C++ Class Diagram

1. library
2. protocol
   - 2-1. service
   - 2-2. master
   - 2-3. slave
3. templates
4. example

Library

string & container
critical section
math
xml & sql driver
event
File tree

Common Library and Utilities

Map Packages

MapContainer Package _Kty, _Ty
: public std::Container<_Ty, MapAllocator<_Kty, _Ty>>

+has(_Kty) -> bool
+get(_Kty) -> _Ty
+erase(_Kty)

MapAllocator _Kty, _Ty
: public allocator<_Ty>

-map: Map<_Kty, _Ty*>

+CriticalAllocator()
+~CriticalAllocator()

+allocate(size_t, _Ty*) -> _Ty*
+deallocate(_Ty*, size_t)
+construct(_Ty*, _Ty&)
+destroy(_Ty*)

template <typename U>
+CriticalAllocator(CriticalAllocator<U>)
template<typename U>
+construct(U*, Args&&...)
template<typename U>
+destroy(U*)

Dictionary _Ty, ...
<<typedef>>
: public Map<string, _Ty, ...>

Map _Kty, _Ty, _Alloc = std::allocator<_Ty>, _Pr = std::less<_Ty>
: public std::map <_Kty, _Ty, _Alloc, _Pr>

+has(_Kty)
+get(_Kty)
+set_Kty, _Ty)

String Utilities

WeakString

-data_: TCHAR*
-size_: size_t

+WeakString(const string &)
+WeakString(char*)
+WeakString(char*, char*)
+WeakString(char*, size_t)

+size() -> size_t
+at(size_t) -> TCHAR
+operator[](size_t) -> TCHAR
+empty() -> bool
+{... same methods of std::string}

+finds(vector<string>, size_t) -> IndexPair<string>
+rfinds(string, size_t) -> size_t

+substring(size_t, size_t) -> WeakString
+{trim, ltrim, rtrim}(...) -> WeakString
+between(string, string) -> WeakString

+split(string) -> vector<string>
+betweens(string, string) -> vector<string>

+replaceAll(vector<pair<string, string>>) -> string
+str() -> string

WeakString
A string class only references characters
reference only

Be careful about destruction of referenced characters

<<static>>
StringUtil

template <class T, class ... Types>
+substitude(string, T, Types ...)
+substitudeSQL(string, T, Types ...)

+numberFormat(double, int, int)
+percentFormat(double, int, int)
+colorNumberFormat(double, int, int)
+colorPercentFormat(double, int, int)

+{... same methods of WeakString}
+toClipboard(String)

Base64

+static encode(ByteArray) -> string
+static decode(string) -> ByteArray

Charset

+enum {ANSI, UTF8}

+static toMultibyte(wstring) -> string
+static toUTF8(wstring) -> string
+static toUnicode(string, int) -> wstring

SmartPointer _Ty

-referCntMap: static Map<_Ty*, size_t>
-mtx: static mutex

-ptr: _Ty*

+explicit SmartPointer(_Ty* := nullptr)
+SmartPointer(cont SmartPointer&)
+SmartPointer(SmartPointer&&)
+~SmartPointer()

-static construct(_Ty*)
-static destruct(_Ty*)

+reset(_Ty*)
+operator*() -> _Ty&
+operator->() -> _Ty*

ByteArray
: public std::vector<BYTE>

-position: size_t

+ByteArray()
+ByteArray(const ByteArray &)
+ByteArray(ByteArray&&)

template <typeanme _Ty>
+read() -> _Ty
template <typename _Ty>
+write(_Ty)

+compress() -> ByteArray
+decompress() -> ByteArray

Date and Datetime

Date
: public chrono::time_point

#static TP_1970: chrono::time_point

+Date()
+Date(int, int, int)
+Date(string)
+Date(long long linuxTime)
+virtual ~Date()

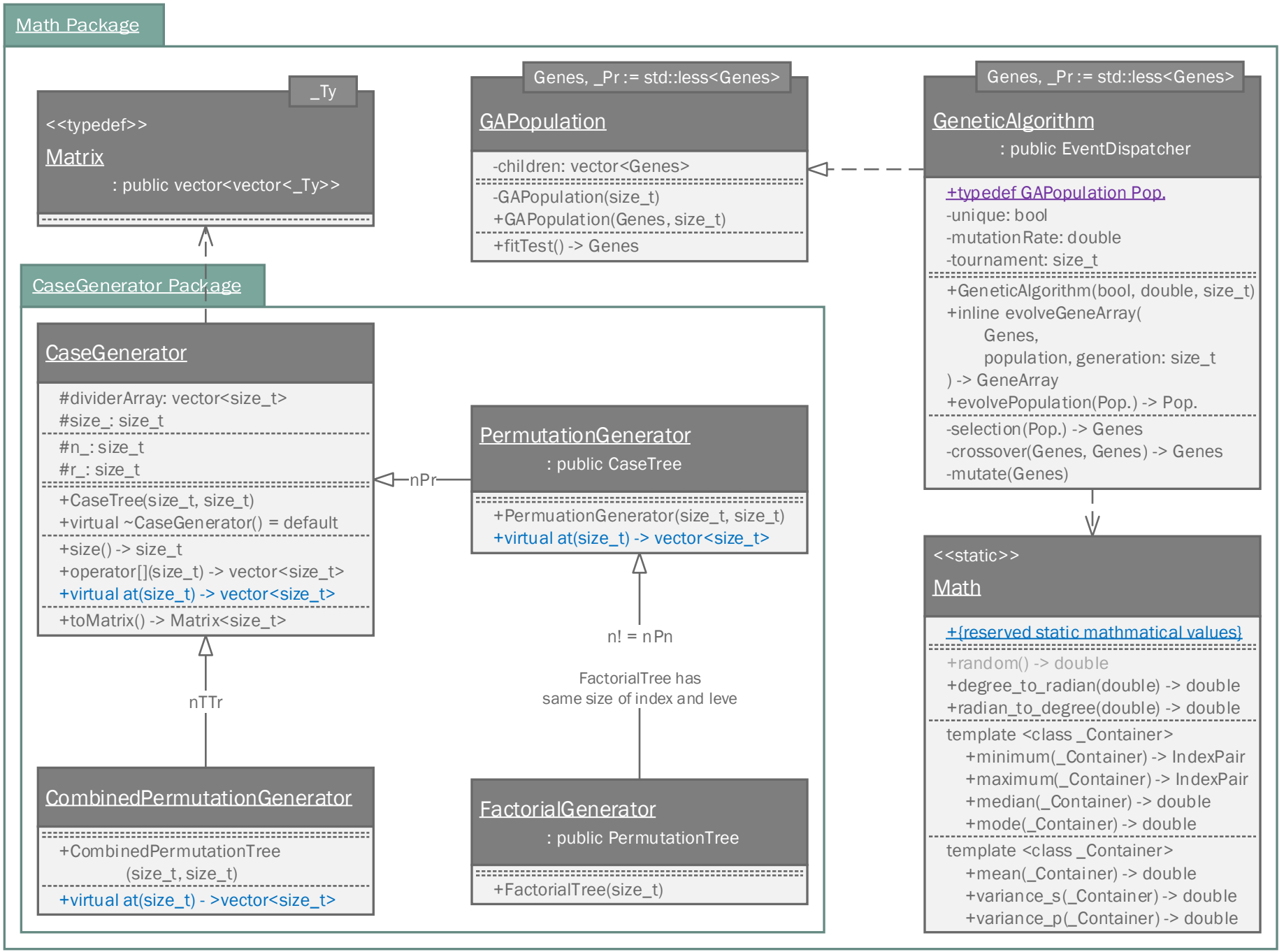
+{addDate(int)}
+{setDate(int)}

+virtual toString() -> string

Datetime
: public Datetime

+using Date::Date()
+Datetime(int, int, int, int, int, int)
+Datetime(string)

+virtual toString() -> string


Math Package
_Ty
<<typedef>>
Matrix
: public vector<vector<_Ty>>
Genes, _Pr := std::less<Genes>
GAPopulation
-children: vector<Genes>
-GAPopulation(size_t)
+GAPopulation(Genes, size_t)
+fitTest() -> Genes
Genes, _Pr := std::less<Genes>
GeneticAlgorithm
: public EventDispatcher
+typedef GAPopulation Pop.
-unique: bool
-mutationRate: double
-tournament: size_t
+GeneticAlgorithm(bool, double, size_t)
+inline evolveGeneArray(
Genes,
population, generation: size_t
) -> GeneArray
+evolvePopulation(Pop.) -> Pop.
-selection(Pop.) -> Genes
-crossover(Genes, Genes) -> Genes
-mutate(Genes)
CaseGenerator Package
CaseGenerator
#dividerArray: vector<size_t>
#size_: size_t
#n_: size_t
#r_: size_t
+CaseTree(size_t, size_t)
+virtual ~CaseGenerator() = default
+size() -> size_t
+operator[](size_t) -> vector<size_t>
+virtual at(size_t) -> vector<size_t>
+toMatrix() -> Matrix<size_t>
nPr
PermutationGenerator
: public CaseTree
+PermuationGenerator(size_t, size_t)
+virtual at(size_t) -> vector<size_t>
n! = nPn
FactorialTree has
same size of index and leve
nTTr
CombinedPermutationGenerator
+CombinedPermutationTree
(size_t, size_t)
+virtual at(size_t) - >vector<size_t>
FactorialGenerator
: public PermutationTree
+FactorialTree(size_t)
<<static>>
Math
+{reserved static mathmatical values}
+random() -> double
+degree_to_radian(double) -> double
+radian_to_degree(double) -> double
template <class _Container>
+minimum(_Container) -> IndexPair
+maximum(_Container) -> IndexPair
+median(_Container) -> double
+mode(_Container) -> double
template <class _Container>
+mean(_Container) -> double
+variance_s(_Container) -> double
+variance_p(_Container) -> double

