C++ Class Diagram

1. library
2. protocol
   - 2-1. service
   - 2-2. master
   - 2-3. slave
3. nam-tree
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()

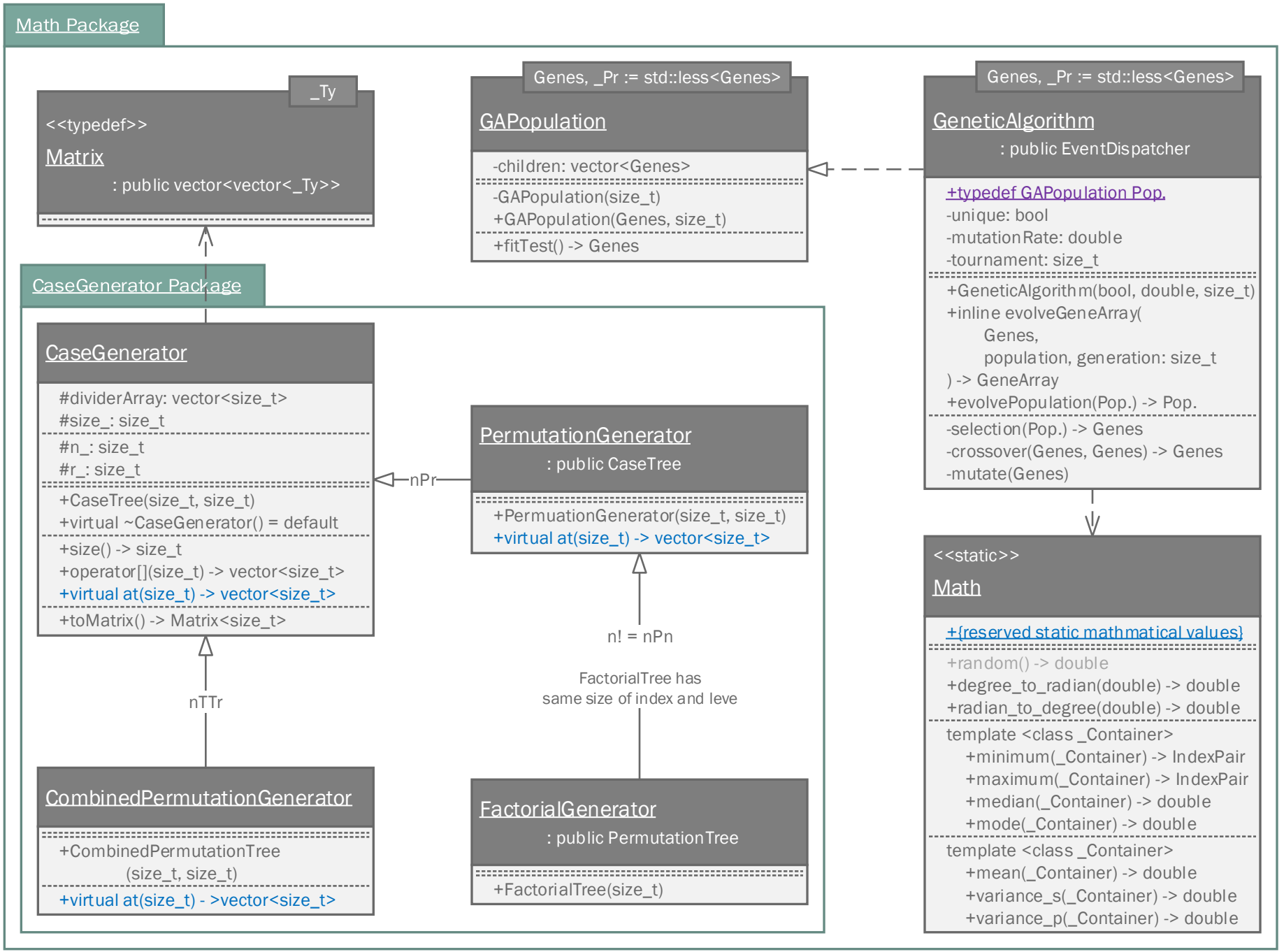
+{addDate(int)}
+{setDate(int)}

+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

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)

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

TSQLStatement : public SQLStatement

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

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)

