Samchon Framework

Team Samchon

남정호

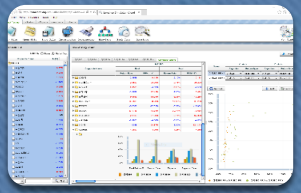
Index

Outline

- Purpose
- Modules
- Languages

Specification

- Message Protocol
- Like OOD
- Utilization



Provides

- Documents
- Examples
- GitHub

Outline

1. Purpose
2. Modules
3. Languages

1. Purpose

- 클라우드 시스템 구축
  - 성능, 메모리 관리가 이슈인 중대형 클라우드 시스템
  - 혹은 기존 C++ 솔루션을 클라우드로 재빌드
- 표준적인 메시지 프로토콜과 데이터 표기법
  - 을 이용하여 통합 시스템을 수월히 구축

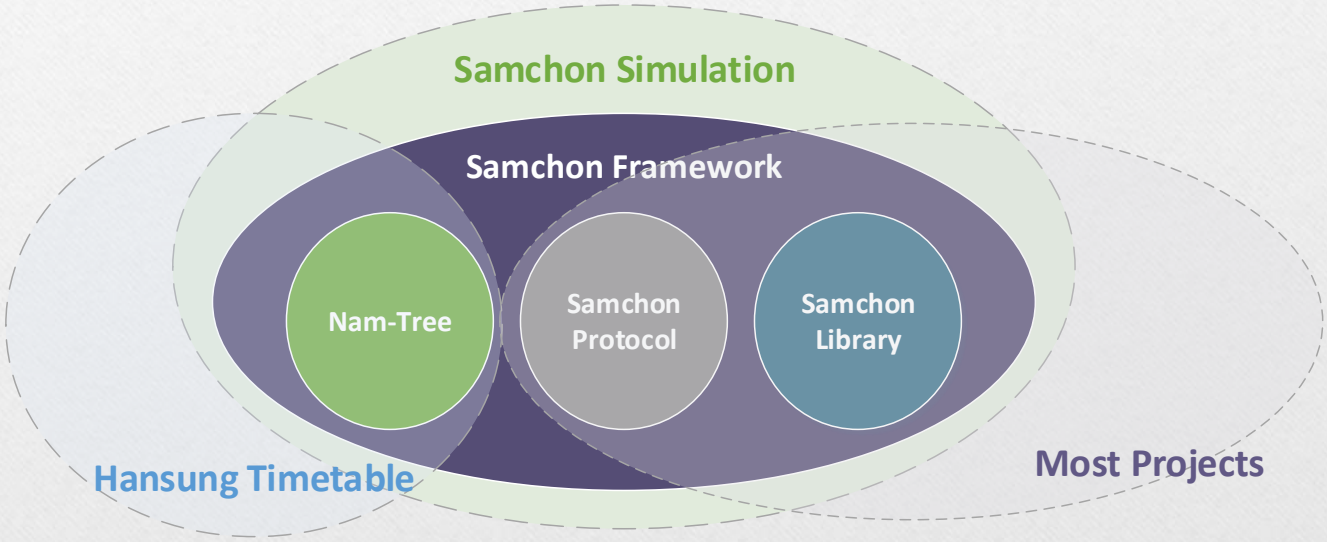
1. Purpose

- 객체지향적 관점에서
  - 네트워크 시스템을 S/W적인 관점에서 수월히 구축
  - ex) 트리 구조의 **분산-병렬처리 시스템**
- 운영체제에 독립적인 C++ 라이브러리 사용
  - 크로스 컴파일 가능

1. Purpose - privately

- 공부
  - 프레임워크를 만들어봄으로써 실력을 늘림
- 인지도, 출세
  - 삼촌 프레임워크의 인지도를 높여
    - 다양한 피드백을 듣고
    - 여러 참여자들을 모집
  - 개인적으로도 출세

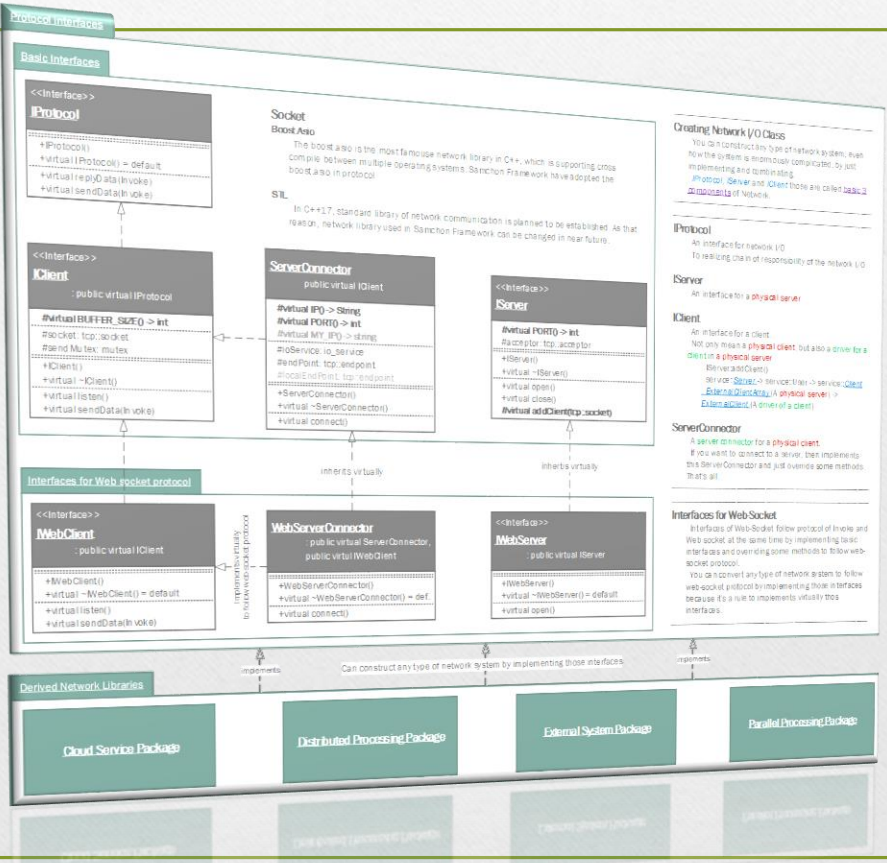
2. Modules


Samchon Simulation
Samchon Framework
Nam-Tree
Samchon Protocol
Samchon Library
Hansung Timetable
Most Projects

2. Modules - Library

- 크로스 컴파일이 가능한 라이브러리
  - WeakString
  - SQL 드라이버 및 XML 파서
  - Event 및 Critical section 모듈 등
- 그 외 유틸리티 클래스
  - Math, Case Generator
  - 유전자 알고리즘 등

2. Modules - Protocol

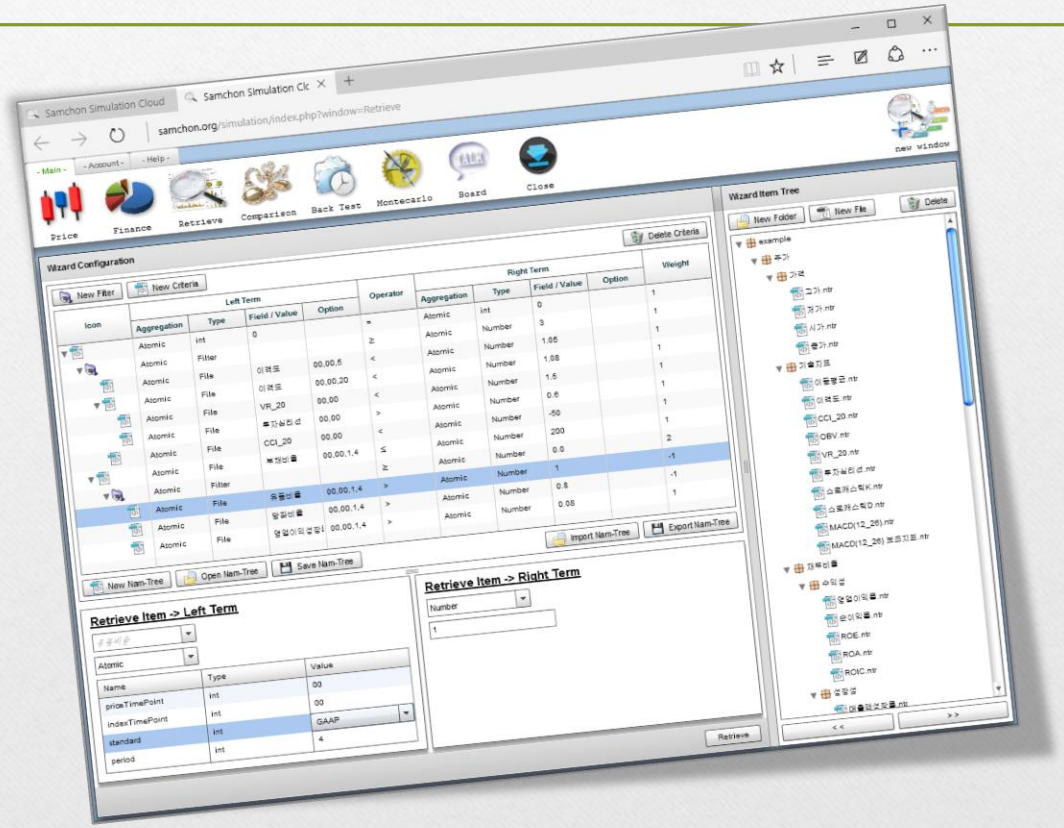


2. Modules - Protocol

★ 본 대회의 주제와 밀접한 연관

- 네트워크 시스템 구성에 관한 전반
  - 클라우드 시스템 구축
  - 분산처리 시스템 구축
  - 여타 네트워크 시스템 설계 및 구현
- 이종 언어간의 통합을 지원
  - C++ with TS or Flex

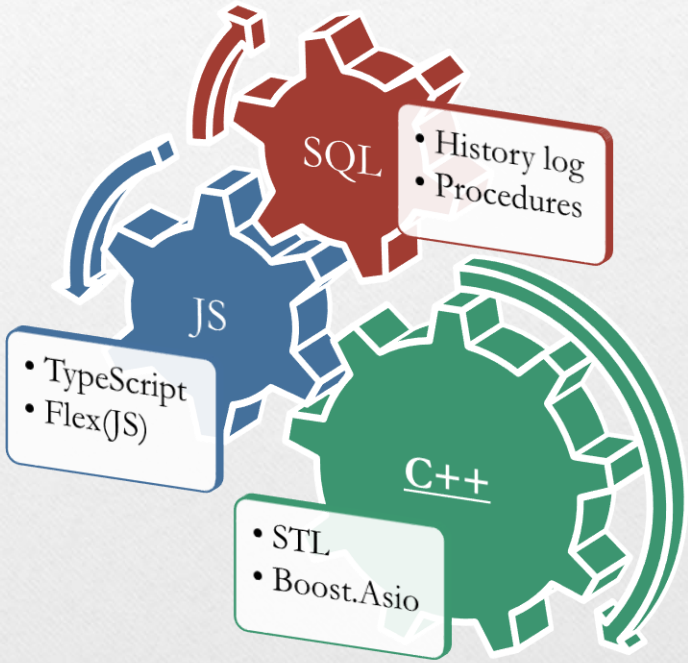
2. Modules - NamTree



3. Modules - NamTree

- 복합 논리조건을 표현할 수 있는 모형
  - 1: N 재귀관계, 트리 구조
- ANN(인공신경망)도 표현할 수 있다.
  - 최적화를 통하여
  - 역으로 최적의 조건을 생성할 수도 있다.