Critical Sections

_Ty

CriticalAllocator
: public allocator<_Ty>

-mutex: RWMutex

+CriticalAllocator()
+~CriticalAllocator()

+allocate(size_t, _Ty*) -> _Ty*
+deallocate(_Ty*, size_t)
+construct(_Ty*, _Ty&)
+destroy(_Ty*)

template <typename U>
+CriticalAllocator(CriticalAllocator<U>)
template<typename U>
+construct(U*, Args&&...)
template<typename U>
+destroy(U*)

contains

RWMutex

-cv: condition_variable
-readMtx: mutex
-writeMtx: mutex

-readingCount: atomic<size_t>
-writingFlag: atomic<bool>
-minusMtx: mutex

+RWMutex()
+~RWMutex()

+readLock() const
+tryReadLock() const -> bool
+readUnlock() const

+writeLock()
+tryWriteLock() -> bool
+writeUnlock()

refers

Unique & Shared locks for RWMutex

UniqueReadLock

-mtx: const RWMutex*
-isLocked: bool

+UniqueReadLock
(const RWMutex &, bool)
+~UniqueReadLock()

+lock() const
+tryLock() const -> bool
+unlock() const

Ensures unlocked status of RWMutex on destruction

SharedReadLock

-mtx: const RWMutex*
-isLocked: atomic<bool>*
-reference: atomic<size_t>*

+SharedReadLock
(const RWMutex &, bool)
+SharedReadLock(SharedReadLock)
+~SharedReadLock()

+lock() const
+tryLock() const
+unlock() const

Unlocks when shareds're all destructed

UniqueWriteLock

-mtx: RWMutex*
-isLocked: bool

+UniqueWriteLock
(const RWMutex, bool)
+~UniqueWriteLock()

+lock()
+tryLock() -> bool
+unlock()

SharedWriteLock

-mtx: RWMutex*
-isLocked: atomic<bool>*
-reference: atomic<size_t>*

+SharedWriteLock(RWMutex &, bool)
+SharedWriteLock(SharedWriteLock)
+~SharedWriteLock()

+lock()
+tryLock()
+unlock()

_Ty

CriticalContainer Package
: public std::Container<_Ty,
CriticalAllocator<_Ty>>

+readLock() const
+readUnlock() const

+writeLock()
+writeUnlock()

Semaphore

-mtx: mutex
-countMtx: mutex
-size_: size_t
-acquired: size_t

+Semaphore(size_t := 1)
~Semaphore()

+size() -> size_t
+acquire()
+tryAcquire() -> bool
+release()

References

Unique & Shared acuiqre for Semaphore

UniqueAcquire

-semaphore: Semaphore*
-isLocked: bool

+UniqueAcquire(const Semaphore &)
+~UniqueAcquire()

+acquire()
+tryAcquire() -> bool
+release()

Ensures released status of Semaphore

SharedAcquire

-semaphore: Semaphore*
-isLocked: atomic<bool>*
-reference: atomic<size_t>*

+UniqueAcquire(Semaphore &)
+UniqueAcquire(UniqueAcquire)
+~UniqueAcquire()

+acquire()
+tryAcquire() -> bool
+release()

RWMutex, Semaphore
There's not rw_mutex and semaphore in STL.
Those're for corss-compiling in multiple operating systems.

CriticalAllocator
CriticalAllocator keeps safety in multi-threading environment. But the CriticalAllcator ensures only safety. If a logic needs a mutual extension not only level of container, don't use it.

UniqueLock
Manages and guarantess a locker to be an unlocked status on destruction. UniqueLock helps to avoid forgetting unlock or a bad situation cannot execute unlock by exception or error.

SharedLock:
Gurantess an unlocked status on destruction of all related SharedLock(s); SharedLock objects referencing same Locker.

Protocol Package

SQL INTERFACE

SQLi

- #virtual DRIVER() -> String = NULL
- #virtual PORT() -> int = NULL
- #connectionHandler: void*
- #stmt: SQLStatement
- #stmtMutex: mutex
- +SQLi()
- +virtual ~SQLi()
- +virtual createStatement() -> SQLStatement* = NULL
- +connect(ip: String, dbName: String, id: String, password: String)
- +disconnect()

MySQLi : public SQLi

- #virtual DRIVER() -> String
- #virtual PORT() -> int := 3389
- +MySQLi()
- +virtual createStatement() -> SQLStatement

TSQLi : public SQLi

- #virtual DRIVER() -> String
- #virtual PORT() -> int := 1433
- +TSQLi()
- +virtual createStatement() -> SQLStatement*

1 1 — 0 1

SQLStatement

- #sqli: SQLi
- #hstmt: void*
- #sizeMap: Map<size_t, size_t>
- #bindedSize: size_t
- -SQLStatement(SQLi)
- +virtual ~SQLStatement()
- +reset(SQLi)
- +free()
- +refresh()
- template<class _Ty, class ... _Types> +prepare(string, _Ty, ... _Types)
- +execute()
- +fetch() -> bool
- +toNextStatement() -> bool
- template <class _Ty> +getData() -> _Ty
- +virtual toXML() -> XML

<<Domain issue>>
T-SQL::{FOR XML}

TSQLStatement : public SQLStatement

- +TSQLStatement()
- +virtual toXML() -> XML

DBMS Server doesn't allow simultaneous query
so that a group of SQLStatement(s) sharing same SQLi will have only a single thread.

In this Samchon Framework,
SQLStatement::repare causes "Lock" and this "Lock" will be unlocked until you calls SQLStatement::free() or SQLStatement::~SQLStatement()

Create new SQLi
If you don't want it and consider multi-thread handling DB, then create multiple SQLi(s)

HTTP Protocol

HTTPService : public EventDispatcher

- #url: String
- #method: enum{GET, POST}
- #header: Map<string, string>
- +HTTPService()
- +HTTPService(string, int)
- +virtual ~HTTPService()
- +load(URLVariables) -> ByteArray
- +send(URLVariables)