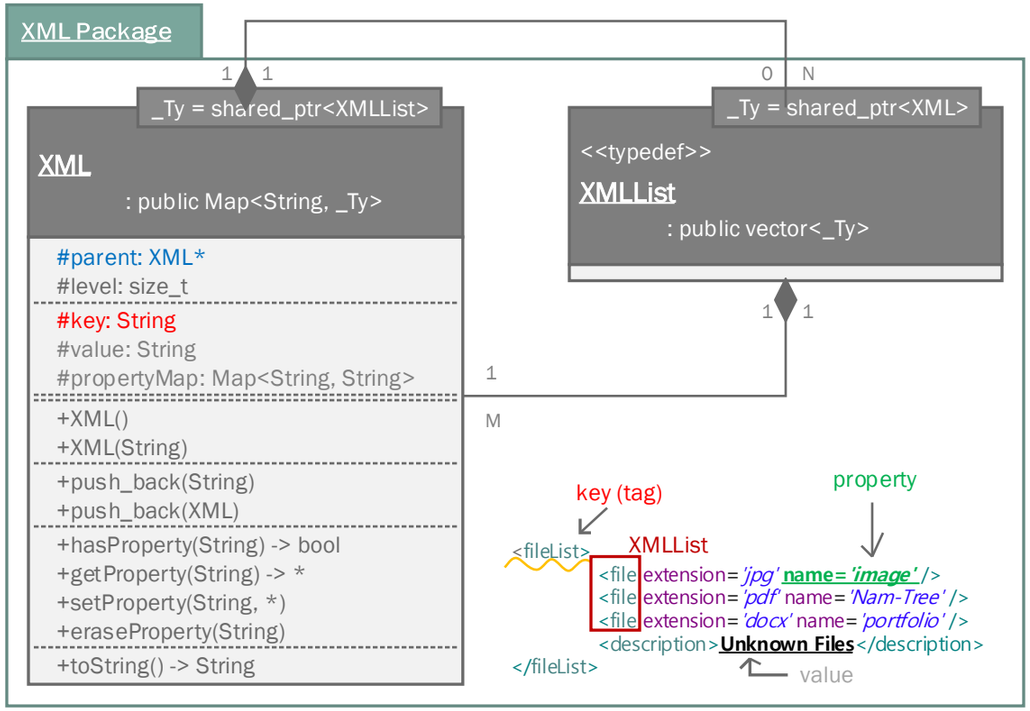
generates when call send

ResultEvent : public Event

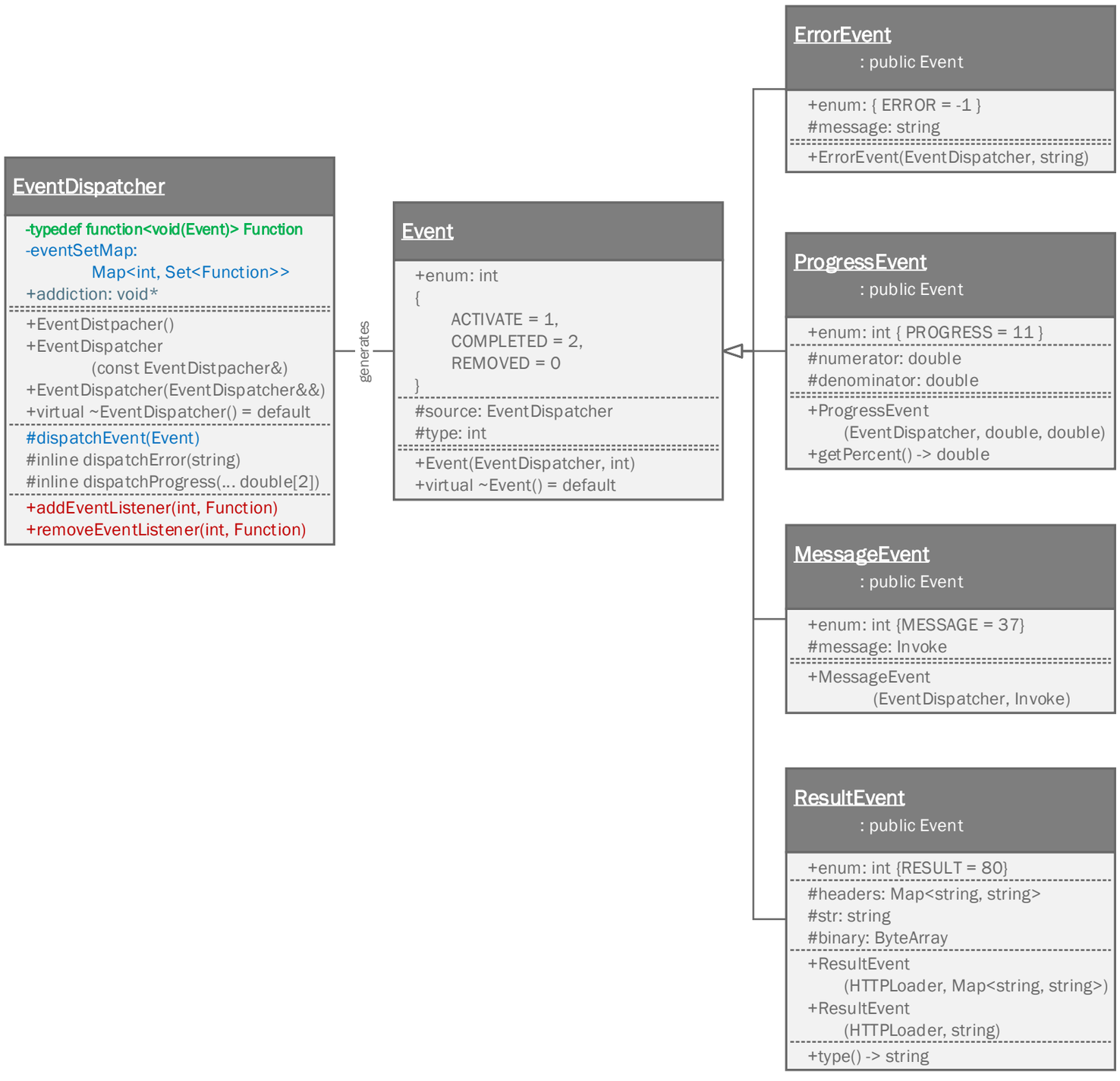
- +enum: int {RESULT = 80}
- #headers: Map<string, string>
- #str: string
- #binary: ByteArray
- +ResultEvent (HTTPLoader, Map<string, string>)
- +ResultEvent (HTTPLoader, string)
- +type() -> string

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.

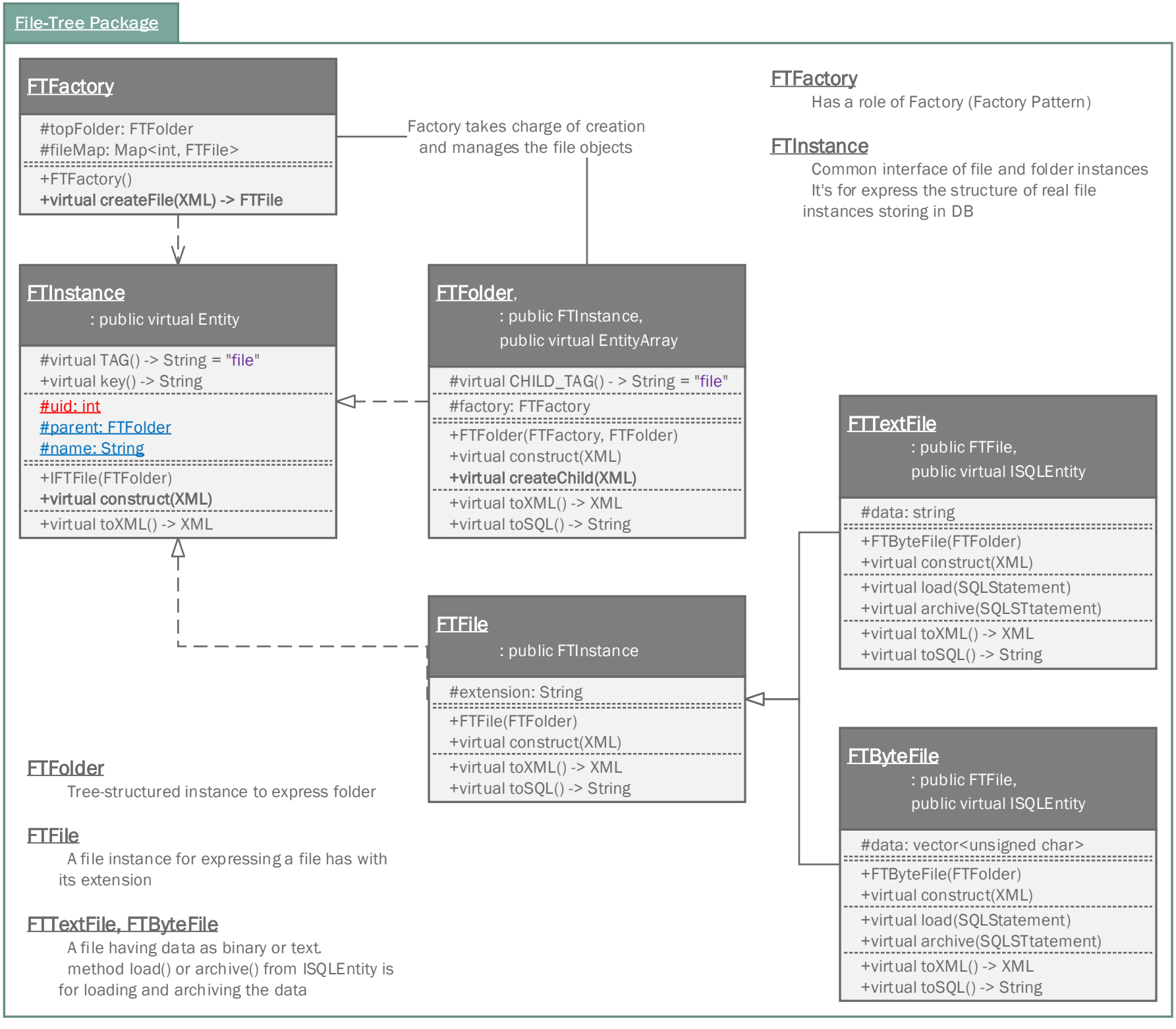
~~MessageEvent~~

~~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

invoke
entity
interfaces
external system
distributed system
parallel system
cloud service

Protocol Interfaces

Basic Interfaces

<<Interface>>
IProtocol
+IProtocol()
+virtual IProtocol() = default
+virtual replyData(Invoke)
+virtual sendData(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 connect()

<<Interface>>
IServer
#virtual PORT() -> int
#acceptor: tcp::acceptor
+IServer()
+virtual ~IServer()
+virtual open()
+virtual close()
#virtual addClient(tcp::socket)

Socket

Boost.Asio

The boost.asio is the most famouse network library in C++, which is supporting cross compile between multiple operating systems. Samchon Framework have adopted the boost.asio in protocol.