3. Languages



- C++
  - STL (C++11/14)
- JavaScript
  - TypeScript
  - Flex(JS)
- SQL
  - T-SQL
  - MySQL

3. Languages; C++

- 삼촌 프레임워크의 주력 언어
  - 클라우드 서버 구현
  - 분산처리시스템 구성
- STL; C++11/14
  - 이른바 Modern C++
  - 이종 운영체제간 크로스 컴파일 가능
- Boost.Asio
  - 네트워크 소켓 라이브러리
  - 이종 운영체제 간 크로스 컴파일이 가능하여 채택

3. Languages; TypeScript

- JavaScript로 변환이 가능한 객체지향언어
  - C++과 설계 및 개념을 동일하게 가져가기 위하여 STL을 일부 차용하여 구현
  - XML 및 Invoke 파서 등 제공
- 클라이언트 역할
  - C++ 서버 접속 모듈
  - 클라우드 시스템 중 UI (홈페이지) 부문
  - 분산처리시스템 중 Slave 부문

3. Languages; Flex(JS)

- Flex와 FlexJS는?
  - Flex는 Flash player에서 구동
  - FlexJS는 JS로 컴파일 가능
- 클라이언트 역할 수행
  - C++ 서버 접속 모듈
  - 클라우드 시스템 중 UI (홈페이지) 부문
- **Nam-Tree 모듈 제공**

Specification

1. Message Protocol
2. Like OOD
3. Utilization

1. Message Protocol

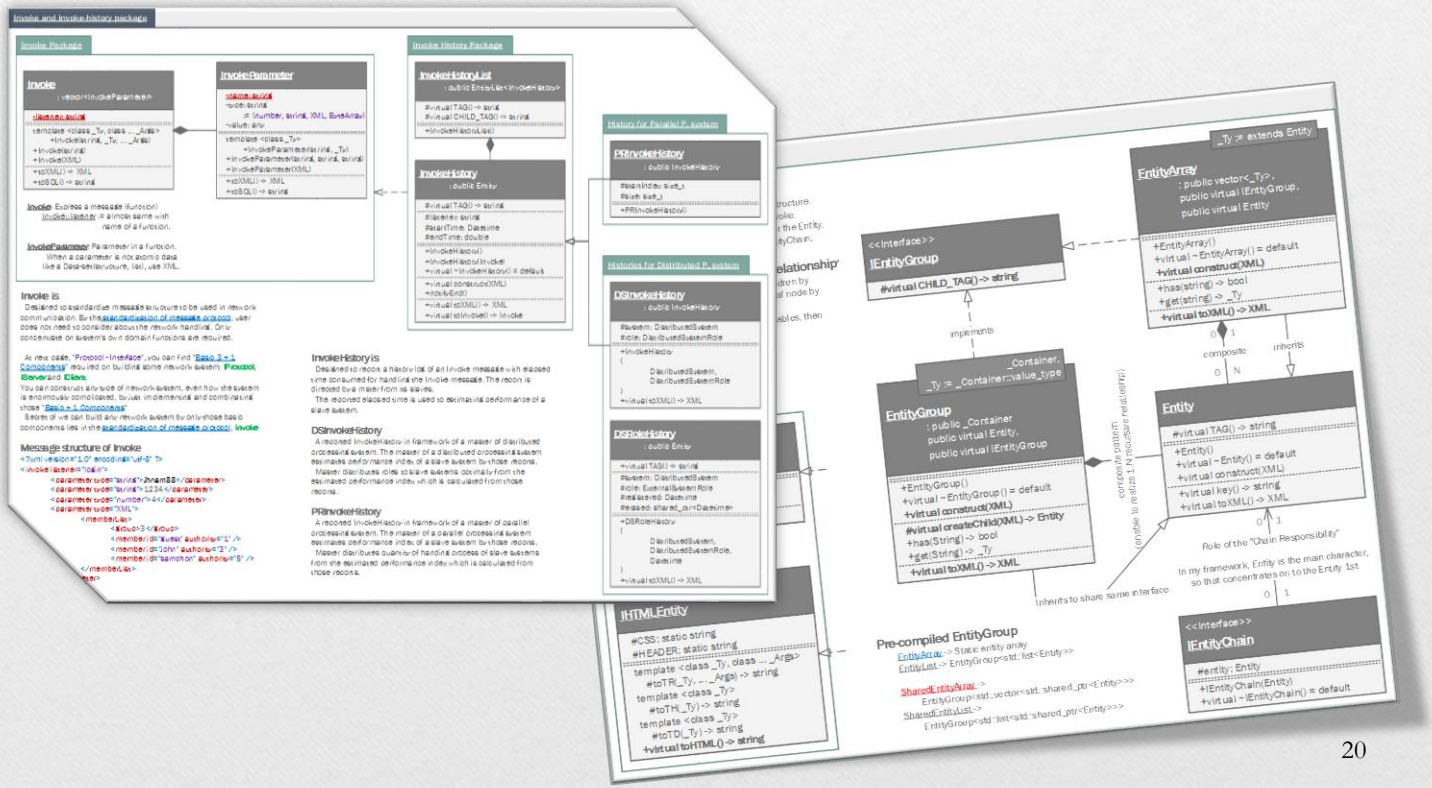
```
<?xml version="1.0" encoding="utf-8" ?>
<invoke listener="login">
        <parameter type="string">jhnam88</parameter>
        <parameter type="string">1234</parameter>
        <parameter type="number">4</parameter>
        <parameter type="XML">
                <memberList>
                    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

1. Message Protocol

```
<?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="“ByteArray">10240</parameter>
</invoke>