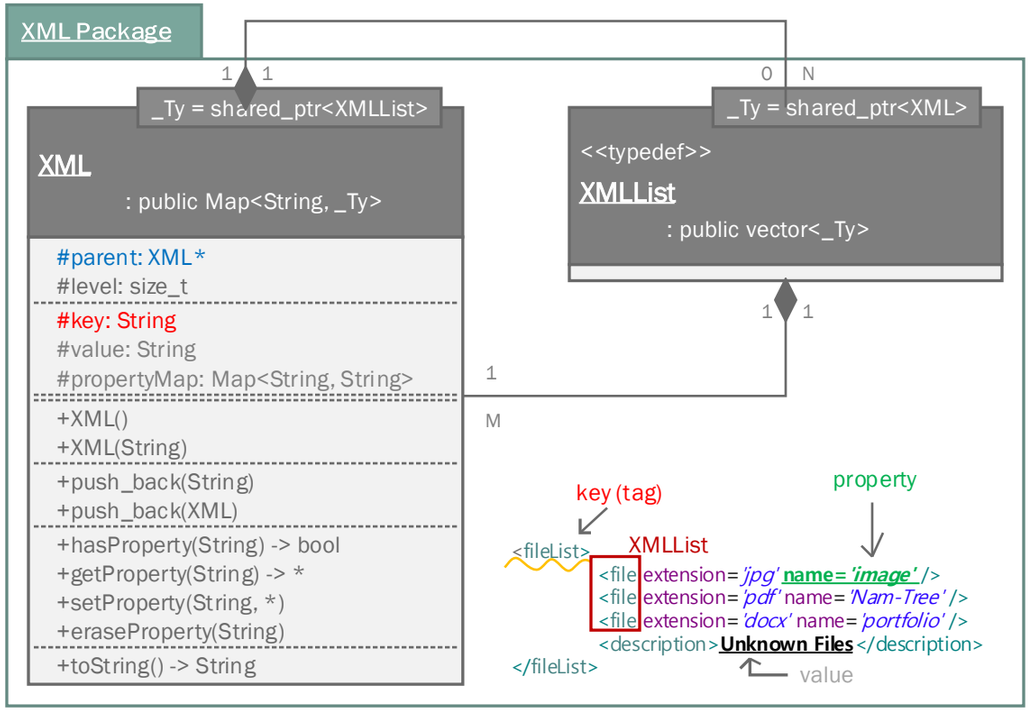
generates when call send

ResultEvent : public Event

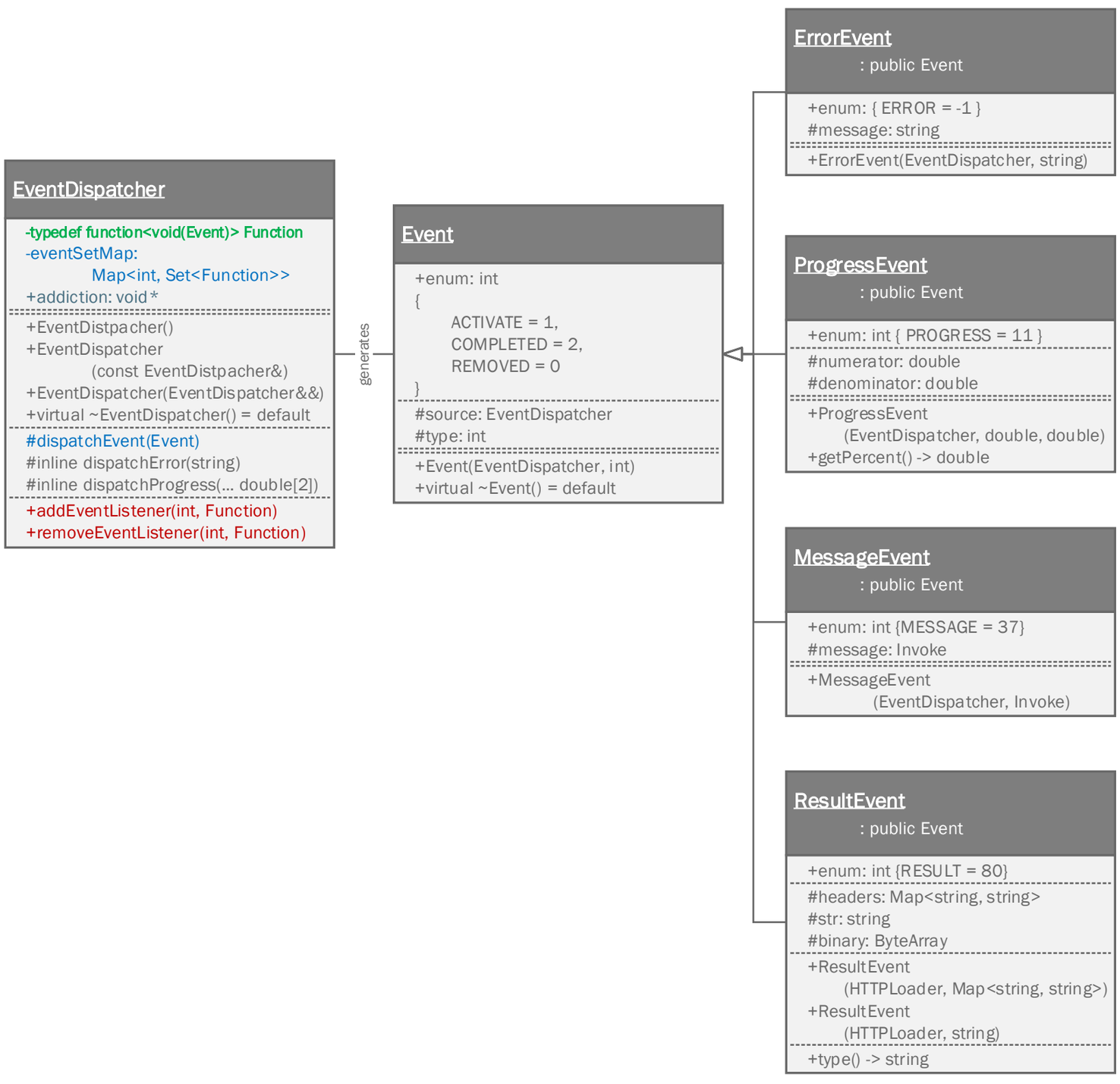
- +enum: int {RESULT = 80}
- #headers: Map<string, string>
- #str: string
- #binary: ByteArray
- +ResultEvent (HTTPLoader, Map<string, string>)
- +ResultEvent (HTTPLoader, string)
- +type() -> string

When call HTTPService::send(), be careful about destruction of HTTPService

XML Package



Event Package



EventDispatcher

All the events are sent asynchronously.

To protect from creating enourmous threads by asynchronous event sending, all event sending process will lock the semahore. The default size of the semaphore is 2

Event listener function has to be global or static

I couldn't specify the class to listen, so I programmed all event listener (function pointer) to be static. To send Events to a class's member method, I'm considering to make an interface to listen, "IEventListener"

Warning!

Since C++11, calling member method of a class by new thread passing by static method and void pointer is recommended to avoid.

By guidance of the STL, using *std::thread* and *std::bind* will be better. As that reason, Event and EventDispatcher can be depreciated in next generation of Samchon Framework

Event

A basic class for expressing an event.

Determined Events are "*ACTIVE*" & "*COMPLETE*"
You can add any new event type, if you want.

ErrorEvent

Cannot throw exception as you called some process asynchronous, you can use this ErrorEvent, insteadly

ProgressEvent

An event representing a progress.
It's good for expressing a progress of a process influences to whole system.

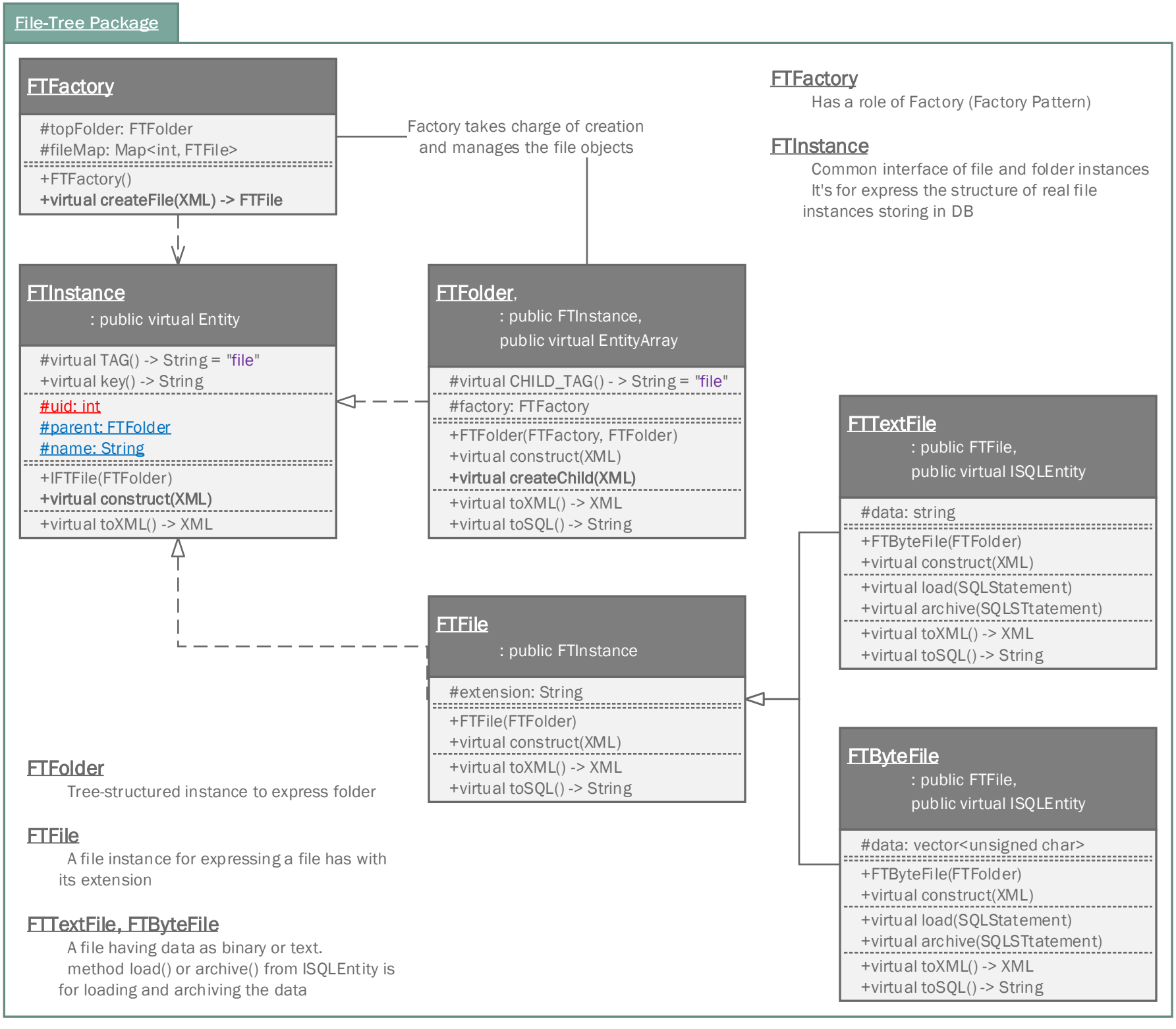
~~MessageEvent~~

~~An event containing an Invoke message~~
Depreciated since v1.0.
Use chain of responsibility pattern with *IProtocol*.

ResultEvent

An event containing result data from a web-page
The result type will be one of *string* and *ByteArray*

ResultEvent::header: Replied headers from a web-page


File-Tree Package
FTFactory
#topFolder: FTFolder
#fileMap: Map<int, FTFile>
+FTFactory()
+virtual createFile(XML) -> FTFile
Factory takes charge of creation and manages the file objects
FTFactory
Has a role of Factory (Factory Pattern)
FTInstance
Common interface of file and folder instances
It's for express the structure of real file instances storing in DB
FTInstance
: public virtual Entity
#virtual TAG() -> String = "file"
+virtual key() -> String
#uid: int
#parent: FTFolder
#name: String
+IFTFile(FTFolder)
+virtual construct(XML)
+virtual toXML() -> XML
FTFolder,
: public FTInstance,
public virtual EntityArray
#virtual CHILD_TAG() - > String = "file"
#factory: FTFactory
+FTFolder(FTFactory, FTFolder)
+virtual construct(XML)
+virtual createChild(XML)
+virtual toXML() -> XML
+virtual toSQL() -> String
FTTextFile
: public FTFile,
public virtual ISQLEntity
#data: string
+FTByteFile(FTFolder)
+virtual construct(XML)
+virtual load(SQLStatement)
+virtual archive(SQLSTtatement)
+virtual toXML() -> XML
+virtual toSQL() -> String
FTFile
: public FTInstance
#extension: String
+FTFile(FTFolder)
+virtual construct(XML)
+virtual toXML() -> XML
+virtual toSQL() -> String
FTByteFile
: public FTFile,
public virtual ISQLEntity
#data: vector<unsigned char>
+FTByteFile(FTFolder)
+virtual construct(XML)
+virtual load(SQLStatement)
+virtual archive(SQLSTtatement)
+virtual toXML() -> XML
+virtual toSQL() -> String
FTFolder
Tree-structured instance to express folder
FTFile
A file instance for expressing a file has with its extension
FTTextFile, FTByteFile
A file having data as binary or text.
method load() or archive() from ISQLEntity is for loading and archiving the data

Protocol

Object Oriented Network

Basic Components
Invoke
Entity

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.