STL

In C++17, standard library of network communication is planned to be established. As that reason, network library used in Samchon Framework can be changed in near future.

inherits virtually

inhertis virtually

Interfaces for Web socket protocol

<<Interface>>
IWebClient
: public virtual IClient
+IWebClient()
+virtual ~IWebClient() = default
+virtual listen()
+virtual sendData(Invoke)

Implements virtually to follow web-socket protocol

WebServerConnector
: public virtual ServerConnector,
public virtul IWebClient
+WebServerConnector()
+virtual ~WebServerConnector() = def.
+virtual connect()

<<Interface>>
IWebServer
: public virtual IServer
+IWebServer()
+virtual ~IWebServer() = default
+virtual open()

implements

Can construct any type of network system by implementing those interfaces

implements

Derived Network Libraries

Cloud Service Package

Distributed Processing Package

External System Package

Parallel Processing Package

Creating Network I/O Class

You can construct any type of network system, even how the system is enormously complicated, by just implementing and combinating:

IProtocol, *IServer* and *IClient* those are called basic 3 components of Network.

IProtocol

An interface for network I/O

To realizing chain of responsibility of the network I/O

IServer

An interface for a physical server

IClient

An interface for a client.

Not only mean a physical client, but also a driver for a client in a physical server

IServer:addClient()

service::*Server* -> service::User -> service::*Client*

ExternalClientArray (A physical server) -> *ExternalClient* (A driver of a client)

ServerConnector

A server connector for a physical client.

If you want to connect to a server, then implements this ServerConnector and just override some methods. That's all.

Interfaces for Web-Socket

Interfaces 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 thos interfaces.

Invoke and invoke-history package

Invoke Package

Invoke
: vector<InvokeParameter>

-listener: string

template <class _Ty, class ... _Args>
+Invoke(string, _Ty, ... _Args)
+Invoke(string)
+Invoke(XML)

+toXML() -> XML
+toSQL() -> string

InvokeParameter

-name: string
-type: string
:= {number, string, XML, ByteArray}
-value: any

template <class _Ty>
+InvokeParameter(string, _Ty)
+InvokeParameter(string, string, string)
+InvokeParameter(XML)

+toXML() -> XML
+toSQL() -> string

Invoke: Express a message (function)
Invoke::listener := almost same with name of a function.

InvokeParameter: Parameter in a function.
When a parameter is not atomic data like a Data-set(structure, list), use XML.

Invoke History Package

InvokeHistoryList
: public EntityList<InvokeHistory>

#virtual TAG() -> strig
#virtual CHILD_TAG() -> string

+InvokeHistoryList()

InvokeHistory
: public Entity

#virtual TAG() -> string

#listener: string
#startTime: Datetime
#endTime: double

+InvokeHistory()
+InvokeHistory(Invoke)
+virtual ~InvokeHistory() = default

+virtual construct(XML)
+notifyEnd()

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

History for Parallel P. system

PRInvokeHistory
: public InvokeHistory

#startIndex: size_t
#size: size_t

+PRInvokeHistory()

Histories for Distributed P. system

DSInvokeHistory
: public InvokeHistory

#system: DistributedSystem
#role: DistributedSystemRole

+InvokeHistory
(
DistributedSystem,
DistributedSystemRole
)
+virtual toXML() -> XML

DSRoleHistory
: public Entity

+virtual TAG() -> string

#system: DistributedSystem
#role: ExternalSystemRole
#registered: Datetime
#erased: shared_ptr<Datetime>

+DSRoleHistory
(
DistributedSystem,
DistributedSystemRole,
Datetime
)
+virtual toXML() -> XML

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 3 + 1 Components" required on building some network system; **IProtocol**, **IServer** and **IClient**.
You can construct any type of network system, even how the system is enormously complicated, by just implementing and combinating those "Basic + 1 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>
                        <group>3</group>
                        <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.

DSInvokeHistory

A reported InvokeHistory in framework of a master of distributed processing system. The master of a distributed processing system estimates performance index of a slave system by those reports.
Master distributes roles to slave systems optimally from the estimated performance index which is calculated from those reports.

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.

