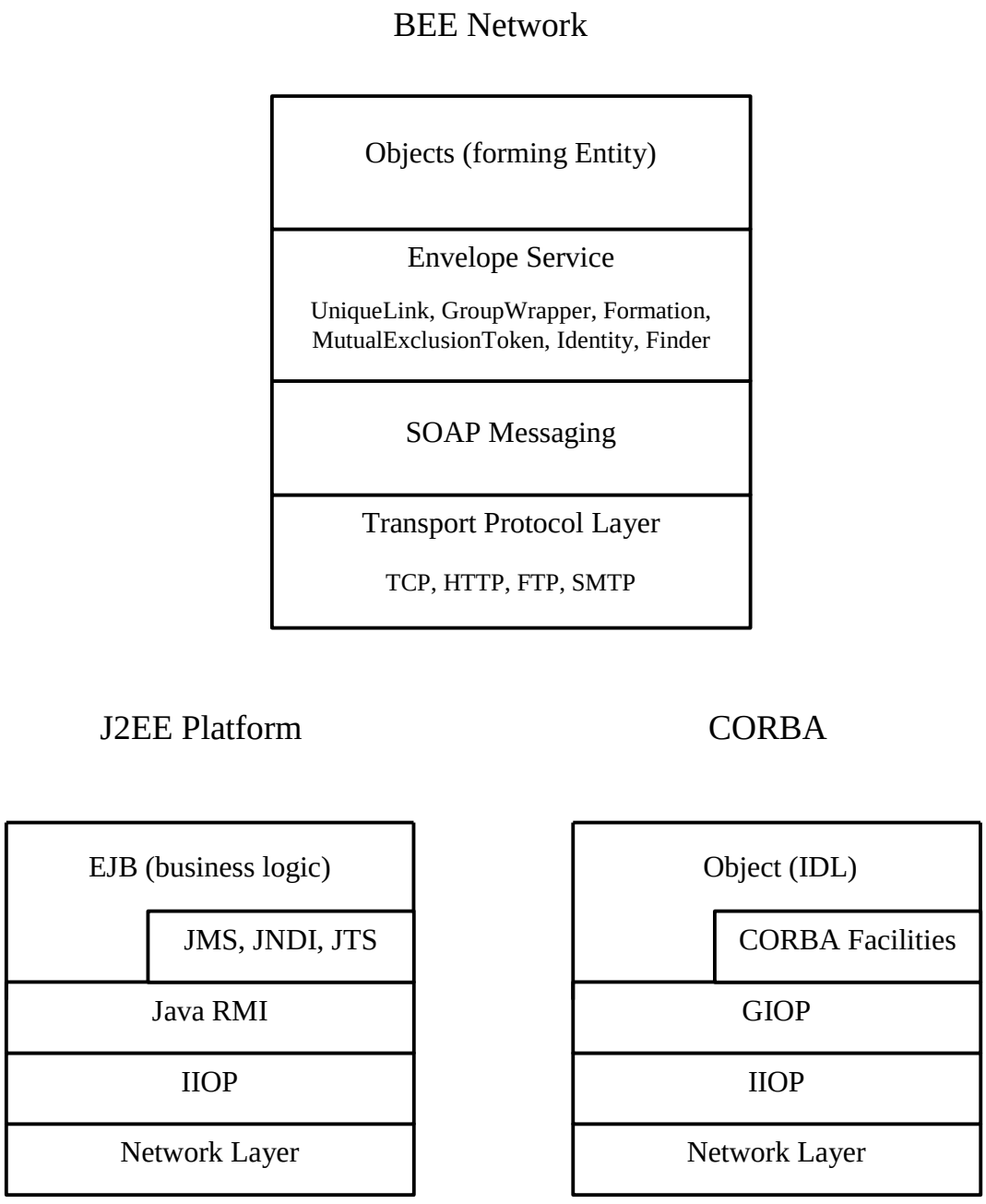
BEE Network using SOAP



Platform Architecture

To summarize that the BEE framework is built based on SOAP messaging layer. The SOAP layer can in turn transported by any protocol. However, the design concept at the object level has not discussed yet.

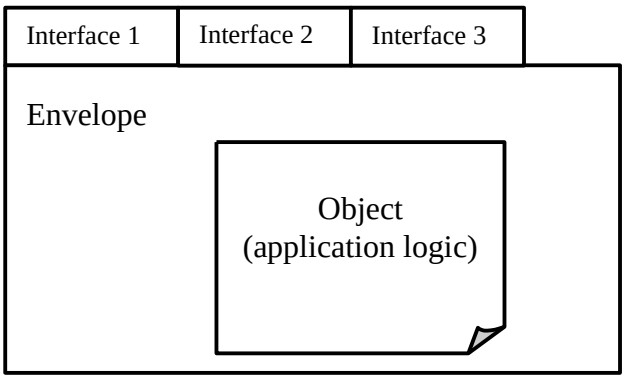
The figure below shows the various high level view of platform:



The fundamental difference between BEE and the two multi-tier systems (e.g. J2EE and CORBA) is there present a SOAP message layer. This messaging layer is designed to integrate tightly with the framework. Therefore, the whole system is independent of underlying network protocol. The problem of heterogeneous components is solved. Thus making all the resources interoperable, which is the basic requirement for peer-to-peer network.

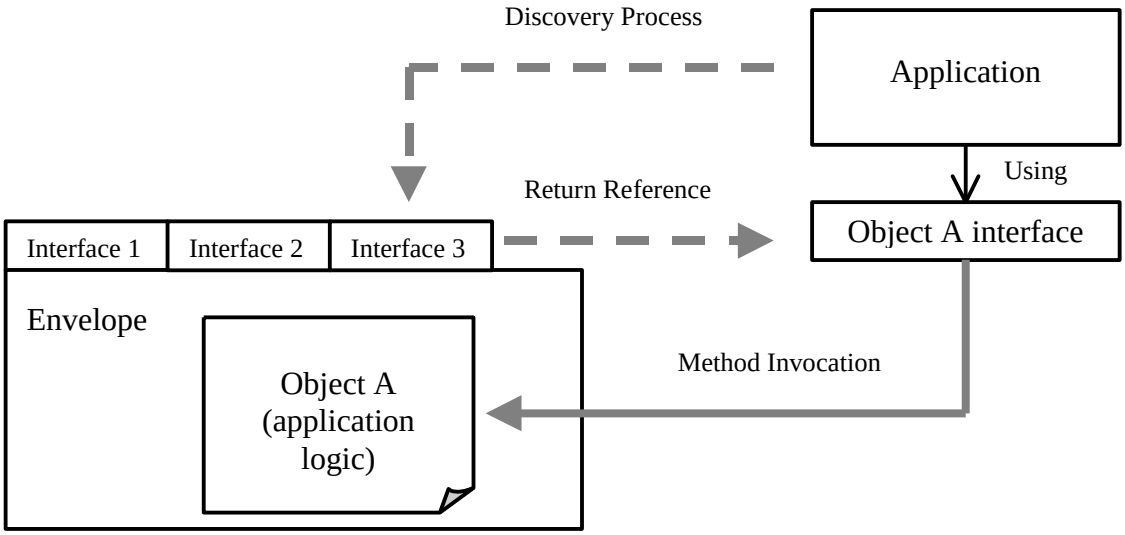
Just above the messaging layer, there are some higher level of functions provided by the framework and they are called Envelope Services. With these envelopes, programming tasks can be made simple. For example, the discovery process is wrapped through by the Finder envelope. Each envelope is associated with specific object. Therefore, a subset of the envelopes provided by framework can be selectively deployed during startup. The envelope can be used by the local object or referenced by remote object.

Relationship of Object and its Envelope



The number of interfaces that the envelope has depends on how many envelope services are deployed. Direct interaction with the application object is not possible without the help of the envelope service. For example, the name and full address in BEE environment of the object is supported by deploying the Identity envelope service. The detail of each envelope service will be explain in the later section.

Interaction of Objects inside BEE Network



Suppose the Interface 3 is the interface exposed by Identity envelope service. The object discovery is just a simple ping pong mechanism. However, when creating the application object, no coding effort is required to implement the concrete ping pong operation. It is because the framework provides this identity service by default.

The separation of application logic to the coordination code is one of the strength of the framework. The coordination code can be implemented in the form of envelope services. Therefore, the capability of the framework can evolve with time.

The dynamic deployment of envelope services makes the framework more flexible and, most importantly, more extensible. Since some envelope services are not used quite often in certain kind of application, removing them can increase the overall efficiency of the system. Each type of application can be optimized by customizing the number of envelope services associated with it.

The above descriptions are just like the Use Case analysis of the framework. The following section will give an in-depth description of the class design.

Framework Design (Java) and Specifications

The objects in BEE environment are defined by using XML. Therefore, a mapping between Java and XML must present. This Java framework is mostly defined in form of Java interfaces. A default reference implementation is also created so that a workable environment can be formed.

Package Description

The BEE framework is mainly composing 6 packages of Java interfaces and classes. They are:

Package - bee.view;

It is the main package that all the core components are defined in it. The interfaces here represent the abstract idea that above the SOAP messaging layer (as stated in section Platform Architecture).

Package - bee.tunneling;

This package defines the objects that work below the SOAP layer. They are usually concerning the network connection code.

Package - bee.lookup;

The package defines the interfaces that are in the Envelope Service layer. For example, the Identity service that gives the complete network address to an object in BEE environment. Finder service gives abstraction to discovery mechanism. The Formation is to provide a simple interface that can wrap the functions of the agents that are doing binding operations.

Package - bee.sync;

This package is going to contain all the distributed coordination and collaboration functions provided in the framework. Currently, there are two distributed programming interfaces. One is to provide distributed mutual exclusion inside peers group. The second is the ability to introduce synchronization across two distributed objects. This is done by using locking mechanism.

Package - bee.replication;

It currently contains one interface that can wrap a group of identical objects. Treating multiple objects as single object can provide load balancing and fail over handling capabilities.

Package - bee.deploy;

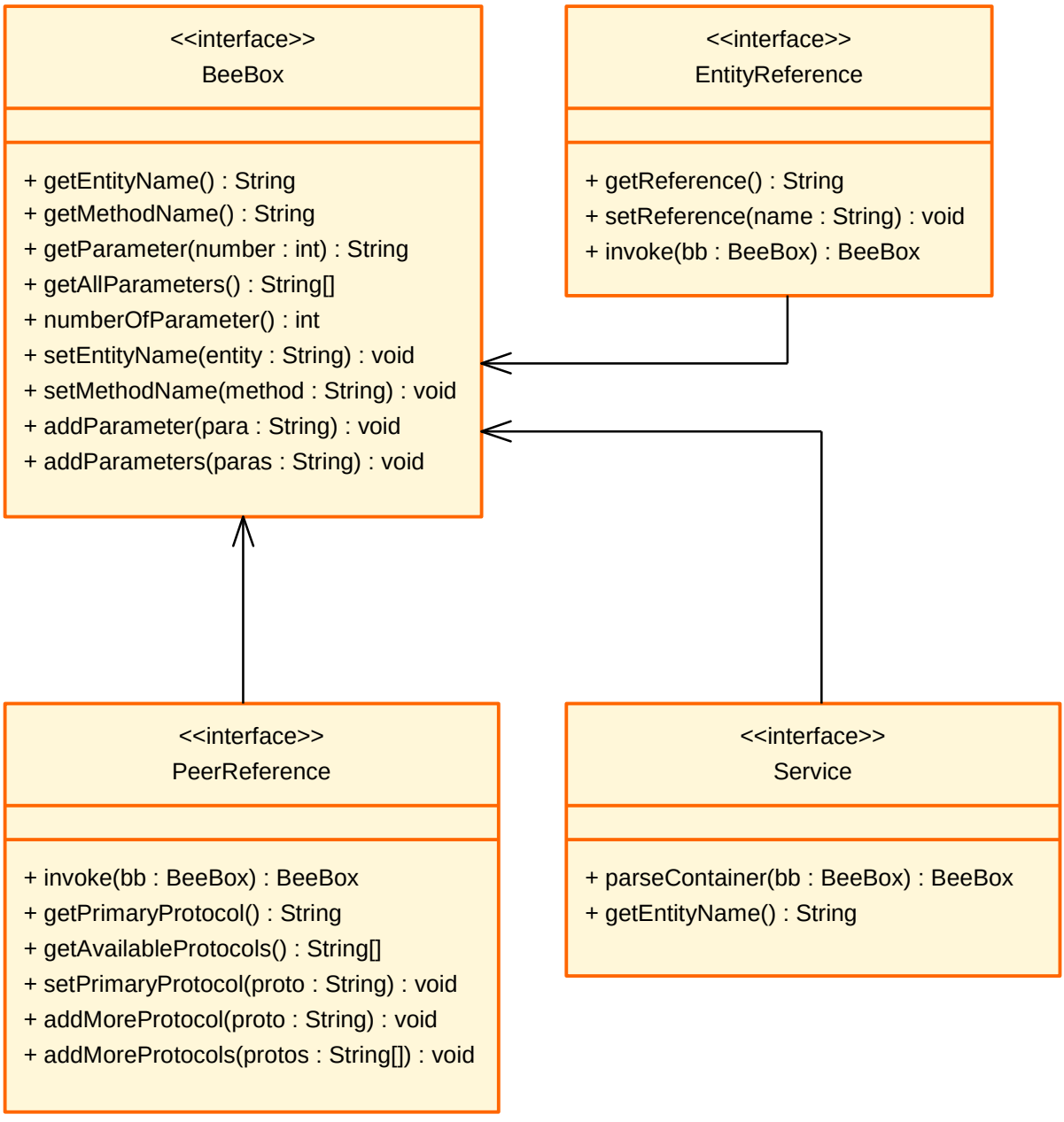
The package mainly contains classes that are used to provide runtime support and deployment selection functions. Furthermore, a compiler written in Java is also included in it. This compiler can create remote object reference from pure Java interface. Therefore, an application can transform itself into distributed services by simply passing all its interfaces to the compiler and generate a set of remote references.