01000111001010101010100100011100101010101
01000111001010101010110110011100101010101
01000111001010101010110100011100101010101
…
…
```

1. Message Protocol

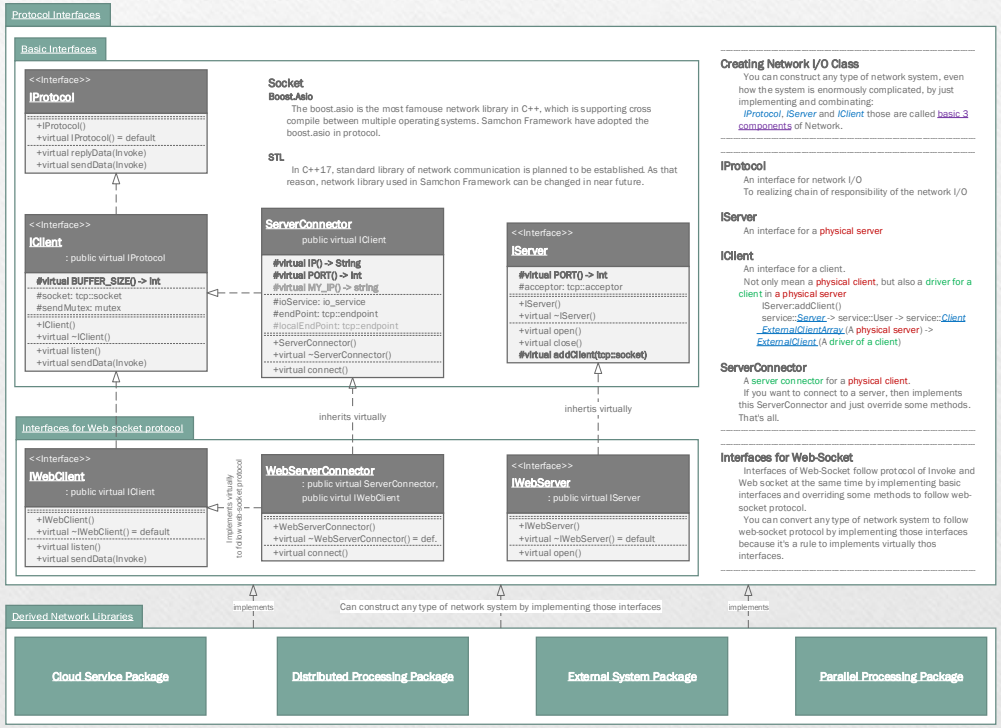


1. Message Protocol

- 표준화된 메시지 프로토콜 -> Invoke
- 표준화된 데이터 표기법 (Entity 모듈)을 통해

- 네트워크 시스템을 S/W, 객체지향적 관점에서 수월히 구현하고 표현할 수 있다. (계속…)

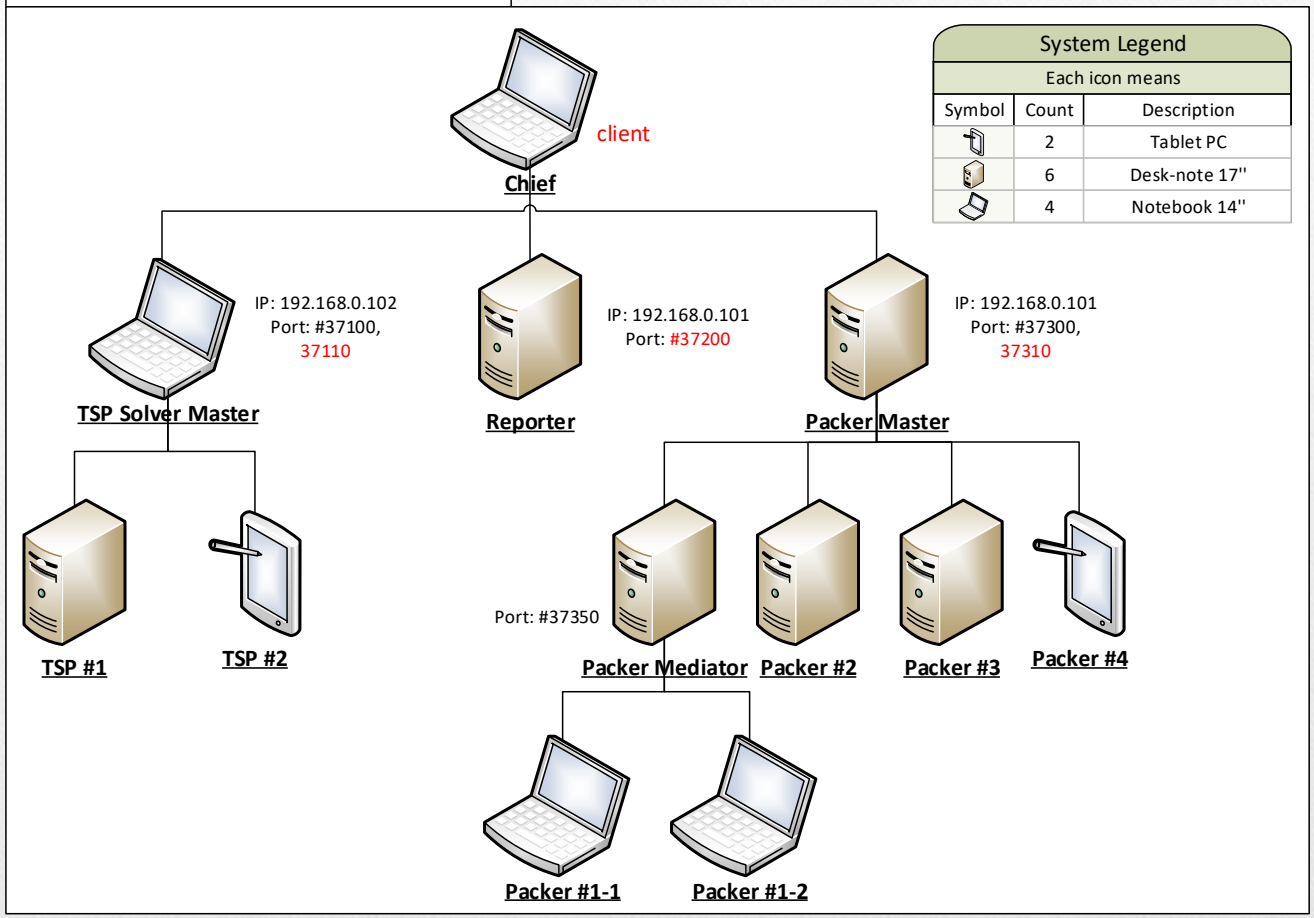
2. Like OOD


Protocol Interfaces
Basic Interfaces
<<Interface>>
IProtocol
+IProtocol()
+virtual IProtocol() = default
+virtual replyData(Invoke)
+virtual sendData(Invoke)
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.
<<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)
inherits virtually
inherits 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()
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.
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

2. Like OOD

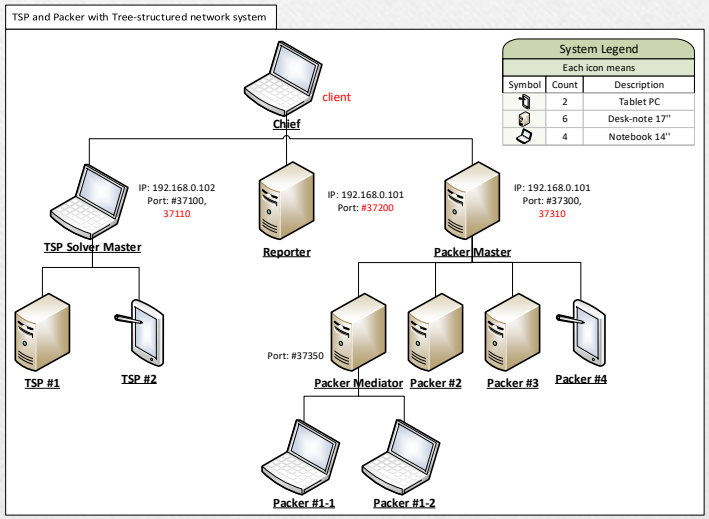
- 네트워크 시스템 구성에 쓰이는
  - Abstarct class (driver)
  - Interface
- 어떠한 복잡한 네트워크 시스템도
  - 이들 Basic 3 + 1 Components의
  - 상속과 조합을 통하여 만들어진다
- 다소 비약을 하자면
  - 이들 모두 또한 활용 예제에 불과
    - 클라우드 서버 모듈
    - 분산처리 시스템 모듈

TSP and Packer with Tree-structured network system



2. Like OOD

분산/병렬처리시스템

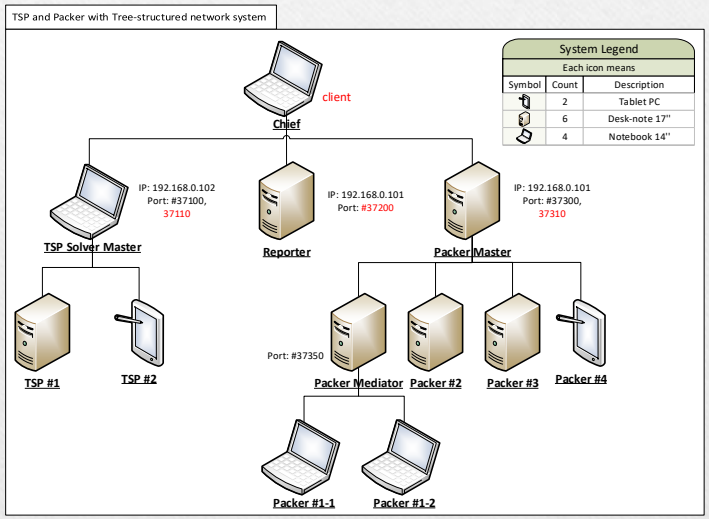


각 시스템 역할 정의

- Chief
  - 명령 전달
- Reporter
  - 연산 결과를 화면에 출력
- Master
  - 명령을 하부 시스템에 전달
  - 연산결과를 집계하여 보고

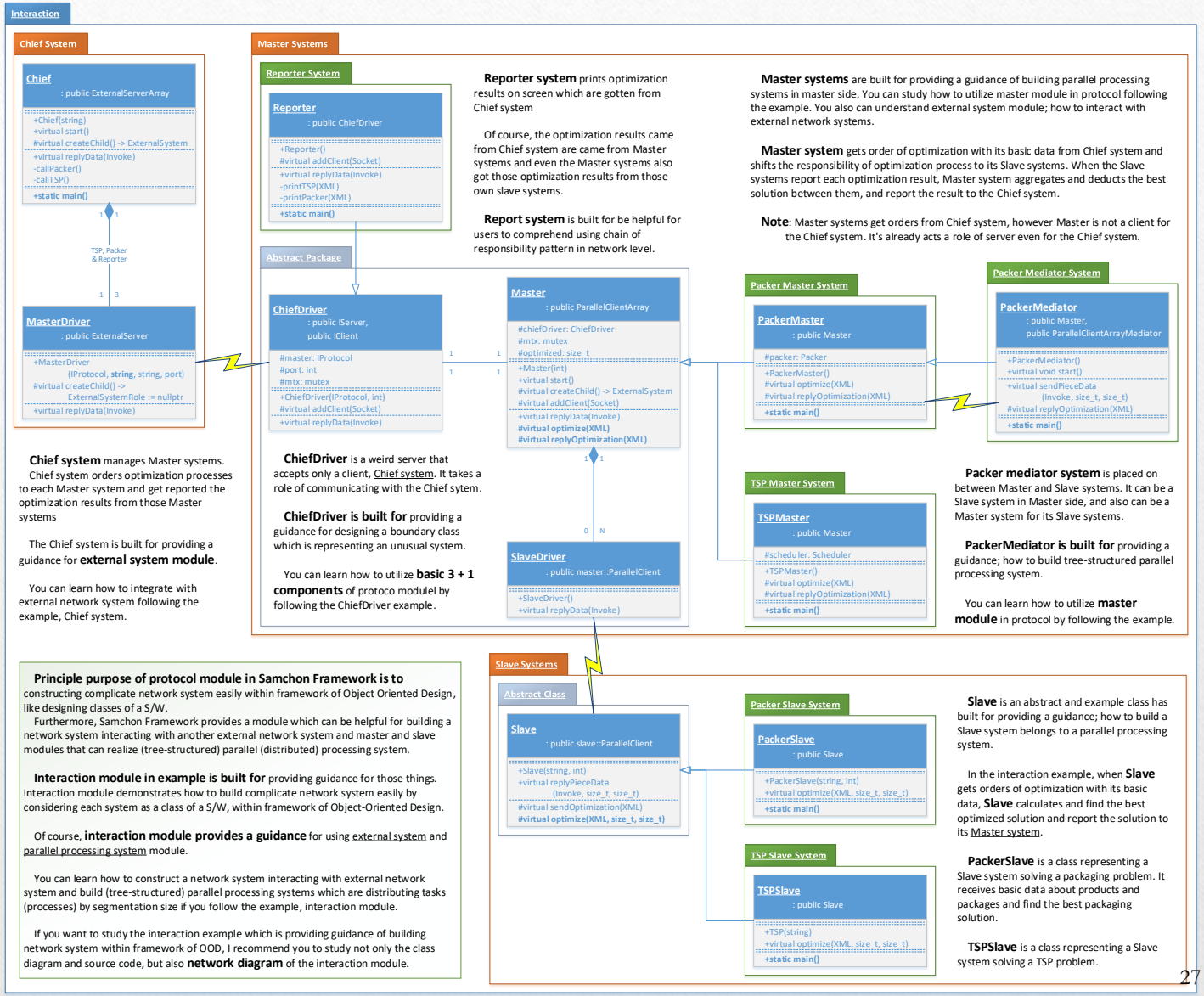
2. Like OOD

분산/병렬처리시스템



각 시스템 역할 정의

- Mediator
  - Master와 Slave 간 중개
- Slave
  - 명령을 받아 연산 수행함
  - 연산 후 상부 Master 또는 Mediator에 보고



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.

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.

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 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.

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.

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.

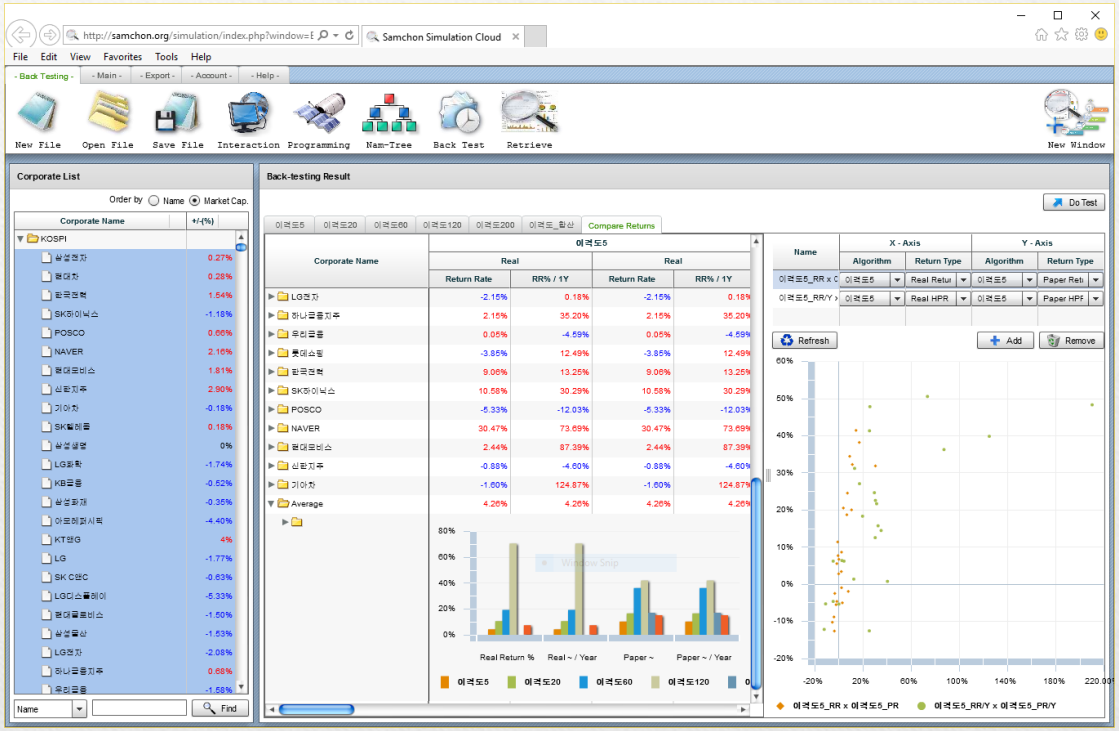
PackerSlave is a class representing a Slave system solving a packaging problem. It receives basic data about products and packages and find the best packaging solution.

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

2. Like OOD

- 표준화와 Basic 3 + 1 Components
- 그 어떤 복잡한 네트워크 시스템도
  - 네트워크 시스템 간 논리적인 연결관계를 정의하고
  - 마치 SW 클래스들을 설계하듯이 네트워크 시스템을 쉽게 구성함
- 클라우드와 분산처리 모듈도
  - 이렇게 만들어졌다.

3. Utilization - Samchon Simulation



3. Utilization - Samchon Simulation

- 주식 시뮬레이션 및 최적 거래 알고리즘 도출
  - 시세 조회
    - 주가 조회
    - 재무정보 조회
  - 종목 검색
  - 시뮬레이션
    - 백 테스팅 – 최적의 거래 알고리즘 도출
    - 몬테카를로 시뮬레이션 – 미래 주가 흐름 추이 예측

3. Utilization - Samchon Simulation

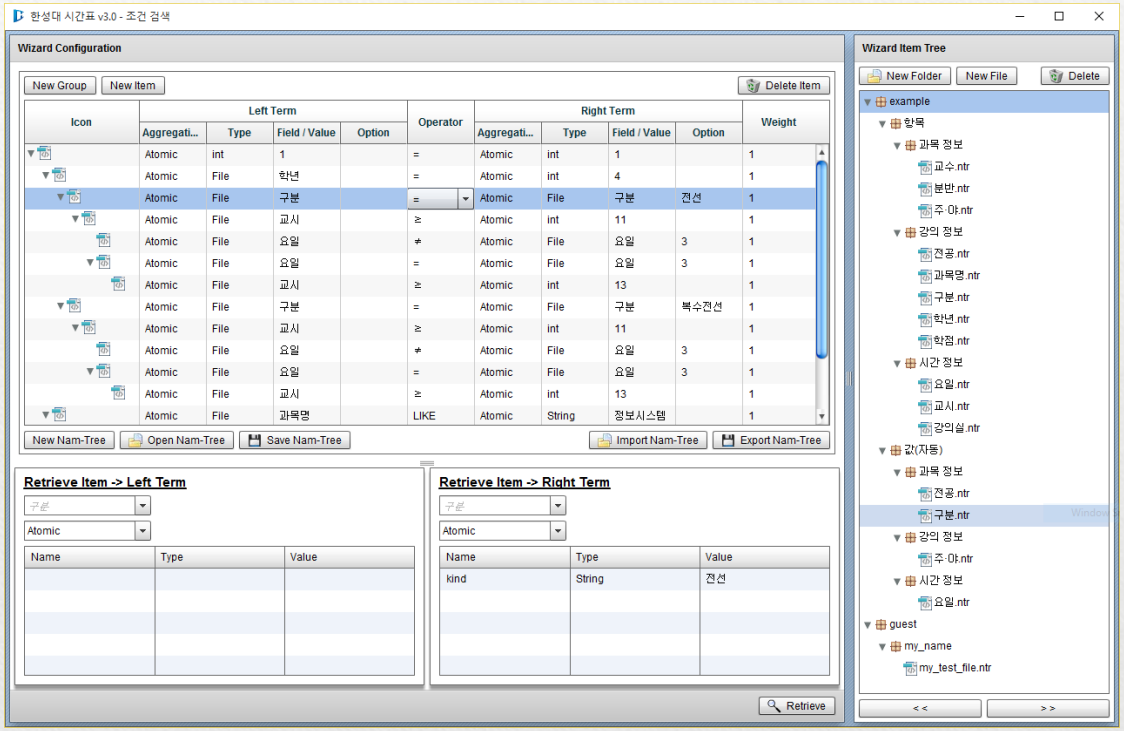
- 활용 예제라기보단
  - 삼촌 프레임워크의 모태가 되는 프로젝트
- 많은 메모리 사용과 빠른 성능이 요구됨
  - 클라우드 시스템 with C++
  - 전반적인 모듈(의 모태)이 사용됨

Samchon Framework

A framework for realizing cloud and distributed processing system

Jeongho Nam

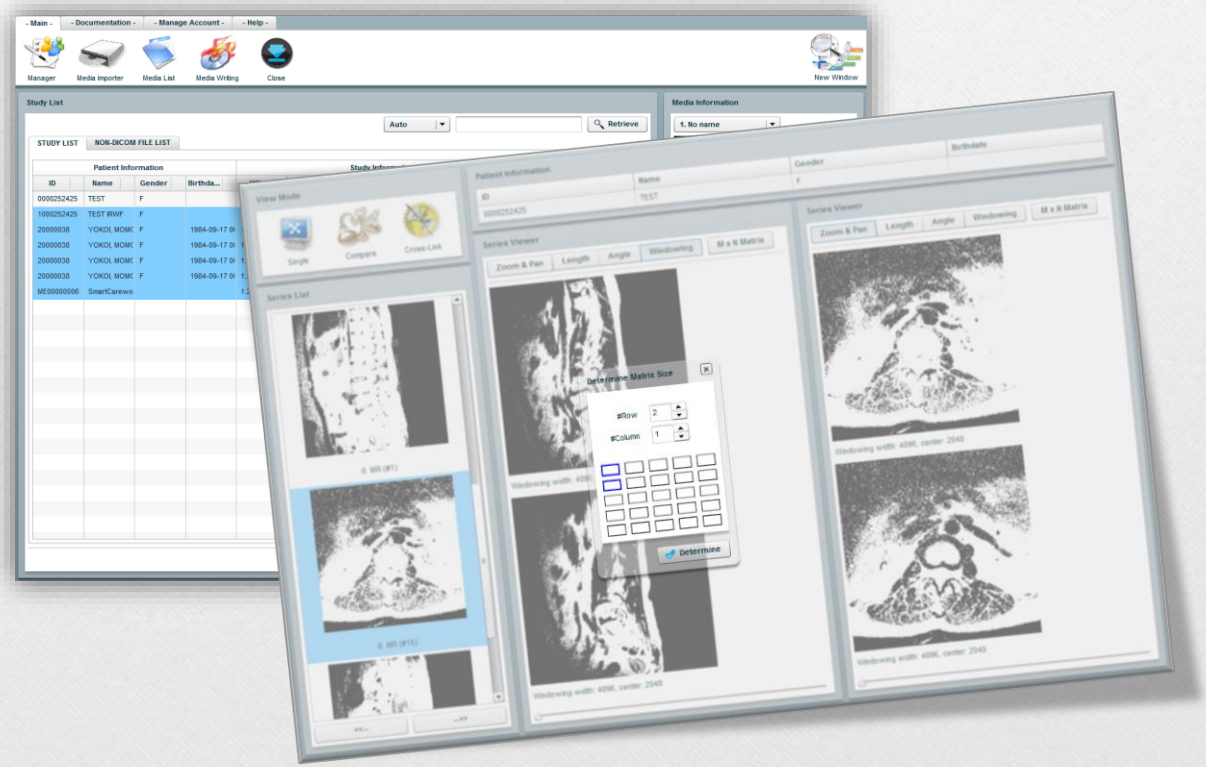
3. Utilization - Timetable



3. Utilization - Timetable

- (클라우드와는 관련 없음)
- 솔루션급 예제
  - 남트리 모듈의 활용 방안을 보이기 위함
  - 클라우드 프로젝트는 아니되, Library, Protocol 및 N am-Tree 모듈이 고루고루 사용됨

3. Utilization - OraQ



3. Utilization - OraQ

- 병원 검사기록 (PACS Media) 관리
  - PACS 미디어 입출납 관리
  - PACS Media 뷰어
  - Data Warehouse 및 백업 역할 겸함
- 위의 역할을 수행하는 서버를
  - 클라우드 서버로 제작해야 한다

3. Utilization - OraQ

- C++/MFC Dependency가 있었음
  - PCAS Media 및 영상처리 API가 DLL
  - PACS, MWL 서버 라이브러리도 DLL
  - 데이터 사용량이 많아 자체 메모리 관리도 필요함
- 하지만, 클라우드 서버로 제작해야
  - 때문에 **Samchon Framework**를 사용하게 됨

Provides

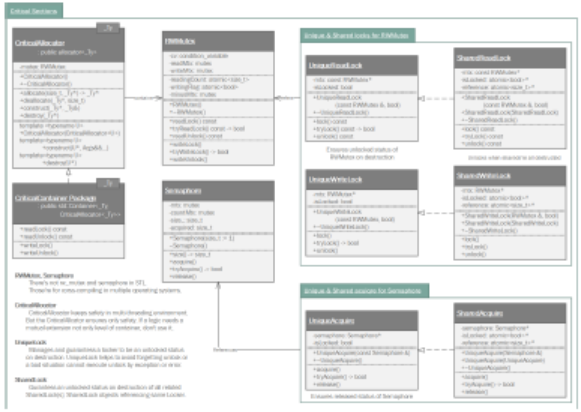
1. Documents
2. Examples
3. GitHub

1. Documents

- 삼촌 프레임워크의 이용자 및 오픈소스 참여자들을 위하여 약 1,600 여 페이지의 문서를 제공
- 본인과 같은 사례(framework)의 오픈소스 프로젝트가 가장 지켜야 할 기본기가 아닐까

1. Documents

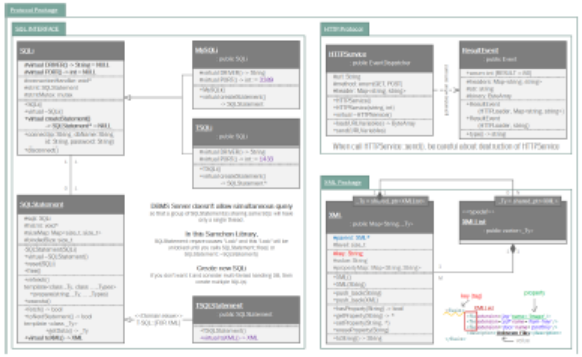
- 삼촌 프레임워크의 성패는 ~에 달림
  - 얼마나 양질의 문서를 제공하냐
  - 그리고 이 문서가 얼마나 이용자들에게 효용이 있냐
- 미시적 거시적 관점에서
  - 상세하고 치밀한 설계와 문서를 제공하고
  - 쉽게 따라하고 이해할 수 있는 예제를 제공해야 함



- Data I/O libraries.

  Provides libraries about data I/O.

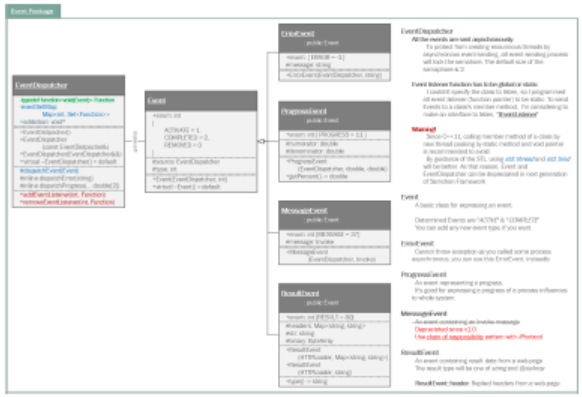
  SQLi and SQLStatement are the ODBC drivier designed to follow principles of OOP by adapter pattern. XML class is designed to follow composite relationship.



- Event libraries.

  Libraries representing events and dispatching those events, which are running on background, with own exclusive thread.

  But there's something to notice. 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.

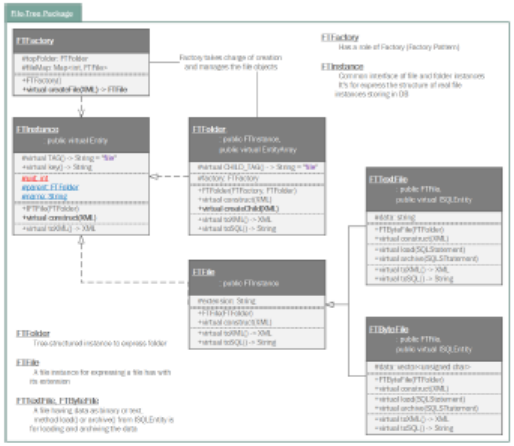


- File-tree libraries.

  A module for expressing folder and file instances. The objects in file-tree module are realized by protocol::Entity.

  They have recursive and hierarchical relationship and created by Factory class (FTFactory, factory pattern). Those objects can be archived in and loaded from Database. Using the pre-defined methods interaction with Database, you can not only express real file and folder instances but also realize virtual file system.

  Classes in the Module of file-tree are all abstract, but package nam-tree can be an example inheriting and having real model from those abstract classes.



Author

Jeongho Nam

5.5.2 Typedef Documentation

5.5.2.1 using CriticalDictionary = CriticalMap<std::string, _Ty, _Pr>

A Dictionary ensures concurrency.

See also

library::CriticalAllocator
samchon::library

Author

Jeongho Nam

5.5.2.2 using CriticalList = std::list<_Ty, CriticalAllocator<_Ty>>

A std::list ensures concurrency.

See also

library::CriticalAllocator
samchon::library

Author

Jeongho Nam

5.5.2.3 using CriticalMap = Map<_Kty, _Ty, _Pr, CriticalAllocator<std::pair<const std::string, _Ty>>>

A Map ensures concurrency.

See also

library::CriticalAllocator
samchon::library

Author

Jeongho Nam

5.5.2.4 using CriticalSet = Set<_Ty, _Pr, CriticalAllocator<_Ty>>

A Set ensures concurrency.

See also

library::CriticalAllocator
samchon::library

Author

Jeongho Nam

5.5.2.5 using CriticalVector = std::vector<_Ty, CriticalAllocator<_Ty>>

A std::vector ensures concurrency.

See also

library::CriticalAllocator
samchon::library

Author

Jeongho Nam

5.5.2.6 typedef std::vector<std::shared_ptr<XML>> XMLList

A list of XML, tags are same.

See also

samchon::library

Author

Jeongho Nam

5.6 samchon::namtree Namespace Reference

Package of Nam-Tree.

Classes

- class INTExplore

  An interface for exploration.

- class NTCriteria

  Criteria, a conditional expression with weight.

- class NTEntityGroup

  A historical, studying data.

- class NTFactory

  A factory for Nam-Tree objects.

- class NTFile

  A file archiving metadata of a function.

- class NTIterator

  Iterator of historical data.

- class NTParameter

  A metadata of a parameter in a function.

- class NTParameterArray

  An Array of NTParamotor.

- class NTParameterDetermined

  A pre-determined value of a parameter.

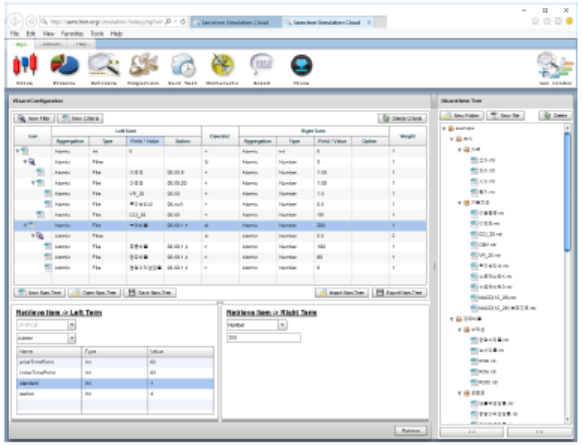
- class NTSide

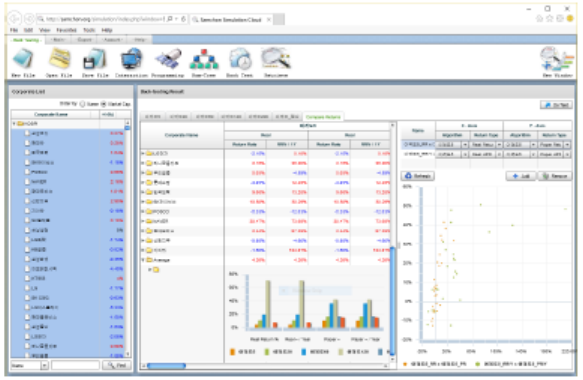
  A side of a conditional expression.

5.6.1 Detailed Description

Package of Nam-Tree.

Nam-Tree is a module of tree-structured complicate logical condition model for realizing A.N.N. (Artificial Neural Network). You can make the ANN model with weight and bias and express not only logical condition by your hand, but also construct the logical condition by automaticaly with optimization of genetic algorithm and grid method.





The basic pricinple of Nam-Tree are follow:

1. A conditional expression with weight

   F(x) = (NTSide < NTSide ? 1 : 0) X weight

   NTCriteria is made up for conditional expression.

   - When the expression is true, returns 1 + weight
   - When the expression is false, returns 0

2. Hierarchical relationship

   - In vertical relationship: multiply (X)
   - In horizontal relationship: plus (+)

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

3. Making bias

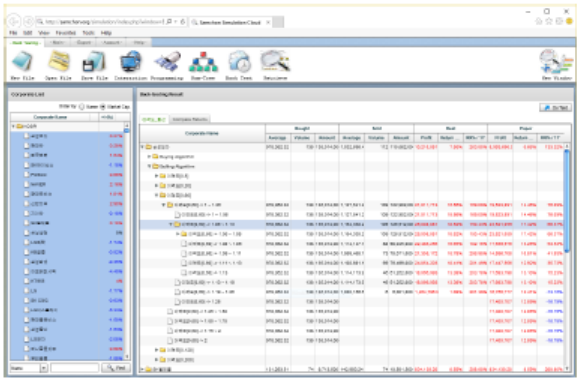
   Just make a NTCriteria returns only true.

   It's the bias returns only weight.

4. Explore

   (a) Exporing 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 minimum to maximum in INTExplore reach to the precision, by the method of multi-demensional grid.

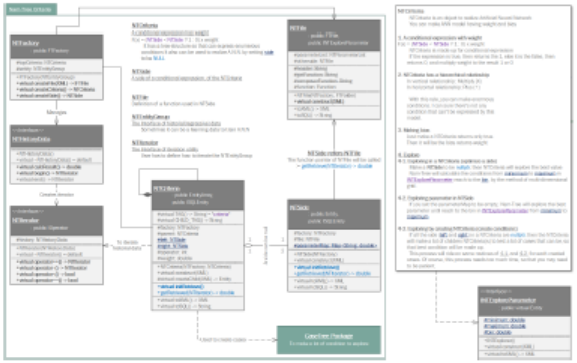
   (b) Exploring parameter in NTSide
   If you set the parameterMNap to be empty, Nam-Tree will explore the best parameter until reach to the precision in INTExploreParameter from minimum to maximum.

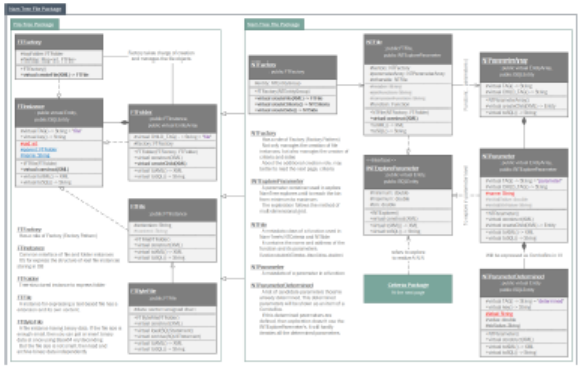


   (c) Exploring by creating NTCriteria
   (Creates lots of hierarchical conditions and tests them)

If all the side (left and right) in a NTCriteria are nullptr, the NTCriteria will make a lot of children NTCriteria(s) to test lots of cases, so that derives the best condition set 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.





Note

Previous version of the Samchon Framework, NTFile class had a function script on the NTFile and Nam-Tree module compiled the script like Nam-Tree moudle of Flex standalone that is keeping the compiling method.

However, modern Samchon Framework's C++ NTFile class doesn't have a function script and does not compile. I don't know a way to compile a script in lots of operating systems, so I dropped the compiling method. Until update, NTFactory has function pointers in a Dictionary and NTFile reference them. Users must put own methods into the Dictionary by their hands.

I will solve the problem sooon. Until next generation of Samchon Framework, put your own function pointers to a Dictionary in NTFactory by your hand please. Sorry for my unripe skill on programming.

Author

Jeongho Nam

5.7 samchon::protocol Namespace Reference

Package of network protocol and libraries.

Namespaces

- master
  Package for external system, within the framework of master.
- service
  Package of cloud service as a server.
- slave
  Package of external system, within the framework of slave.

Classes

- class Entity
  An entity, a standard data class.
- class EntityArray
  An Entity and a static array containing Entity objects.
- class EntityGroup
  An Entity and a container of children Entity objects.
- class EntityList
  An Entity and a static list containing Entity objects.
- class ExternalClient
  A network driver for an external client.
- class ExternalClientArray
  An array of ExternalClient(s)
- class ExternalServer
  A network driver for an external server.
- class ExternalServerArray
  An array of ExternalServer(s)
- class ExternalSystem
  A network driver for an external system.
- class ExternalSystemArray

An array of ExternalSystem(s).

- class ExternalSystemRole
  A role belongs to an external system.
- class FlashPolicyServer
  A flash policy server.
- class IClient
  An interface for a client.
- class IEntityChain
  A chain of entity.
- class IEntityGroup
  An interface for entity group.
- class IEntityPtrGroup
- class IHTMLEntity
  An interface supporting conversion to html.
- class Invoke
  Standard message of network I/O.
- class InvokeParameter
  A parameter of an Invoke.
- class IProtocol
  An interface of Invoke message chain.
- class IServer
  An interface of a physical server.
- class ISQLEntity
  An interface supporting DB-I/O.
- class IWebClient
  An interface for a web-client.
- class IWebServer
  An interface for a physical server following web-socket.
- class ServerConnector
  A server connector for a physical client.
- class SystemRole
  A role belongs to a system.
- class WebServerConnector
  A web-socket server connector.

Typedefs

- template<typename _Ty = Entity>
  using SharedEntityArray = EntityGroup< std::vector< std::shared_ptr< _Ty >>, _Ty, std::shared_ptr< _Ty > >
  An EntityGroup with vector container and children capsuled in shared pointers.
- template<typename _Ty = Entity>
  using SharedEntityList = EntityGroup< std::list< std::shared_ptr< _Ty >>, _Ty, std::shared_ptr< _Ty > >
  An EntityGroup with list container and children capsuled in shared pointers.
- template<typename _Ty = Entity>
  using UniqueEntityArray = EntityGroup< std::vector< std::unique_ptr< _Ty >>, _Ty, std::unique_ptr< _Ty > >
  An EntityGroup with vector container and children capsuled in unique pointers.
- template<typename _Ty = Entity>
  using UniqueEntityList = EntityGroup< std::list< std::unique_ptr< _Ty >>, _Ty, std::unique_ptr< _Ty > >
  An EntityGroup with list container and children capsuled in unique pointers.

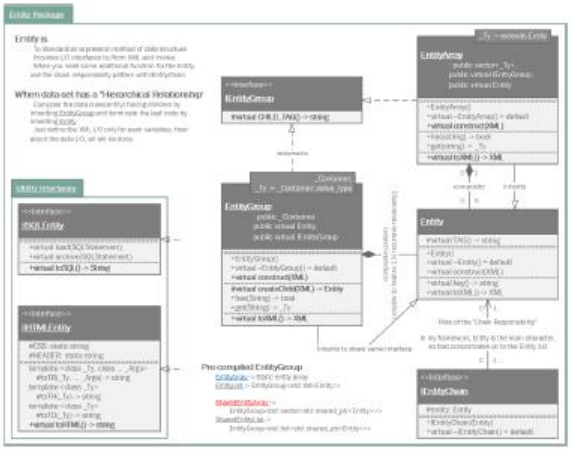
5.7.1 Detailed Description

Package of network protocol and libraries.

- **Entity** - data classes with utility methods

  In Samchon Framework, with entity, boundary and control, entity is the main. The entity package provides I/O interface to/from XML, Invoke and DBMS. If you need some additional function (role of boundary or control) for an entity, use chain of responsibility pattern starting from the entity.

  When data-set has a "hierarchical relationship", do not have the children objects by having a container member variable. Directly inherits container set. Composes the data class (entity) having children by inheriting EntityGroup and terminate the leaf node by inheriting Entity (single object).



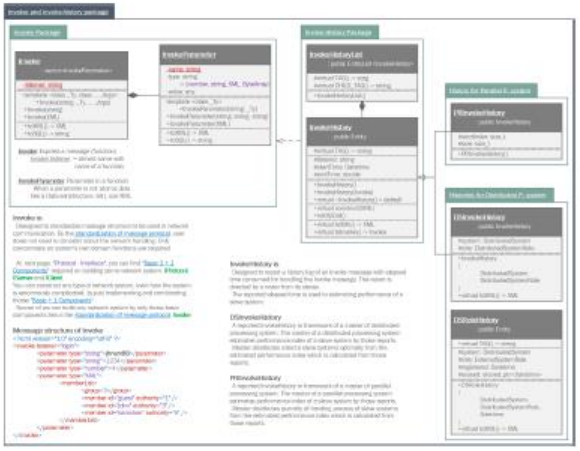
- **Invoke** - standard network I/O message

  Invoke is a standard message structure using network I/O in Samchon Framework.

  The Invoke message has an XML structure like the placture in below. We can enjoy lots of benefits by the normalized and standardized message structure used in network I/O.

  The greatest advantage is that we can make any type of network system, even how the system is enourmously complicated. As network communication message is standardized, we only need to concentrate on logical relationships between network systems. We can handle each network system like a object (class) in OOD. And those relationships can be easily designed by using design pattern.

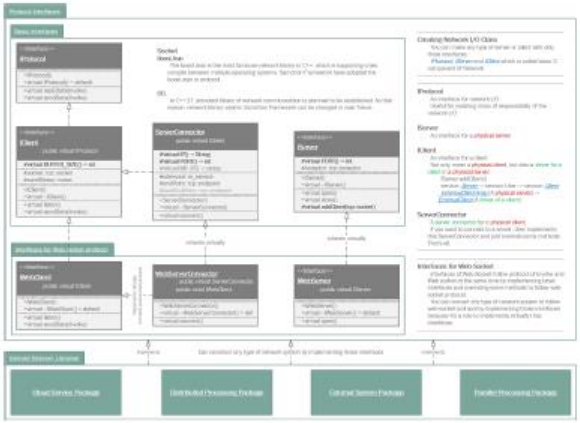
  Furthermore, The module invoke not only provides parsing and converting from/to Invoke message, but also history classes can be used to archiving log or estimating performance of a system. The purpose of estimating performance a system, especially, used in module of distributed processing and parallel processing systems.

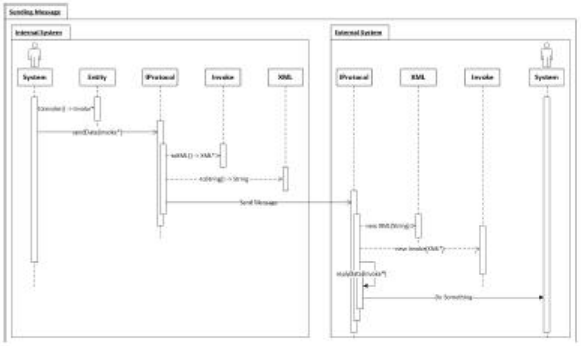


- Intefaces

  You can make any type of network system with only three + one interfaces; IProtocol, IServer and IClient + ServerConnector which are called basic 3 + 1 components. As I've mentiond in previous invoke module, because of standardization of message of network I/O, we only need to concentrate on logical relationship like handling OOD objects

  The basic 3 + 1 components are top level abstract interfaces for designing network system like handling OOD objects. When you see the protocol module from a different view, out of framework, you can find that all the other classes in protocol are another examples combinationing and utilizing those basic 3 + 1 components (interaces).





- External System

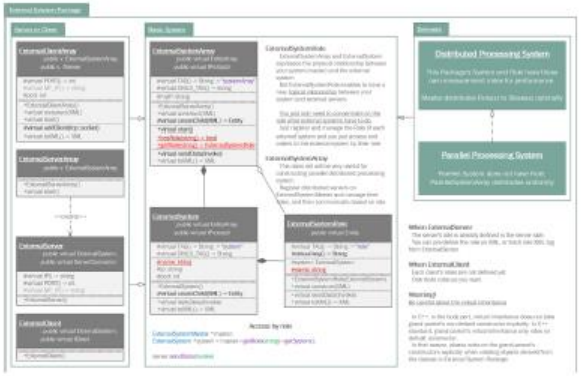
  Module *external_system* provides interfaces for interaction with external network system. Although, the module *external_system* acts boundary as main role, what you've to concentrate on is the entity. Samchon Framework takes responsibility of network communication and you only consider about relationship and role of each *external network systems*.

ExternalSystem objects are managed by ExternalSystemArray and the ExternalSystemArray can access to an ExternalSystemRole belongs to an ExternalSystem directly. When you send an Invoke message to ExternalSystemArray, the ExternalSystemArray finds matched ExternalSystemRole and the External-SystemRole shifts the network I/O responsibility to belonged ExternalSystem.

The relationship called as "Proxy Pattern". With the pattern, "Proxy", you can concentrate on roles irrespective of where each role is belonged to (you can only concentrate on ExternalSystemRole itself, what to do with Invoke message, irrespective of the ExternalSystemRole is belonged to which ExternalSystem).

- ExternalSystemArray::sendData() -> ExternalSystemRole(Proxy)::sendData() -> ExternalSystem-::sendData()
- ExternalSystem::replyData() -> ExternalSystemRole(Proxy)::replyData()

Whether using the "Proxy pattern" is on your mind in *external_system module* level. "Proxy pattern" is recommend to use in *external_system* module, but not forced. However, since *parallel_processing_system* module, you've to follow the pattern.



- Packages in protocol

6.24.2.6 virtual auto toXML ( ) const -> std::shared_ptr<library::XML> [override], [virtual]

Get an XML object represents the EntityGroup.

Archives the EntityGroup's own member variables only to the returned XML object.

Do not consider about archiving children Entity objects' data in EntityGroup::toXML(). Those children Entity objects will converted to XML object by their own toXML() method. The insertion of XML objects representing children are done by abstract method of EntityGroup::toXML().

Archives only data of EntityGroup's own.

Inherited

Get an XML object represents the Entity.

Returns an XML object that can represents the Entity containing member variables into properties.

A member variable (not object, but atomic value like number, string or date) is categorized as a property within the framework of entity side. Thus, when overriding a toXML() method and archiving member variables to an XML object to return, puts each variable to be a property belongs to only an XML object.

Don't archive the member variable of atomic value to XML::value causing enormouse creation of XML objects to number of member variables. An Entity must be represented by only an XML instance (tag).

Standard Usage	Non-standard usage abusing value
<memberList> <member id='jhnam88' name='Jeongho+Nam' birthdate='1988-03-11' /> <member id='master' name='Administartor' birthdate='2011-07-28' /> </memberList>	<member> <id>jhnam88</id> <name>Jeongho+Nam</name> <birthdate>1988-03-11</birthdate> </member>

Returns

An XML object representing the Entity.

Reimplemented from EntityGroup< _Container, _ETy, _Ty >.

Reimplemented in DistributedClientArray.

6.24.3 Member Data Documentation

6.24.3.1 struct library::GAParameters gaParameters [protected]

A structure of parameters for genetic algorithm.

.

The documentation for this class was generated from the following file:

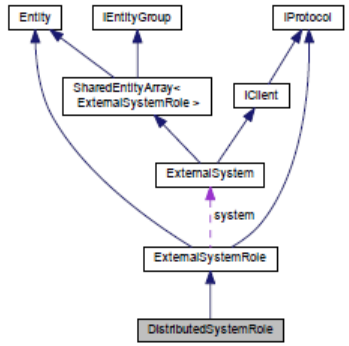
- D:/OneDrive/Project/Samchon/framework/cpp/samchon/protocol/master/DistributedSystemArray.hpp

6.25 DistributedSystemRole Class Reference

A role of distributed processing system.

`#include <DistributedSystemRole.hpp>`

Collaboration diagram for DistributedSystemRole:



Public Member Functions

- DistributedSystemRole ()
  Default Constructor.
- auto getSystem () const -> ExternalSystem *--delete
  Get an external system -> deprecated.
- auto getPerformance () const -> double
  Get performance.
- auto getAllocationHistoryList () const -> DSRoleHistoryList *
  Get allocation histories.
- auto getInvokeHistoryList () const -> DSInvokeHistoryList *
  Get invoke histories.
- virtual void sendData (std::shared_ptr< Invoke >) override
  Send a message.
- virtual auto toXML () const -> std::shared_ptr< library::XML > override
  Get an XML object represents the Entity.

Protected Attributes

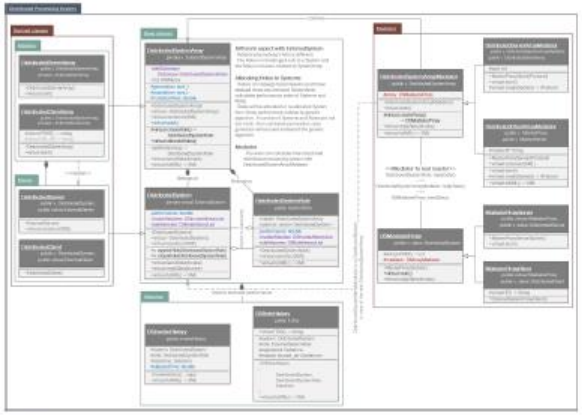
- std::set< DistributedSystem * > allocatedSystems
  Allocated systems of about the role, at now.
- double performance
  A required performance index.

- DSRoleHistoryList * allocationHistoryList
  A history log for archiving role allocation to a system.
- DSInvokeHistoryList * invokeHistoryList
  A history log for archiving elapsed time for each process.

6.25.1 Detailed Description

A role of distributed processing system.

DistributedSystemRole is an ExternalSystemRole allocates for distributed processing system(s). Unlike the ExternalSystemRole, DistributedSystemRole can belonged to multiple DistributedSystem(s).

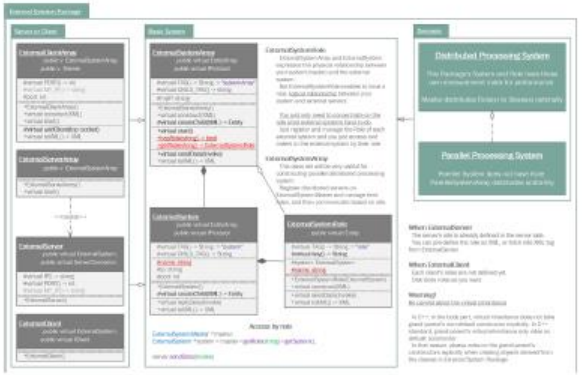


Inherited

ExternalSystemRole is a 'control' class groupping methods, handling Invoke messages interacting with an external system that the ExternalSystemRole is belonged to, by a subject or unit of a module.

ExternalSystemRole can be a "logical proxy" for an ExternalSystem which is containing the ExternalSystemRole. Of course, the ExternalSystemRole is belonged to an ExternalSystem. However, if you access an ExternalSystemRole from an ExternalSystemArray directly, not passing by a belonged ExternalSystem, and send an Invoke message even you're not knowing which ExternalSystem is related in, the ExternalSystemRole acted a role of proxy.

It's called as "Proxy pattern". With the pattern, you can only concentrate on ExternalSystemRole itself, what to do with Invoke message, irrespective of the ExternalSystemRole is belonged to which ExternalSystem.



See also

samchon::protocol

Author

Jeongho Nam

6.25.2 Member Function Documentation

6.25.2.1 auto getSystem ( ) const -> ExternalSysem * = delete [delete]

Get an external system -> depreciated.

Unlike ExternalSysemRole, DistributedSystemRole can be belonged to multiple DistributedSystem(s). By that reason, the getSystem(), a getter of a single ExternalSystem is depreciated.

6.25.2.2 virtual void sendData (std::shared_ptr< Invoke >) [override], [virtual]

Send a message.

Sends Invoke message to a network system or shifts the responsibility to related chain.

Parameters

invoke	An Invoke message to send to a network system

Reimplemented from ExternalSystemRole.

6.25.2.3 virtual auto toXML ( ) const -> std::shared_ptr<library::XML> [override], [virtual]

Get an XML object represents the Entity.

Returns an XML object that can represents the Entity containing member variables into properties.

A member variable (not object, but atomic value like number, string or date) is categorized as a property within the framework of entity side. Thus, when overriding a toXML() method and archiving member variables to an XML object to return, puts each variable to be a property belongs to only an XML object.

Don't archive the member variable of atomic value to XML::value causing enormouse creation of XML objects to number of member variables. An Entity must be represented by only an XML instance (tag).

Standard Usage	Non-standard usage abusing value
<memberList> <member id='jhnam88' name='Jeongho+Nam' birthdate='1988-03-11' /> <member id='master' name='Administartor' birthdate='2011-07-28' /> </memberList>	<member> <id>jhnam88</id> <name>Jeongho+Nam</name> <birthdate>1988-03-11</birthdate> </member>

Returns

An XML object representing the Entity.

Reimplemented from ExternalSystemRole.

The documentation for this class was generated from the following file:

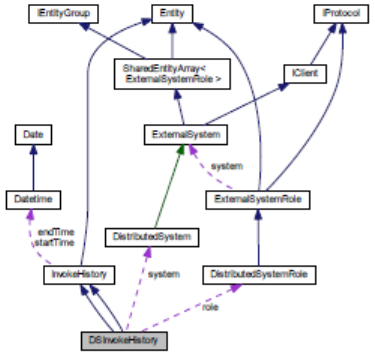
- D:/OneDrive/Project/Samchon/framework/cpp/samchon/protocol/master/DistributedSystemRole.hpp

6.26 DSInvokeHistory Class Reference

A reported history of an Invoke message.

