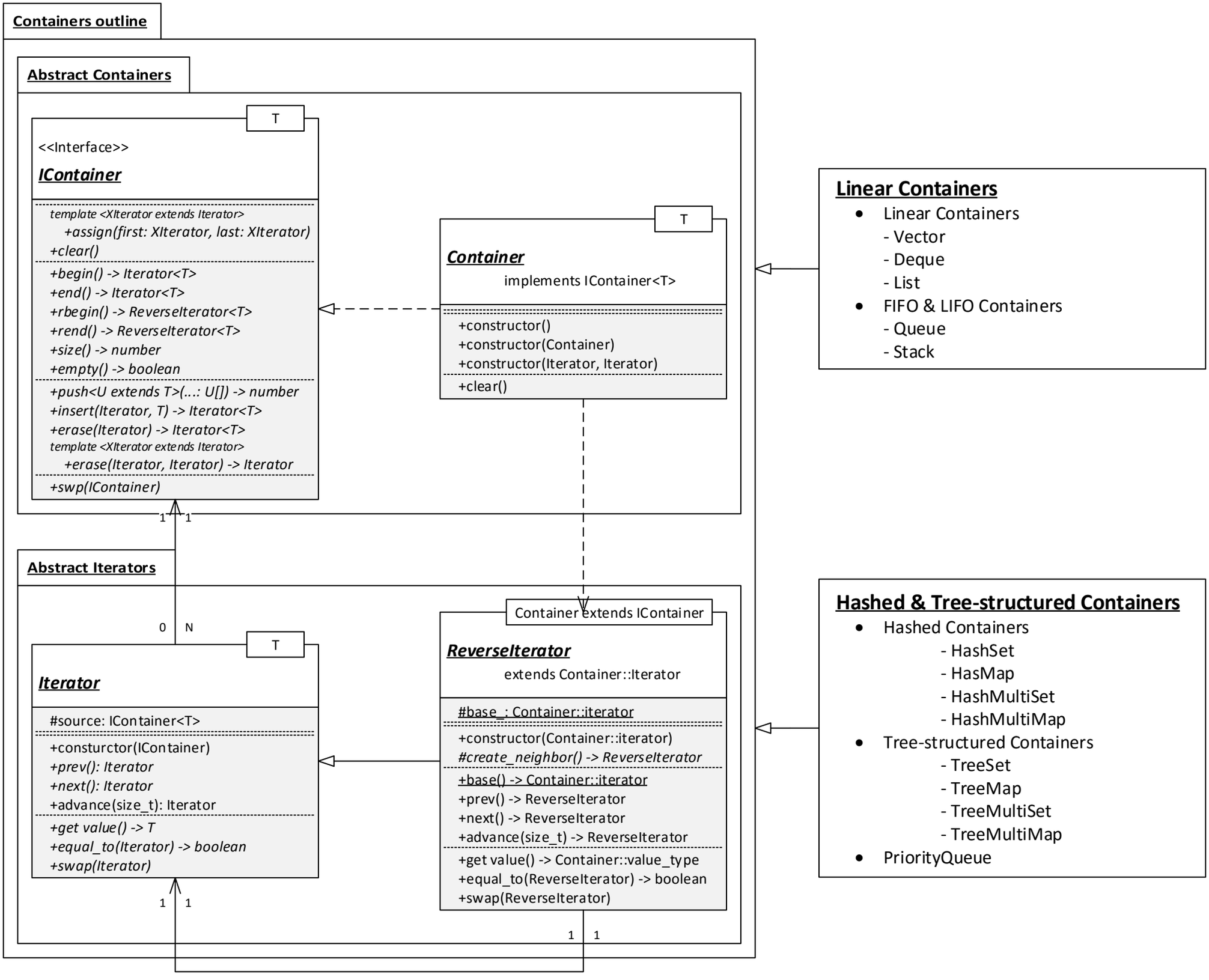
JS Class Diagram

1. STL
2. Library
3. Protocol
4. Example

TypeScript-STL

TypeScript-STL (Standard Template Library)

Basics
Linear Containers
Set Containers
Map Containers


Containers outline
Abstract Containers
T
<<Interface>>
IContainer
template <XIterator extends Iterator>
+assign(first: XIterator, last: XIterator)
+clear()
+begin() -> Iterator<T>
+end() -> Iterator<T>
+rbegin() -> ReverseIterator<T>
+rend() -> ReverseIterator<T>
+size() -> number
+empty() -> boolean
+push<U extends T>(...: U[]) -> number
+insert(Iterator, T) -> Iterator<T>
+erase(Iterator) -> Iterator<T>
template <XIterator extends Iterator>
+erase(Iterator, Iterator) -> Iterator
+swp(IContainer)
T
Container
implements IContainer<T>
+constructor()
+constructor(Container)
+constructor(Iterator, Iterator)
+clear()
1
1
Abstract Iterators
0
N
T
Iterator
#source: IContainer<T>
+consturctor(IContainer)
+prev(): Iterator
+next(): Iterator
+advance(size_t): Iterator
+get value() -> T
+equal_to(Iterator) -> boolean
+swap(Iterator)
1
1
Container extends IContainer
ReverseIterator
extends Container::Iterator
#base_: Container::iterator
+constructor(Container::iterator)
#create_neighbor() -> ReverseIterator
+base() -> Container::iterator
+prev() -> ReverseIterator
+next() -> ReverseIterator
+advance(size_t) -> ReverseIterator
+get value() -> Container::value_type
+equal_to(ReverseIterator) -> boolean
+swap(ReverseIterator)
1
1
Linear Containers
Linear Containers
- Vector
- Deque
- List
FIFO & LIFO Containers
- Queue
- Stack
Hashed & Tree-structured Containers
Hashed Containers
- HashSet
- HasMap
- HashMultiSet
- HashMultiMap
Tree-structured Containers
- TreeSet
- TreeMap
- TreeMultiSet
- TreeMultiMap
PriorityQueue

Linear Containers

FIFO and LIFO Containers

Stack

T

#container_: ILinearContainer<T>

+Stack()
+Stack(Stack)

+top() -> T
+size() -> size_t
+empty() -> boolean

+push(T)
+pop()

Stack

T

#container_: ILinearContainer<T>

+Stack()
+Stack(Stack)

+front() -> T
+back() -> T
+size() -> size_t
+empty() -> boolean

+push(T)
+pop()

1 1 1 1

Interfaces for linear containers

1 1 1 1

<<Interface>>

ILinearContainer

extends IContainer<T>

T

+using super::assign
+assign(n: size_t, val: T)

+front(): T
+back(): T
+push_back(T)
+pop_back()

+using super::insert
+insert(Iterator, size_t, T) -> Iterator
template <XIterator extends Iterator>
+ insert(Iterator, XIterator, XIterator) -> Iterator

<<Interface>>

IDequeContainer

extends ILinearContainer<T>

T

+push_front(T)
+pop_front()

<<Interface>>

IArrayContainer

extends ILinearContainer<T>

T

+reserve(size_t)
+capacity() -> size_t

+at(size_t) -> T
+set(size_t, T)

List container and its Iterators

List

T

extends Container<T>
implements IDequeContainer<T>

#begin_: ListIterator
#end_: ListIterator
#size_: size_t

+override super::constructor
+constructor(size_t, T)
+override super::assign
+clear()

+push<U extends T>(...: U[]): size_t
+override super::insert
+override super::erase

+{<algorithm> methods}

1 1 2 1 N

ListIterator

T

extends Iterator<T>

#prev_: ListIterator
#next_: ListIterator
#value_: T

+constructor
(List, ListIterator, ListIterator, T)
+prev() -> ListIterator
+next() -> ListIterator
+advance(size_t) -> ListIterator

+get value(): T
+set value(T)
+equal_to(ListIterator) -> boolean

+swap(ListIterator)

1 1 1 1

ListReverseIterator

T

extends ReverseIterator<List<T>>

+constructor(ListIterator)
#create_neighbor()
-> ListReverseIterator

+set value(T)

List container and its Iterators

Deque

T

extends Container<T>
implements IArrayContainer<T>,
IDequeContainer<T>

#matrx_: T[][]
#size_: size_t
#capacity_: size_t

+override super::constructor
+constructor(size_t, T)
+override super::assign
+clear()
+reserve(size_t)

+push<U extends T>(...: U[]): size_t
+at(size_t) -> T
+set(size_t, T)

+override super::insert
+override super::erase
-fetch_index(size_t)
-> Pair<size_t, size_t>

1 1 0 1 N

DequeIterator

T

extends Iterator<T>
implements IArrayIterator<T>

#index_: size_t

+constructor(Deque<T>, size_t)
+prev() -> DequeIterator
+next() -> DequeIterator
+advance(size_t) -> DequeIterator

+get index(): size_t
+get value(): T
+set value(T)
+equal_to(DequeIterator) -> boolean

+swap(DequeIterator)

1 1 1 1

DequeReverseIterator

T

extends ReverseIterator<Deque<T>>
implements IArrayIterator<T>

+constructor(DequeIterator)
#create_neighbor()
-> DequeReverseIterator

+get index() -> size_t
+set value(T)

List container and its Iterators

Vector

T

extends Array<T>
implements IArrayContainer<T>

+override super::constructor
+constructor(size_t, T)
+override super::assign
+clear()

+override super::insert
+override super::erase

1 1 0 1 N

VectorIterator

T

extends Iterator<T>

#index_: size_t

+constructor(Vector<T>, size_t)
+prev() -> VectorIterator
+next() -> VectorIterator
+advance(size_t) -> VectorIterator

+getindex(): size_t
+get value(): T
+set value(T)
+equal_to(VectorIterator) -> boolean

+swap(VectorIterator)

1 1 0 1 1

VectorReverseIterator

T

extends ReverseIterator<Vector<T>>
implements IReverseIterator<T>

+constructor(VectorIterator)
#create_neighbor()
-> VectorReverseIterator

+get index() -> size_t
+set value(T)