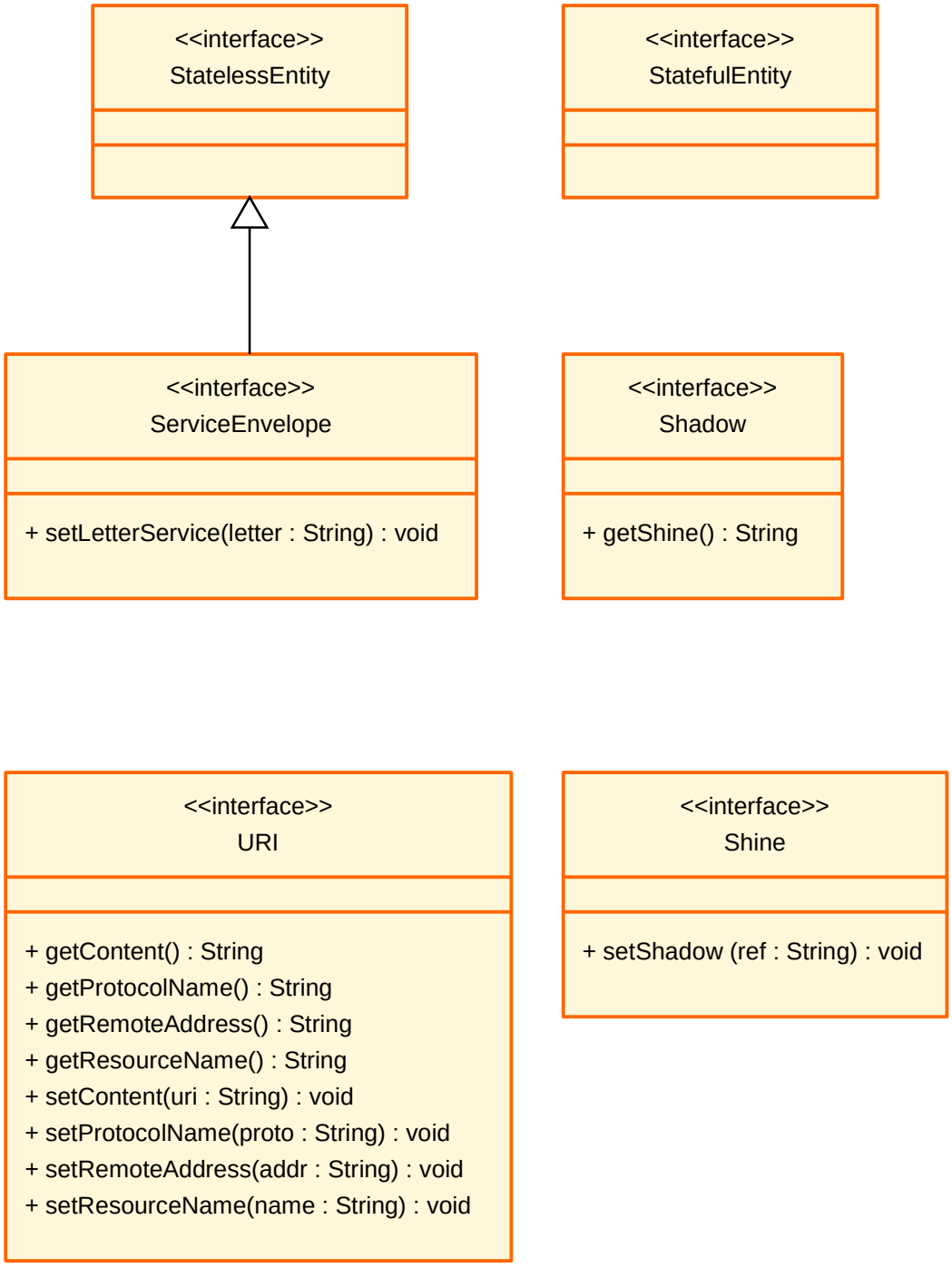
The framework has provided the function to search for the required object that is represented by those remote references. Any program can make use of the services across the network by simply getting a complete set of lightweight remote reference objects while the concrete implementation codes are reside remotely. No version upgrade is required even if the remote service changed its implementation as long as the object interface has not changed.

More than one file is generated by the compiler when passing in a Java interface. The second one is the template for creating the service. The template includes all the method names and the necessary conversion code for method calling of this object.

Class Diagram

Package - bee.view;




<<interface>>
StatelessEntity
<<interface>>
StatefulEntity
<<interface>>
ServiceEnvelope
+ setLetterService(letter : String) : void
<<interface>>
Shadow
+ getShine() : String
<<interface>>
URI
+ getContent() : String
+ getProtocolName() : String
+ getRemoteAddress() : String
+ getResourceName() : String
+ setContent(uri : String) : void
+ setProtocolName(proto : String) : void
+ setRemoteAddress(addr : String) : void
+ setResourceName(name : String) : void
<<interface>>
Shine
+ setShadow (ref : String) : void

Interface function

BeeBox

BeeBox interface encapsulates all the actions that can be carried out on the SOAP message. Since multiple APIs for SOAP can be used, this interface provide a standard across the different brands of product to be used in the BeeNetwork.

EntityReference

EntityReference interface is used to wrap the remote peer's service into a local object interface.

PeerReference

PeerReference contains the information used to interact with a peer at low level. It manages the pairing of abstract BeeEntity reference and the protocol URI.

Service

The Service interface defines the methods that all the servicing objects should implement. The methods define the invocation and naming query interface. It is the super interface of all the interfaces that want to provide services.

ServiceEnvelope

ServiceEnvelope is a kind of service that wrap an ordinary service. It is mainly used by the framework.

Shadow

Shadow interface is implemented by all the logical reference in the BeeNetwork. It is used for getting the reference string.

Shine

Shine interface is implemented by all the logical reference in the BeeNetwork. It is used for setting the reference string.

StatefulEntity

This interface is implemented by the object that wants to provide stateful service. It is used as identification by the compiler.

StatelessEntity

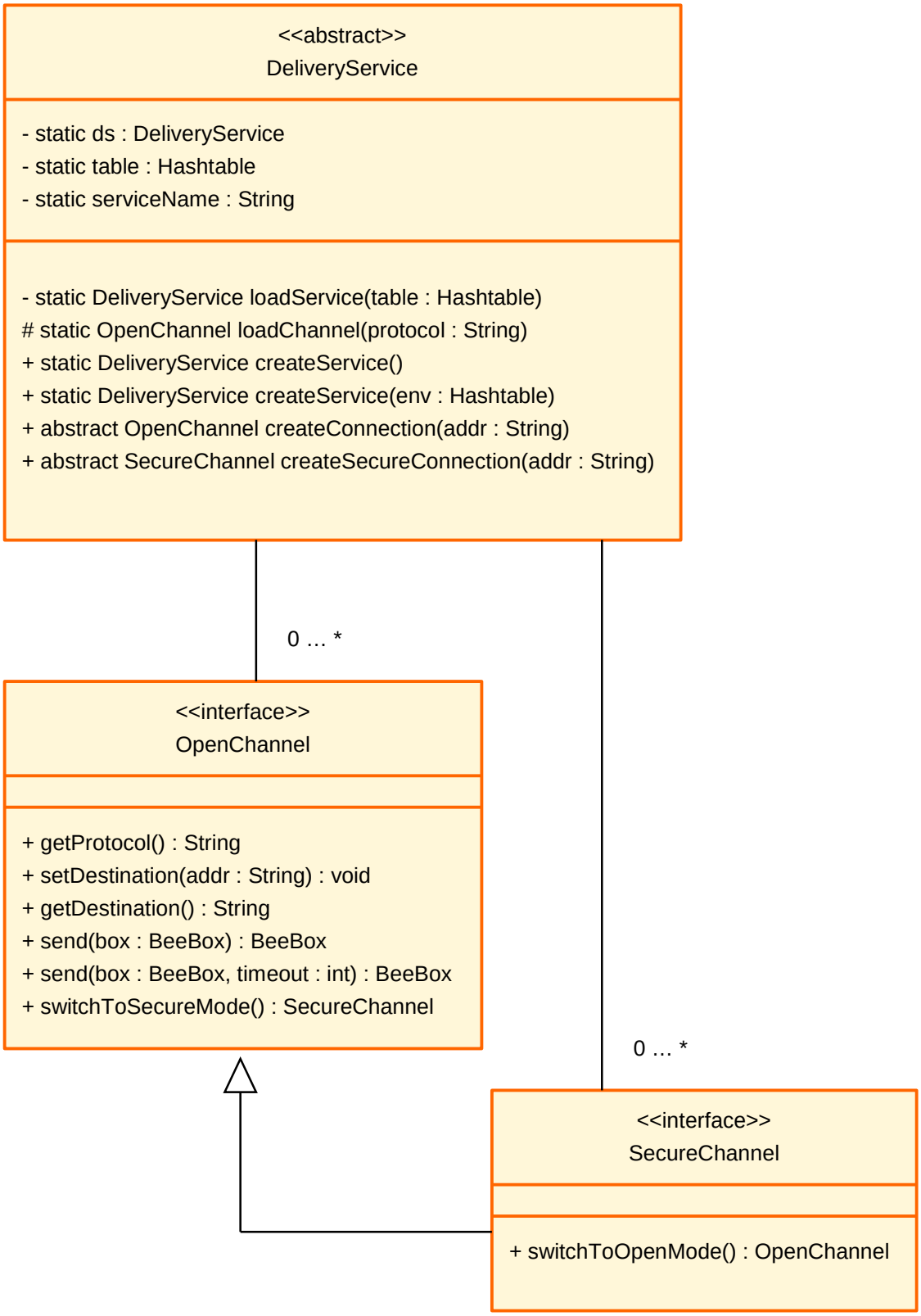
This interface is implemented by the object that wants to provide stateless service. It is used as identification by the compiler.