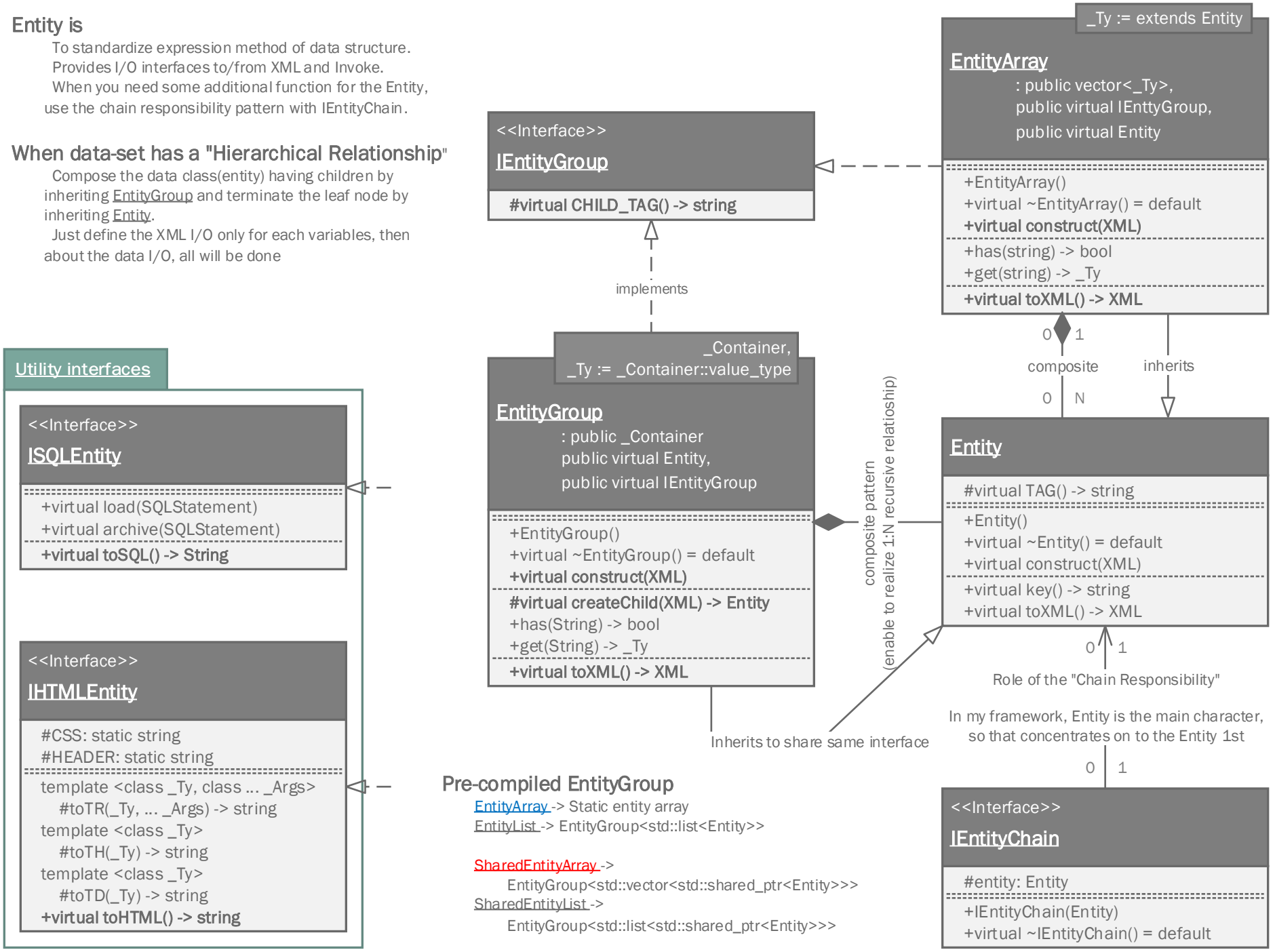
Entity Package

Entity is

To standardize expression method of data structure.
Provides I/O interfaces to/from XML and Invoke.
When you need some additional function for the Entity, use the chain responsibility pattern with IEntityChain.

When data-set has a "Hierarchical Relationship"

Compose the data class(entity) having children by inheriting EntityGroup 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

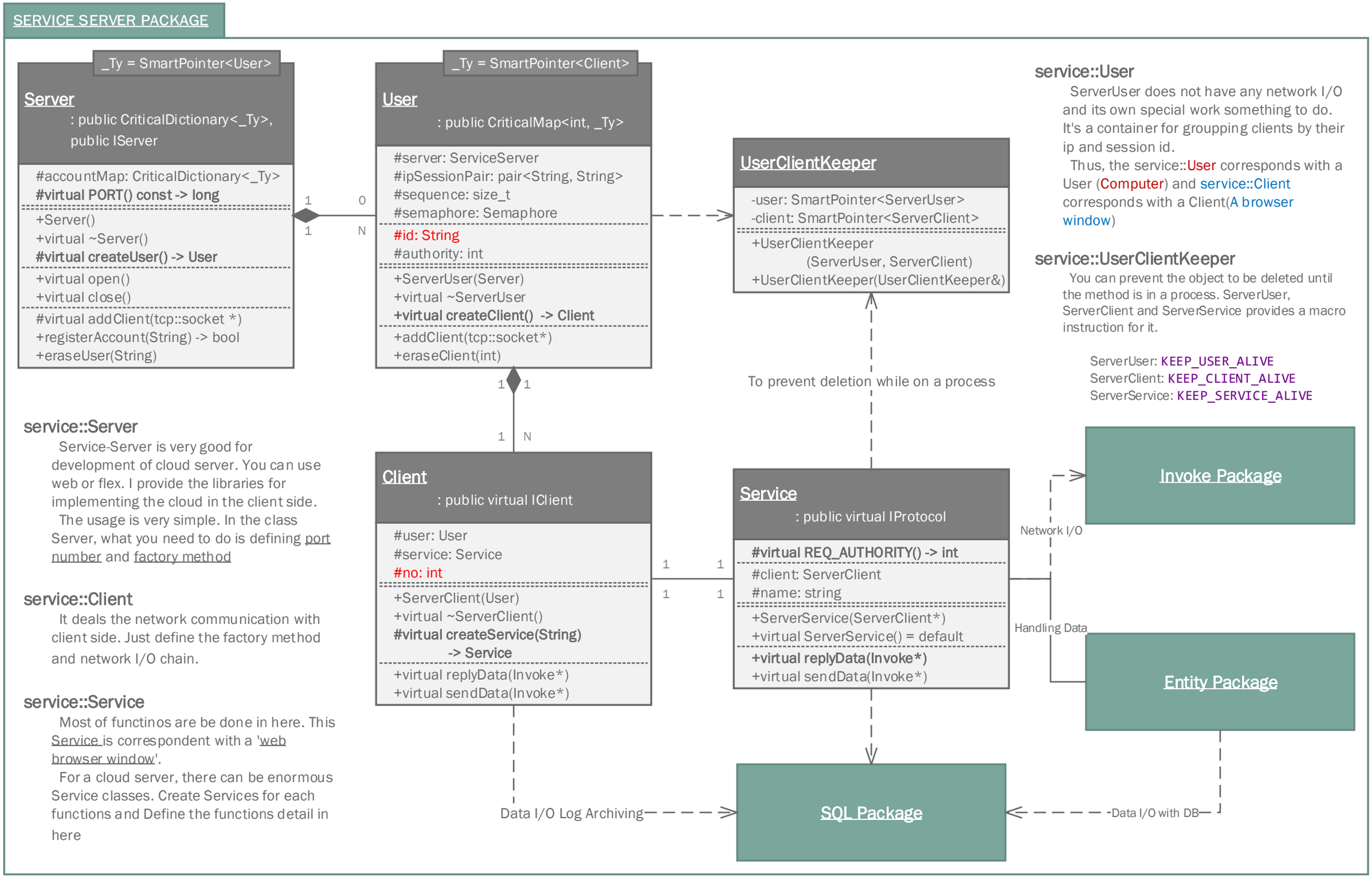


Utility interfaces

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>>>


SERVICE SERVER PACKAGE
_Ty = SmartPointer<User>
Server
: public CriticalDictionary<_Ty>,
public IServer
#accountMap: CriticalDictionary<_Ty>
#virtual PORT() const -> long
+Server()
+virtual ~Server()
#virtual createUser() -> User
+virtual open()
+virtual close()
#virtual addClient(tcp::socket *)
+registerAccount(String) -> bool
+eraseUser(String)
1
0
1
N
_Ty = SmartPointer<Client>
User
: public CriticalMap<int, _Ty>
#server: ServiceServer
#ipSessionPair: pair<String, String>
#sequence: size_t
#semaphore: Semaphore
#id: String
#authority: int
+ServerUser(Server)
+virtual ~ServerUser
+virtual createClient() -> Client
+addClient(tcp::socket*)
+eraseClient(int)
UserClientKeeper
-user: SmartPointer<ServerUser>
-client: SmartPointer<ServerClient>
+UserClientKeeper
(ServerUser, ServerClient)
+UserClientKeeper(UserClientKeeper&)
To prevent deletion while on a process
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.
ServerUser: KEEP_USER_ALIVE
ServerClient: KEEP_CLIENT_ALIVE
ServerService: KEEP_SERVICE_ALIVE
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
1
1
1
N
Client
: public virtual IClient
#user: User
#service: Service
#no: int
+ServerClient(User)
+virtual ~ServerClient()
#virtual createService(String)
-> Service
+virtual replyData(Invoke*)
+virtual sendData(Invoke*)
1
1
1
1
Service
: public virtual IProtocol
#virtual REQ_AUTHORITY() -> int
#client: ServerClient
#name: string
+ServerService(ServerClient*)
+virtual ServerService() = default
+virtual replyData(Invoke*)
+virtual sendData(Invoke*)
Invoke Package
Network I/O
Handling Data
Entity Package
SQL Package
Data I/O Log Archiving
Data I/O with DB