Set Containers

Abstract Set Containers

SetContainer (T)
extends Container<T>

- #data_: List<T>
- +constructor()
- +constructor(Array<T>)
- +constructor(IContainer<T>)
- template <U extends T, InputIterator extends Iterator<T>>
- +constructor(InputIterator, InputIterator)
- +assign(InputIterator, InputIterator)
- #init()
- +clear()
- *+find(T) -> SetIterator*
- +begin() -> SetIterator
- +end() -> SetIterator
- +rbegin() -> ReverseSetIterator
- +rend() -> ReverseSetIterator
- *+count(T) -> size_t*
- +has(T) -> boolean
- +size() -> size_t
- +push(Pair<L, U>[]): size_t
- *+insert(SetIterator, T) -> SetIterator*
- *template <U extends T, XIterator extends Iterator<U>>*
- *+insert(InputIterator, InputIterator)*
- +erase(T) -> size_t
- template <XIterator extends SetIterator>
- +erase(XIterator, XIterator)
- template <XIterator extends SetIterator<Key, T>>
- #handle_insert(XIterator, XIterator)
- #handle_erase(XIterator, XIterator)

Specified Setss about uniqueness

UniqueSet (T)
extends SetContainer< T>

- +using super::constructor
- +count(Key) -> size_t
- +using super::insert
- *+insert(T) -> Pair<SetIterator, boolean>*
- +swap(UniqueSet)

MultiSet (T)
extends SetContainer<T>

- +using super::constructor
- +using super::insert
- *+insert(T) -> SetIterator*
- +swap(MultiSet)

Iterators

SetIterator (T)
extends Iterator<T>

- #list_it_: ListIterator<T>
- +constructor(SetContainer, ListIterator)
- +prev() -> SetIterator
- +next() -> SetIterator
- +advance(size_t) -> SetIterator
- +get value() -> Pair<Key, T>
- +get first() -> Key
- +get second() -> T
- +set second(T)
- +equal_to(Set_Iterator) -> boolean
- +less(SetIterator) -> boolean
- +hash() -> size_t

SetReverseIterator (T)
extends ReverseIterator <SetContainer<T>>

- +constructor(SetIterator)
- #create_reverse_iterator() -> SetReverseIterator

Final Sets

HashSet (T)
extends SpecifiedSet<T>

- #hash_buckets_: SetHashBuckets
- +using super::constructor
- #init()
- template <U extends T, XIterator extends Iterator<U>>
- +assign(InputIterator, InputIterator)
- +clear()
- **+find(T) -> SetIterator**
- +override super::push
- +override super::insert
- template <XIterator extends SetIterator<T>>
- #handle_insert(XIterator, XIterator)
- #handle_erase(XIterator, XIterator(

Tree(Multi)Set (T)
extends SpecifiedSet<T>

- #tree_: PairTree
- +using super::constructor
- #init()
- template <U extends T, XIterator extends Iterator<U>>
- +assign(InputIterator, InputIterator)
- +clear()
- **+find(T) -> SetIterator**
- +lower_bound(T) -> SetIterator
- +upper_bound(T) -> SetIterator
- +equal_range(T) -> Pair<SetIterator, SetIterator>
- +override super::insert
- template <XIterator extends SetIterator<T>>
- #handle_insert(XIterator, XIterator)
- #handle_erase(XIterator, XIterator(

PriorityQueue (T)
extends TreeMultiSet<T>

- #container_: TreeMultiSet<T>
- +constructor()
- +constructor(T[])
- +constructor(IContainer<T>)
- +constructor(Iterator, Iterator)
- +top() -> T
- +size() -> size_t
- +empty() -> boolean
- +push(T)
- +pop()

Hash functions

MapHashBuckets (T)
extends HashBuckets <SetIterator<T>>

- -set_: SetContainer< T>
- +constructor(SetContainer<T>)
- +find(T) -> SetIterator

HashBuckets (T)

- #buckets_: Array<Array<T>>
- #item_size_: size_t
- +constructor()
- +reserve(size_t)
- +clear()
- +at(size_t) -> Array<T>
- -hash_index(T) -> size_t
- +insert(T)
- +erase(T)

Red-Black Tree

AtomicTree (T)
extends XTree<T>

- -compare_: (Key, Key) => boolean
- +public constructor(Compare = std.less)
- +using super::find
- +find(SetIterator) -> XTreeNode
- template <XIterator extends SetIterator<T>>
- +is_less(XIterator, XIterator) -> boolean
- +is_equal_to(XIterator, XIterator) -> boolean

XTree (T)

- #root_: XTreeNode<T>
- +public constructor()
- +find(T) -> XTreeNode
- #fetch_maximum(XTreeNode) -> XTreeNode
- -fetch_color(XTreeNode) -> Color
- +insert(T)
- +erase(T)
- -insert_case_{1~5}(XTreeNode)
- -erase_case_{1~6}(XTreeNode)
- #rotate_left(XTreeNode)
- #rotate_right(XTreeNode)
- #replace_node(XTreeNode, XTreeNode)
- *+is_less(T, T) -> boolean*
- *+is_equal_to(T, T) -> boolean*

XTreeNode (T)

- parent_: XTreeNode
- left_: XTreeNode
- right_: XTreeNode
- value: T
- color: Color
- +constructor(T, Color)

Map Containers

Abstract Map Containers

MapContainer (Key, T)
extends Container<Pair<Key, T>>

- **#data_: List<Pair<Key, T>>**
- +constructor()
- +constructor(Pair<Key, T>[])
- +constructor(IContainer)
- template <L extends Key, U extends T, XIterator extends Iterator<Pair<L, U>>>
- +constructor(InputIterator, InputIterator)
- +assign(InputIterator, InputIterator)
- #init()
- +clear()
- ***+find(Key) -> MapIterator***
- +begin() -> MapIterator
- +end() -> MapIterator
- +rbegin() -> ReverseMapIterator
- +rend() -> ReverseMapIterator
- *+count(Key) -> size_t*
- +has(Key) -> boolean
- +size() -> size_t
- +push(Pair<L, U>[]): size_t
- *+insert(MapIterator, Pair<Key, T>) -> MapIterator*
- *template <L extends Key, U extends T, XIterator extends Iterator<Pair<L, U>>>*
- *+insert(InputIterator, InputIterator)*
- +erase(Key) -> size_t
- template <XIterator extends MapIterator>
- +erase(XIterator, XIterator)
- template <XIterator extends MapIterator<Key, T>>
- #handle_insert(XIterator, XIterator)
- #handle_erase(XIterator, XIterator)

Specified Maps about uniqueness

UniqueMap (Key, T)
extends MapContainer<<Key, T>

- +using super::constructor
- +count(Key) -> size_t
- +get(Key) -> T
- +set(Key, T)
- +using super::insert
- *+insert(Pair<Key, T>) -> Pair<MapIterator, boolean>*
- +swap(UniqueMap)

MultiMap (Key, T)
extends MapContainer<<Key, T>

- +using super::constructor
- +using super::insert
- *+insert(Pair<Key, T>) -> MapIterator*
- +swap(MultiMap)

Iterators

MapIterator (Key, T)
extends Iterator<Pair<Key, T>>

- #list_it_: ListIterator<Pair<Key, T>>
- +constructor(MapContainer, ListIterator)
- +prev() -> MapIterator
- +next() -> MapIterator
- +advance(size_t) -> MapIterator
- +get value() -> Pair<Key, T>
- +get first() -> Key
- +get second() -> T
- +set second(T)
- +equal_to(MapIterator) -> boolean
- +less(MapIterator) -> boolean
- +hash() -> size_t
- +swap(MapIterator)

MapReverseIterator (Key, T)
extends ReverseIterator<MapContainer<Key, T>>

- +constructor(MapIterator)
- #create_neighbor() -> MapReverseIterator
- +get first() -> Key
- +get second() -> T
- +set second(T)

Final Maps

HashMap (Key, T)
extends SpecifiedMap<Key, T>

- **#hash_buckets_: MapHashBuckets**
- +override super::constructor
- #init()
- template <L extends Key, U extends T, XIterator extends Iterator<Pair<L, U>>>
- +assign(InputIterator, InputIterator)
- +clear()
- **+find(Key) -> MapIterator**
- +override super::push
- +override super::insert
- template <XIterator extends MapIterator<Key, T>>
- #handle_insert(XIterator, XIterator)
- #handle_erase(XIterator, XIterator(

Tree(Multi)Map (Key, T)
extends SpecifiedMap<Key, T>

- **#tree_: PairTree**
- +override super::constructor
- #init()
- template <L extends Key, U extends T, XIterator extends Iterator<Pair<L, U>>>
- +assign(InputIterator, InputIterator)
- +clear()
- **+find(Key) -> MapIterator**
- +lower_bound(Key) -> MapIterator
- +upper_bound(Key) -> MapIterator
- +equal_range(Key) -> Pair<MapIterator, MapIterator>
- +override super::insert
- template <XIterator extends MapIterator<Key, T>>
- #handle_insert(XIterator, XIterator)
- #handle_erase(XIterator, XIterator(

Hash functions

MapHashBuckets (Key, T)
extends HashBuckets<MapIterator<Key, T>>

- -map_: MapContainer<Key, T>
- +constructor(MapContainer<Key, T>)
- +find(Key) -> MapIterator

HashBuckets (T)

- #buckets_: Array<Array<T>>
- #item_size_: size_t
- +constructor()
- +reserve(size_t)
- +clear()
- +at(size_t) -> Array<T>
- -hash_index(T) -> size_t
- +insert(T)
- +erase(T)

Red-Black Tree

PairTree (Key, T)
extends XTree<Pair<Key, T>>

- -compare_: (Key, Key) => boolean
- +public constructor(Compare = std.less)
- +using super::find
- +find(MapIterator) -> XTreeNode
- template <XIterator extends MapIterator<Key, T>>
- +is_less(XIterator, XIterator) -> boolean
- +is_equal_to(XIterator, XIterator) -> boolean

XTree (T)