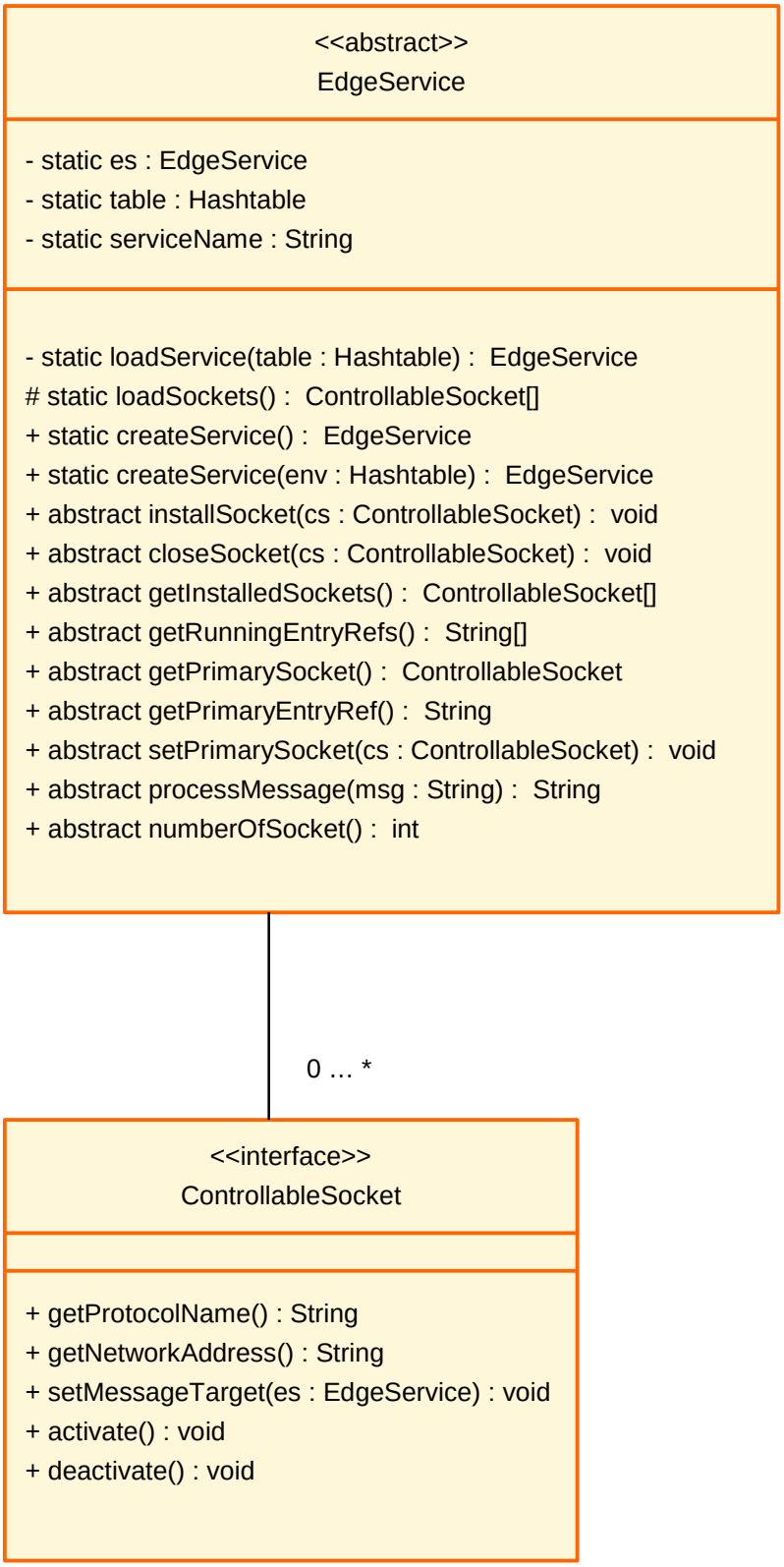
URI

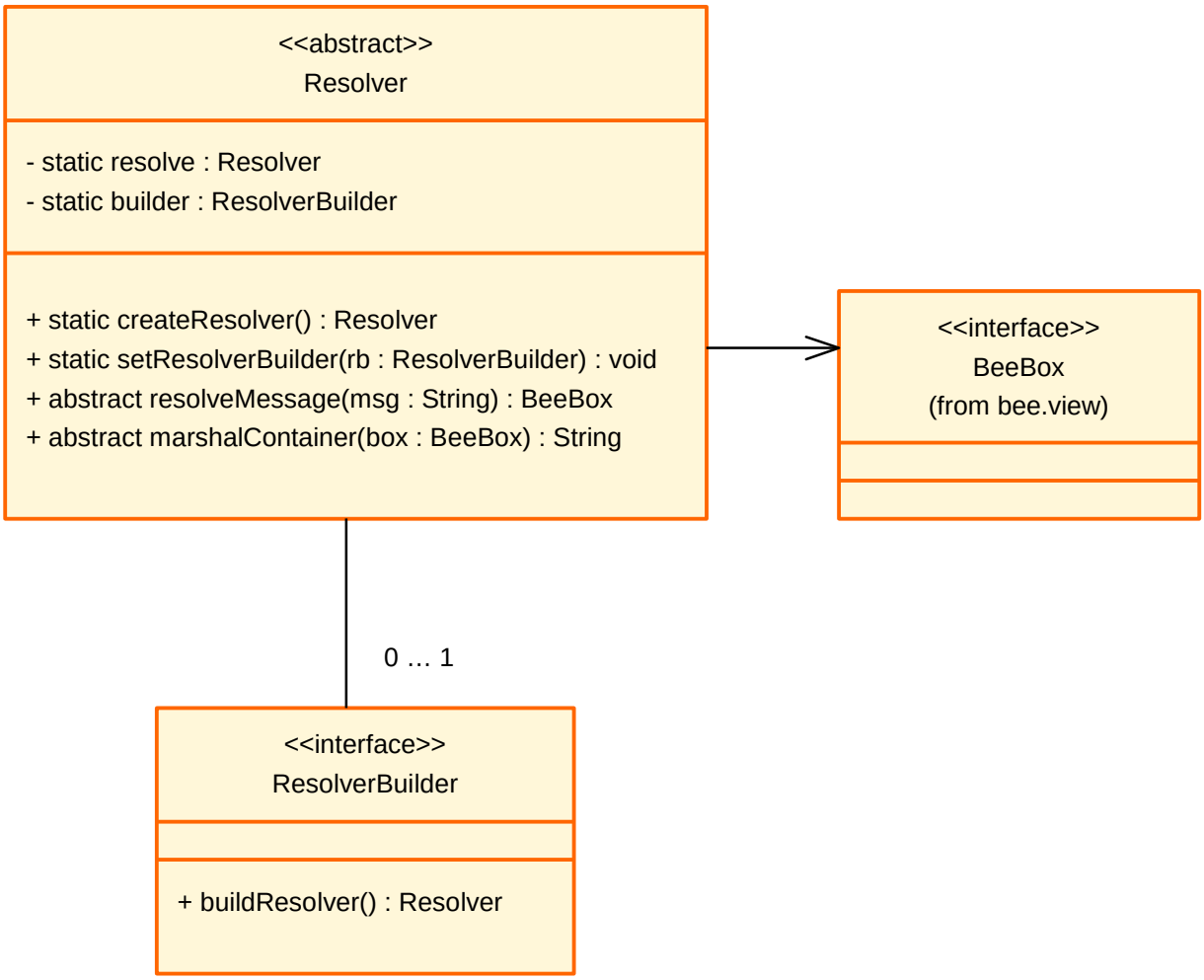
URI interface describe the basic function that an URI should have. Any URI provides the information of describing access mechanism, address of the remote host and the name of the required resource.

Please see the appendix for more detail of classes and methods description

Package - bee.tunneling;




<<abstract>>
EdgeService
- static es : EdgeService
- static table : Hashtable
- static serviceName : String
- static loadService(table : Hashtable) : EdgeService
static loadSockets() : ControllableSocket[]
+ static createService() : EdgeService
+ static createService(env : Hashtable) : EdgeService
+ abstract installSocket(cs : ControllableSocket) : void
+ abstract closeSocket(cs : ControllableSocket) : void
+ abstract getInstalledSockets() : ControllableSocket[]
+ abstract getRunningEntryRefs() : String[]
+ abstract getPrimarySocket() : ControllableSocket
+ abstract getPrimaryEntryRef() : String
+ abstract setPrimarySocket(cs : ControllableSocket) : void
+ abstract processMessage(msg : String) : String
+ abstract numberOfSocket() : int
0 ... *
<<interface>>
ControllableSocket
+ getProtocolName() : String
+ getNetworkAddress() : String
+ setMessageTarget(es : EdgeService) : void
+ activate() : void
+ deactivate() : void



Interface function

ControllableSocket

ControllableSocket interface defines the behavior that a server socket should have. The socket can be any protocol implementation.

DeliveryService

DeliveryService is a singleton class that manages all the low-level transportation mechanism of the SOAP message. It does so by using the OpenChannel object, which responses to handle the actual mechanism.

DeliveryService

EdgeService is the super class of all the monitors that take care of the control of server side sockets. It attempts to load the default reference implementation of EdgeService from the system properties.

OpenChannel

The interface that implemented by all the non encrypted connection. It defines the operations that a public channel should have.

SecureChannel

It is an extension of the OpenChannel interface. The differences are not noticeable to the programmer, but the underlying transfer mechanism provides encryption.

Resolver

The Resolver is responsible for translating the input message into standard BeeNetwork function call interface BeeBox. Since Resolver is replacable, the message that the framework can handle can be changed easily. Or faster SOAP parser can be installed quickly.

ResolverBuilder

ResolverBuilder interface is implemented by the builder object for the Resolver class. It defines only one creation method for Resolver.

Package - bee.lookup;

Interface function

Identity

This is one of the interfaces that supported by envelope service. The envelope service that implements this interface would wrap the identity of an object in BEE environment.

Formation

Define the basic messages that the BEE network required to build up its topology.

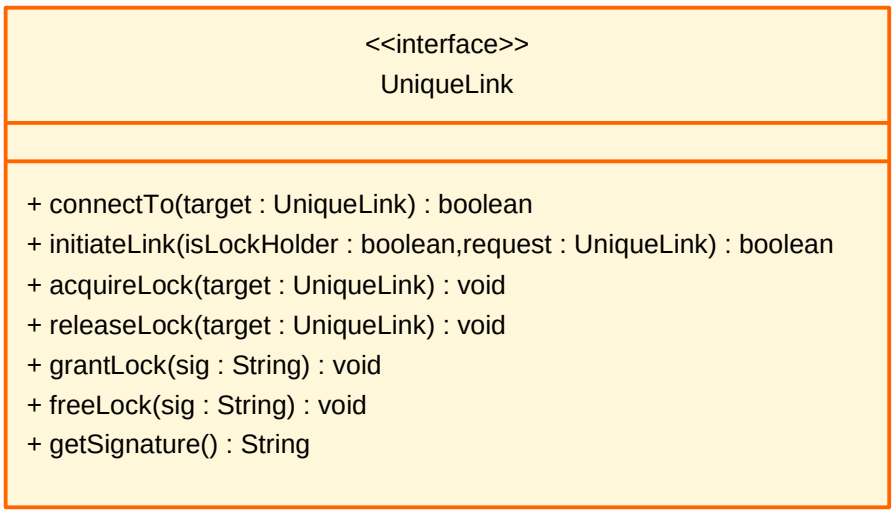
Finder

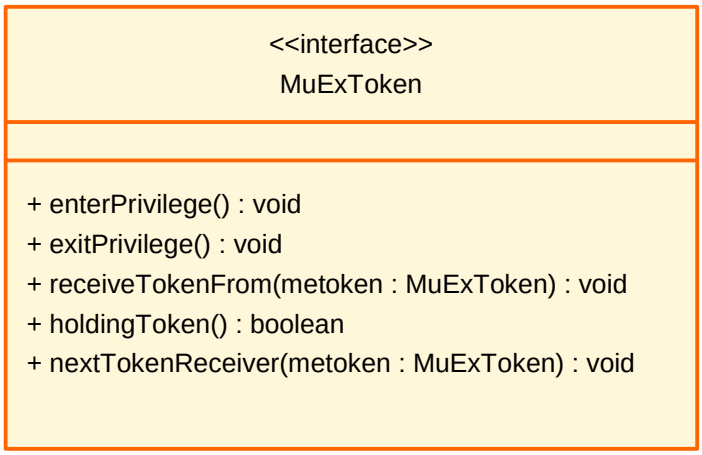
Finder interface encapsulates the logic in creating the discovery process.

Folder

Define the lookup service that the framework provides.

Package - bee.sync;





Interface function

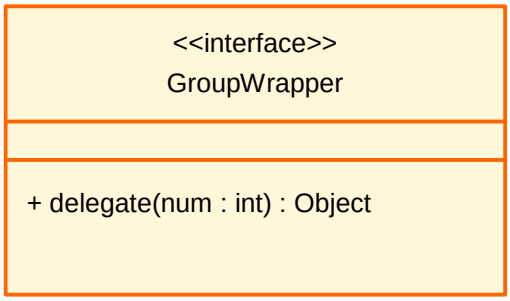
UniqueLink

UniqueLink is an envelope service. It allows the control of single and unique connection to the target peer.

MuExToken

MuExToken is an envelope service. It allows the use of distributed mutual exclusion within the target group of peers.

Package - bee.replication;