External Sytstem Package

Server or Client

ExternalClientArray
: public v. ExternalSystemArray, public v. IServer

#virtual PORT() -> int
#virtual MY_IP() -> string
#port: int

+ExternalClientArray()
+virtual construct(XML)
+virtual start()

#virtual addClient(tcp::socket)
+virtual toXML() -> XML

ExternalServerArray
: public v. ExternalSystemArray

+ExternalServerArray()
+virtual start()

<<creates>>

ExternalServer
: public virtual ExternalSystem, public virtual ServerConnector

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

+ExternalServer()

ExternalClient
: public virtual ExternalSystem, public virtual IClient

+ExternalClient()

Basic System

ExternalSystemArray
: public virtual EntityArray, public virtual IProtocol

#virtual TAG() -> String := "systemArray"
#virtual CHILD_TAG() -> string

#myIP: string

+ExternalServerArray()
+virtual construct(XML)
#virtual createChild(XML) -> Entity

+virtual start()
+hasRole(string) -> bool
+getRole(string) -> ExternalSystemRole

+virtual sendData(Invoke)
+virtual toXML() -> XML

ExternalSystem
: public virtual EntityArray, public virtual IProtocol

#virtual TAG() -> String := "system"
#virtual CHILD_TAG() -> String

#name: string
#ip: string
#port: int

+ExternalSystem()
#virtual createChild(XML) -> Entity

+virtual replyData(Invoke)
+virtual toXML() -> XML

ExternalSystemRole
: public virtual Entity

#virtual TAG() -> String := "role"
#virtual key() -> String

#system: ExternalSystem
#name: string

+ExternalSystemRole(ExternalSystem)
+virtual construct(XML)

+virtual sendData(Invoke)
+virtual toXML() -> XML

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

ExternalSystemArray

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

Access by role

```
ExternalSystemMaster *master;
ExternalSystem *system = master->getRole(string)->getSystem();

server.sendData(Invoke)
```

Deriveds

Distributed Processing System

This Package's System and Role have those own measurement index for performance.

Master distributes Role(s) to Slavess) optimally

Parallel Processing System

Parellel System does not have Role.
ParellelSystemArray distributes uniformly

When ExternalServer

The server's role is already defined in the server side.
You can pre-define the role on XML, or fetch role XML tag from ExternalServer.

When ExternalClient

Each client's roles are not defined yet.
Distribute roles as you want.

Warning!

Be careful about the virtual inheritance

In C++, in the body part, virtual inheritance does not take grand-parent's non-default constructor implicitly. In C++ standard, grand-parent's virtual inheritance only rides on defualt constructor.
In that reason, please write on the grand-parent's constructors explicitly when creating objects derived from the classes in External System Package

External Sytstem Package II - Alternative model; No virtual inheritance **Depreciated in C++, but will be adopted to other language**

Deriveds

Distributed Processing System

This Package's System and Role have those own index for performance.
Master distributes Role(s) to Slavess) optimally

Also, Master and Slave can be delegated by composing tree-structure

Parallel Processing System

Parellel System does not have Role.
ParellelSystemMaster distributes works uniformly

External System Network Chain

ExternalSystemClientSocket
: public ExternalSystemSocket,
public virtual IClient

+ExternalSystemClientSocket
(ExternalSystem, tcp::socket)
+virtual start()

ExternalSystemServerSocket
: public ExternalSystemSocket,
public virtual IServerConnector

+ExtSysServerSocket(ExternalSystem)
+virtual start()

ExternalSystemSocket
: public virtual IClient

#system: ExternalSystem
+ExtSysSocket(ExternalSystem)
+virtual ~ExtSysSocket = default
+virtual start()
+virtual replyData(Invoke)

Basic System

ExternalSystemArray
: public virtual EntityArray,
public virtual IProtocol,
private IServer

#virtual TAG() -> String := "systemArray"
#virtual CHILD_TAG() -> String
+virtual DIRECTION()
-> enum {SERVER, CLIENT}
+virtual BUFFER_SIZE() -> int
#parent: IProtocol
#roleMap: Map<String, ExtSysRole>
-myIP: String
-port: int
+ExternalSystemArray(IProtocol)
+virtual construct(XML)
#virtual createChild(XML) -> Entity
+virtual start()
+hasRole(String) -> bool
+getRole(String) -> ExtSysRole
+virtual sendData(Invoke)
+virtual replyData(Invoke)

ExternalSystem
: public virtual EntityArray,
public virtual IProtocol

#virtual TAG() -> String := "system"
#virtual CHILD_TAG() -> String
#master: ExternalSystemArray
-socket: ExternalSystemSocket
-ip: String
-myIP: String
-port: int
+ExternalSystem(ExternalSystemArray)
+virtual construct(XML)
#virtual createChild(XML) -> Entity
+virtual start()
+virtual sendData(Invoke)
+virtual replyData(Invoke)

<<Interface>>
ExternalSystemRole
: public virtual Entity

#virtual TAG() -> String := "role"
#virtual key() -> String
#system: ExternalSystem
+ExternalSystemRole(Slave)
+virtual ~ExternalServerRole()

ExternalSystemRole

ExternalSystemMaster 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

ExternalSystemArray

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

Access by Role

```
ExternalSystemMaster *master;
ExternalSystem *system =
    master->getRole(String)->getSystem();
server.sendData(invoke)
```

Distributed Processing System

Derives

Derived classes

Masters

DistributedServerArray
: public v. DistributedSystemArray,
private v. ExternalClientArray

+DistributedServerArray()
+virtual start()

DistributedClientArray
: public v. DistributedSystemArray,
private v. ExternalServerArray

#virtual PORT() -> string
#virtual MY_IP() -> string

+DistributedClientArray()
+virtusl start()

<<creates>>

Slaves

DistributedIServer
: public v. DistributedSystem,
public virtual ExternalServer

+ExternalServer()
+virtual construct(XML)

DistributedClient
: public v. DistributedSystem,
public virtual ExternalClient

+DistributedClient()

Base classes

DistributedSystemArray
: private v. ExternalSystemArray

-roleDictionary:
Dictionary<DistributedSystemRole>
-mtx: RWMutex
#generation: size_t
#population: size_t
#mutationRate: double

+DistributedSystemArray()
+virtual ~DistributedSystemArray()
+virtual construct(XML)
+virtual start()
#virtual createRole() ->
DistributedSystemRole
+virtual allocateRoles()
+getRole(string) ->
DistributedSystemRole
+virtual sendData(Invoke)
+virtual toXML() -> XML

Different aspect with ExternalSystem

Relationship between Role is different.
The Role is not belonged only to a System and the Role is not even created by SystemArray.

Allocating Roles to Systems

History of message transmissions and those elapsed times are archived. SystemArray calculates performance index of Systems and Roles.
Roles will be allocated or re-allocated System from those performance indices by genetic algorithm. If number of Systems and Roles are not too much, then combined permuation case generator will be used instead of the genetic algorithm.