- #root_: XTreeNode<T>
- +public constructor()
- +find(T) -> XTreeNode
- #fetch_maximum(XTreeNode) -> XTreeNode
- -fetch_color(XTreeNode) -> Color
- +insert(T)
- +erase(T)
- -insert_case_{1~5}(XTreeNode)
- -erase_case_{1~6}(XTreeNode)
- #rotate_left(XTreeNode)
- #rotate_right(XTreeNode)
- #replace_node(XTreeNode, XTreeNode)
- *+is_less(T, T) -> boolean*
- *+is_equal_to(T, T) -> boolean*

XTreeNode (T)

- parent_: XTreeNode
- left_: XTreeNode
- right_: XTreeNode
- value: T
- color: Color
- +constructor(T, Color)

Utilities

File References

FileReferenceList
extends EventDispatcher

- -file_list: Vector<FileReference>

- +constructor()
- +browse(...extensions: string[])

- +get fileList() -> Vector<FileReference>

FileReference
extends EventDispatcher

- -file_: File
- -data_: string | Buffer

- +constructor()
- +browse(...extensions: string)
- +load()

- +get data() -> string | Buffer
- +get name() -> string
- +get extension() -> string
- +get modificationDate() -> Date

- +save(data: string | Buffer, name: string)

Events

<<Interface>>
IEventDispatcher

- +hasEventListener(string) -> boolean
- +dispatchEvent(Event) -> boolean

- +addEventListener(string, Listener)
- +removeEventListener(string, Listener)

EventDispatcher
implements IEventDispatcher

- -event_dispatcher_: IEventDispatcher
- -event_listeners_: HashMap
  <string, Pair<Listener, Object>>

- +constructor()
- +constructor(IEventDispatcher)

BasicEvent
implements Event

- -type_: string
- -target_: IEventDispatcher
- -current_target_: IEventDispatcher
- -timestamp_: Date

- +constructor(string, boolean, boolean)

XML & String Utils

XML
extends HashMap<string, XMLList>

- -tag: string
- -value: string
- -properties: HashMap<string, string>

- +constructor()
- +constructor(string)

- +hasProperty(string) -> boolean
- +setProperty(string, string)
- +getProperty(string) -> string
- +push(...: XML[])
- +push(...: XMLList[])

- +toString() -> string
- +toHTML() -> string

XMLList
extends Deque<XML>

- +using super::constructor

- +toString() -> string
- +toHTML() -> string

<<Static>>
StringUtil

- -SPACE_ARRAY: string[]

- +substitute(string, ...args: any[]) -> string
- +replaceAll(string, string, string) -> string
- +replaceAll(string, Pair[]) -> string

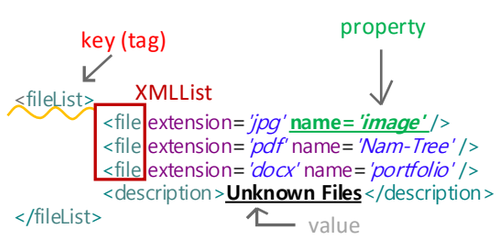
- +trim(string, ...args: any[]) -> string
- +ltrim(string, ...args: any[]) -> string
- +rtrim(string, ...args: any[]) -> string

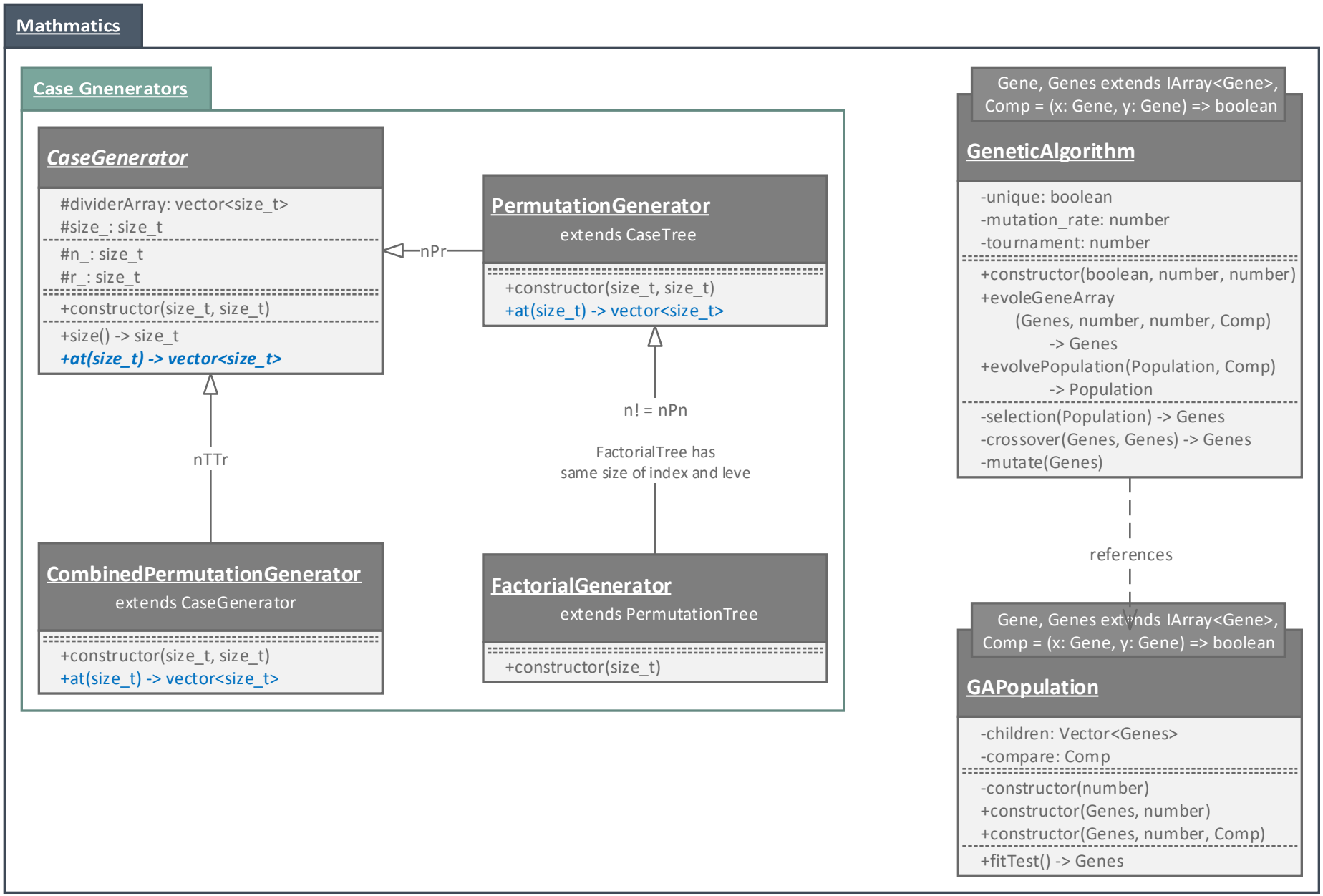
- +between(string, string, string) -> string
- +betweens(string, string, string)
  -> string[]

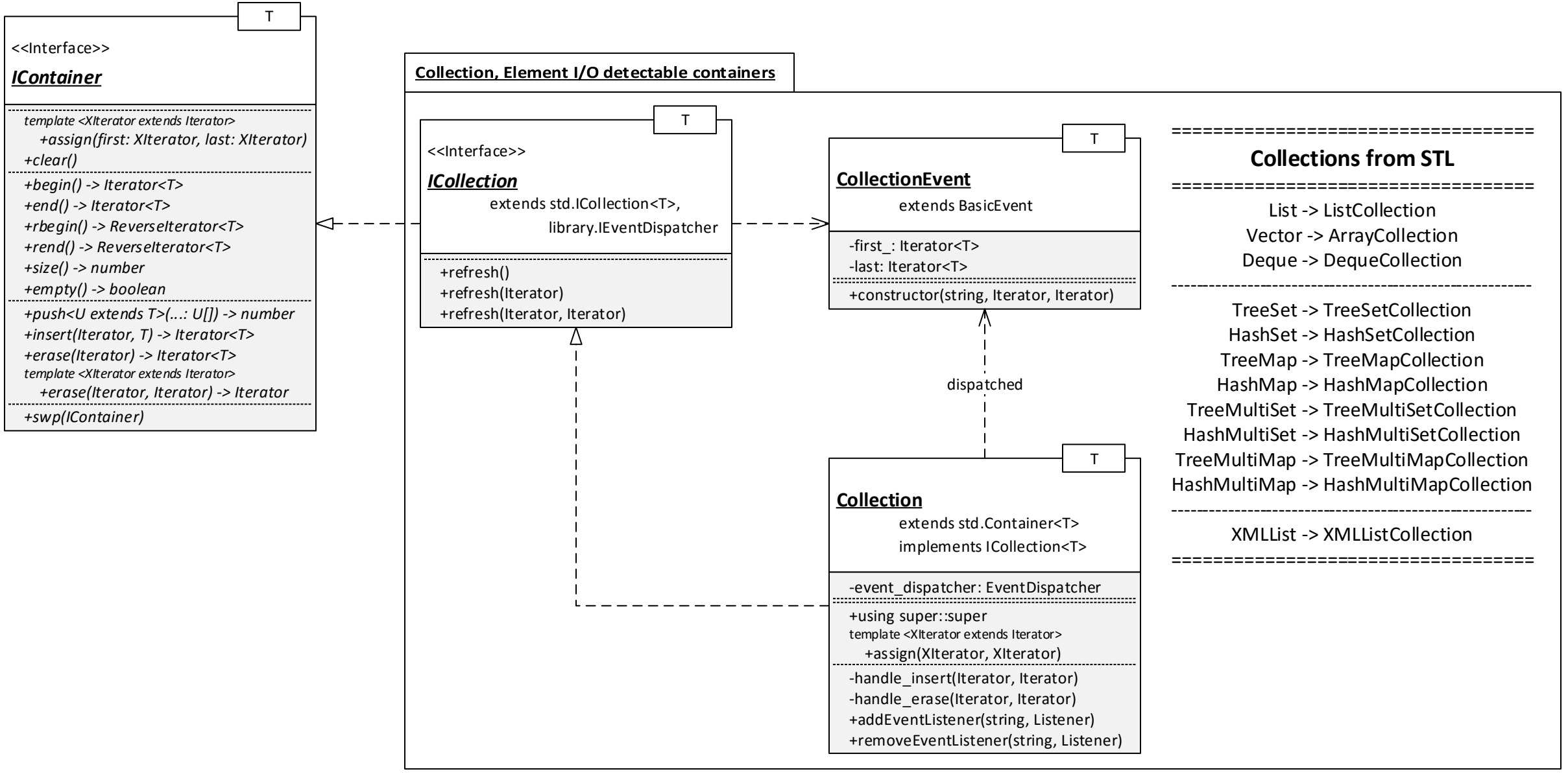
URLVariables
extends HashMap<string, string>