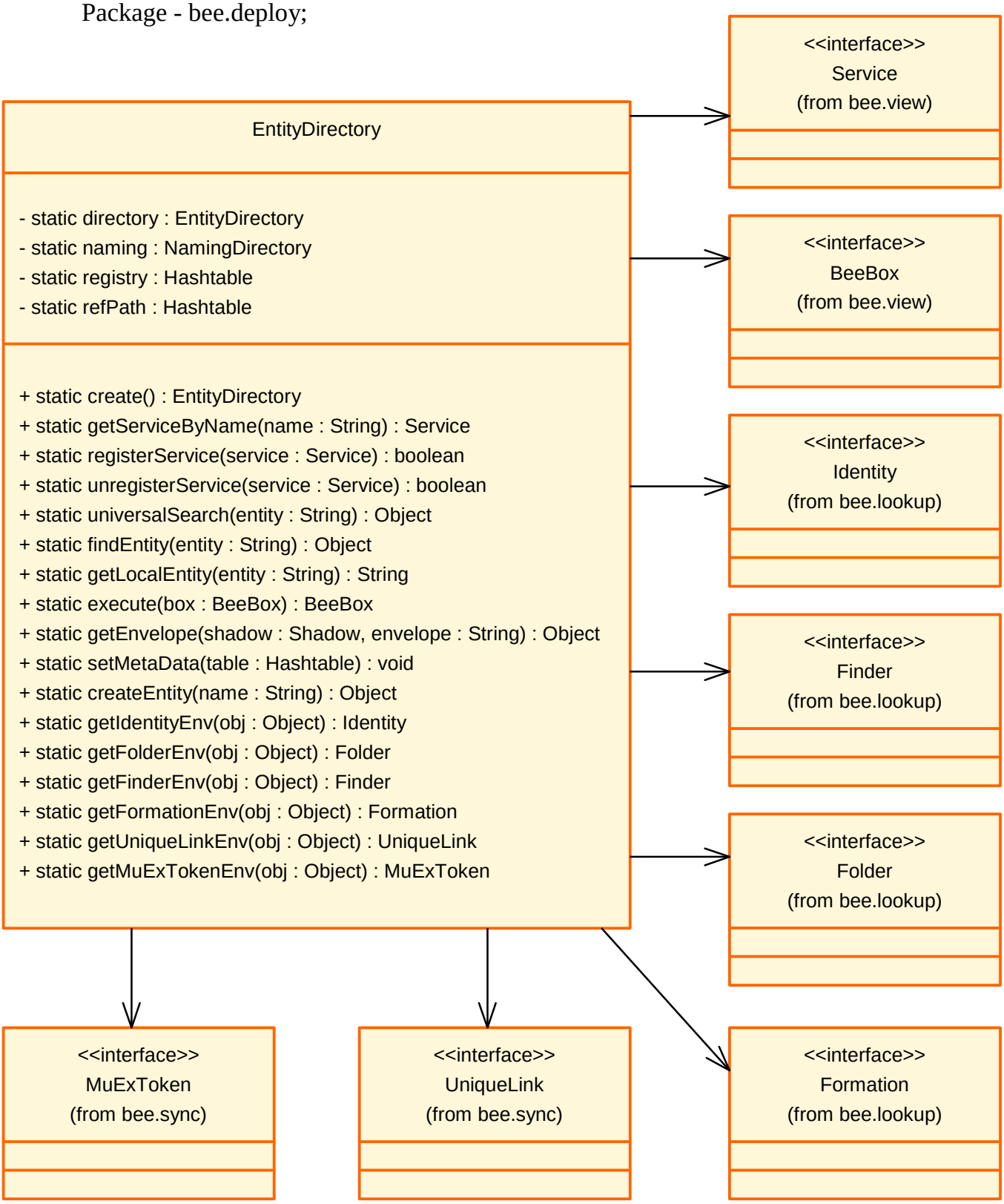


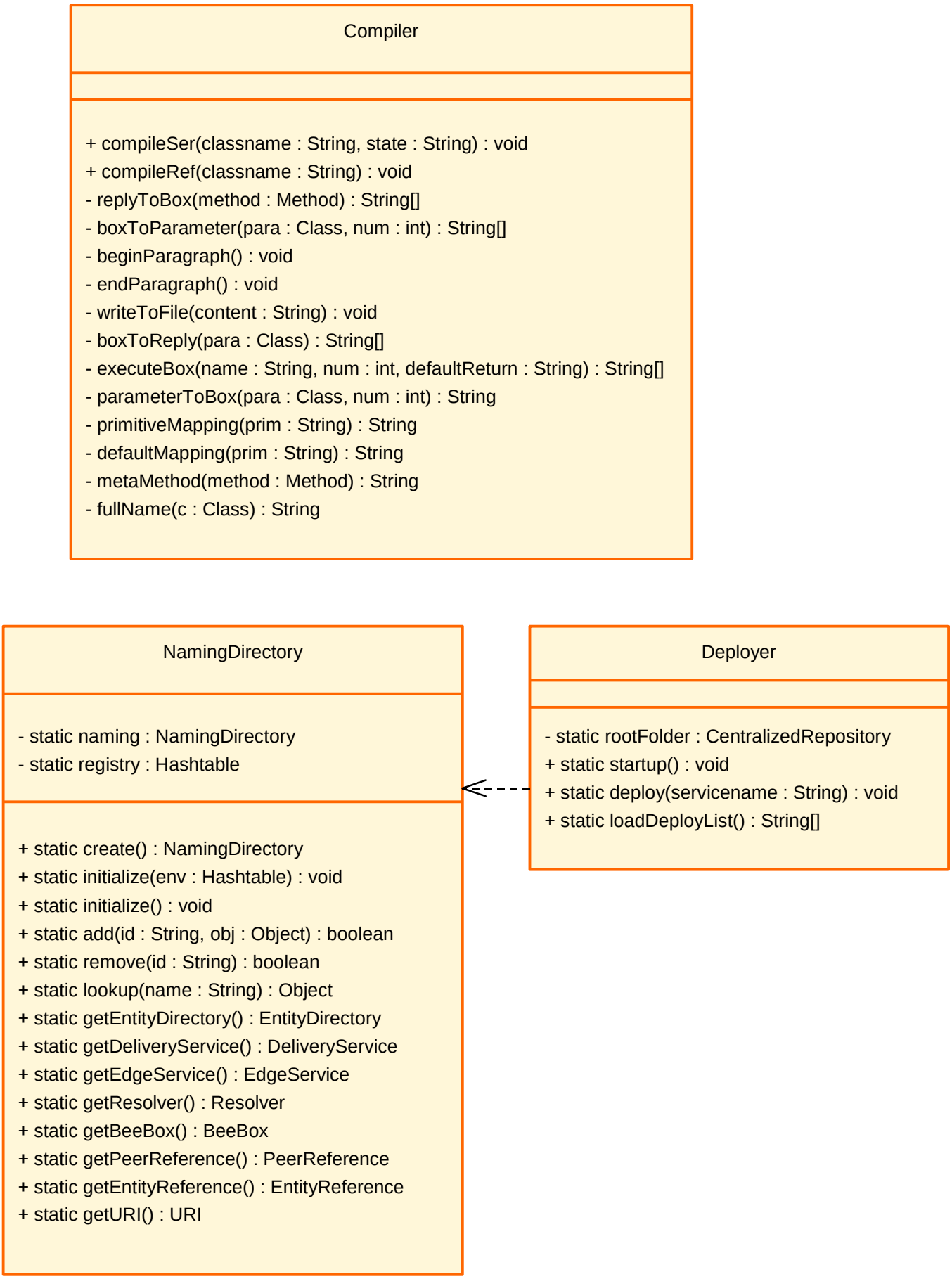
Interface function

GroupWrapper

GroupWrapper is an envelope service. It allows the introduction of load balancing capability in the target group of peers.

Package - bee.deploy;




Compiler
+ compileSer(classname : String, state : String) : void
+ compileRef(classname : String) : void
- replyToBox(method : Method) : String[]
- boxToParameter(para : Class, num : int) : String[]
- beginParagraph() : void
- endParagraph() : void
- writeToFile(content : String) : void
- boxToReply(para : Class) : String[]
- executeBox(name : String, num : int, defaultReturn : String) : String[]
- parameterToBox(para : Class, num : int) : String
- primitiveMapping(prim : String) : String
- defaultMapping(prim : String) : String
- metaMethod(method : Method) : String
- fullName(c : Class) : String
NamingDirectory
- static naming : NamingDirectory
- static registry : Hashtable
+ static create() : NamingDirectory
+ static initialize(env : Hashtable) : void
+ static initialize() : void
+ static add(id : String, obj : Object) : boolean
+ static remove(id : String) : boolean
+ static lookup(name : String) : Object
+ static getEntityDirectory() : EntityDirectory
+ static getDeliveryService() : DeliveryService
+ static getEdgeService() : EdgeService
+ static getResolver() : Resolver
+ static getBeeBox() : BeeBox
+ static getPeerReference() : PeerReference
+ static getEntityReference() : EntityReference
+ static getURI() : URI
Deployer
- static rootFolder : CentralizedRepository
+ static startup() : void
+ static deploy(servicename : String) : void
+ static loadDeployList() : String[]

Class function descriptions

EntityDirectory

EntityDirectory is the class that stores all the deployed objects and provides runtime functions. Accessing the envelope service is also through this singleton object.

Deployer

The Deployer class responses for managing the deployment setting. The list of service names is read from a file.

NamingDirectory

Provides name binding for various local objects. This NamingDirectory object is mainly returning the interfaces that are defined in BEE framework specification. By using this singleton object, different version of reference implementations can be installed easily. It provides default mapping for the concrete classes and the standard interfaces, but the setting can be changed easily by using the Deployer class.

Compiler

The compiler generates two files from a Java interface. Both files have the same name as the interface and suffixes with either 'Ref' or 'Service'. The file ended with 'Ref' can compile successfully and is expected to be distributed as API for application. For the 'Service' file, a concrete implementation must be filled into it before any system deploys this service.

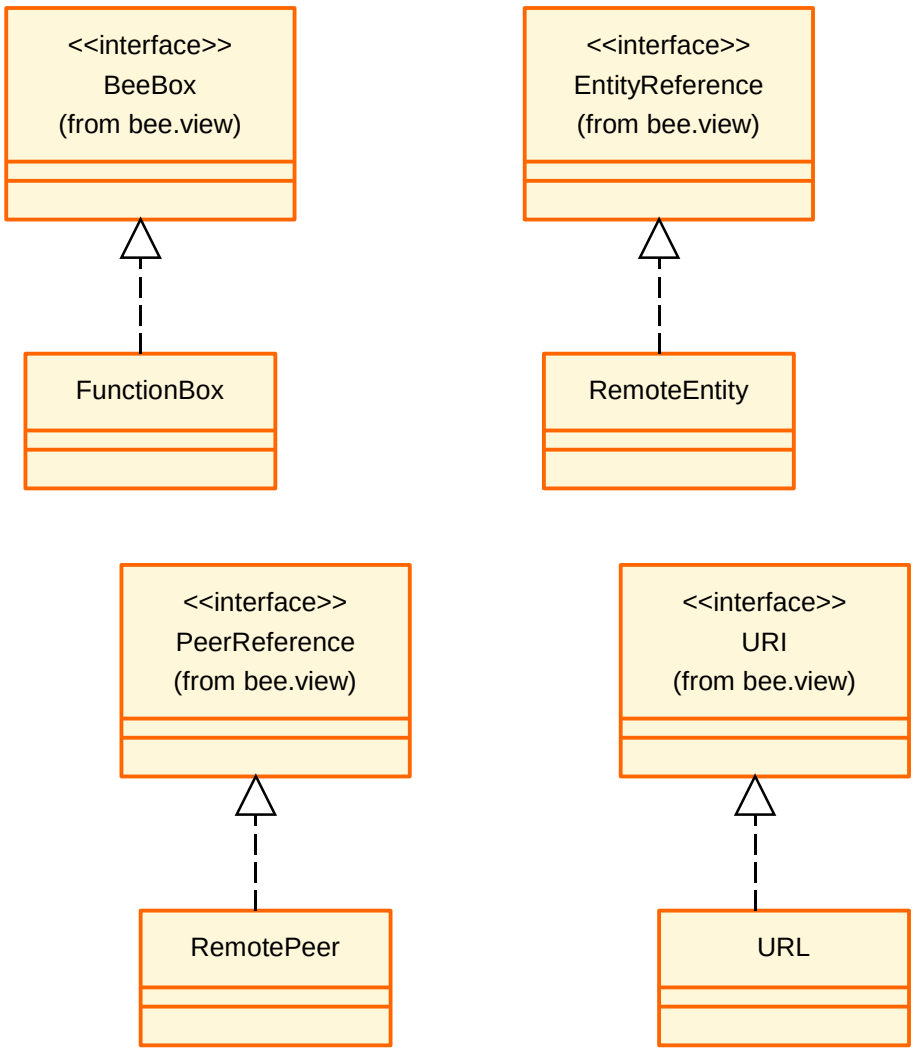
Reference Implementation

Every package of the reference implementation is using the same name as the standard package except prefixed with the word 'framework'. For example:

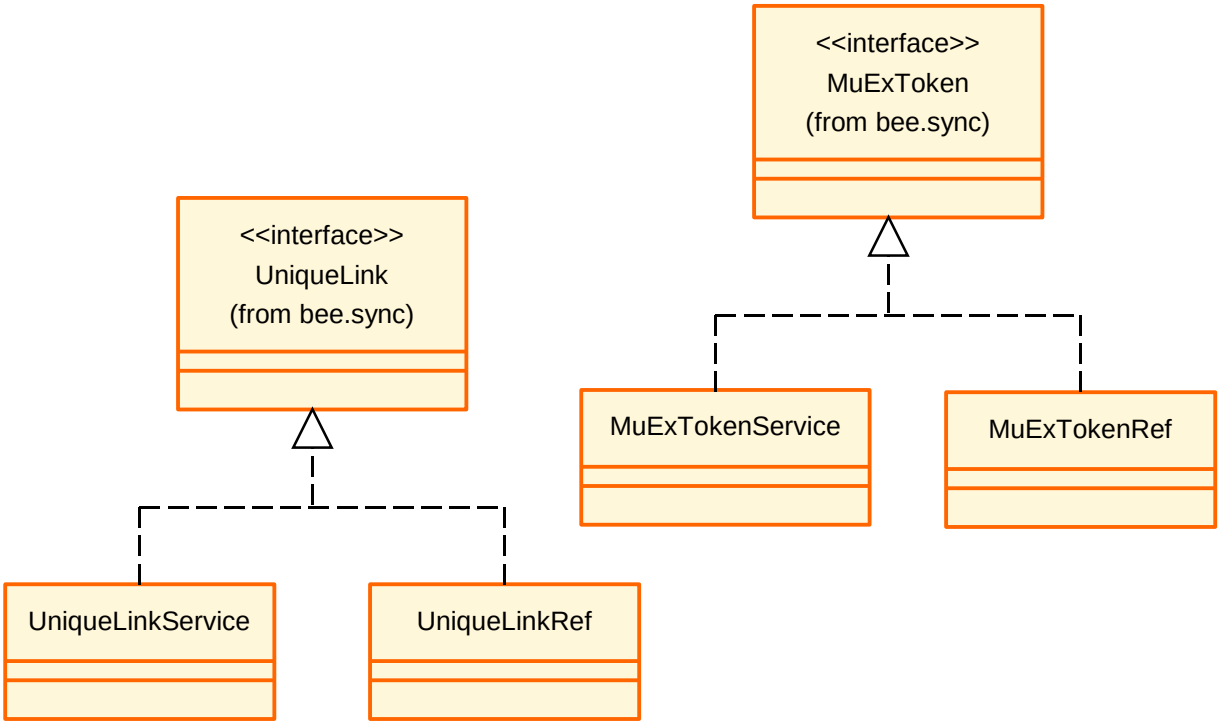
RI (package)	Standard interface (package)
bee.framework.view	bee.view
bee.framework.tunneling	bee.tunneling
bee.framework.lookup	bee.lookup
bee.framework.sync	bee.sync
bee.framework.replication	bee.replication