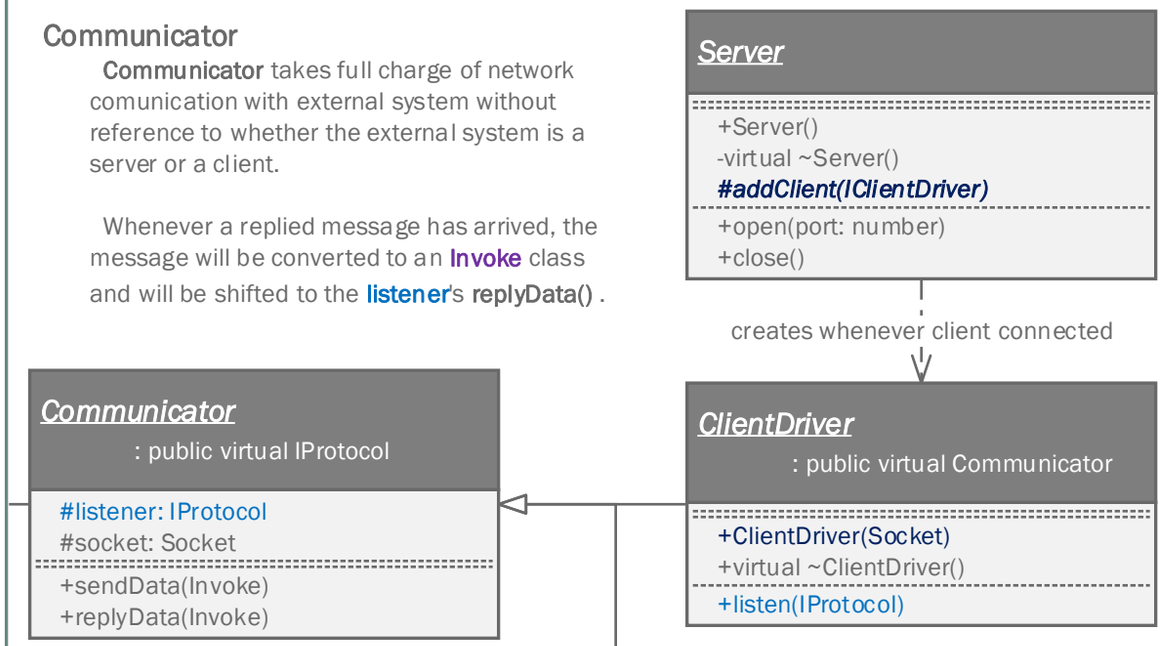
All the system templates in this framework are also being implemented by extending and combination of the **basic components**.

Communicators

Communicator

Communicator 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()** .



Server

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

- Server
- WebServer

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

ClientDriver

ClientDriver is a type of **Communicator**, taking full charge of network communication with the remote client.

The **ClientDriver** object is created by the **Server** object whenever a remote client has newly connected. Starts communication by the method; **ClientDriver.listen**(**listener**). Then replied message from the remote client will be delivered to **listener**'s **replyData()**.

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.

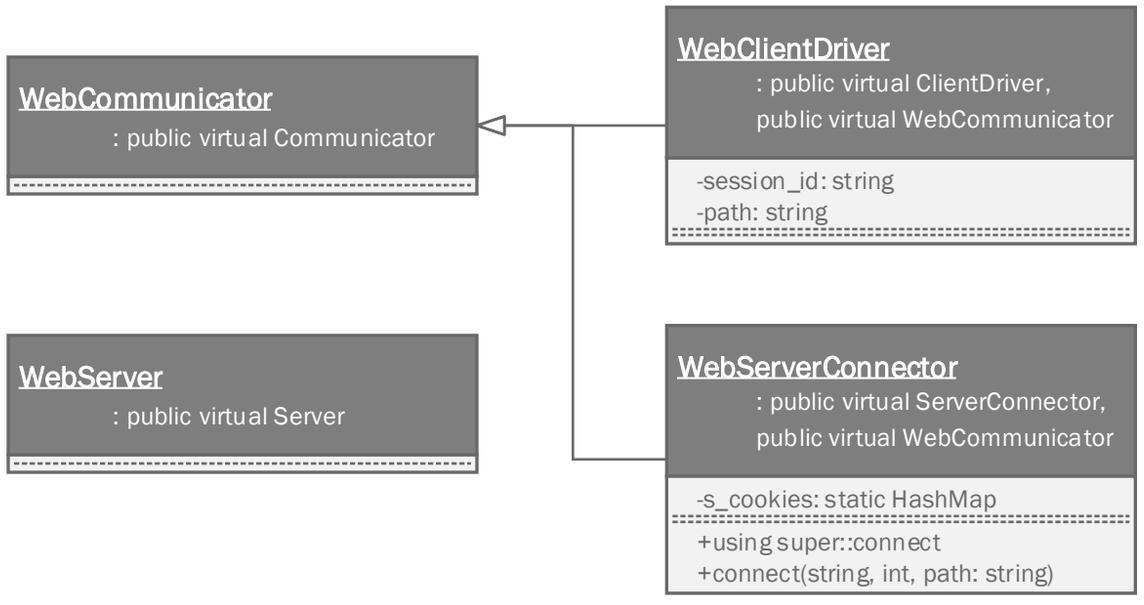
You can see that all classes with related network I/O and handling **Invoke** message are implementing the **IProtocol** interface with **IServer** and **communicator classes**.

ServerConnector

ServerConnector is a server connector who can connect to an remote server system as a client.

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

Web Communicators



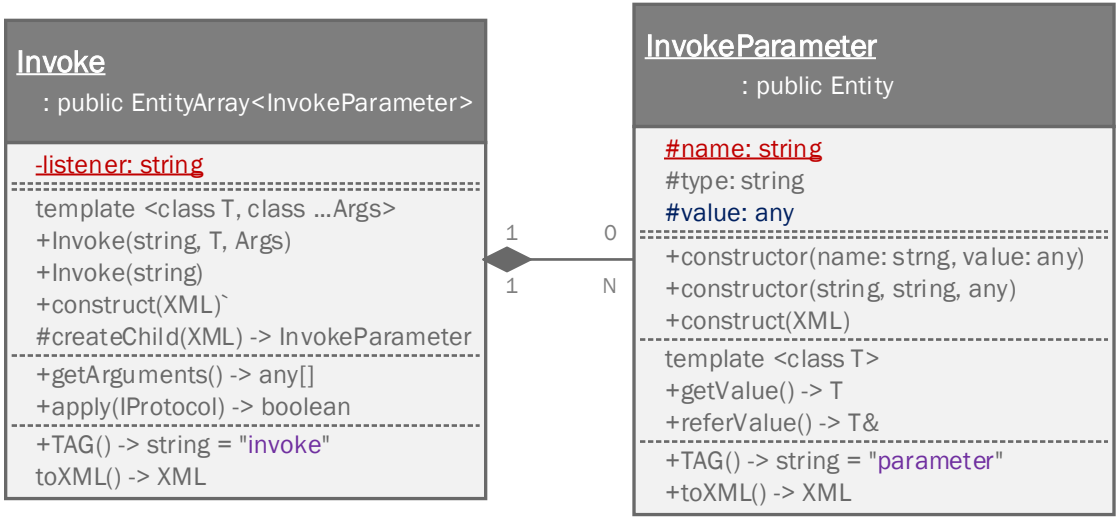
Classes for Web-Socket

Classes of Web-Socket follow protocol of Invoke and Web socket at the same time by implementing basic interfaces and overriding some methods to follow web-socket protocol.

You can convert any type of network system to follow web-socket protocol by implementing those interfaces because it's a rule to implements virtually those classes.