Mediator

You even can compose tree-structured distributed processing system with DistributedSystemArrayMediator

Belongs to

Belongs to

DistributedSystem
: private virtual ExternalSystem

-performance: double
-invokeHistories: DSInvokeHistoryList
-roleHistories: DSRoleHistoryList

+DistributedSystem()
+virtual ~DistributedSystem()
+virtual construct(XML)
+v. registerRole(DistributedSystemRole)
+v. eraseRole(DistributedSystemRole)
+virtual sendData(Invoke)
+virtual replyData(Invoke)
+virtual toXML() -> XML

Has M: N relationship

DistributedSystemRole
: public SystemRole

-master: DistributedSystemArray
-systems: vector<DistributedSystem>
-performance: double
-invokeHistories: DSInvokeHistoryList
-roleHistories: DSRoleHistoryList

+DistributedSystemRole()
+virtual construct(XML)
+virtual toXML() -> XML

Used to estimate performance

Histories

DSInvokeHistory
: public InvokeHistory

#system: DistributedSystem
#role: DistributedSystemRole
#datetime: Datetime
#elapsedTime: double

+InvokeHistory(... args)
+virtual toXML() -> XML

DSRoleHistory
: public Entity

+virtual TAG() -> string
#system: DistributedSystem
#role: ExternalSystemRole
#registered: Datetime
#erased: shared_ptr<Datetime>

+DSRoleHistory
(
DistributedSystem,
DistributedSystemRole,
Datetime
)
+virtual toXML() -> XML

DistributedSystemArrayMediator is a DistributedSystem
in view of the real-DistributedSystemArray

Mediator

DistributedSystemArrayMediator
: public v. DistributedSystemArray

#proxy: DSMediatorProxy

+DistributedSystemArrayMediator()
+virtual start()
#virtual createProxy()
-> DSMediatorProxy
+virtual replyData(Invoke)
+virtual toXML() -> XML

<<Mediator to real master>>
DistributedSystemRole::replyData()
-->>
DistributedSystemArrayMediator::replyData()
-->>
DSMediatorProxy::sendData()

DSMediatorProxy
: public v. slave::DistributedSystem

#virtual PORT() -> int
#mediator: DSArrayMediator

+MasterProxySocket()
+virtual start()
+virtual replyData(Invoke)

DistributedServerArrayMediator
: public v. DistributedSystemArrayMediator,
public v. DistributedServerArray

#port: int

+MasterProxyClient(IProtocol)
+virtual start()
#virtual createSocket() -> IProtocol

DistributedClientArrayMediator
: public v. MasterProxy,
public v. MasterServer

#masterIP: String

+MasterProxyServer(IProtocol)
+virtual construct(XML)
+virtual start()
#virtual createSocket() -> IProtocol
+virtual toXML() -> XML

MediatorProxyServer
: public virtual MediatorProxy,
public v. slave::DistributedServer

+MasterProxyServerSocket()
+virtual start()

MediatorProxyClient
: public virtual MediatorProxy
public v. slave::DistributedClient

#virtual IP() -> String
+ExternalSystemProxyClient()

Parallel Processing System

Parallel Server Package

ParallelServerArray
: public v. ParallelSystemArray

+ParallelServerArray()
+virtual ~ParallelServerArray() = def.
+virtual start()

ParallelServer
: public virtual ParallelSystem,
public virtual ServerConnector

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

+ParallelServer()
+virtual ~ParallelServer() = default

Parallel Client Package

ParallelClientArray
: public v. ParallelSystemArray,
public virtual IServer

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

+ParallelClientArray()
+virtual ~ParallelClientArray() = def.
+virtual start()

ParallelClient
: public virtual ParallelSystem,
public virtual IClient

+ParallelClient()
+virtual ~ParallelClient() = def.

Parallel System Pakcage

ParallelSystemArray
: public SharedEntityArray,
public virtual IProtocol

#virtual TAG() -> string := "systemArray"
#virtual CHILD_TAG -> string := "system"

#myIP: string

+ParallelSystemArray()
+virtual ~ParallelSystemArray() = def.
+virtual start()

+virtual replyData(Invoke)
+virtual sendData(Invoke)
+sendData(Invoke, size_t, size_t)

+virtual toXML() -> XML

ParallelSystem
: public Entity,
public virtual IProtocol

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

#parent: ParallelSystemArray
#ip: string
#port: int
#performance: double

+ParallelSystem()
+virtual ~ParallelSystem() = default
+virtual constrcut(XML)

+virtual replyData(Invoke)
+sendData(Invoke, size_t, size_t)

+virtual toXML() -> XML

Mediator

ParallelSystemArrayMediator
: public v. ParallelSystemArray

#proxy: PRMediatorProxy

+DistributedSystemArrayMediator()
+virtual start()

#virtual createProxy()
-> PRMediatorProxy
+virtual replyData(Invoke)

+virtual toXML() -> XML

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

PRMediatorProxy
: public virtual IProtocol

#virtual PORT() -> int
#mediator:
DistributedSystemArrayMediator

+MasterProxySocket(MasterProxy)
+virtual start()
+virtual replyData(Invoke)

DistributedServerArrayMediator
: public v. DistributedSystemArrayMediator,
public v. DistributedServerArray

#port: int

+MasterProxyClient(IProtocol)
+virtual start()
#virtual createSocket() -> IProtocol

DistributedClientArrayMediator
: public v. MasterProxy,
public v. MasterServer

#masterIP: String

+MasterProxyServer(IProtocol)
+virtual construct(XML)

+virtual start()
#virtual createSocket() -> IProtocol

+virtual toXML() -> XML

DSMediatorProxyServer
: public v. MasterProxySocket,
public v. OneToOneServer

+MasterProxyServerSocket
(MasterProxy)
+virtual start()

DSMediatorProxyClient
: public v. MasterProxySocket
public v. ServerConnector

#virtual IP() -> String

+ExternalSystemProxyClient
(ExternalSystemProxy)

Nam-Tree

file
criteria

Nam-Tree File Package

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

FTInstance
: public virtual Entity,
public ISQLEntity

#virtual TAG() -> String = "file"
+virtual key() -> String

#uid: int
#parent: FTFolder
#name: String

+IFTFile(FTFolder)
+virtual construct(XML)

+virtual toXML() -> XML
+virtual toSQL() -> String

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

FTFile
: public FTInstance

#extension: String
#content: String

+FTFile(FTFolder)
+virtual construct(XML)

+virtual toXML() -> XML
+virtual toSQL() -> String

FTByteFile
: public FTFile

#data: vector<unsigned char>

+FTByteFile(FTFolder)
+virtual construct(XML)

+virtual load(SQLStatement)
+virtual archive(SQLSTtatement)

+virtual toXML() -> XML
+virtual toSQL() -> String

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

FTFolder
Tree-structured instance to express folder

FTFile
A instance for expressing a text-based file has a extension and its own centent.