Class Diagram

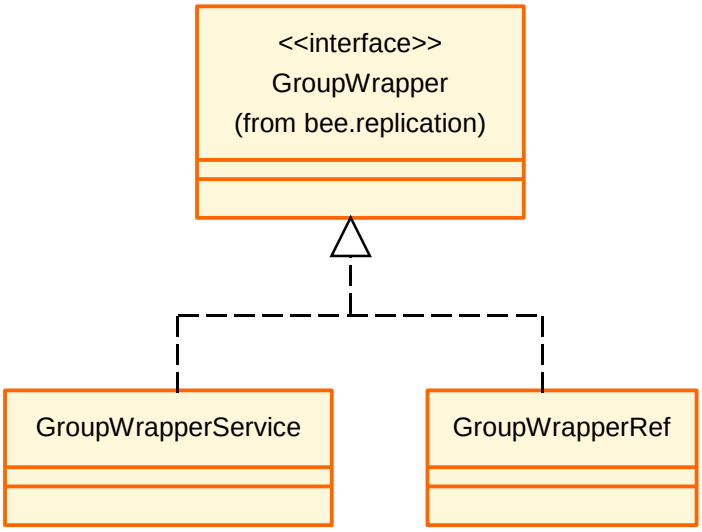
bee.framework.view



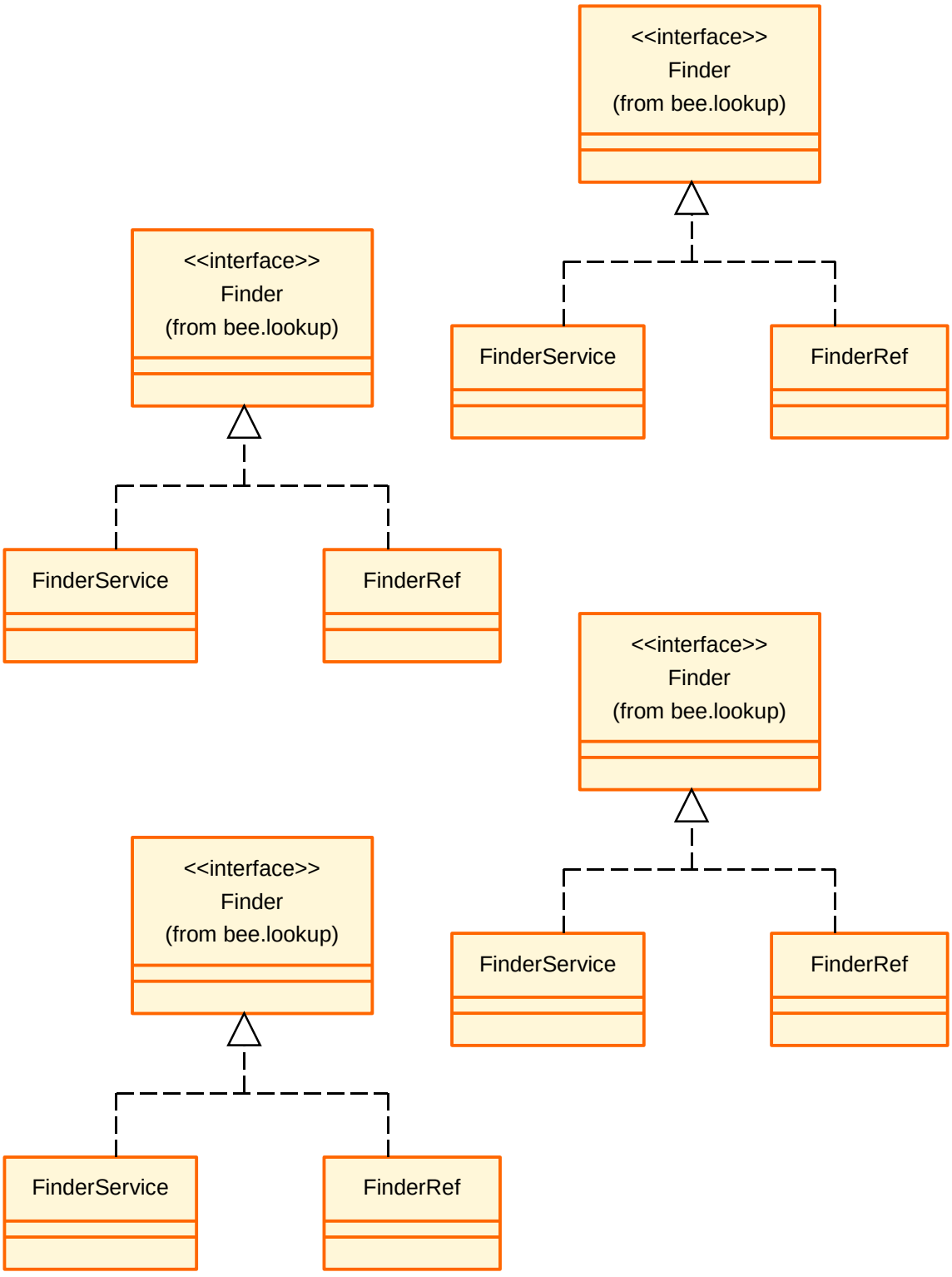
bee.framework.sync



bee.framework.replication



bee.framework.lookup



Programming Challenges and Solutions

Developing this framework involves many system level programming designs since its purpose is to provide low level mechanism for distributed programming. Programming algorithm efficiency, resource management and performance tuning become significant for this large system development which is not handled seriously in small application creation. Here are the major challenges encountered and their solutions:

Imperfect Network IO

The TCP sockets that are provided natively by the operating system vary greatly across different OS. The number of connections that can be initialized is limited by the OS. If there are multiple simultaneous connection requests, some of them would be lost if the response time is not fast enough.

Since the thread used to listen connection requests is blocked during its operation, the frequency of the polling is significantly affecting the connection's successfulness. Therefore, when the computer is heavily loaded, connections establishing exceptions would more likely to occur.

Solution

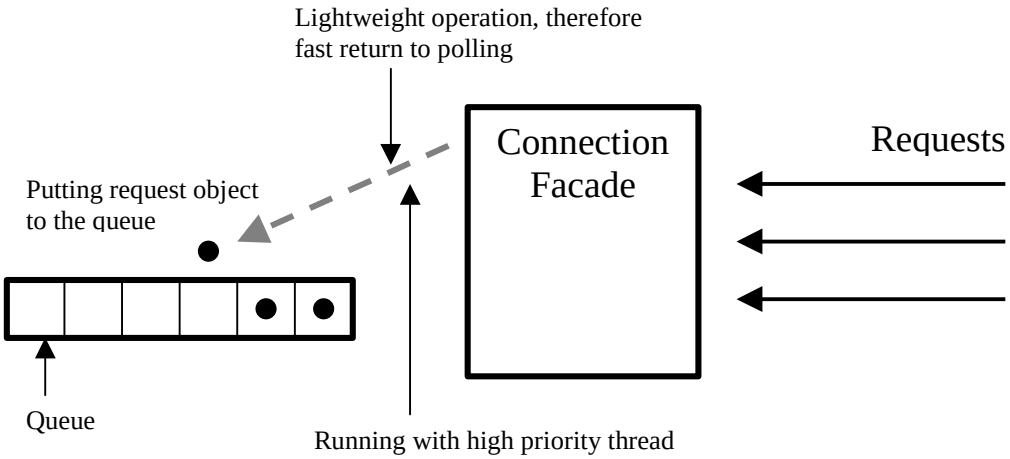
For the solution of the polling efficiency problem, changing the thread running priority is used. Inside the Java VM, every thread is running with its own priority. Changing the polling thread to higher priority can reserve the required processing power to handle IO operation.

Since the number of active connection is limited by the operating system, a memory queue can be used to store the requests and keeps the number of active connection under a desire level. A Facade design pattern is used to handle this condition.

A facade object encapsulates the existing services and the associated data provided by the platform. This class can export a cohesive abstraction that provides a specific type of

functionality. The methods in the facade class are simply forwarding the connection requests to the queue. The underlying mechanism is hidden within the private portion of the facade object.

Connection Handling Mechanism



Thread per Request Causes Overloading

In response to the remote method invocations, the initial approach is to try to service them as long as they arrived. This solution seems straight forward, but problems come out when more services are running. During the beginning stage of the framework development, the number of deployed services is still small. Later, when there is a test of running over 10 peers (i.e. over 10 servers and 10 clients) in one local computer, the problem of method invocation exceptions occur frequently and the whole system down.

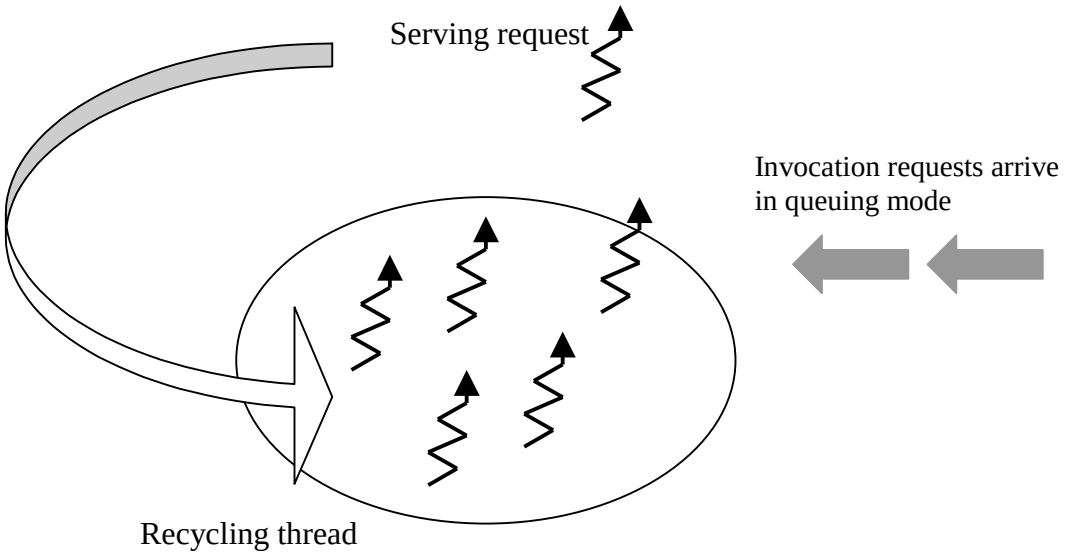
After the code checking of several important parts, it seems there are no logical mistake from the programming point of views. The same test is re-run again with only few peers, and the problems occurred before are disappeared. It can therefore be concluded that there are problems in resources management.

Solution

The solution to this challenge is to try to manage the amount of resources that consuming during system running. Since the services deployed with the framework is event triggered, limiting the number of running threads can solve that problem.

The object pooling technique is used to develop this solution. A number of worker threads are pooled and they are waiting for asynchronous wake up event. Therefore, the number of invocation requests that can be served at one time is manageable. The capability that can provide asynchronous event dispatching is by the use of the wait() and notify() methods of the Java object (java.lang.Object).

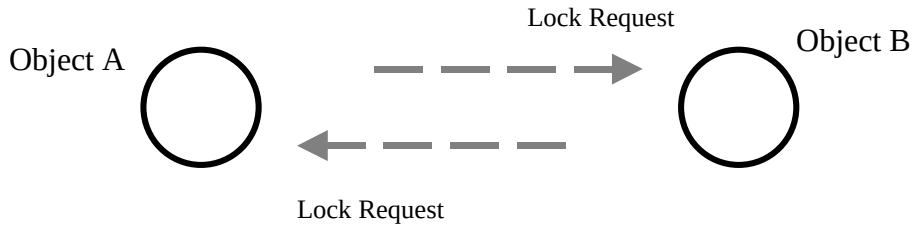
The Concept of Pooling



Distributed Synchronization

Having synchronization capability within objects in the same Java VM is very simple and easy to use. The reason for the efficient implementation of the local synchronization is that a central coordinator can be found easily. In the case for Java programming language, the Java VM is the coordinator.

However, for two distributed objects, it is hard to do so especially in a peer-to-peer situation. The explanation is show below:



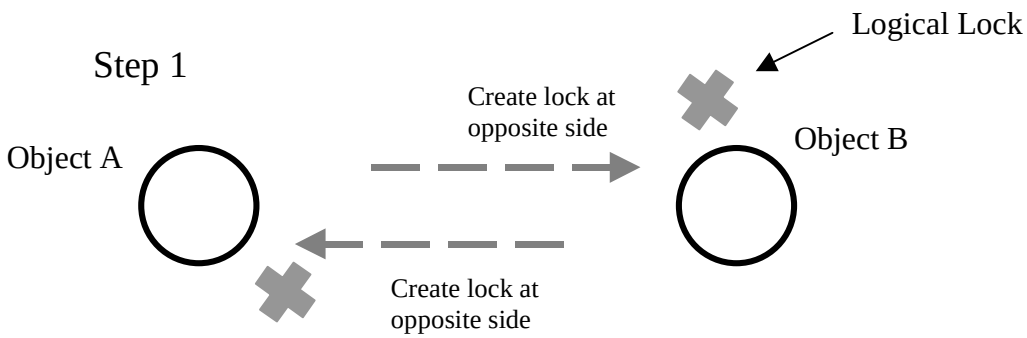
There would be conflict if two peers want to acquire lock onto each other. One possible solution is that both peers wait for a random of time and initiate the request again. However, if this solution is implemented at such high level like programming level, the efficiency is not enough. It is because waiting for seconds of time if two distributed objects are conflict in their method call would make the whole application runs slowly. If the waiting time is in micro second level, the difference is not significant when the method call passed to the low level network connection again. Therefore, the two objects would send requests to each other in non-stop way.