WebServer
WebClientDriver
WebServerConnector

Invoke Message



Invoke is

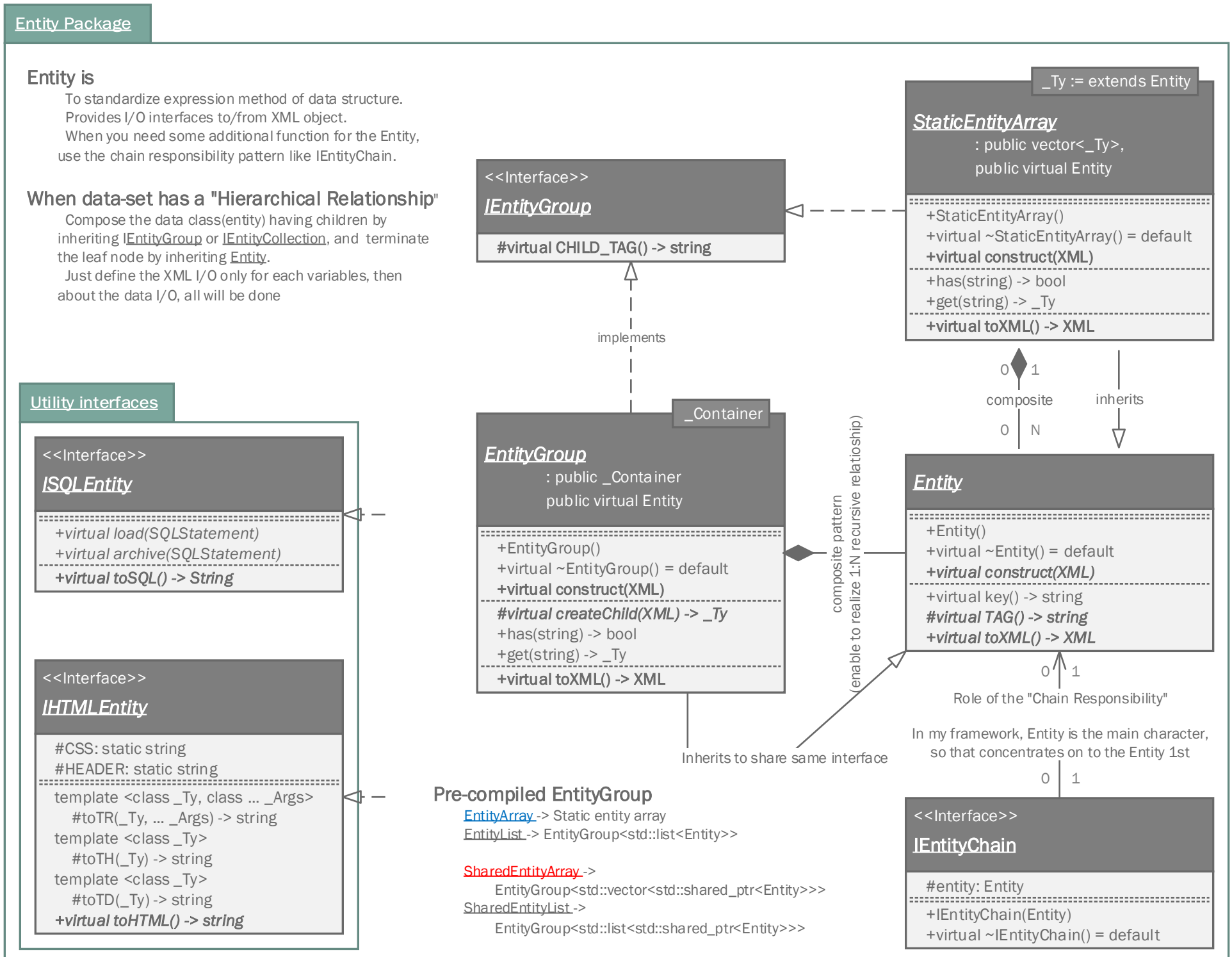
Designed to standardize message structure to be used in network communication. By the standardization of message protocol, user does not need to consider about the network handling. Only concentrate on system's own domain functions are required.

At prev page, "Protocol – Basic Components", you can find out some objects 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