```
#include <DSInvokeHistory.hpp>
```

Collaboration diagram for DSInvokeHistory:



Public Member Functions

- DSInvokeHistory (master::DistributedSystem *, master::DistributedSystemRole *)
  Construct from a system and role.
- virtual auto toXML () const -> std::shared_ptr< library::XML > override
  Get an XML object represents the Entity.
- DSInvokeHistory (master::ParallelSystem *)
  Construct from a system.
- virtual auto toXML () const -> std::shared_ptr< library::XML > override
  Get an XML object represents the Entity.

Protected Attributes

- master::DistributedSystem * system
  Source system.
- master::DistributedSystemRole * role
  Source role.
- master::ParallelSystem * system
  Source system.

6.26.1 Detailed Description

A reported history of an Invoke message.

Inherited

6.26.2 Constructor & Destructor Documentation

6.26.2.1 DSInvokeHistory (master::DistributedSystem *, master::DistributedSystemRole *)

Construct from a system and role.

Parameters

system	a source system
role	a source role

6.26.3 Member Function Documentation

6.26.3.1 virtual auto toXML ( ) const -> std::shared_ptr<library::XML> [override], [virtual]

Get an XML object represents the Entity.

Returns an XML object that can represents the Entity containing member variables into properties.

A member variable (not object, but atomic value like number, string or date) is categorized as a property within the framework of entity side. Thus, when overriding a toXML() method and archiving member variables to an XML object to return, puts each variable to be a property belongs to only an XML object.

Don't archive the member variable of atomic value to XML::value causing enormouse creation of XML objects to number of member variables. An Entity must be represented by only an XML instance (tag).

Standard Usage	Non-standard usage abusing value
<memberList> <member id='jhnam88' name='Jeongho+Nam' birthdate='1988-03-11' /> <member id='master' name='Administartor' birthdate='2011-07-28' /> </memberList>	<member> <id>jhnam88</id> <name>Jeongho+Nam</name> <birthdate>1988-03-11</birthdate> </member>

Returns

An XML object representing the Entity.

Reimplemented from InvokeHistory.

6.26.3.2 virtual auto toXML ( ) const -> std::shared_ptr<library::XML> [override], [virtual]

Get an XML object represents the Entity.

Returns an XML object that can represents the Entity containing member variables into properties.

A member variable (not object, but atomic value like number, string or date) is categorized as a property within the framework of entity side. Thus, when overriding a toXML() method and archiving member variables to an XML object to return, puts each variable to be a property belongs to only an XML object.

Don't archive the member variable of atomic value to XML::value causing enormouse creation of XML objects to number of member variables. An Entity must be represented by only an XML instance (tag).

Standard Usage	Non-standard usage abusing value
<memberList> <member id='jhnam88' name='Jeongho+Nam' birthdate='1988-03-11' /> <member id='master' name='Administartor' birthdate='2011-07-28' /> </memberList>	<member> <id>jhnam88</id> <name>Jeongho+Nam</name> <birthdate>1988-03-11</birthdate> </member>

Returns

An XML object representing the Entity.

Reimplemented from InvokeHistory.

The documentation for this class was generated from the following files:

- D:/OneDrive/Project/Samchon/framework/cpp/samchon/protocol/master/DSInvokeHistory.hpp
- D:/OneDrive/Project/Samchon/framework/cpp/samchon/protocol/master/PRInvokeHistory.hpp

6.27 Entity Class Reference

An entity, a standard data class.