FTByteFile
A file instance having binary data. If the file size is enough small, then you can get or insert binary data at once using Base64 en/deconding
But the file size is not small, then load and archive binary data independently

Nam-Tree File Package

NTFactory
: public FTFactory

#entity: NTEntityGroup

+FTFactory(NTEntityGroup)

+virtual createFile(XML) -> FTFile
+virtual createCriteria() -> NTCriteria
+virtual createSide() -> NTSide

NTFile
: public FTFile,
public INTExploreParameter

#factory: NTFactory
#parameterArray: NTParameterArray
#otherside: NTFile
~~#header: String~~
~~#getFunction: String~~
~~#composerFunction: String~~
#function: Function

+NTFile(NTFactory, FTFolder)
+virtual construct(XML)

+toXML() -> XML
+toSQL() -> String

Function(... parameters)

NTParameterArray
: public virtual EntityArray,
public ISQLEntity

#virtual TAG() -> String
#virtual CHILD_TAG() -> String

+NTParameterArray()
#virtual createChild(XML) -> Entity

+virtual toSQL() -> String

<<Interface>>
INTExploreParameter
: public virtual Entity,
public ISQLEntity

#minimum: double
#maximum: double
#bin: double

+INTExplorer()
+virtual construct(XML)

+virtual toXML() -> XML
+virtual toSQL() -> String

refers to explore
to realize A.N.N

Criteria Package
At the next page

To explore in parameter level

NTParameter
: public virtual EntityArray,
public INTExploreParameter

#virtual TAG() -> String = "parameter"
#virtual CHILD_TAG() -> String

#name: String
#initialValue: double
~~#initialStrValue: String~~

+NTParameter()
+virtual construct(XML)
#virtual createChild(XML) -> Entity

+virtual toXML() -> XML
+virtual toSQL() -> String

Will be expressed as ComboBox in UI

NTParameterDetermined
: public virtual Entity
public ISQLEntity

#virtual TAG() -> String = "determined"
+virtual key() -> String

#label: String
#value: double
~~#strValue: String~~

+NTParameter()
+virtual construct(XML)

+virtual toXML() -> XML
+virtual toSQL() -> String

NTFactory
Has a role of Factory (Factory Pattern)
Not only manages the creation of file instances, but also manages the creaion of criteria and sides
About the additional creation role, may better to read the next page, criteria

INTExploreParameter
A parameter container used in explore NamTree explores until to reach the bin from minimum to maximum.
The exploration follows the method of multi-dimensional grid.

NTFile
A metadata class of a function used in Nam-Tree's NTCriteria and NTSide
It contains the name and address of the function and its parameters.
Function<double(NTIterator, Map<String, double>)

NTParameter
A metadata of a parameter in a function

NTParameterDetermined
A list of candidate parameters those're already determined. This determined parameters will be shown as an item of a ComboBox.
If this determined parameters are defined, then exploration doesn't use the INTExploreParemeter's. It will hardly iterates all the determined parameters.

Nam-Tree Criteria

NTFactory
: public FTFactory

#topCriteria: NTCriteria
#entity: NTEntityGroup

+FTFactory(NTEntityGroup)
+virtual createFile(XML) -> IFTFile
+virtual createCriteria() -> NTCriteria
+virtual createSide() -> NTSide

Manages

<<Interface>>
NTHistoryData

+NTHistoryData()
+virtual ~NTHistoryData() = default
+virtual calcResult() -> double
+virtual begin() -> NTIterator
+virtual end() -> NTIterator

Creates iterator

<<Interface>>
NTIterator
: public IOperator

#history: NTHistoryData

+NTIterator(NTHistoryData)
+virtual ~NTIterator() = default
+virtual operator++() -> NTIterator
+virtual operator–() -> NTIterator
+virtual operator<() -> bool
+virtual operator==() -> bool

To iterate historial data

NTCriteria

A conditional expression has weight
F(x) = (NTSide < NTSide ? 1 : 0) x weight
It has a tree-structure so that can express enumerous conditions It also can be used to realize A.N.N. by setting side to be NULL

NTSide

A side of a conditional expression, of the NTCriteria

NTFile

Definition of a function used in NTSide

NTEntityGroup

The Interface of historial(regressive) data
Sometimes it can be a learning data to train A.N.N

NTIterator

The Interface of iteration utility
User has to define how to iterate the NTEntityGroup

NTCriteria
: public EntityArray,
public ISQLEntity

#virtual TAG() -> String = "criteria"
#virtual CHILD_TAG() -> String

#factory: NTFactory
#parent: NTCriteria
#left: NTSide
#right: NTSide
#operator: int
#weight: double

+NTCriteria(NTFactory, NTCriteria)
+virtual construct(XML)
#cirtual createChild(XML) -> Entity

+virtual initRetrieve()
+getRetrieved(NTIterator) -> double

+virtual toXML() -> XML
+virtual toSQL() -> String

1 — 1 — A side can be null — 1 — 2

NTSide
: public Entity,
public ISQLEntity

#factory: NTFactory
#file: NTFile
#parameterMap: Map<String, double>

+NTSide(NTFactory)
+virtual construct(XML)

+virtual initRetrieve()
+getRetrieved(NTIterator) -> double

+virtual toXML() -> XML
+virtual toSQL() -> String

Used to create cases

CaseTree Package
To make a lot of condition to explore

NTFile
: public FTFile,
public INTExploreParameter

#parameterList: NTParameterList
#otherside: NTFile

~~#header: String~~
~~#getFunction: String~~
~~#composerFunction: String~~
#function: Function

+NTFile(NTFactory, FTFolder)
+virtual construct(XML)

+toXML() -> XML
+toSQL() -> String

NTSide refers NTFile
The function pointer of NTFile will be called
:= getRetrieve(NTIterator) -> double

NTCriteria

NTCriteria is an object to realize Artificial Neural Network
You can make ANN model having weight and bias

1. A conditional expression with weight
F(x) = (NTSide < NTSide ? 1 : 0) x weight
NTCriteria is made up for conditional expression
If the expression is true, then returns the 1, else it is the false, then returns 0, and multiply weight to the result 1 or 0

2. NTCriteria has a hierarchical relatioship
In vertical relationship: Multiply (X)
In horizontal relationship: Plus (+)

With this rule, you can make enormous conditions. I can sure there's not any condition that can't be expressed by this model.

3. Making bias
Just make a NTCriteria returns only true.
Then it will be the bias returns weight

4. Explore
4-1. Exploring in a NTCriteria (optimize a side)
Make a **NTSide** to be nullptr, then NTCriteria will explore the best value
Nam-Tree will calculate the conditions from mininmum to maximum in INTExploreParameter reach to the bin, by the method of multi-dimensional grid.

4-2. Exploring parameter in NTSide
If you set the parameterMap to be empty, Nam-Tree will explore the best parameter until reach to the bin in INTExploreParameter from minimum to maximum

4-3. Exploring by creating NTCriteria (create conditions)
If all the side (left and right) in a NTCriteria are nullptr, then the NTCriteria will make a lot of children NTCriteria(s) to test a lot of cases that can be, so that best condition will be made up.
This process will ride on same routines of 4-1. and 4-2. for each created cases. Of course, this process needs too much time, so that you may need to be patient.