- +constructor()
- +constructor(string)

- +decode(string)
- +toString() -> string




Mathmatics
Case Gnenerators
CaseGenerator
#dividerArray: vector<size_t>
#size_: size_t
#n_: size_t
#r_: size_t
+constructor(size_t, size_t)
+size() -> size_t
+at(size_t) -> vector<size_t>
nPr
PermutationGenerator
extends CaseTree
+constructor(size_t, size_t)
+at(size_t) -> vector<size_t>
n! = nPn
FactorialTree has
same size of index and leve
nTTr
CombinedPermutationGenerator
extends CaseGenerator
+constructor(size_t, size_t)
+at(size_t) -> vector<size_t>
FactorialGenerator
extends PermutationTree
+constructor(size_t)
Gene, Genes extends IArray<Gene>,
Comp = (x: Gene, y: Gene) => boolean
GeneticAlgorithm
-unique: boolean
-mutation_rate: number
-tournament: number
+constructor(boolean, number, number)
+evoleGeneArray
(Genes, number, number, Comp)
-> Genes
+evolvePopulation(Population, Comp)
-> Population
-selection(Population) -> Genes
-crossover(Genes, Genes) -> Genes
-mutate(Genes)
references
Gene, Genes extends IArray<Gene>,
Comp = (x: Gene, y: Gene) => boolean
GAPopulation
-children: Vector<Genes>
-compare: Comp
-constructor(number)
+constructor(Genes, number)
+constructor(Genes, number, Comp)
+fitTest() -> Genes


T
<<Interface>>
IContainer
template <XIterator extends Iterator>
+assign(first: XIterator, last: XIterator)
+clear()
+begin() -> Iterator<T>
+end() -> Iterator<T>
+rbegin() -> ReverseIterator<T>
+rend() -> ReverseIterator<T>
+size() -> number
+empty() -> boolean
+push<U extends T>(...: U[]) -> number
+insert(Iterator, T) -> Iterator<T>
+erase(Iterator) -> Iterator<T>
template <XIterator extends Iterator>
+erase(Iterator, Iterator) -> Iterator
+swp(IContainer)
Collection, Element I/O detectable containers
T
<<Interface>>
ICollection
extends std.ICollection<T>,
library.IEventDispatcher
+refresh()
+refresh(Iterator)
+refresh(Iterator, Iterator)
T
CollectionEvent
extends BasicEvent
-first_: Iterator<T>
-last: Iterator<T>
+constructor(string, Iterator, Iterator)
dispatched
T
Collection
extends std.Container<T>
implements ICollection<T>
-event_dispatcher: EventDispatcher
+using super::super
template <XIterator extends Iterator>
+assign(XIterator, XIterator)
-handle_insert(Iterator, Iterator)
-handle_erase(Iterator, Iterator)
+addEventListener(string, Listener)
+removeEventListener(string, Listener)
Collections from STL
List -> ListCollection
Vector -> ArrayCollection
Deque -> DequeCollection
TreeSet -> TreeSetCollection
HashSet -> HashSetCollection
TreeMap -> TreeMapCollection
HashMap -> HashMapCollection
TreeMultiSet -> TreeMultiSetCollection
HashMultiSet -> HashMultiSetCollection
TreeMultiMap -> TreeMultiMapCollection
HashMultiMap -> HashMultiMapCollection
XMLList -> XMLListCollection

Protocol

Basic components
Invoke
Entity
Cloud Service
External System
Parallel System
Distributed system

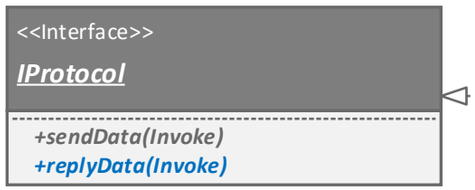
Basic Components of Protocol

Basic Components of Protocol

You can construct any type of network system, even how the system is enormously scaled and complicated, by just combinating the basic components.

All the system templates in this framework are also being implemented by extending and combination of the **basic components**.

- Service
- External System
- Parallel System
- Distributed System



IProtocol

IProtocol is an interface for **Invoke** message, standard message of network I/O in Samchon Framework, chain.

IProtocol is used in network drivers (ICommunicator) or some classes which are in a relationship of chain of responsibility of those network drivers (**ICommunicator** objects) and handling **Invoke** messages.

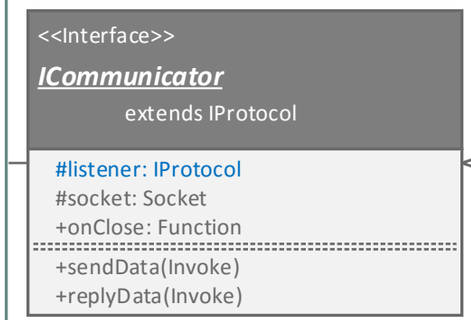
You can see that all classes with related network I/O and handling **Invoke** message are implementing the **IProtocol** interface with **IServer** and **communicator classes**.

Communicators

ICommunicator

ICommunicator takes full charge of network comunication with external system without reference to whether the external system is a server or a client.

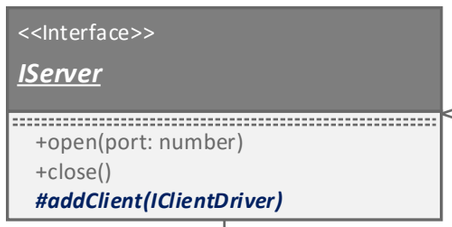
Whenever a replied message has arrived, the message will be converted to an **Invoke** class and will be shifted to the **listener**'s **replyData()** .

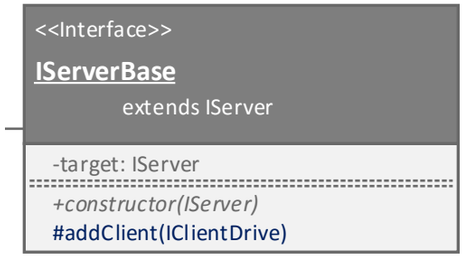


IServerConnector

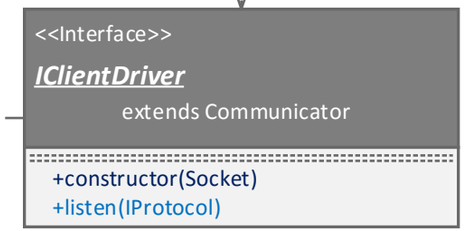
IServerConnector is a server connector who can connect to an external server system as a client.

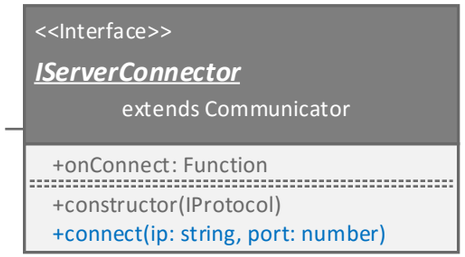
IServerConnector is extended from the **ICommunicator**, thus, it also takes full charge of network communication and delivers replied message to **listener**'s **replyData()**.





creates whenever client connected





IServer

The easiest way to defining a server class is to extending one of them, who are derived from the **IServer**.

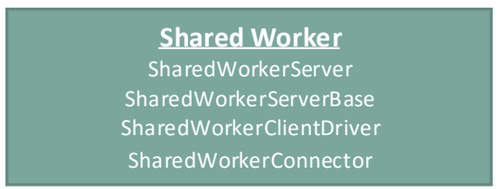
- **Server**
- **WebServer**
- **SharedWorkerServer**