Solution

The developed solution is to use implicit distributed lock. The medium that solves the distributed synchronization problem changed from time to number of trials.

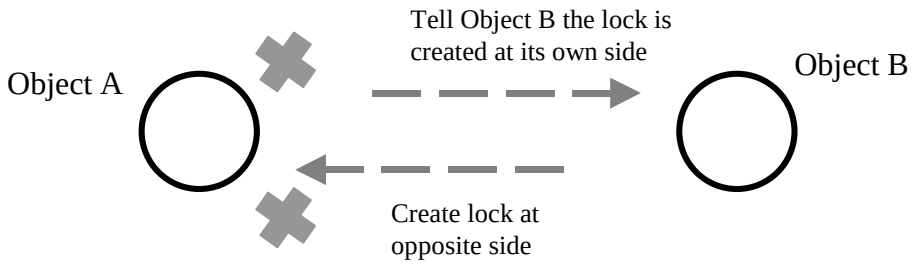
Each object would try to create a logical lock randomly first when they want to have distributed synchronization. After the successful creation, each of them needs to acquire lock from this virtual central coordinator.

Mechanism of Implicit Distributed Lock



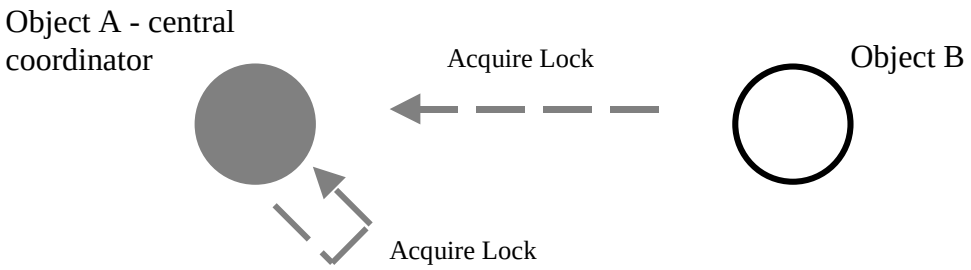
Step 1 is not success because both objects do not have a common agreement on the position of the lock. Therefore, the same operation has to repeat.

Step 2



Step 2 can successfully create a virtual central coordinator.

Step 3



After the existence of a central coordinator, the distributed synchronization can be done in a straightforward way.

The efficiency of this implicit distributed lock is good because the time for successful connection is by the means of number of trials rather than time. Having few distributed method calls needs only mini-second of time.

There is 50% of chance will be success when both objects try to create the lock at the same time. If only one object is trying the creation, this will be 100% to be successful.

Transforming Methodologies into Java Objects

In programming the framework, some more functions are also added to make it more versatile. There are the implementations of load balancing and mutual exclusion algorithms.

Both of these services are in form of envelope, so they can be used by all application objects that are running within the framework.

The mutual exclusion function is the implementation of an algorithm called CSL. It enables queuing up process of requests in the distributed environment to have minimum overhead.

A wrapper object can provide load balancing across a number of distributed identical objects. The algorithm that using is Round Robin, because it is efficient to use and simple to implement.

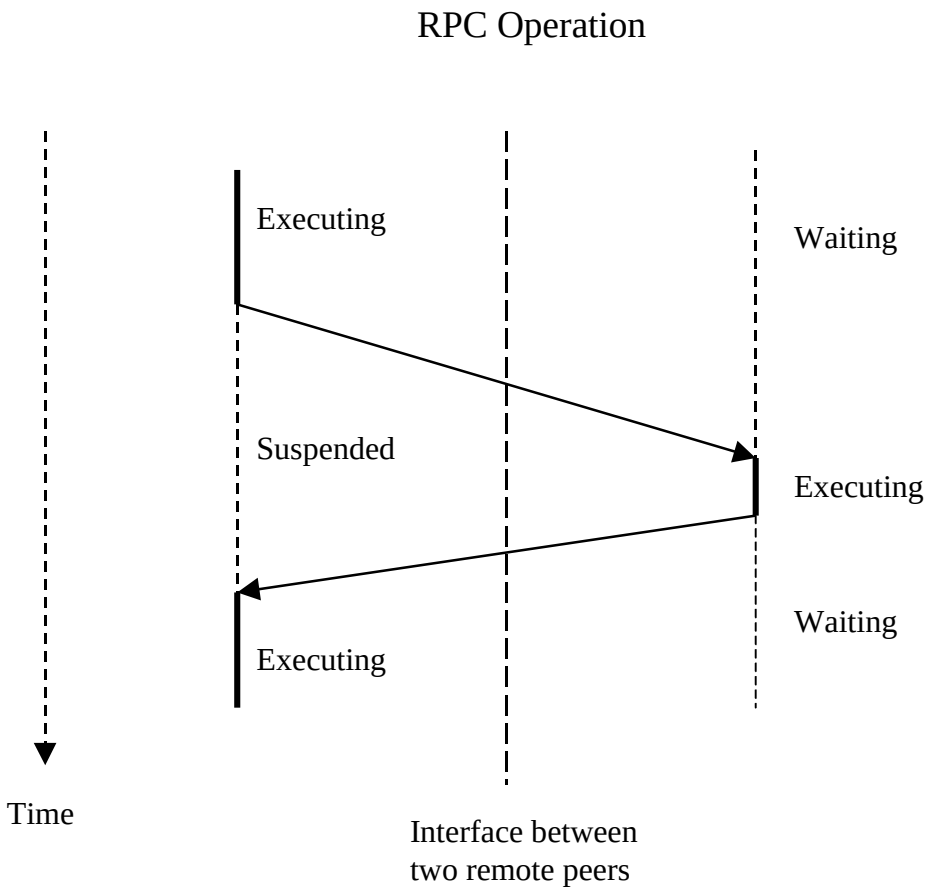
Remote Procedure Call (RPC) over Asynchronous Messaging

The SOAP messaging layer as presented in the Platform Architecture section is asynchronous in nature. However, the method calls in Java are synchronous operations and the idea of RPC is used in the framework.

Remote procedure call (RPC) is a generalization of the standard procedure call as provided in many higher level programming languages like C, Pascal or Java. RPC is defined as a synchronous flow of control and data passing method achieved through procedure calls between processes running in separate hosts where the needed communication is via channels. The channels can be any level of protocols.

The strength of RPC lies in the fact that neither the client nor server assumes that the procedure call is performed over a network, so asynchronous messaging or different

protocols can be involved. Both the client and server just execute a standard method call. Below shows the mechanism of RPC:



The flow of control as well as the data passing operation are synchronous. That means the client peer after having called the method, it suspends its execution and waits for the server peer to return the result. To avoid endless blocking due to crashing of peers or the unreliability of networks, timeout frees the suspended client peer and allows it to react properly with response to these kinds of problems.

A Logistics Application

In order to demonstrate the functionality of the framework, a logistics application is built to achieve this goal.

The strengths of the framework that demonstrated are:

Decentralized management

Every peer in the network can deploy a subset of the complete set of services. The whole system is functional as long as there existing at least one object instance for every service in the deploy list.

Collaboration of distributed objects

All the available services are displayed on the service tree at the left hand side. The physical location of these object are distributed across the network.

Awareness of data integrity

Every object instance can be started up and shut down, the corresponding service would still be available as long as at least one such kind object instance exists.

Logistics Application

A logistics application is built, which is providing the follow functions:

Routing recording checking

Enable to search for an existing record by entering its number.

Update of the delivery path

Allow to update the transportation path and method in case of any except. For example, traffic jams, weather condition or the cost of the path.

Real time chat

In case of an exception, the real time chat function allows the discussion of both managers in real time.

Instant traffic condition

Display the current traffic condition, which is expected to be provided the traffic monitoring station.

Weather information

Display the weather forecasting information.

Tunnel cost

Display the cost of using various tunnel and help to select an optimum path.

Centralized Repository Application

Since an initial startup mechanism is required to construct the BEE environment, any discovery can be used.

For convenience, a centralized repository application is built as a form of application by using the BEE framework. The reason is that it enables direct access to this repository service within the BEE environment. A LDAP directory server can be used to provide the same function, but more code is required to handle the bridging between the Java VM and the native directory server application.

Furthermore, this centralized repository can be used as another demonstration of building application by using the BEE framework.

Package Description

Package - app.discovery;

This package contains the discovery application that is used by the framework to startup the construction of network structure.

CentralizedRepository

The CentralizedRepository interface encapsulates the low level discovery mechanism that the framework used to find the other peers in the most elementary level. This interface can abstract out different searching methods, e.g. ADS, NDS, DNS etc.

Package - app.Logistics.FastCompany;

Contains the company serving objects for the logistics application.

PathChanging

Provide the functions to update the transportation path and method.

RealTimeChat

Allow the direct chatting between two peers.

RoutingRecord

Enable to search for an existing record by supplying the record number.

Package - app.Logistics.InstantInfo

Contains the instant information objects for the logistics application. They can be deployed in traffic monitor station.

TrafficCondition

Provide instant information about traffic condition.

Tunnel

Provide cost about tunnel usage.

Weather

Provide weather forecasting information.

Package - app.ui;

This package contains all the visual components for both standard user interface and logistics application specific windows.

StandardUI

Standard user interface that comes with the framework.

GenericWindow

The super class of all the internal windows for the logistics application.

ReadingInputPanel

The visual component that response for getting input from user.

PathChangingWindow

The visual component for the path changing function.

RealTimeChatWindow

The visual component for the real time chat function.

RoutingRecordWindow

The visual component for the routing record function.

TrafficConditionWindow

The visual component for the traffic condition function.

TunnelWindow

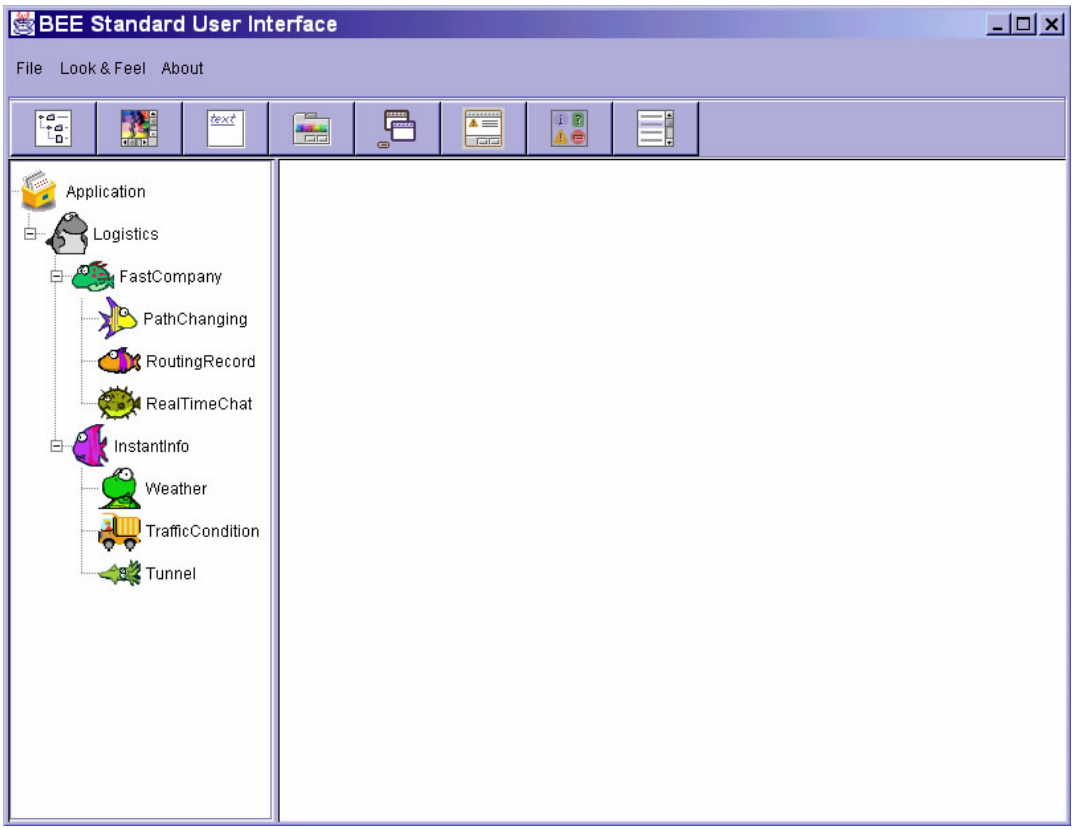
The visual component for the tunnel usage cost function.