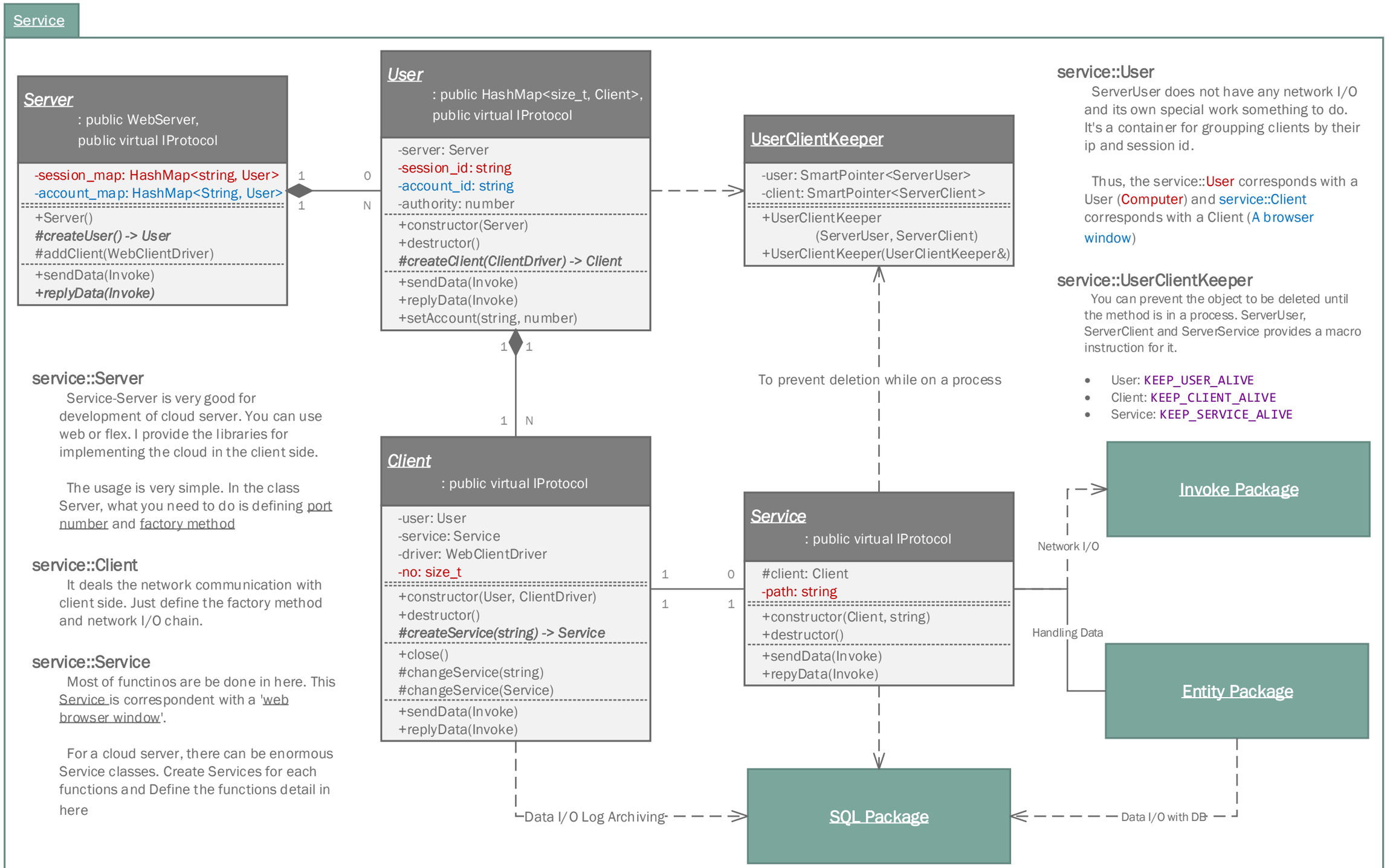
```
<?xml version="1.0" encoding="utf-8" ?>
<invoke listener="login">
        <parameter type="string">jhnam88</parameter>
        <parameter type="string">1234</parameter>
        <parameter type="number">4</parameter>
        <parameter type="XML">
                <memberList>
                    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


Entity Package
Entity is
To standardize expression method of data structure.
Provides I/O interfaces to/from XML object.
When you need some additional function for the Entity, use the chain responsibility pattern like IEntityChain.
When data-set has a "Hierarchical Relationship"
Compose the data class(entity) having children by inheriting IEntityGroup or IEntityCollection, and terminate the leaf node by inheriting Entity.
Just define the XML I/O only for each variables, then about the data I/O, all will be done
Utility interfaces
<<Interface>>
ISQLEntity
+virtual load(SQLStatement)
+virtual archive(SQLStatement)
+virtual toSQL() -> String
<<Interface>>
IHTMLEntity
#CSS: static string
#HEADER: static string
template <class _Ty, class ... _Args>
#toTR(_Ty, ... _Args) -> string
template <class _Ty>
#toTH(_Ty) -> string
template <class _Ty>
#toTD(_Ty) -> string
+virtual toHTML() -> string
<<Interface>>
IEntityGroup
#virtual CHILD_TAG() -> string
implements
_Ty := extends Entity
StaticEntityArray
: public vector<_Ty>,
public virtual Entity
+StaticEntityArray()
+virtual ~StaticEntityArray() = default
+virtual construct(XML)
+has(string) -> bool
+get(string) -> _Ty
+virtual toXML() -> XML
0
1
composite
inherits
0
N
_Container
EntityGroup
: public _Container
public virtual Entity
+EntityGroup()
+virtual ~EntityGroup() = default
+virtual construct(XML)
#virtual createChild(XML) -> _Ty
+has(string) -> bool
+get(string) -> _Ty
+virtual toXML() -> XML
composite pattern
(enable to realize 1:N recursive relatioship)
Entity
+Entity()
+virtual ~Entity() = default
+virtual construct(XML)
+virtual key() -> string
#virtual TAG() -> string
+virtual toXML() -> XML
0
1
Role of the "Chain Responsibility"
In my framework, Entity is the main character, so that concentrates on to the Entity 1st
0
1
Inherits to share same interface
<<Interface>>
IEntityChain
#entity: Entity
+IEntityChain(Entity)
+virtual ~IEntityChain() = default
Pre-compiled EntityGroup
EntityArray -> Static entity array
EntityList -> EntityGroup<std::list<Entity>>
SharedEntityArray ->
EntityGroup<std::vector<std::shared_ptr<Entity>>>
SharedEntityList ->
EntityGroup<std::list<std::shared_ptr<Entity>>>

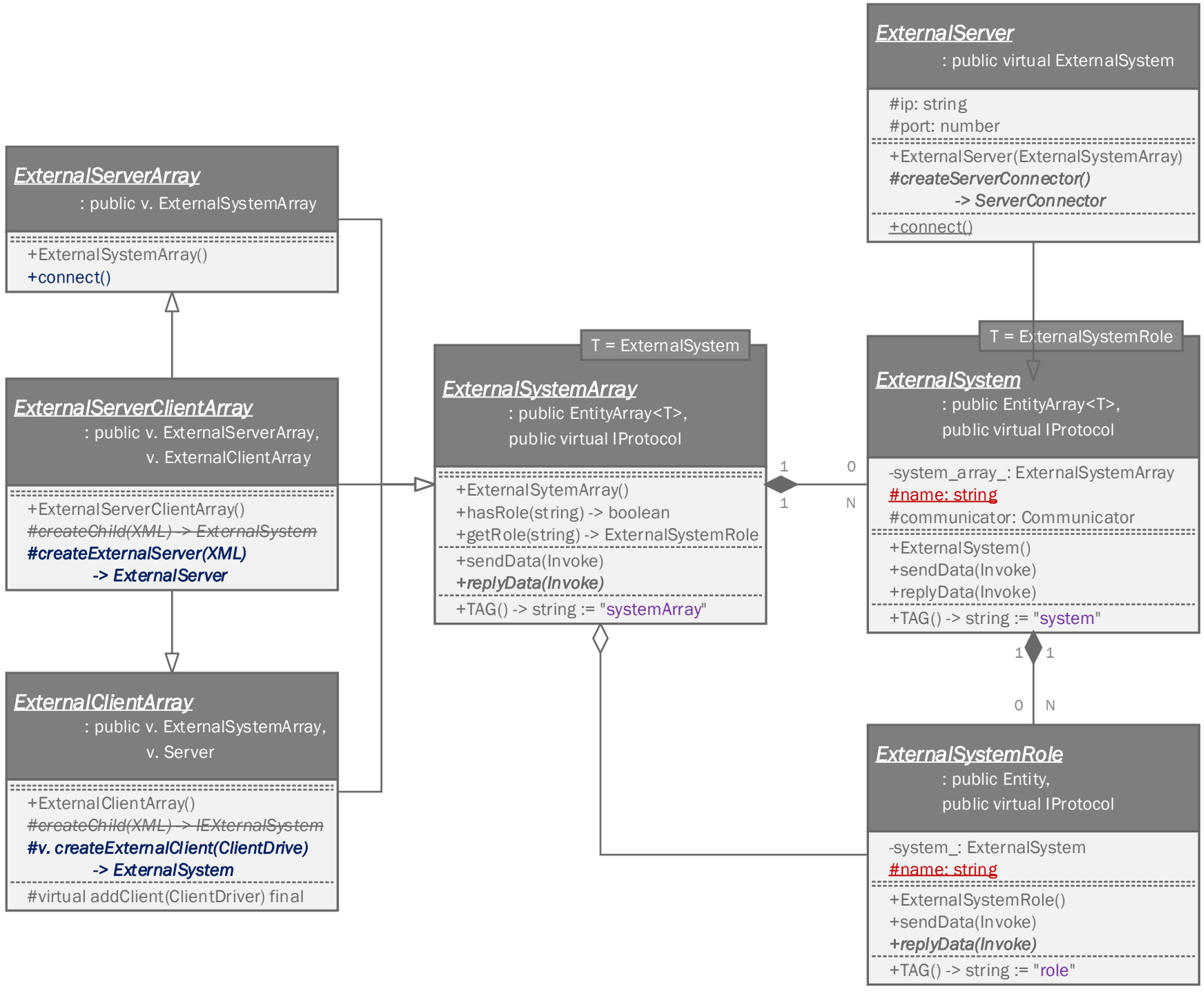
Templates

Pre-defined Network System Modules

Cloud Service
External Systems
Parallel Processing System
Distributed Processing System


Service
Server
: public WebServer,
public virtual IProtocol
-session_map: HashMap<string, User>
-account_map: HashMap<String, User>
+Server()
#createUser() -> User
#addClient(WebClientDriver)
+sendData(Invoke)
+replyData(Invoke)
1
1
0
N
User
: public HashMap<size_t, Client>,
public virtual IProtocol
-server: Server
-session_id: string
-account_id: string
-authority: number
+constructor(Server)
+destructor()
#createClient(ClientDriver) -> Client
+sendData(Invoke)
+replyData(Invoke)
+setAccount(string, number)
1
1
1
N
UserClientKeeper
-user: SmartPointer<ServerUser>
-client: SmartPointer<ServerClient>
+UserClientKeeper
(ServerUser, ServerClient)
+UserClientKeeper(UserClientKeeper&)
To prevent deletion while on a process
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.
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
Client
: public virtual IProtocol
-user: User
-service: Service
-driver: WebClientDriver
-no: size_t
+constructor(User, ClientDriver)
+destructor()
#createService(string) -> Service
+close()
#changeService(string)
#changeService(Service)
+sendData(Invoke)
+replyData(Invoke)
1
1
0
1
Service
: public virtual IProtocol
#client: Client
-path: string
+constructor(Client, string)
+destructor()
+sendData(Invoke)
+repyData(Invoke)
Network I/O
Handling Data
Invoke Package
Entity Package
Data I/O Log Archiving
SQL Package
Data I/O with DB
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::UserClientKeeper
You can prevent the object to be deleted until the method is in a process. ServerUser, ServerClient and ServerService provides a macro instruction for it.
User: KEEP_USER_ALIVE
Client: KEEP_CLIENT_ALIVE
Service: KEEP_SERVICE_ALIVE

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 Module

Parallel System

ParallelSystemArrayMediator
: public v. ParallelSystemArray

-mediator_: MediatorSystem

+using super::super
#v. createMediator() -> Mediator
#v. startMediator()

#v. _Complete_history(InvokeHistory)

1 1

<<Mediator to real master>>
ParallelSystem::replyData()
–>> ParallelSystemArrayMediator::replyData()
–>> Mediator::sendData()

1 1

MediatorSystem
: public virtual SlaveSystem,
public virtual IListener

-system_array_: PSystemArrayMediator

+MediatorSystem
(ParallelSystemArrayMediator)
+virtual start()

#virtual _Reply_data(Invoke)
+virtual replyData(Invoke)
-complete_history(number)

ParallelSystemArray
public v. ExternalSystemArray

-history_sequence_: number

+ParallelSystemArray()
+sendSegmentData(Invoke, number)
+sendPieceData
(Invoke, number, number)

#v. _Complete_history(InvokeHistory)
#v. _Normalize_performance()

1 1

0 N

ParallelSystem
: public v. ExternalSystem

-progress_list_, history_list_:
HashMap<number, InvokeHistory>
-performance: number
-enforced_: boolean

+ParallelSystem(ParallelSystemArray)
+ParallelSystem
(PSystemArray, ClientDriver)
+virtual ~ParallelSystem()

-send_piece_data
(Invoke, number, number)
#virtual _Send_back_history
(Invoke, InvokeHistory)
#virtual _Reply_data(Invoke)
#virtual _Report_history(XML)

1 1

ParallelSystem also can have Role

0 N

ExternalSystemRole
: public Entitiy,
public v. IProtocol

#system: ExternalSystem
#name: string

+ExternalSystemRole(ExternalSystem)
+virtual sendData(Invoke)
+virtual replyData(Invoke)

+TAG() -> string := "role"

Derived Classes