```
#include <Entity.hpp>
```

Public Member Functions

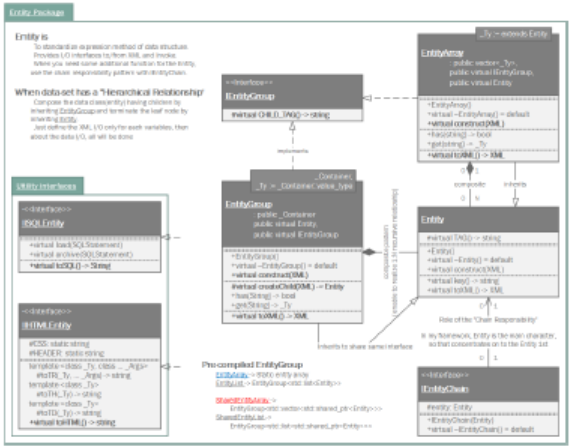
- virtual auto TAG () const -> std::string=0
  A tag name when represented by XML.
- Entity ()
  Default Constructor.
- virtual void construct (std::shared_ptr< library::XML >)=0
  Construct data of the Entity from an XML object.
- virtual auto key () const -> std::string
  Get a key that can identify the Entity uniquely.
- virtual auto toXML () const -> std::shared_ptr< library::XML >
  Get an XML object represents the Entity.

6.27.1 Detailed Description

An entity, a standard data class.

Entity is a class for standardization of expression method using on network I/O by XML. If Invoke is a standard message protocol of Samchon Framework which must be kept, Entity is a recommended semi-protocol of message for expressing a data class. Following the semi-protocol Entity is not imposed but encouraged.

As we could get advantages from standardization of message for network I/O with Invoke, we can get additional advantage from standardizing expression method of data class with Entity. We do not need to know a part of network communication. Thus, with the Entity, we can only concentrate on entity's own logics and relationships between another entities. Entity does not need to how network communications are being done.



Example source