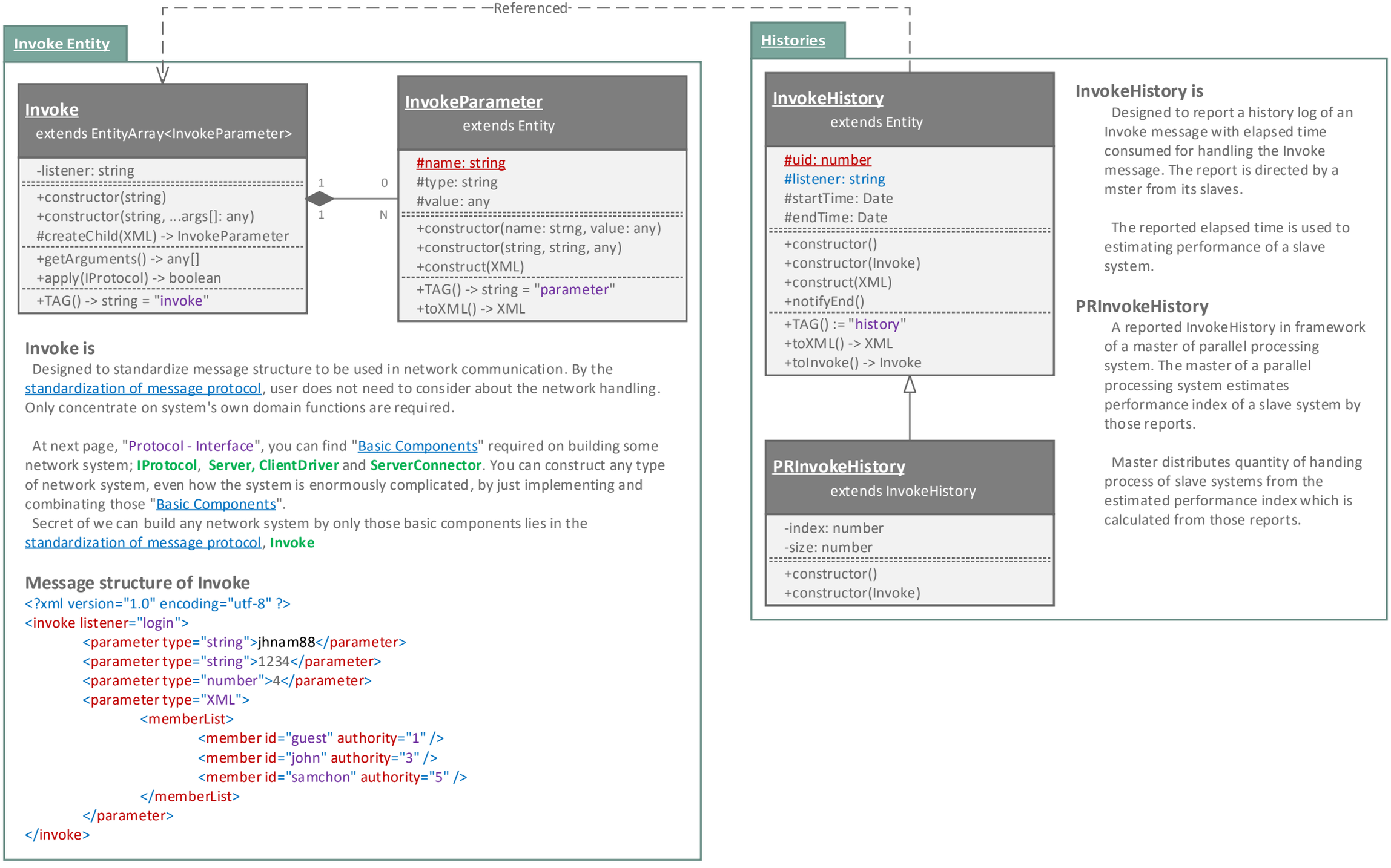
Whenever a client has newly connected, then **addClient()** will be called with a **IClientDriver** object, who takes responsibility of network communication with the client.

IServerBase

However, it is impossible (that is, if the class is already extending another class), you can instead implement the **IServer** interface, create an **IServerBase** member, and write simple hooks to route calls into the aggregated **IServerBase**.

Derived Communicators





Invoke is

Designed to standardize message structure to be used in network communication. By the standardization of message protocol, user does not need to consider about the network handling. Only concentrate on system's own domain functions are required.

At next page, "Protocol - Interface", you can find "Basic Components" required on building some network system; **IProtocol**, **Server, ClientDriver** and **ServerConnector**. You can construct any type of network system, even how the system is enormously complicated, by just implementing and combinating those "Basic Components".

Secret of we can build any network system by only those basic components lies in the standardization of message protocol, **Invoke**

Message structure of Invoke

```
<?xml version="1.0" encoding="utf-8" ?>
<invoke listener="login">
        <parameter type="string">jhnam88</parameter>
        <parameter type="string">1234</parameter>
        <parameter type="number">4</parameter>
        <parameter type="XML">
                <memberList>
                        <member id="guest" authority="1" />
                        <member id="john" authority="3" />
                        <member id="samchon" authority="5" />
                </memberList>
        </parameter>
</invoke>
```

InvokeHistory is

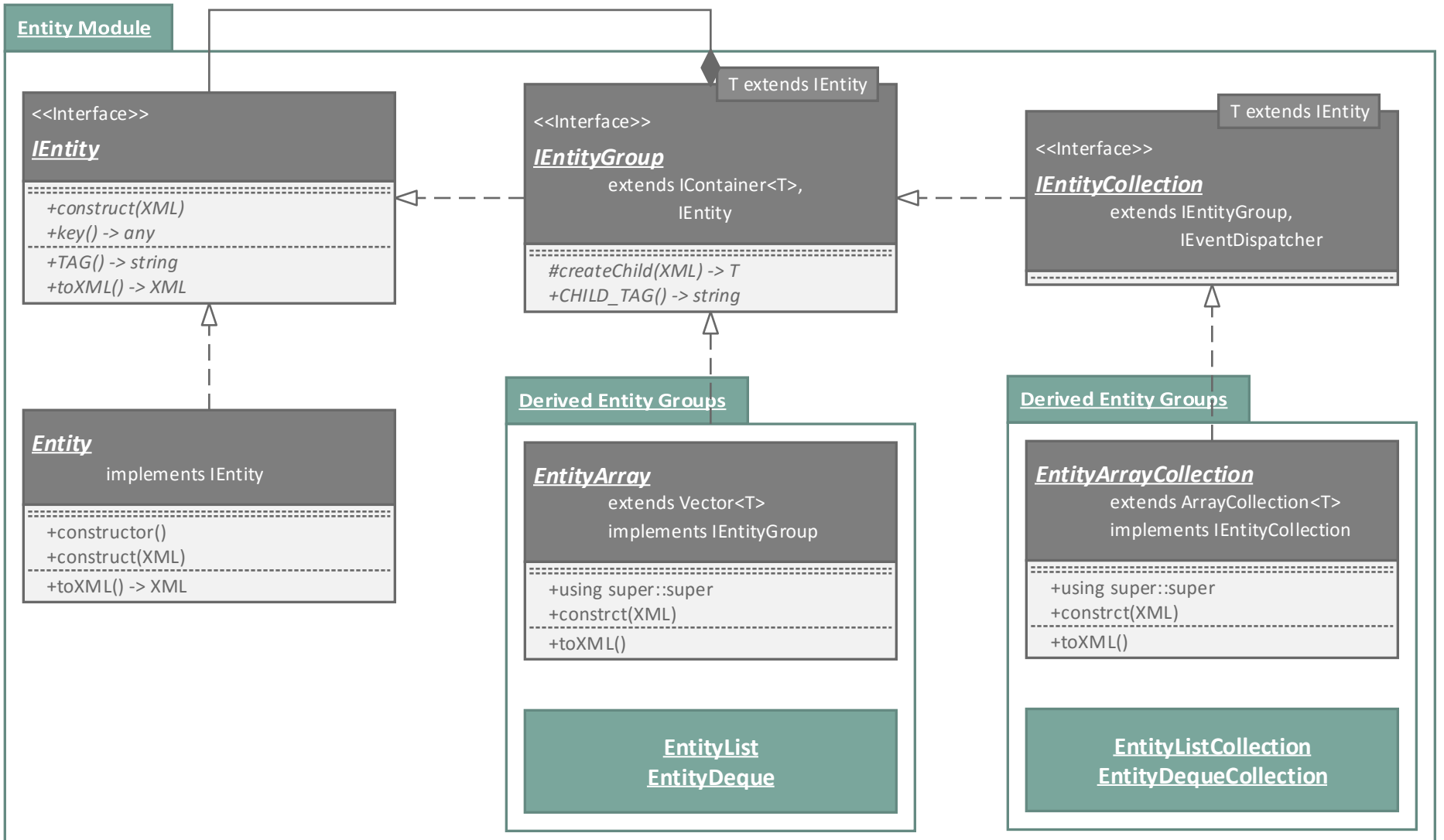
Designed to report a history log of an Invoke message with elapsed time consumed for handling the Invoke message. The report is directed by a mster from its slaves.

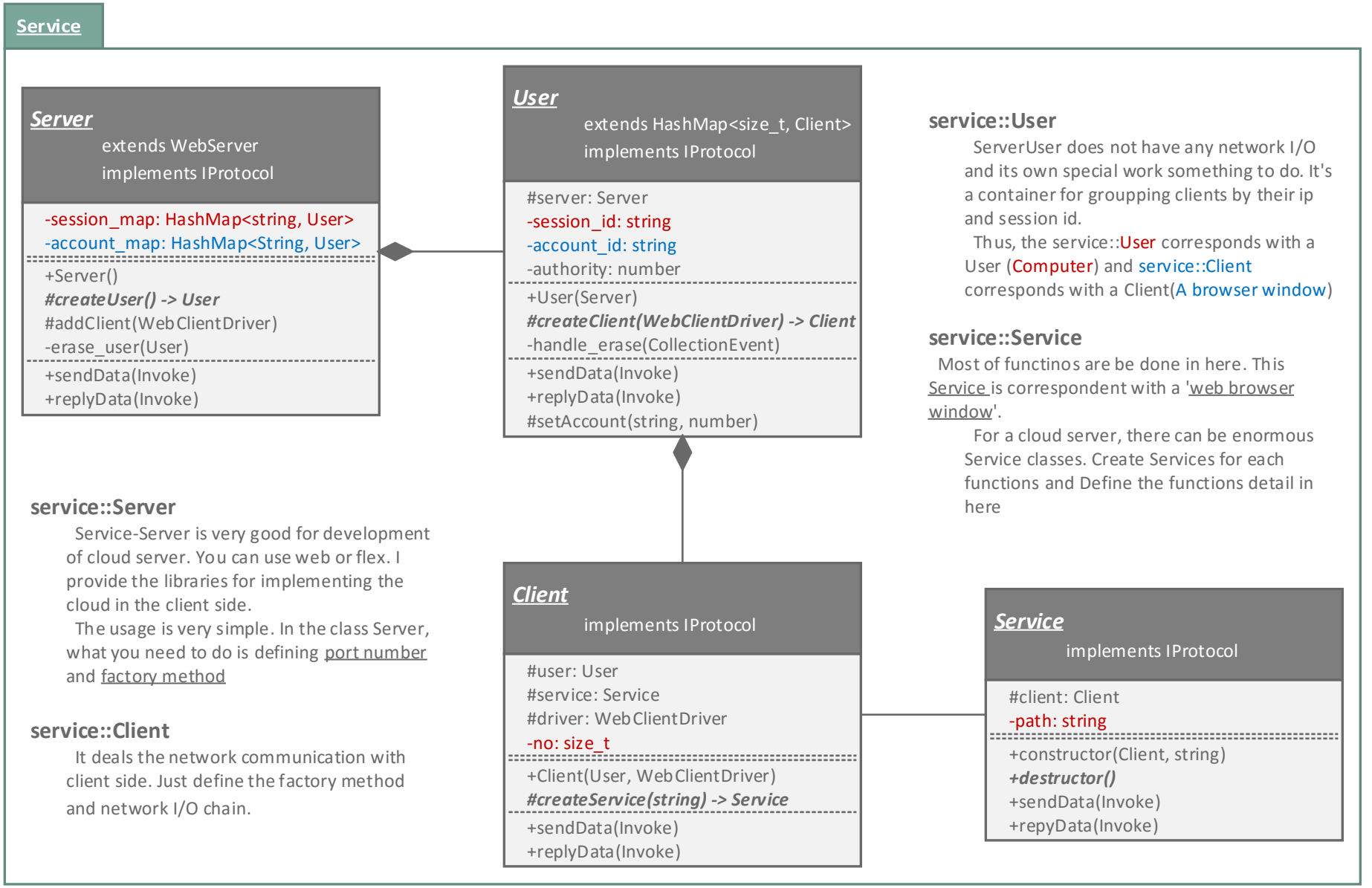
The reported elapsed time is used to estimating performance of a slave system.