ParallelSystemArrayMediator
ParallelServerArrayMediator
ParallelClientArrayMediator
ParallelServerClientArrayMediator

ParallelSystemArray
ParallelServerArray
ParallelClientArray
ParallelServerClientArray

ParallelSystem
ParallelServer

AUTO_INCREMENTS

1 1

Histories

InvokeHistory
: public Entity<size_t>

-uid: size_t
-listener: string
-start_time_: Date
-end_time_: Date

+InvokeHistory()
+InvokeHistory(Invoke)
+virtual construct(XML)
+complete()

+virtual TAG() := "history"
+virtual toXML() -> XML
+toInvoke() -> Invoke

1 1

0 N

PRInvokeHistory
: public InvokeHistory

-first: number
-last: number

+PRInvokeHistory()
+PRInvokeHistory(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.

Distributed Processing System Module

System and related Classes

DistributedSystemArrayMediator
: public v. DistributedSystemArray,
public v. ParallelSystemArrayMediator

+DistributedSystemArrayMediator()
#v. _CompleteHistory(InvokeHistory)

1 1

<<Mediator to real master>>
ParallelSystem::replyData()
-->> ParallelSystemArrayMediator::replyData()
-->> Mediator::sendData()

1 1

MediatorSystem
: public virtual SlaveSystem,
public virtual IListener

-system_array_: PSystemArrayMediator
+MediatorSystem
(ParallelSystemArrayMediator)
+virtual start()
#virtual _Reply_data(Invoke)
+virtual replyData(Invoke)
-complete_history(number)

DistributedSystemArray
: public v. ParallelSystemArray

-process_map: HashMap
<string, DistributedProcess>
+DistributedSystemArray()
+virtual construct(XML)
#virtual createProcess(XML)
-> DistributedProcess
#vir. _Complete_history(InvokeHistory)
#vir. _Normalize_performance()
+virtual toXML() -> XML

1 1

0 N

DistributedSystem
: public virtual. ParallelSystem

+using super::super
+~virtual ~DistributedSystem()
+virtual replyData(Invoke)
#virtual _Send_back_history
(Invoke, InvokeHistory)
#virtual _Report_history(InvokeHistory)

1 1

DistributedSystem also can have Role

0 N

ExternalSystemRole
: public Entity

#system: ExternalSystem
#name: string
+ExternalSystemRole(ExternalSystem)
+virtual construct(XML)
#virtual key() -> String
+virtual sendData(Invoke)
+virtual replyData(Invoke)
#virtual TAG() -> String := "role"
+virtual toXML() -> XML

1 1

0 N

DistributedProcess
: public Entity

-systemArray: DistributedSystemArray
-progress_list, history_list:
HashMap<size_t, DSInvokeHistory>
-name: string
-resource: double
+DistributedProcess(DSystemArray)
+virtual ~DistributedProcess()
+virtual construct(XML)
+virtual replyData(Invoke)
+virtual sendData(Invoke)
+virtual sendData(Invoke, double)
+virtual TAG() -> string := "process"
+virtual toXML() -> XML

0 M

M: N Relationship

0 N

Derived Classes

DistributedSystemArrayMediator
DistributedServerArrayMediator
DistributedClientArrayMediator
DistributedServerClientArrayMediator

DistributedSystemArray
DistributedServerArray
DisttributedClientArray
DistributedServerClientArray

DistributedSystem
DistributedServer

Histories

InvokeHistory
: public Entity<size_t>

-uid: size_t
-listener: string
-start_time_: Date
-end_time_: Date
+InvokeHistory()
+InvokeHistory(Invoke)
+virtual construct(XML)
+complete()
+virtual TAG() := "history"
+virtual toXML() -> XML
+toInvoke() -> Invoke

DSInvokeHistory
: public InvokeHistory

-system: DistributedSystem
-process: DistributedProcess
-weight: double
+DSInvokeHistory(DistributedSystem*)
+DSInvokeHistory(
DistributedSystem,
DistributedProcess,
Invoke, double)
+virtual construct(XML)
+virtual toXML() -> XML

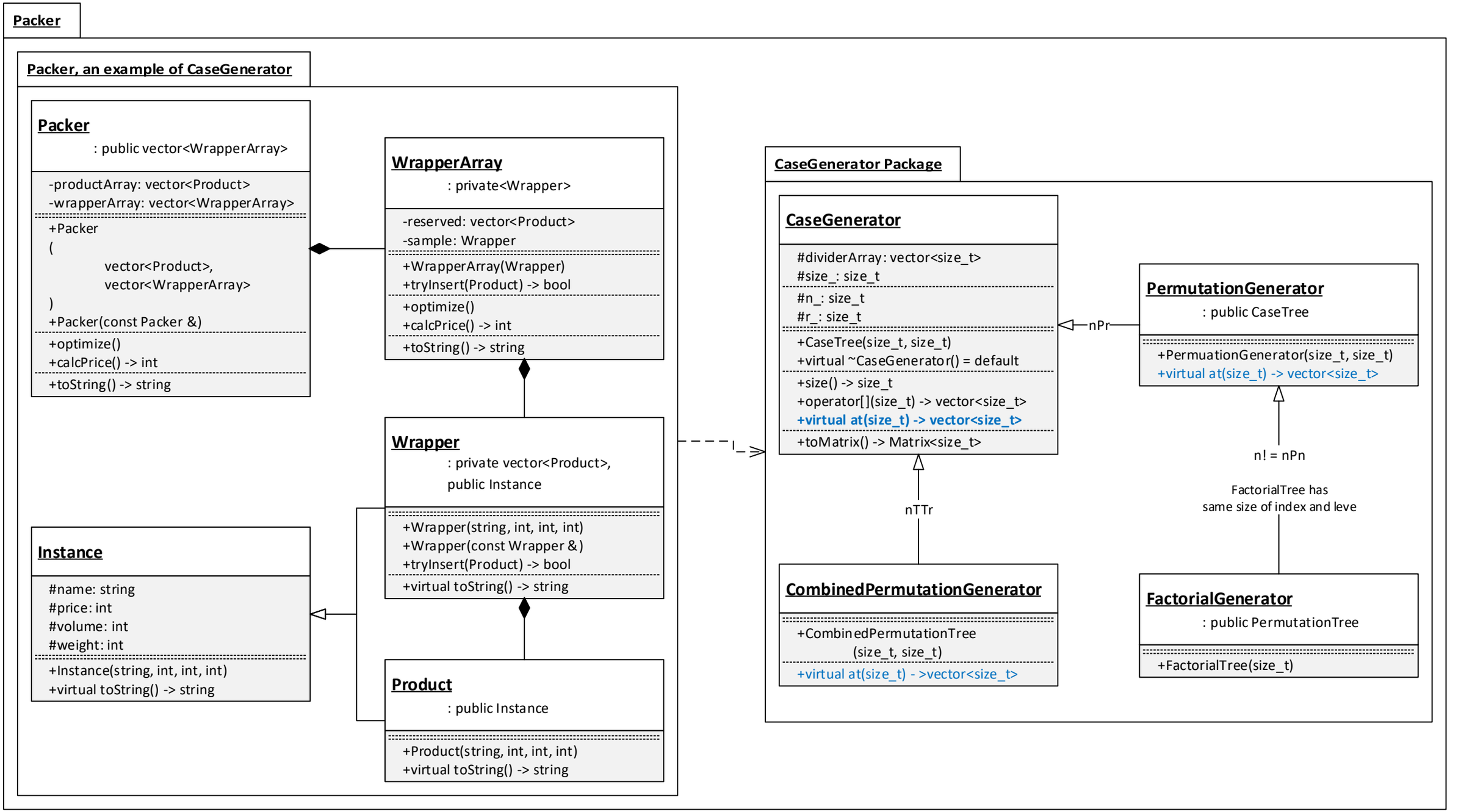
DSInvokeHistory

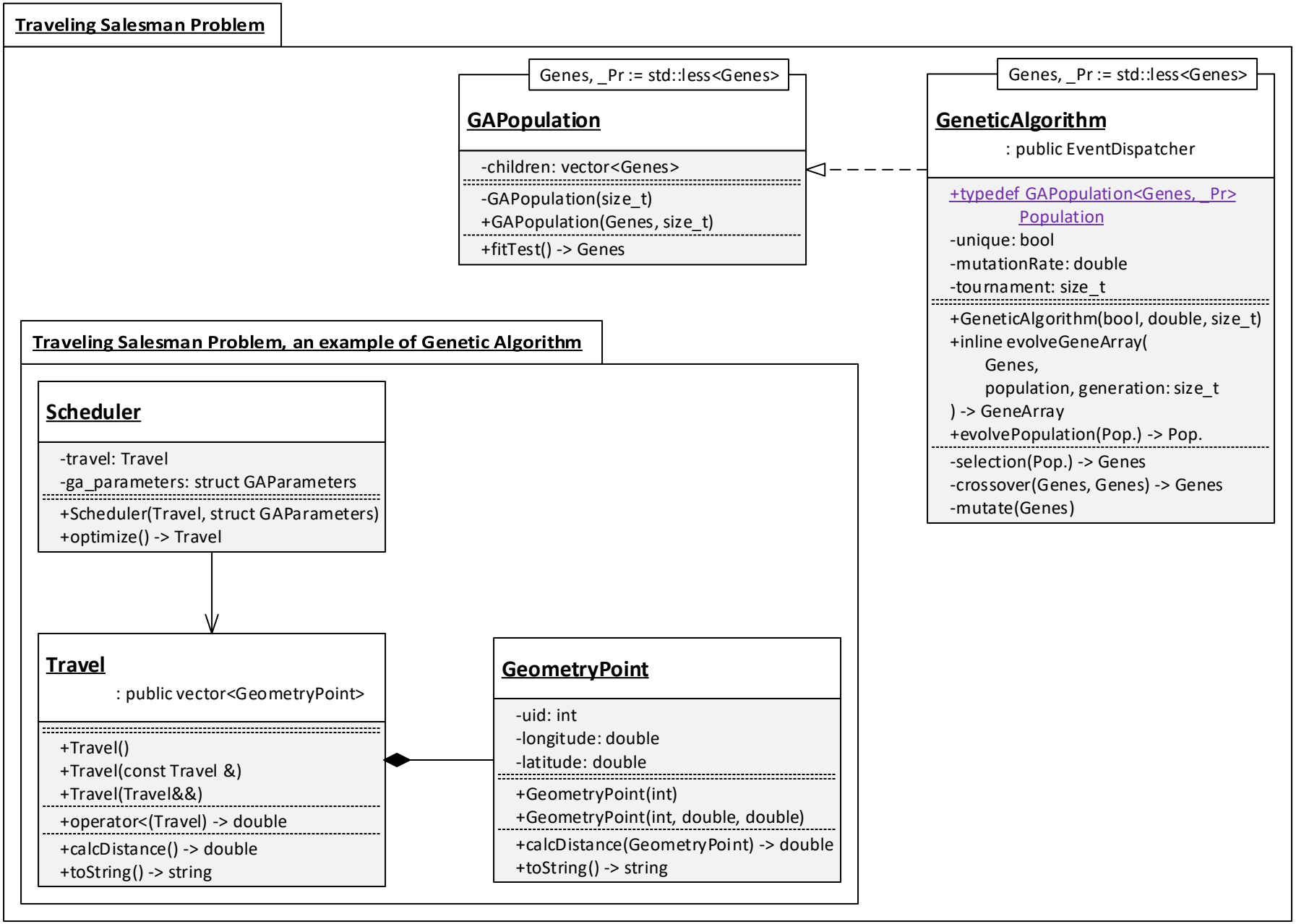
A reported InvokeHistory in framework of a master of parallel processing system. The master of a parallel processing system estimates performance index of a slave system by those reports.

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

Example

packer – case generator
traveling salesman problem – genetic alogirhtm
console chat – interfaces of protocol


Packer
Packer, an example of CaseGenerator
Packer
: public vector<WrapperArray>
-productArray: vector<Product>
-wrapperArray: vector<WrapperArray>
+Packer
(
vector<Product>,
vector<WrapperArray>
)
+Packer(const Packer &)
+optimize()
+calcPrice() -> int
+toString() -> string
WrapperArray
: private<Wrapper>
-reserved: vector<Product>
-sample: Wrapper
+WrapperArray(Wrapper)
+tryInsert(Product) -> bool
+optimize()
+calcPrice() -> int
+toString() -> string
Wrapper
: private vector<Product>,
public Instance
+Wrapper(string, int, int, int)
+Wrapper(const Wrapper &)
+tryInsert(Product) -> bool
+virtual toString() -> string
Instance
#name: string
#price: int
#volume: int
#weight: int
+Instance(string, int, int, int)
+virtual toString() -> string
Product
: public Instance
+Product(string, int, int, int)
+virtual toString() -> string
I_
CaseGenerator Package
CaseGenerator
#dividerArray: vector<size_t>
#size_: size_t
#n_: size_t
#r_: size_t
+CaseTree(size_t, size_t)
+virtual ~CaseGenerator() = default
+size() -> size_t
+operator[](size_t) -> vector<size_t>
+virtual at(size_t) -> vector<size_t>
+toMatrix() -> Matrix<size_t>
nPr
PermutationGenerator
: public CaseTree
+PermuationGenerator(size_t, size_t)
+virtual at(size_t) -> vector<size_t>
n! = nPn
FactorialTree has
same size of index and leve
nTTr
CombinedPermutationGenerator
+CombinedPermutationTree
(size_t, size_t)
+virtual at(size_t) - >vector<size_t>
FactorialGenerator
: public PermutationTree
+FactorialTree(size_t)


Traveling Salesman Problem
Genes, _Pr := std::less<Genes>
GAPopulation
-children: vector<Genes>
-GAPopulation(size_t)
+GAPopulation(Genes, size_t)
+fitTest() -> Genes
Genes, _Pr := std::less<Genes>
GeneticAlgorithm
: public EventDispatcher
+typedef GAPopulation<Genes, _Pr> Population
-unique: bool
-mutationRate: double
-tournament: size_t
+GeneticAlgorithm(bool, double, size_t)
+inline evolveGeneArray(
Genes,
population, generation: size_t
) -> GeneArray
+evolvePopulation(Pop.) -> Pop.
-selection(Pop.) -> Genes
-crossover(Genes, Genes) -> Genes
-mutate(Genes)
Traveling Salesman Problem, an example of Genetic Algorithm
Scheduler