```
1 #include <iostream>
2
3 #include <samchon/protocol/Entity.hpp>
4 #include <samchon/protocol/IHTMLEntity.hpp>
5 #include <samchon/protocol/SharedEntityArray.hpp>
6
7 #include <samchon/library/XML.hpp>
8
9 #ifdef _WIN64
10 #   ifdef _DEBUG
11 #       pragma comment(lib, "x64/Debug/SamchonFramework.lib")
12 #   else
13 #       pragma comment(lib, "x64/Release/SamchonFramework.lib")
14 #   endif
15 #else
16 #   ifdef _DEBUG
17 #       pragma comment(lib, "Debug/SamchonFramework.lib")
18 #   else
19 #       pragma comment(lib, "Release/SamchonFramework.lib")
20 #   endif
21 #endif
```

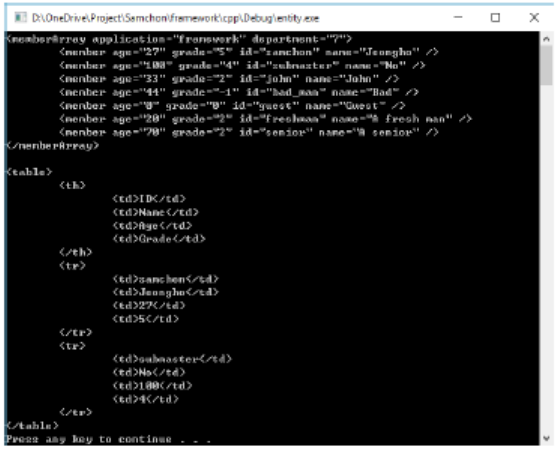
```
22
23 using namespace std;
24 using namespace samchon::library;
25 using namespace samchon::protocol;
26
27 class Member
28     : public Entity, public virtual IHTMLEntity
29 {
30 protected:
31     typedef Entity super;
32
33     string id;
34     string name;
35     int age;
36     int grade;
37
38 public:
39     /* ---------------------------------------------------------
40         CONSTRUCTORS
41     --------------------------------------------------------- */
42     Member()
43         : super(), IHTMLEntity()
44     {
45     }
46     Member(const string &id, const string &name, int age, int grade)
47         : super(), IHTMLEntity()
48     {
49         this->id = id;
50         this->name = name;
51         this->age = age;
52         this->grade = grade;
53     };
54     virtual ~Member() = default;
55
56     virtual void construct(shared_ptr<XML> xml) override
57     {
58         this->id = xml->getProperty("id");
59         this->name = xml->getProperty("name");
60         this->age = xml->getProperty<int>("age");
61         this->grade = xml->getProperty<int>("grade");
62     };
63
64     /* ---------------------------------------------------------
65         GETTERSs
66     --------------------------------------------------------- */
67     virtual auto key() const -> std::string override
68     {
69         return this->id;
70     };
71
72     /* ---------------------------------------------------------
73         XML EXPORTERS
74     --------------------------------------------------------- */
75     virtual auto TAG() const -> string override
76     {
77         return "member";
78     };
79     virtual auto toXML() const -> shared_ptr<XML>
80     {
81         shared_ptr<XML> xml = super::toXML();
82         xml->setProperty("id", id);
83         xml->setProperty("name", name);
84         xml->setProperty("age", age);
85         xml->setProperty("grade", grade);
86
87         return move(xml);
88     };
89     virtual auto toHTML() const -> string
90     {
91         return toTR(id, name, age, grade);
92     };
93 };
94
95 class MemberArray
96     : public SharedEntityArray<Member>,
97     public virtual IHTMLEntity
98 {
99 protected:
100     typedef SharedEntityArray<Member> super;
101
102     string application;
103     int department;
104     Member *chief;
105
106 public:
107     /* ---------------------------------------------------------
108         CONSTRUCTORS
```