PRInvokeHistory

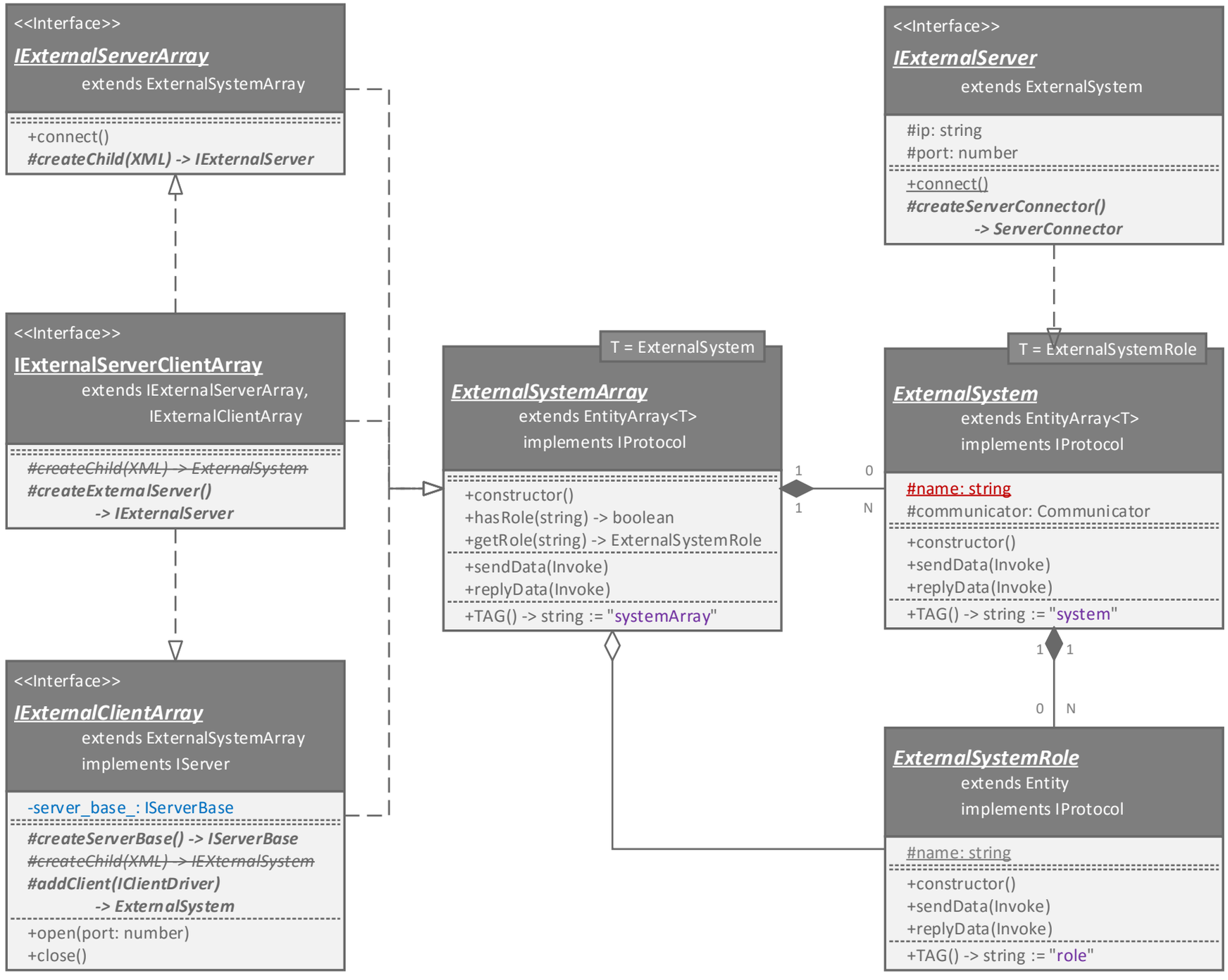
A reported InvokeHistory in framework of a master of parallel processing system. The master of a parallel processing system estimates performance index of a slave system by those reports.

Master distributes quantity of handing process of slave systems from the estimated performance index which is calculated from those reports.


Entity Module
<<Interface>>
IEntity
+construct(XML)
+key() -> any
+TAG() -> string
+toXML() -> XML
T extends IEntity
<<Interface>>
IEntityGroup
extends IContainer<T>,
IEntity
#createChild(XML) -> T
+CHILD_TAG() -> string
T extends IEntity
<<Interface>>
IEntityCollection
extends IEntityGroup,
IEventDispatcher
Entity
implements IEntity
+constructor()
+construct(XML)
+toXML() -> XML
Derived Entity Groups
EntityArray
extends Vector<T>
implements IEntityGroup
+using super::super
+constrct(XML)
+toXML()
EntityList
EntityDeque
Derived Entity Groups
EntityArrayCollection
extends ArrayCollection<T>
implements IEntityCollection
+using super::super
+constrct(XML)
+toXML()
EntityListCollection
EntityDequeCollection


Service
Server
extends WebServer
implements IProtocol
-session_map: HashMap<string, User>
-account_map: HashMap<String, User>
+Server()
#createUser() -> User
#addClient(WebClientDriver)
-erase_user(User)
+sendData(Invoke)
+replyData(Invoke)
User
extends HashMap<size_t, Client>
implements IProtocol
#server: Server
-session_id: string
-account_id: string
-authority: number
+User(Server)
#createClient(WebClientDriver) -> Client
-handle_erase(CollectionEvent)
+sendData(Invoke)
+replyData(Invoke)
#setAccount(string, number)
service::User
ServerUser does not have any network I/O and its own special work something to do. It's a container for groupping clients by their ip and session id.
Thus, the service::User corresponds with a User (Computer) and service::Client corresponds with a Client(A browser window)
service::Service
Most of functinos are be done in here. This Service is correspondent with a 'web browser window'.
For a cloud server, there can be enormous Service classes. Create Services for each functions and Define the functions detail in here
service::Server
Service-Server is very good for development of cloud server. You can use web or flex. I provide the libraries for implementing the cloud in the client side.
The usage is very simple. In the class Server, what you need to do is defining port number and factory method
service::Client
It deals the network communication with client side. Just define the factory method and network I/O chain.
Client
implements IProtocol
#user: User
#service: Service
#driver: WebClientDriver
-no: size_t
+Client(User, WebClientDriver)
#createService(string) -> Service
+sendData(Invoke)
+replyData(Invoke)
Service
implements IProtocol
#client: Client
-path: string
+constructor(Client, string)
+destructor()
+sendData(Invoke)
+repyData(Invoke)

External Systems



ExternalSystemArray

This class set will be very useful for constructing parallel distributed processing system.
Register distributed systems on **ExternalSystemArray** and manage their roles, and then communicate based on role.

ExternalSystem

If an external system is a server that I've to connect, then implements **IExternalServer** and define the abstract method, ***createServerConnector***().
Meanwhile, an external system is a client who connects to my server, then nothing to define especially.

ExternalSystemRole

ExternalSystemArray and ExternalSystem expresses the physical relationship between your system(master) and the external system.
But ExternalSystemRole enables to have a new, logical relationship between your system and external servers.

You just only need to concentrate on the role what external systems have to do.
Just register and manage the Role of each external system and you just access and orders to the external system by their role