<<Interface>>
INTExploreParameter
: public virtual Entity

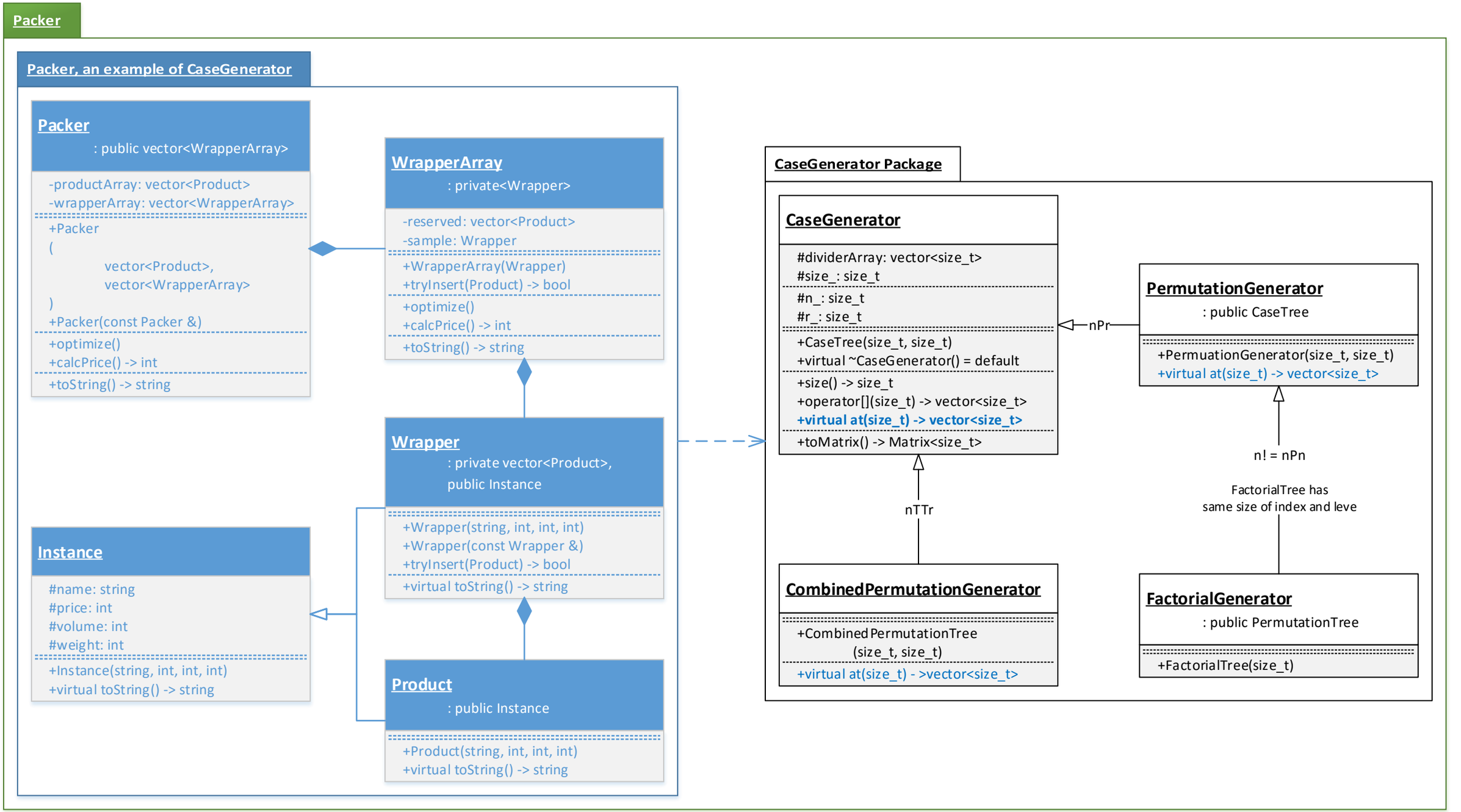
#minimum: double
#maximum: double
#bin: double

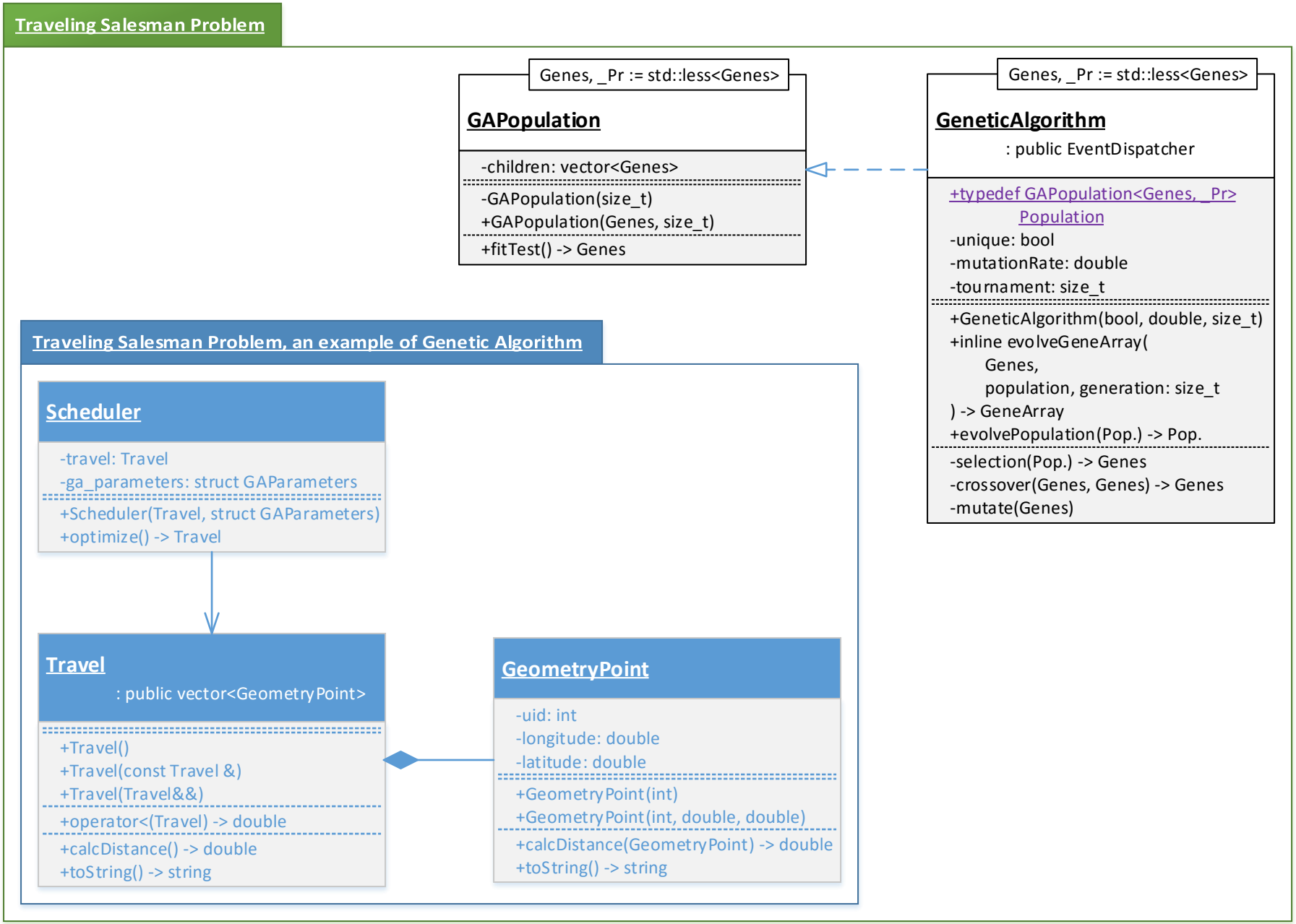
+INTExplorer()
+virtual construct(XML)

+virtual toXML() -> XML

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
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()

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.

TSP Master System

TSPMaster
: public Master

#scheduler: Scheduler
+TSPMaster()
#virtual optimize(XML)
#virtual replyOptimization(XML)
+static main()

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()

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.

TSP Slave System

TSPSlave
: public Slave

+TSP(string)
+virtual 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.