```
109     --------------------------------------------------------- */
110     MemberArray()
111         : super(), IHTMLEntity()
112     {
113         this->chief = nullptr;
114     };
115     virtual ~MemberArray() = default;
116
117     // You don't need to consider children(Member) objects
118     // Just concentrate on constructing MemberArray's own member variables
119     virtual void construct(shared_ptr<XML> xml) override
120     {
121         super::construct(xml);
122
123         this->application = xml->getProperty("application");
124         this->department = xml->getPropertyInt("department");
125
126         if(xml->hasProperty("chief") == true && this->has( xml->getProperty("chief") ) == true)
127             this->chief = this->get( xml->getProperty("chief") ).get();
128     };
129
130 protected:
131     //FACTORY METHOD FOR MEMBER
132     virtual auto createChild(shared_ptr<XML> = nullptr) -> Member* override
133     {
134         return new Member();
135     };
136
137     /* ---------------------------------------------------------
138         XML EXPORTERS
139     --------------------------------------------------------- */
140 public:
141     virtual auto TAG() const -> string override
142     {
143         return "memberArray";
144     };
145     virtual auto CHILD_TAG() const -> string override
146     {
147         return "member";
148     };
149
150     // You don't need to consider children(Member) objects
151     // Just concentrate on expressing MemberArray's own member variables
152     virtual auto toXML() const -> shared_ptr<XML>
153     {
154         shared_ptr<XML> xml = super::toXML();
155         xml->setProperty("application", application);
156         xml->setProperty("department", department);
157
158         if(chief != nullptr)
159             xml->setProperty("chief", chief->key());
160
161         return move(xml);
162     };
163     virtual auto toHTML() const -> string
164     {
165         string html = "<table>\n";
166         html += toTR("ID", "Name", "Age", "Grade") + "\n";
167
168         for (size_t i = 0; i < 2; i++)
169             html += at(i)->toHTML() + "\n";
170
171         html += "</table>";
172         return move(html);
173     };
174 };
175
176 void main()
177 {
178     string str = string("") +
179         "<memberArray application='framework' department='3' chief='samchon'>\n" +
180         "    <member id='samchon' name='Jeongho Nam' age='23' grade='5' />" +
181         "    <member id='submaster' name='No Name' age='100' grade='4' />" +
182         "    <member id='john' name='John Doe' age='33' grade='2' />" +
183         "    <member id='bad_man' name='Bad Man' age='44' grade='-1' />" +
184         "    <member id='guest' name='Guest' age='5' grade='0' />" +
185         "</memberArray>";
186     shared_ptr<XML> xml(new XML(str));
187
188     MemberArray memberArray;
189     memberArray.construct(xml);
190
191     memberArray.emplace_back(new Member("freshman", "A fresh man", 20, 2));
192     memberArray.emplace_back(new Member("senior", "A senior", 70, 2));
193
194     cout << memberArray.toXML()->toString() << endl << endl;
195     cout << memberArray.toHTML() << endl;
```