Access by Role

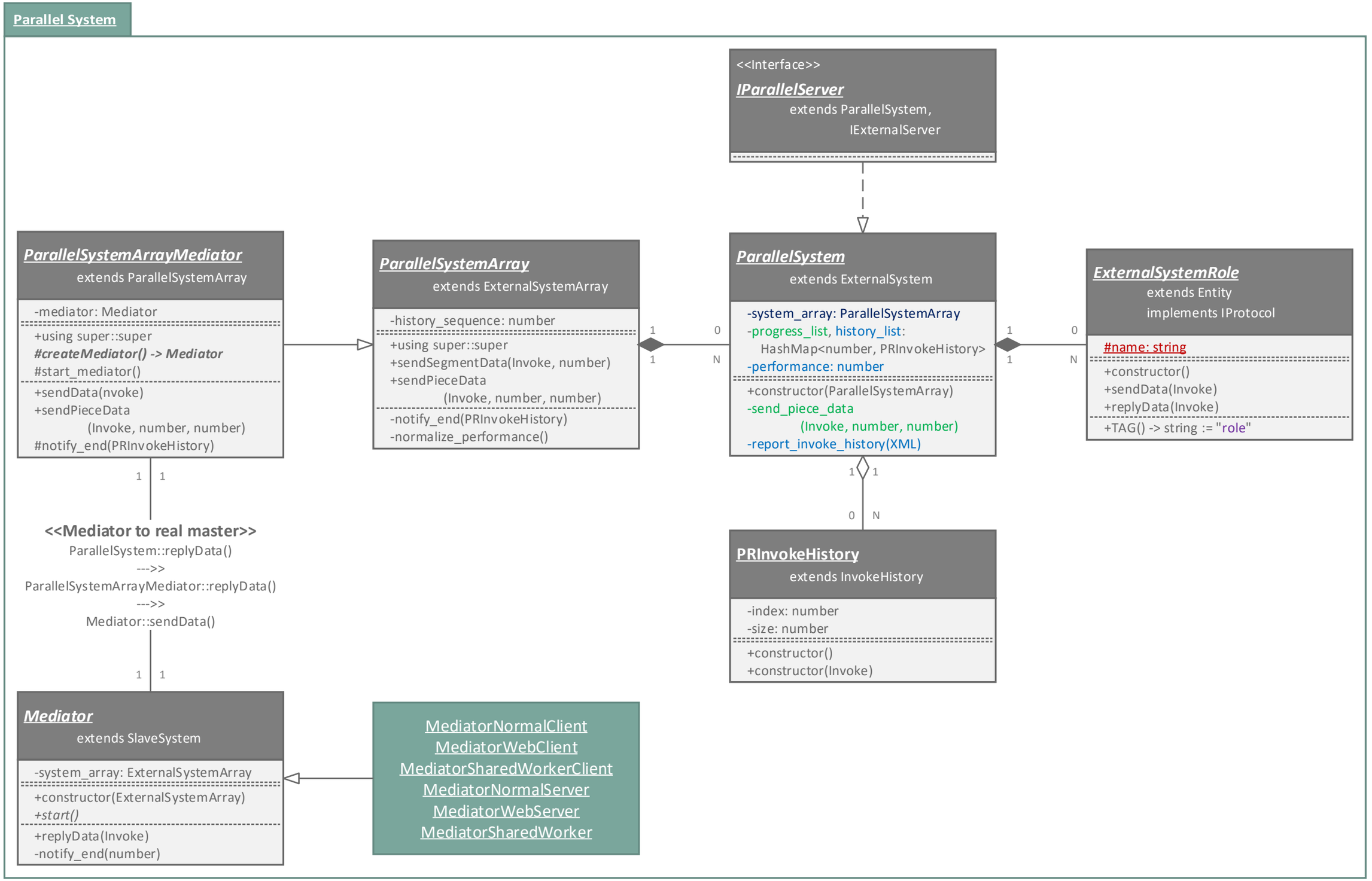
```
ExternalSystemArray *master;
ExternalSystemRole *role = master->getRole(String);
role->sendData(invoke)
```