WeatherWindow

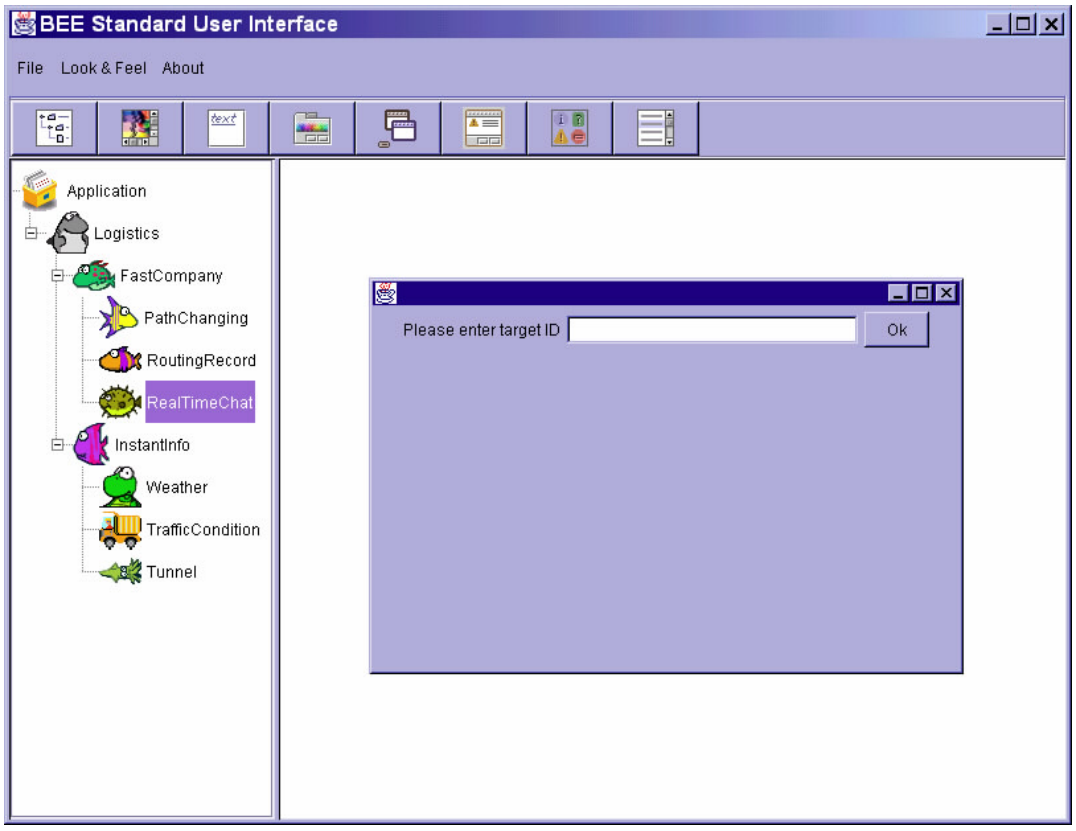
The visual component for the weather forecasting function.

Screen Dump of the Logistics Application

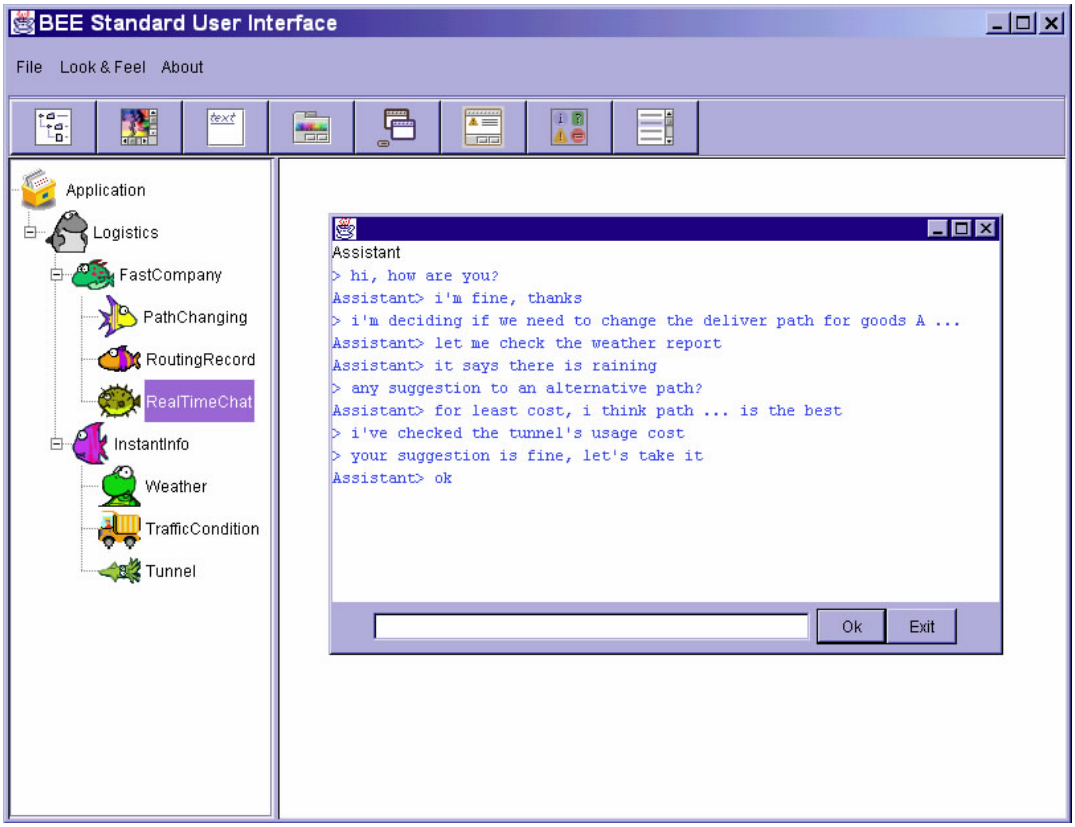
The main window



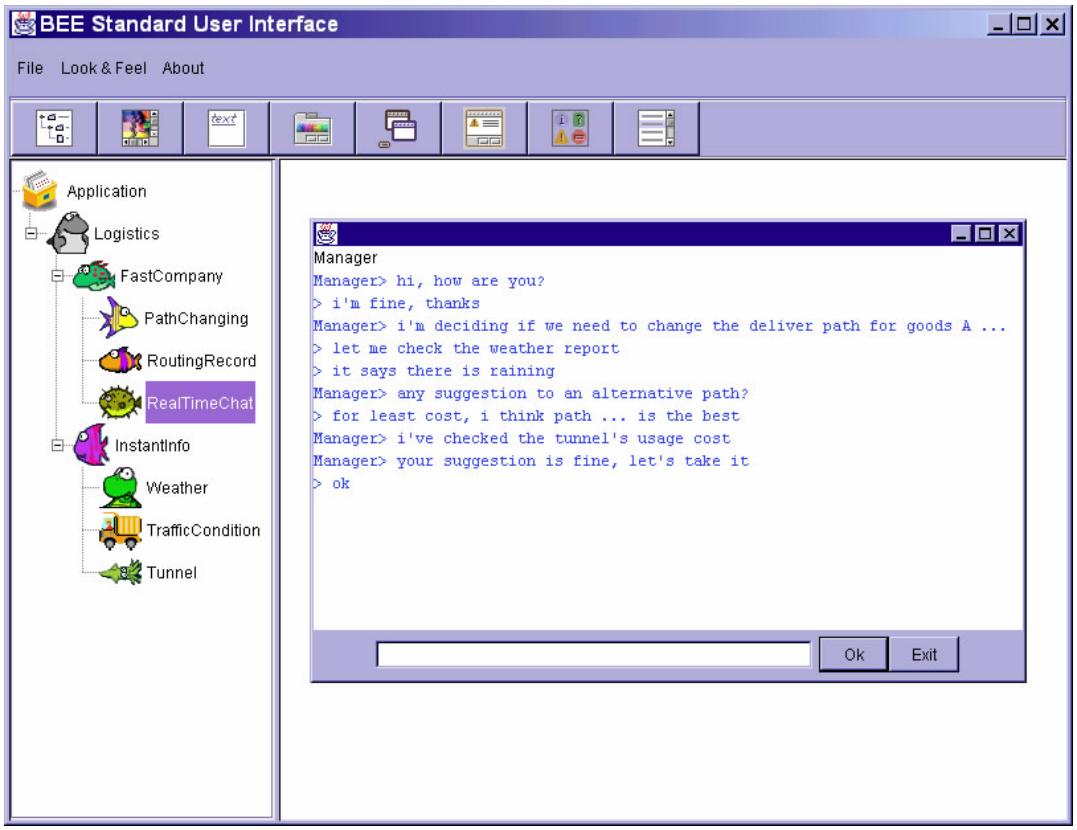
Opening up the chat window and search for the required person



The view of the manager (see the Assistant at the chat window)



The view of the assistant (see the Manager at the chat window)



Appendix (classes and methods descriptions)

Package - bee.view;

BeeBox

BeeBox interface encapsulates all the actions that can be carried out on the SOAP message. Since multiple APIs for SOAP can be used, this interface provide a standard across the different brands of product to be used in the BeeNetwork.

+ getEntityName() : String

Get the entity name of the function call.

+ getMethodName() : String

Get the method name of the function call.

+ getParameter(number : int) : String

Get all the parameter of the specified number.

+ getAllParameters() : String[]

Get all the parameters of the function call.

+ numberOfParameter() : int

Get the number of parameters of the function call.

+ setEntityName(entity : String) : void

Set the entity name of the function call.

+ setMethodName(method : String) : void

Set the method name of the function call.

+ addParameter(para : String) : void

Add a parameter of the function call.

+ addParameters(paras : String) : void

Add all parameters of the function call.

EntityReference

EntityReference interface is used to wrap the remote peer's service into a local object interface.

+ getReference() : String

This method return the logical address of the entity. Regardless whether it is stateful or stateless entity.

+ setReference(name : String) : void

This method set the default logical address of the entity. The string is used to construct the URI of this entity.

+ invoke(bb : BeeBox) : BeeBox

The default entity name associated with this interface overrides the entity name in the BeeBox object when calling this method.

PeerReference

PeerReference contains the information used to interact with a peer at low level. It manages the pairing of abstract BeeEntity reference and the protocol URI.

+ invoke(bb : BeeBox) : BeeBox

Execute the object method as described in the BeeBox object.

+ getPrimaryProtocol() : String

Each PeerReference is associated with more than one network protocol. This method returns the name of primary protocol, which is usually the most efficient one.

+ getAvailableProtocols() : String[]

Return the name of all the supported protocols. The number of protocol that supports is restricted by the object bee.tunneling.DeliveryService.

+ setPrimaryProtocol(proto : String) : void

Change the primary protocol to the specified protocol name.

+ addMoreProtocol(proto : String) : void

Add the protocols that the remote object is supporting. The information can be gained by invoking the getSupportedProtocol(: int) method of the Identity envelope service of the remote object (interface: bee.lookup.Identity).

+ addMoreProtocols(protos : String[]) : void

Add a set of supporting protocols. Same as the above addMoreProtocol(: String) method except an array of protocols is supplied.

Service

The Service interface defines the methods that all the servicing objects should implement. The methods define the invocation and naming query interface. It is the super interface of all the interfaces that want to provide services.

+ parseContainer(bb : BeeBox) : BeeBox

Calling this method would cause the object implementing this method to give specific responds. Return null if the request is invalid.

+ getEntityName() : String

Return the location of this object in the current machine. It is used to call the method of the object.

ServiceEnvelope

ServiceEnvelope is a kind of service that wrap an ordinary service. It is mainly used by the framework.

+ setLetterService(letter : String) : void

Put a letter into this envelop.

Shadow

Shadow interface is implemented by all the logical reference in the BeeNetwork. It is used for getting the reference string.

+ getShine() : String

Return the full object ID and network address.

Shine

Shine interface is implemented by all the logical reference in the BeeNetwork. It is used for setting the reference string.

+ setShadow (ref : String) : void

Set the reference to the specified value.

StatefulEntity

This interface is implemented by the object that wants to provide stateful service. It is used as identification by the compiler.

StatelessEntity

This interface is implemented by the object that wants to provide stateless service. It is used as identification by the compiler.

URI

URI interface describe the basic function that an URI should have. Any URI provides the information of describing access mechanism, address of the remote host and the name of the required resource.

+ getContent() : String

Return a String object to represent this URI object.

+ getProtocolName() : String

Return the access mechanism's name.

+ getRemoteAddress() : String

Return the address of the host, which is protocol specific.

+ getResourceName() : String

Return the name of resource.

+ setContent(uri : String) : void

Initialize the URI using the input string.

+ setProtocolName(proto : String) : void

Set the protocol name.

+ setRemoteAddress(addr : String) : void

Set host address.

+ setResourceName(name : String) : void

Set resource name.

Package - bee.tunneling;

ControllableSocket

ControllableSocket interface defines the behavior that a server socket should have. The socket can be any protocol implementation.

+ getProtocolName() : String

Return the name of the protocol that this socket supporting.

+ getNetworkAddress() : String

Return a string that defines the address of this peer in the network. This string is protocol specific.

+ setMessageTarget(es : EdgeService) : void

Since the socket only responsible for receiving the message, it won't do any analysis procedures. Therefore, the resolving and delegation processes are passed to the EdgeService.

+ activate() : void

Activate this socket.

+ deactivate() : void

Deactivate this socket.

DeliveryService

DeliveryService is a singleton class that manages all the low-level transportation mechanism of the SOAP message. It does so by using the OpenChannel object, which responses to handle the actual mechanism.

- static DeliveryService loadService(table : Hashtable)

Loads the delivery service from the path that specified in the input Hashtable object. This method is used internally.

static OpenChannel loadChannel(protocol : String)

Initializes a channel for transportation and the name of the protocol is provided. Return null if the protocol is not supported.

+ static DeliveryService createService()

Creates a default DeliveryService is not available. This method can ensure that there is only single instance of that service.

+ static DeliveryService createService(env : Hashtable)

Loads the DeliveryService with new setting. The method gives the same function as the no argument version.

+ abstract OpenChannel createConnection(addr : String)

Creates a concrete object that provides open connection from the specified network address. Since various parsing techniques can be used, it needs to be implemented by subclass. The ability to support new type of protocol can be enhanced by providing different version of this method.