-travel: Travel
-ga_parameters: struct GAParameters
+Scheduler(Travel, struct GAParameters)
+optimize() -> Travel
Travel
: public vector<GeometryPoint>
+Travel()
+Travel(const Travel &)
+Travel(Travel&&)
+operator<(Travel) -> double
+calcDistance() -> double
+toString() -> string
GeometryPoint
-uid: int
-longitude: double
-latitude: double
+GeometryPoint(int)
+GeometryPoint(int, double, double)
+calcDistance(GeometryPoint) -> double
+toString() -> string

Console Chat

Console chat - An example of interfaces of protocol

console_chat_server.exe

ChatServer

: public IServer,
public IProtocol,
public set<ChatClient>

#virtual PORT() -> int := 37749
-clients: set<Client>
-mtx: RWMutex

+ChatServer()
#virtual addClient(Socket)
+virtual replyData(Invoke)
+virtual sendData(Invoke)

ChatClient

: public IProtocol

-server: ChatServer
-socket: Socket

+ChatClient(ChatServer, Socket)
+virtual ChatClient()
+virtual replyData(Invoke)

console_chat_client.exe

ChatClient

: public ServerConnector

-id: string
-ip: string
-port: int

+ChatServerConnector(string, string, int)
+virtual replyData(Invoke)
+sendMessage(string)

Interfaces of protocol

<<Interface>>

IServer

#virtual PORT() -> int
#acceptor: tcp::acceptor

+IServer()
+virtual ~IServer()

+virtual start()
+virtual close()
#virtual addClient(tcp::socket)

<<Interface>>

IProtocol

+IProtocol()
+virtual ~IProtocol() = default

+virtual sendData(Invoke)
+virtual replyData(Invoke)

<<Interface>>

IClient

: public virtual IProtocol

#virtual BUFFER_SIZE() -> int

#socket: tcp::socket
#sendMutex: mutex

+IClient()
+virtual ~IClient()

+virtual listen()
+virtual sendData(Invoke)

ServerConnector

public virtual IClient

#virtual IP() -> String
#virtual PORT() -> int
#virtual MY_IP() -> string

#ioService: io_service
#endPoint: tcp::endpoint
#localEndPoint: tcp::endpoint

+ServerConnector()
+virtual ~ServerConnector()

+virtual start()

Chat service

Chat service - an example of clouse service

ChatServer
: public Server

#virtual PORT() -> int := 37749
-rooms:RoomArray

+ChatServer()
#virtual createUser() -> User

RoomArray
: public CriticalDictionary<Room>,
public IEntityGroup

#virtual TAG() -> string = "roomArray"
#virtual CHILD_TAG() -> string = "room"

-server: ChatServer

+RoomArray(ChatServer)
+notify()

-toXML() -> XML
-toInvoke() -> Invoke

ChatUser
: public User

+ChatUser(Server)
#virtual createClient(int, Socket)

#virtual doJoin(Invoke) -> bool
#virtual doLogin(Invoke) -> book

ListService
: public Service

+ListService(Client)
+virtual replyData(Invoke)

ChatClient
: public Client

+ChatClient(User)
#virtual createService(string) -> Service

ChatService
: public Service

-room: Room

+ChatRoomService(Client, ChatRoom)
+virtual ~ChatRoomService()

+virtual replyData(Invoke)

Room
public Entity,
public IProtocol

#virtual TAG() -> string = "room"

-name: string
-host: ChatUser
-parcitipants: CriticalSet<CharClient>

+Room(string, ChatUser)
+virtual ~ChatRoom()
~~+virtual construct(XML)~~

+void registerParticipant(ChatService)
+void eraseParticipant(ChatService)

+virtual replyData(Invoke)
+virtual sendData(Invoke)

+virtual key() -> string
+virtual toXML() -> XML

ChatMessage
: public Entity

#virtual TAG() -> string := "message"

-orator: string
-listeners: list<string>
-message: string

+ChatMessage()
+virtual construct(XML)

+virtual toXML() -> XML

Service Package in Protocol

A package for building cloud service

Interaction

Chief System

Chief
: public ExternalServerArray

+Chief(string)
+virtual start()
#virtual createChild() -> ExternalSystem
+virtual replyData(Invoke)
-callPacker()
-callTSP()
+static main()

1 1
TSP, Packer & Reporter
1 3

MasterDriver
: public ExternalServer

+MasterDriver
(IProtocol, **string**, string, port)
#virtual createChild() ->
ExternalSystemRole := nullptr
+virtual 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.

Master Systems

Reporter System

Reporter
: public ChiefDriver

+Reporter()
#virtual addClient(Socket)
+virtual 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: 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 Package

ChiefDriver
: public IServer,
public IClient

#master: IProtocol
#port: int
#mtx: mutex
+ChiefDriver(IProtocol, int)
#virtual addClient(Socket)
+virtual replyData(Invoke)

1 1
1 1

Master
: public ParallelClientArray

#chiefDriver: ChiefDriver
#mtx: mutex
#optimized: size_t
+Master(int)
+virtual start()
#virtual createChild() -> ExternalSystem
#virtual addClient(Socket)
+virtual replyData(Invoke)
#virtual optimize(XML)
#virtual replyOptimization(XML)

1 1
0 N

SlaveDriver
: public master::ParallelClient

+SlaveDriver()
+virtual replyData(Invoke)

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 3 + 1 components** of protoco modulel by following the ChiefDriver example.

Packer Master System

PackerMaster
: public Master

#packer: Packer
+PackerMaster()
#virtual optimize(XML)
#virtual replyOptimization(XML)
+static main()

Packer Mediator System

PackerMediator
: public Master,
public ParallelClientArrayMediator

+PackerMediator()
+virtual void start()
+virtual sendPieceData
(Invoke, size_t, size_t)
#virtual replyOptimization(XML)
+static main()

TSP Master System

TSPMaster
: public Master

#scheduler: Scheduler
+TSPMaster()
#virtual optimize(XML)
#virtual 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.

You can learn how to utilize **master module** in protocol by following the example.

Slave Systems

Abstract Class

Slave
: public slave::ParallelClient

+Slave(string, int)
+virtual replyPieceData
(Invoke, size_t, size_t)
#virtual sendOptimization(XML)
#virtual optimize(XML, size_t, size_t)

Packer Slave System

PackerSlave
: public Slave

+PackerSlave(string, int)
+virtual optimize(XML, size_t, size_t)
+static main()

TSP Slave System

TSPSlave
: public Slave

+TSP(string)
+virtual optimize(XML, size_t, size_t)
+static main()

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.

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.

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.