Derived Modules

Parallel System

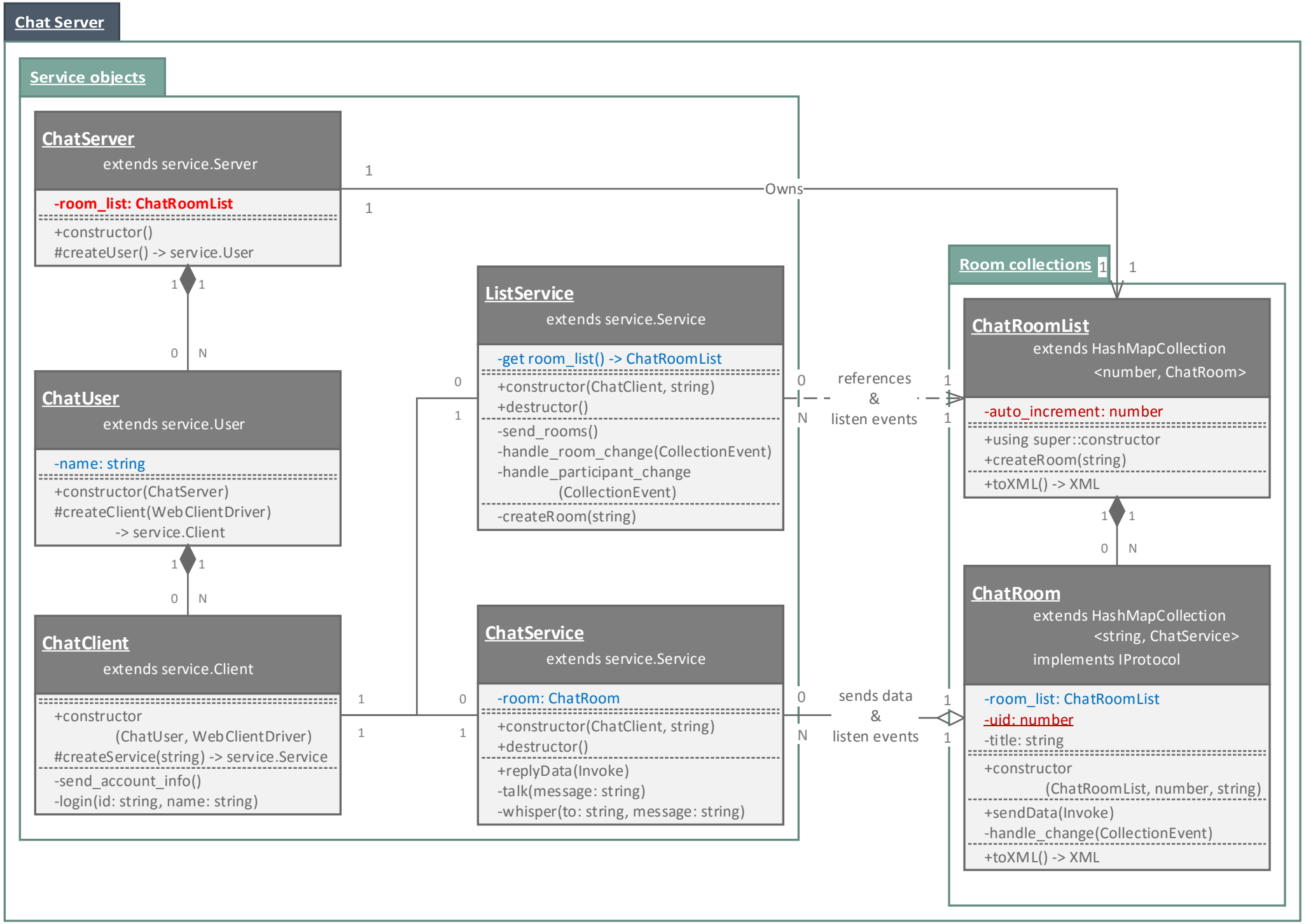
Distributed System

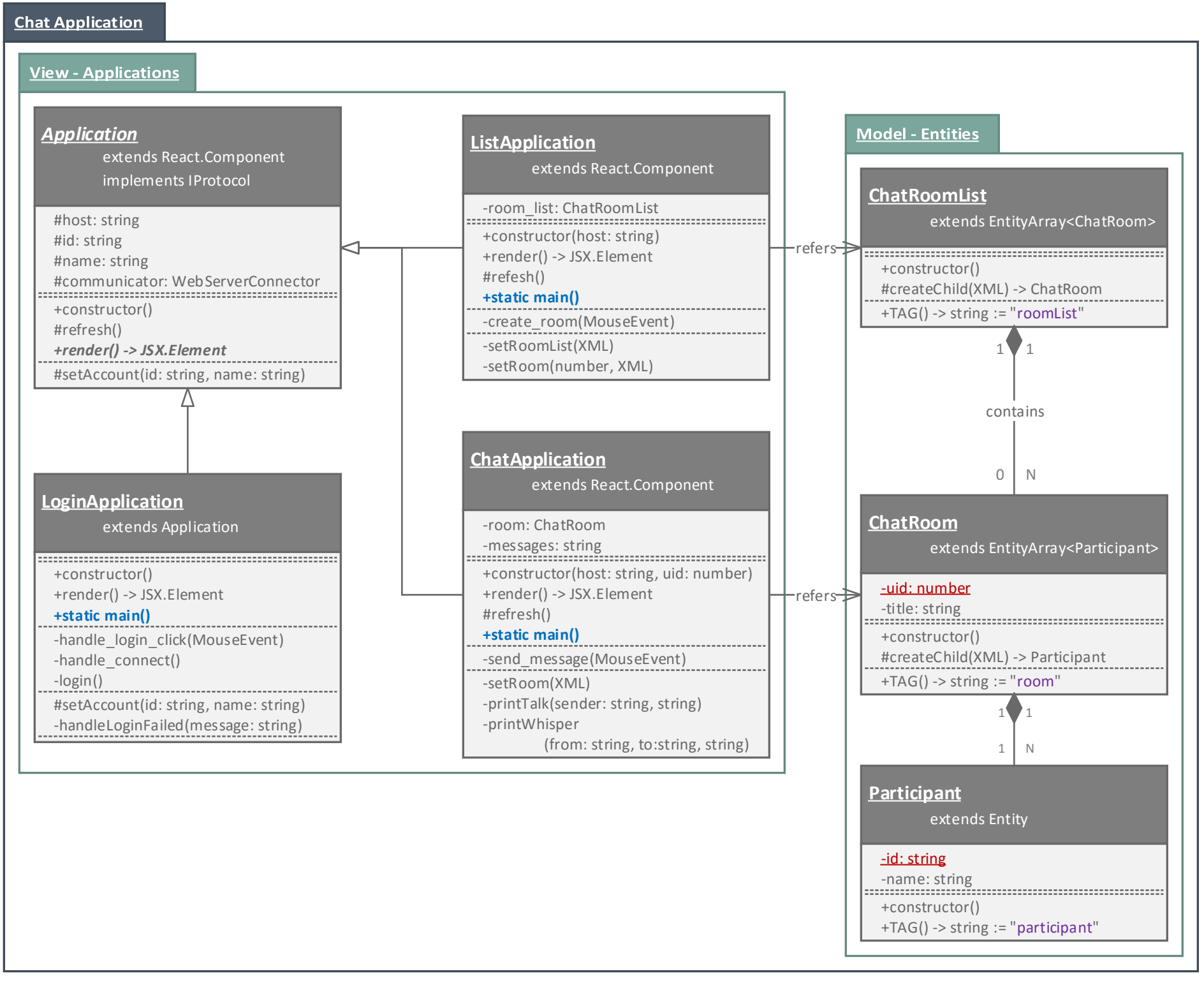
Slave System


Parallel System
<<Interface>>
IParallelServer
extends ParallelSystem,
IExternalServer
ParallelSystemArrayMediator
extends ParallelSystemArray
-mediator: Mediator
+using super::super
#createMediator() -> Mediator
#start_mediator()
+sendData(nvoke)
+sendPieceData
(Invoke, number, number)
#notify_end(PRInvokeHistory)
ParallelSystemArray
extends ExternalSystemArray
-history_sequence: number
+using super::super
+sendSegmentData(Invoke, number)
+sendPieceData
(Invoke, number, number)
-notify_end(PRInvokeHistory)
-normalize_performance()
ParallelSystem
extends ExternalSystem
-system_array: ParallelSystemArray
-progress_list, history_list:
HashMap<number, PRInvokeHistory>
-performance: number
+constructor(ParallelSystemArray)
-send_piece_data
(Invoke, number, number)
-report_invoke_history(XML)
ExternalSystemRole
extends Entity
implements IProtocol
#name: string
+constructor()
+sendData(Invoke)
+replyData(Invoke)
+TAG() -> string := "role"
1
1
0
N
1
1
0
N
1
1
0
N
<<Mediator to real master>>
ParallelSystem::replyData()
--->>
ParallelSystemArrayMediator::replyData()
--->>
Mediator::sendData()
1
1
1
1
PRInvokeHistory
extends InvokeHistory
-index: number
-size: number
+constructor()
+constructor(Invoke)
Mediator
extends SlaveSystem
-system_array: ExternalSystemArray
+constructor(ExternalSystemArray)
+start()
+replyData(Invoke)
-notify_end(number)
MediatorNormalClient
MediatorWebClient
MediatorSharedWorkerClient
MediatorNormalServer
MediatorWebServer
MediatorSharedWorker

Example

Chat-Server & Chat-Application Interaction


Chat Server
Service objects
ChatServer
extends service.Server
-room_list: ChatRoomList
+constructor()
#createUser() -> service.User
Owns
1
1
1
1
0
N
ChatUser
extends service.User
-name: string
+constructor(ChatServer)
#createClient(WebClientDriver)
-> service.Client
1
1
0
N
ChatClient
extends service.Client
+constructor
(ChatUser, WebClientDriver)
#createService(string) -> service.Service
-send_account_info()
-login(id: string, name: string)
1
1
ListService
extends service.Service
-get room_list() -> ChatRoomList
+constructor(ChatClient, string)
+destructor()
-send_rooms()
-handle_room_change(CollectionEvent)
-handle_participant_change
(CollectionEvent)
-createRoom(string)
0
1
ChatService
extends service.Service
-room: ChatRoom
+constructor(ChatClient, string)
+destructor()
+replyData(Invoke)
-talk(message: string)
-whisper(to: string, message: string)
0
1
0
N
references
&
listen events
sends data
&
listen events
1
1
Room collections
1
1
ChatRoomList
extends HashMapCollection
<number, ChatRoom>
-auto_increment: number
+using super::constructor
+createRoom(string)
+toXML() -> XML
1
1
0
N
ChatRoom
extends HashMapCollection
<string, ChatService>
implements IProtocol
-room_list: ChatRoomList
-uid: number
-title: string
+constructor
(ChatRoomList, number, string)
+sendData(Invoke)
-handle_change(CollectionEvent)
+toXML() -> XML


Chat Application
View - Applications
Application
extends React.Component
implements IProtocol
#host: string
#id: string
#name: string
#communicator: WebServerConnector
+constructor()
#refresh()
+render() -> JSX.Element
#setAccount(id: string, name: string)
ListApplication
extends React.Component
-room_list: ChatRoomList
+constructor(host: string)
+render() -> JSX.Element
#refesh()
+static main()
-create_room(MouseEvent)
-setRoomList(XML)
-setRoom(number, XML)
LoginApplication
extends Application
+constructor()
+render() -> JSX.Element
+static main()
-handle_login_click(MouseEvent)
-handle_connect()
-login()
#setAccount(id: string, name: string)
-handleLoginFailed(message: string)
ChatApplication
extends React.Component
-room: ChatRoom
-messages: string
+constructor(host: string, uid: number)
+render() -> JSX.Element
#refresh()
+static main()
-send_message(MouseEvent)
-setRoom(XML)
-printTalk(sender: string, string)
-printWhisper
(from: string, to:string, string)
refers
refers
Model - Entities
ChatRoomList
extends EntityArray<ChatRoom>
+constructor()
#createChild(XML) -> ChatRoom
+TAG() -> string := "roomList"
1
1
contains
0
N
ChatRoom
extends EntityArray<Participant>
-uid: number
-title: string
+constructor()
#createChild(XML) -> Participant
+TAG() -> string := "room"
1
1
1
N
Participant
extends Entity
-id: string
-name: string
+constructor()
+TAG() -> string := "participant"

Interaction

node chief

Chief
extends ExternalServerArray

+constructor()
#createChild(XML) -> ExternalSystem
+solveTSP()
+solvePacker()
+replyData(Invoke)

1 1
TSP, Packer & Reporter
3 3

MasterDriver
extends ExternalServer

-chief: Chief
+constructor(Chief, string)
#createConnector() -> IServerConnector
+replyData(Invoke)

Chief system manages Master systems.
Chief system orders optimization processes to each Master system and get reported the optimization results from those Master systems

The Chief system is built for providing a guidance for **external system module**.

You can learn how to integrate with external network system following the example, Chief system.

connects

Master Systems

node reporter

Reporter
extends ChiefDriver

+constructor()
+replyData(Invoke)
-printTSP(XML)
-printPacker(XML)
+static main()

Reporter system prints optimization results on screen which are gotten from Chief system

Of course, the optimization results came from Chief system are came from Master systems and even the Master systems also got those optimization results from those own slave systems.

Report system is built for be helpful for users to comprehend using chain of responsibility pattern in network level.

Master systems are built for providing a guidance of building parallel processing systems in master side. You can study how to utilize master module in protocol following the example. You also can understand external system module; how to interact with external network systems.

Master system gets order of optimization with its basic data from Chief system and shifts the responsibility of optimization process to its Slave systems. When the Slave systems report each optimization result, Master system aggregates and deducts the best solution between them, and report the result to the Chief system.

Note that, Master systems get orders from Chief system, however Master is not a client for the Chief system. It's already acts a role of server even for the Chief system.

Abstract master classes

ChiefDriver
extends WebServer
implements IProtocol

-master: Master
-client_drvier: WebClientDriver
+constructor(Master)
+addClient(WebClientDriver)
+sendData(Invoke)
+replyData(Invoke)

1 1
1 1

Master
extends ParallelClientArray

#chiefDriver: ChiefDriver
#completed_count: number
#requested_size: number
+constructor(port_for_chief: number)
#createServerBase() -> IServerBase
#createExternalClient(IClientDriver) -> ExternalSystem
#optimize(XML)
#replyOptimization(XML)

1 1
contains and manages
0 N

SlaveDriver
extends ParallelSystem

+using super::constructor
+replyOptimization(XML)

ChiefDriver is a weird server that accepts only a client, Chief system. It takes a role of communicating with the Chief sytem.

ChiefDriver is built for providing a guidance for designing a boundary class which is representing an unusual system.

You can learn how to utilize **basic components** by following the ChiefDriver example.

node packer-master

PackerMaster
extends Master

-best_solution: Packer
+constructor()
#optimize(XML)
#replyOptimization(XML)
+static main()

connected & intermediated

node packer-mediator

PackerMediator
extends ParallelClientArrayMediator

-best_solution: Packer
-completed_count: number
-requested_size: number
+PackerMediator(master_ip: string)
#createMediator() -> MediatorSystem
#createServerBase() -> IServerBase
#createExternalClient(IClientDriver) -> ExternalSystem
+sendData(Invoke)
#replyOptimization(XML)
+static main()

Packer mediator system is placed on between Master and Slave systems. It can be a Slave system in Master side, and also can be a Master system for its Slave systems.

PackerMediator is built for providing a guidance; how to build tree-structured parallel processing system..

node tsp-master

TSPMaster
extends Master

-best_solution: Scheduler
+constructor()
#optimize(XML)
#replyOptimization(XML)
+static main()

Being Connected

Slave Systems

Abstract Slave

Slave
extends SlaveClient

+using super::super
#createConnector() -> IServerConnector
~~#createChild(XML) -> ExtSystemRole~~
#optimize(XML, size_t, size_t)

Slave is an abstract and example class has built for providing a guidance; how to build a Slave system belongs to a parallel processing system.

In the interaction example, when **Slave** gets orders of optimization with its basic data, **Slave** calculates and find the best optimized solution and report the solution to its Master system.

node packer-slave

PackerSlave
extends Slave

+using super::constructor
#optimize(XML, size_t, size_t)
+static main()

PackerSlave is a class representing a Slave system solving a packaging problem. It receives basic data about products and packages and find the best packaging solution.

node tsp-slave

TSPSlave
extends Slave

+using super::constructor
#optimize(XML, size_t, size_t)
+static main()

TSPSlave is a class representing a Slave system solving a TSP problem.

Principle purpose of protocol module in Samchon Framework is to constructing complicate network system easily within framework of Object Oriented Design, like designing classes of a S/W.
Furthermore, Samchon Framework provides a module which can be helpful for building a network system interacting with another external network system and master and slave modules that can realize (tree-structured) parallel (distributed) processing system.

Interaction module in example is built for providing guidance for those things. Interaction module demonstrates how to build complicate network system easily by considering each system as a class of a S/W, within framework of Object-Oriented Design.

Of course, **interaction module provides a guidance** for using external system and parallel processing system module.

You can learn how to construct a network system interacting with external network system and build (tree-structured) parallel processing systems which are distributing tasks (processes) by segmentation size if you follow the example, interaction module.

If you want to study the interaction example which is providing guidance of building network system within framework of OOD, I recommend you to study not only the class diagram and source code, but also **network diagram** of the interaction module.