+ abstract SecureChannel createSecureConnection(addr : String)

Creates a concrete object that provides encrypted connection from the specified network address. The function is similar to createConnection(: String).

DeliveryService

EdgeService is the super class of all the monitors that take care of the control of server side sockets. It attempts to load the default reference implementation of EdgeService from the system properties.

- static loadService(table : Hashtable) : EdgeService

Loads the delivery service from the path that specified in the input Hashtable object. This method is used internally.

static loadSockets() : ControllableSocket[]

The method returns the controllable sockets as specified in properties.

+ static createService() : EdgeService

Loads the default edge service and return it.

+ static createService(env : Hashtable) : EdgeService

Loads the EdgeService with new setting.

+ abstract installSocket(cs : ControllableSocket) : void

Add one socket to the control list.

+ abstract closeSocket(cs : ControllableSocket) : void

Deactivate the specified socket.

+ abstract getInstalledSockets() : ControllableSocket[]

Get all the running sockets. These running server sockets are supporting different protocols, but they are all implement the ControllableSocket interface.

+ abstract getRunningEntryRefs() : String[]

Get the network addresses of all the running sockets.

+ abstract getPrimarySocket() : ControllableSocket

Get the primary socket.

+ abstract getPrimaryEntryRef() : String

Get the network address of the primary socket.

+ abstract setPrimarySocket(cs : ControllableSocket) : void

Set the primary socket. This socket is usually the most efficient one.

+ abstract processMessage(msg : String) : String

Callback by the socket so that the EdgeService can delegate the request.

+ abstract numberOfSocket() : int

Return the number of server sockets that the peer have.

OpenChannel

The interface that implemented by all the non encrypted connection. It defines the operations that a public channel should have.

+ getProtocol() : String

Return the name of the protocol.

+ setDestination(addr : String) : void

Set the network address of the required destination. Repeating the use of this method can provide objects recycling capability.

+ getDestination() : String

Get the destination's network address.

+ send(box : BeeBox) : BeeBox

Sending the SOAP message in synchronous mode. The reply is also wrapped through by the BeeBox interface.

+ send(box : BeeBox, timeout : int) : BeeBox

Similar function as the single argument version except a timeout parameter is provided to terminate the blocked connection after time expired.

+ switchToSecureMode() : SecureChannel

Return a secure connection with encryption during data sending.

SecureChannel

It is an extension of the OpenChannel interface. The differences are not noticeable to the programmer, but the underlying transfer mechanism provides encryption.

+ switchToOpenMode() : OpenChannel

The method returns the reference to the connection object that does not provide encryption during transmission.

Resolver

The Resolver is responsible for translating the input message into standard BeeNetwork function call interface BeeBox. Since Resolver is replacable, the message that the framework can handle can be changed easily. Or faster SOAP parser can be installed quickly.

+ static createResolver() : Resolver

Returns the singleton object reference of the class Resolver. The method would try to load one if there is no existing singleton object. The actual procedures of initializing the

required object are relied on the ResolverBuilder object. If no builder is available, the default one is used.

+ static setResolverBuilder(rb : ResolverBuilder) : void
Set the builder for the Resolver object.

+ abstract resolveMessage(msg : String) : BeeBox
Maps the message (possibly XML - SOAP) into a Java object that is implementing the BeeBox interface.

+ abstract marshalContainer(box : BeeBox) : String
Does the reverse of resolveMessage(: String) method. The method transforms the content of the Java object into a message.

ResolverBuilder

ResolverBuilder interface is implemented by the builder object for the Resolver class. It defines only one creation method for Resolver.

+ buildResolver() : Resolver
Returns an instance for the abstract class Resolver.

Package - bee.tunneling;

Identity

This is one of the interfaces that supported by envelope service. The envelope service that implements this interface would wrap the identity of an object in BEE environment.

+ pingPong() : String
The string representing the protocol and logical name for connecting to this peer is returned.

+ resourcePolling() : String

Return the performance index in turn of a string representation.

+ getSupportedProtocol(num : int) : String

Return all the supported protocols of this peer.

Formation

Define the basic messages that the BEE network required to build up its topology.

+ void bind(peerAddr : Object)

The peer is bound by another remote peer specified by the peerAddr object.

+ Formation getRootPeer()

Tell the peer point by PeerReference whether this peer is the root node or not. If this is not, forward this message to its parent node.

+ void changeOfIntegrity()

Calling this method would block until there is a change in the integrity of the topology.

+ int numberOfPeers()

Return the number of peers that binding with this peer.

+ Object getConnectedPeer(num : int)

Return the entity reference strings to locate the address of the specified connecting peer. The peer with number zero i.e. num == 0 is the peer that pointed by the forward arrow.

+ Formation getConnectedGroup(num : int)

Return the entity reference strings to locate the address of the specified connecting peer group. Formation object is representing a peer group.

Finder

Finder interface encapsulates the logic in creating the discovery process.

+ getContainerFolder() : Folder

Return the folder that containing the letter entity. Any BEE application that implements the Folder interface can provide the discovery process. A simple centralized repository application is built to handle the startup process of the BEE system, as stated later in the application package.

Folder

Define the lookup service that the framework provides.

+ getUpperFolder(index : int) : Folder

Return the upper folder in the hierarchy.

+ searchFolderFrom(source : Folder,index : int) : Folder

Used for recursive searching of upper folder across the tree.

+ getEntity(name : String, index : int) : Object

Return the entity with specified name. It can be a folder since folder is a kind of entity. The entity object returned not necessary be the same for the same index input in each method call.

+ searchEntityFrom(source : Folder, name : String, index : int) : Object

A generic version of the method searchFolderFrom(: Folder, : int).

+ pushTo(folder : Folder) : void

Pushes all the contents containing in the current folder object to the specified folder. Since pushing all the data would cause heavy load to low level network connection, it is not recommended to use very often. This method is useful when one folder object is first initialized as it can request the nearby folders to push their content to it. Merging the received data can provide caching function.

+ setUpperFolder(name : String, folder : Folder) : void

Stores the address of the upper folder with respect to the local one. This is usually used during folder startup.

+ putEntity(name : String, entity : Object) : void

Binds an object's network address to the specified name in the current folder. This method would cause chain reaction to the network because the folder objects are currently programmed to be event triggered. Therefore, adding one object to the current folder would cause multiple replications across the group of folders.

Package - bee.sync;

UniqueLink

UniqueLink is an envelope service. It allows the control of single and unique connection to the target peer.

+ connectTo(target : UniqueLink) : boolean
Used to initiate a primitive link between two distributed object.
+ initiateLink(isLockHolder : boolean,request : UniqueLink) : boolean
Used for inter UniqueLink interfaces communication.

+ acquireLock(target : UniqueLink) : void
Acquires the lock of the established primitive channel.

+ releaseLock(target : UniqueLink) : void
Releases the lock of the established primitive channel.

+ grantLock(sig : String) : void
Used for inter UniqueLink interfaces communication.

+ freeLock(sig : String) : void
Used for inter UniqueLink interfaces communication.

+ getSignature() : String
Return the signature of the current UniqueLink object.

MuExToken

MuExToken is an envelope service. It allows the use of distributed mutual exclusion within the target group of peers.

+ enterPrivilege() : void
Try to enter the privilege area.

+ exitPrivilege() : void

Exit the privilege area.

+ receiveTokenFrom(metoken : MuExToken) : void

Pass the token to this object.

+ holdingToken() : boolean

Telling whether this object is holding the token.

+ nextTokenReceiver(metoken : MuExToken) : void

Set the next token user after this object releasing the holding token. The request would be forwarded if there already exists one.

Package - bee.replication;

GroupWrapper

GroupWrapper is an envelope service. It allows the introduction of load balancing capability in the target group of peers.

+ delegate(num : int) : Object

The possible object that can provide the service as specified in input number is returned.

Appendix (programming with the framework)

The usage is:

Compiles the interface below and fills in the implementation of service as defined in the interface (listing 1). Then deploy it with the deployer provided by the framework. The deployed service can then be discovered decentralizely and used by any machine that participate the BEE environment.

An example code of using the envelope service is listed in listing 2.

Listing 1

```
/*
 *
 * Project BeeNetwork
 *
 * author: Cheng Wing Chuen
 *
 */

package bee.sync;



/**
 * UniqueLink is an envelope service. It allows the control of single and unique
 * connection to the target peer.
 *
 *
 *
 */



public interface UniqueLink {

    public static final String SIGNATURE = ".UniqueLink";

    public boolean connectTo(UniqueLink target);

    public boolean initiateLink(boolean isLockHolder, UniqueLink request);

    public void acquireLock(UniqueLink target);

    public void releaseLock(UniqueLink target);

    public void grantLock(String sig);

    public void freeLock(String sig);

    public String getSignature();
}
```

```
public static void main(String[] args) throws Exception {

        // test program
```

Listing 2

```
        NamingDirectory.initialize(null);
        NamingDirectory naming = NamingDirectory.create();
        EntityDirectory dir = EntityDirectory.create();
        Service s1 = null;
        Service s2 = null;

        // using none standard way to register service
        // all the services are registered at the local machine
        // the UniqueLinkService object is cast to Service in both s1 and s2

        dir.registerService(s1 = new UniqueLinkService("TestObject1"));
        dir.registerService(s2 = new UniqueLinkService("TestObject2"));
        dir.registerService(new IdentityService("TestObject1"));
        dir.registerService(new IdentityService("TestObject2"));

        // UniqueLink is envelope service
        // i.e. function provided by the framework

        final UniqueLink link1 = dir.getUniqueLinkEnv(s1);
        final UniqueLink link2 = dir.getUniqueLinkEnv(s2);


       // link1 and link2 are the references to the remote objects
       // they are used like local object during programming
       // but the physical locations are purely decentralized
       link1.connectTo(link2);

        System.out.println(link1.connectTo(link2));

        Runnable r1 = new Runnable() {
            public void run() {
              link1.acquireLock(link2);
              System.out.println("Link 1 acquired lock to Link 2");
              System.out.println("Link 1 sleep for 0.5 seconds");
              try {
                Thread.currentThread().sleep(500);
              } catch (Exception ex) {
              }
              System.out.println("Link 1 releasing lock");
              link1.releaseLock(link2);
              System.out.println("Link 1 released lock");
            }
        };

        Runnable r2 = new Runnable() {
            public void run() {
              link2.acquireLock(link1);
              System.out.println("Link 2 acquired lock to Link 1");
              System.out.println("Link 2 sleep for 0.5 seconds");
              try {
                Thread.currentThread().sleep(500);
              } catch (Exception ex) {
              }
              System.out.println("Link 2 releasing lock");
              link2.releaseLock(link1);
              System.out.println("Link 2 released lock");
            }
        };

        new Thread(r1).start();
        new Thread(r2).start();

    }
```

References

Java Web Services Developer Pack

http://java.sun.com/webservices/webservicespack.html

Microsoft .NET

http://www.microsoft.com/net/default.asp

.NET P2P

http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnmag01/html/netpeers.asp

Simple Object Access Protocol (SOAP) 1.2. W3C, 2001

http://www.w3c.org/2002/ws

Christensen, E., Curbera, F., Meredith, G. and Weerawarana., S. Web Services Description Language (WSDL) 1.1. W3C, 2001

http://www.w3.org/TR/wsdl

UDDI: Universal Description, Discovery and Integration

http://www.uddi.org

Web Services Flow Language

http://www-4.ibm.com/software/solutions/webservices/pdf/WSFL.pdf

Project JXTA

http://www.jxta.org

Brittenham, P. An Overview of the Web Services Inspection Language 2001

http://www-106.ibm.com/developerworks/webservices/library/ws-wsilover

David Karger: Consistent Hashing and Random Trees

http://theory.lcs.mit.edu/~karger/Talks/Hash/index.htm

De Roure, D., Jennings, N. and Shadbolt, N. Research Agenda for the Semantic Grid: A Future e-Science Infrastructure. UK National eScience Center, 2002
http://www.semanticgrid.org

Fallside, D.C. XML Schema Part 0: Primer. W3C, Recommendation, 2001
http://www.w3.org/TR/xmlschema-0

DataSynapse bridges P2P profit gap, by Om Malik, Red Herring, December 21, 2000
http://www.redherring.com/vc/2000/1221/vc-distributed122100.html

Intel Says Think Like Napster, by Leander Kahney, Wired News, Aug 24, 2000
http://www.wired.com/news/technology/0,1282,38413,00.html

Making P2P Safe For The Enterprise, by Darryl K. Taft, CRN, February 16, 2001
http://www.crn.com/sections/news/top_news.asp?ArticleID=24008

Peer-to-Peer Working Group Web Site
http://www.peer-to-peerwg.org

I. C. Lai and C. L. Lei, "A High Performance Dynamic Token-Based Distributed Synchronization Algorithm," Proceedings of the 1997 International Symposium on Parallel Architectures, Algorithms and Networks, pp. 150-156, IEEE Computer Society Press, 1997

Weighted Round-Robin Schedulers for Advanced QoS in High Speed Networks
http://www.ics.forth.gr/proj/arch-vlsi/muqpro/wrrSched.html

Round-Robin Analysis
http://choices.cs.uiuc.edu/~f-kon/RoundRobin/node2.html

Load balancing serves the farms, EE Times, January 24, 2000
http://www.eetimes.com/story/OEG20000124S0040