```
196     system("pause");
197 }
```

Result of the example



Note

I say repeatedly. Expression method of Entity is recommended, but not imposed. It's a semi protocol for network I/O but not a essential protocol must be kept. The expression method of Entity, using on network I/O, is expressed by XML string.

If your own network system has a critical performance issue on communication data class, it would be better to using binary communication (with ByteArray or boost::serialization). Don't worry about the problem! Invoke also provides methods for binary data (ByteArray).

See also

protocol

Author

Jeongho Nam

6.27.2 Member Function Documentation

6.27.2.1 virtual auto TAG ( ) const -> std::string [pure virtual]

A tag name when represented by XML.

Returns

A tag name

Implemented in InvokeParameter, Invoke, InvokeHistory, NTCriteria, ExternalSystemArray, ExternalSystem, ExternalSystemRole, NTParameter, NTParameterDetermined, SystemRole, ChatRoom, NTSide, NTParameterArray, ChatMessage, and FTInstance.

6.27.2.2 void construct (std::shared_ptr< library::XML >) [pure virtual]

Construct data of the Entity from an XML object.

Overrides the construct() method and fetch data of member variables from the XML.

By recommended guidance, data representing member variables are contained in properties of the put XML object.

Parameters

xml	An xml used to construct data of entity

Implemented in InvokeParameter, DistributedSystem, NTCriteria, NTFile, InvokeHistory, EntityGroup< _Container, _ETy, _Ty >, EntityGroup< _Container, std::unique_ptr< Entity > >, EntityGroup< _Container, std::shared_ptr< Entity > >, EntityGroup< _Container, Entity * >, EntityList< _Ty >, ExternalSystem, ExternalSystemRole, EntityArray< _Ty >, Invoke, DistributedSystemArray, ChatMessage, INTExplore, NTParameterDetermined, NTParameter, SystemRole, NTSide, ExternalClientArray, ExternalServer, DistributedClientArray, and DistributedServer.

6.27.2.3 auto key ( ) const -> std::string [virtual]

Get a key that can identify the Entity uniquely.

If identifier of the Entity is not atomic value, returns a string represents the composite identifier. If identifier of the Entity is not string, converts the identifier to string and returns the string.

Returns

An identifier

Reimplemented in InvokeParameter, ExternalSystem, ExternalSystemRole, FTInstance, NTParameter, NTParameterDetermined, and SystemRole.

6.27.2.4 auto toXML ( ) const -> std::shared_ptr<library::XML> [virtual]

Get an XML object represents the Entity.

Returns an XML object that can represents the Entity containing member variables into properties.

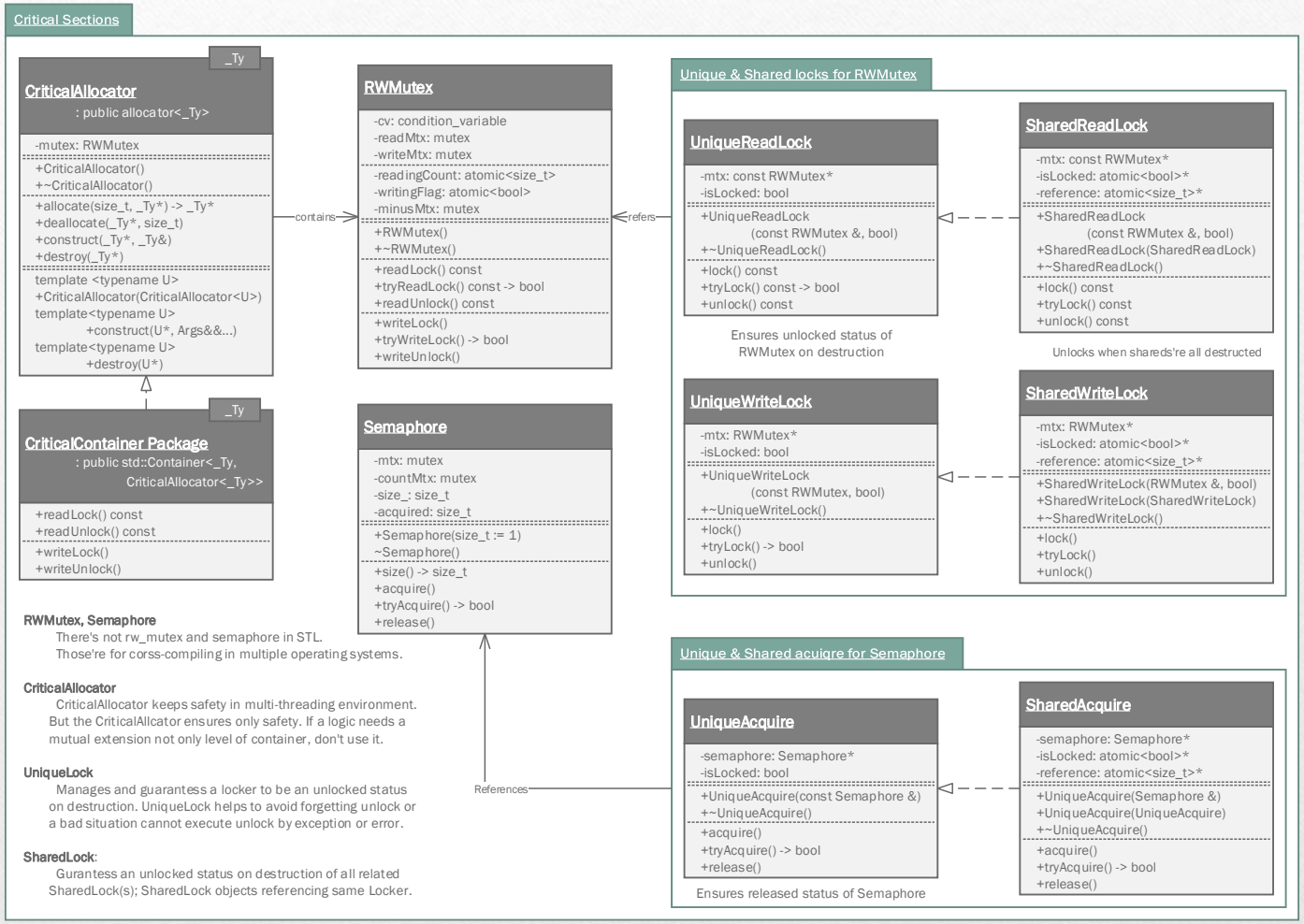
A member variable (not object, but atomic value like number, string or date) is categorized as a property within the framework of entity side. Thus, when overriding a toXML() method and archiving member variables to an XML object to return, puts each variable to be a property belongs to only an XML object.

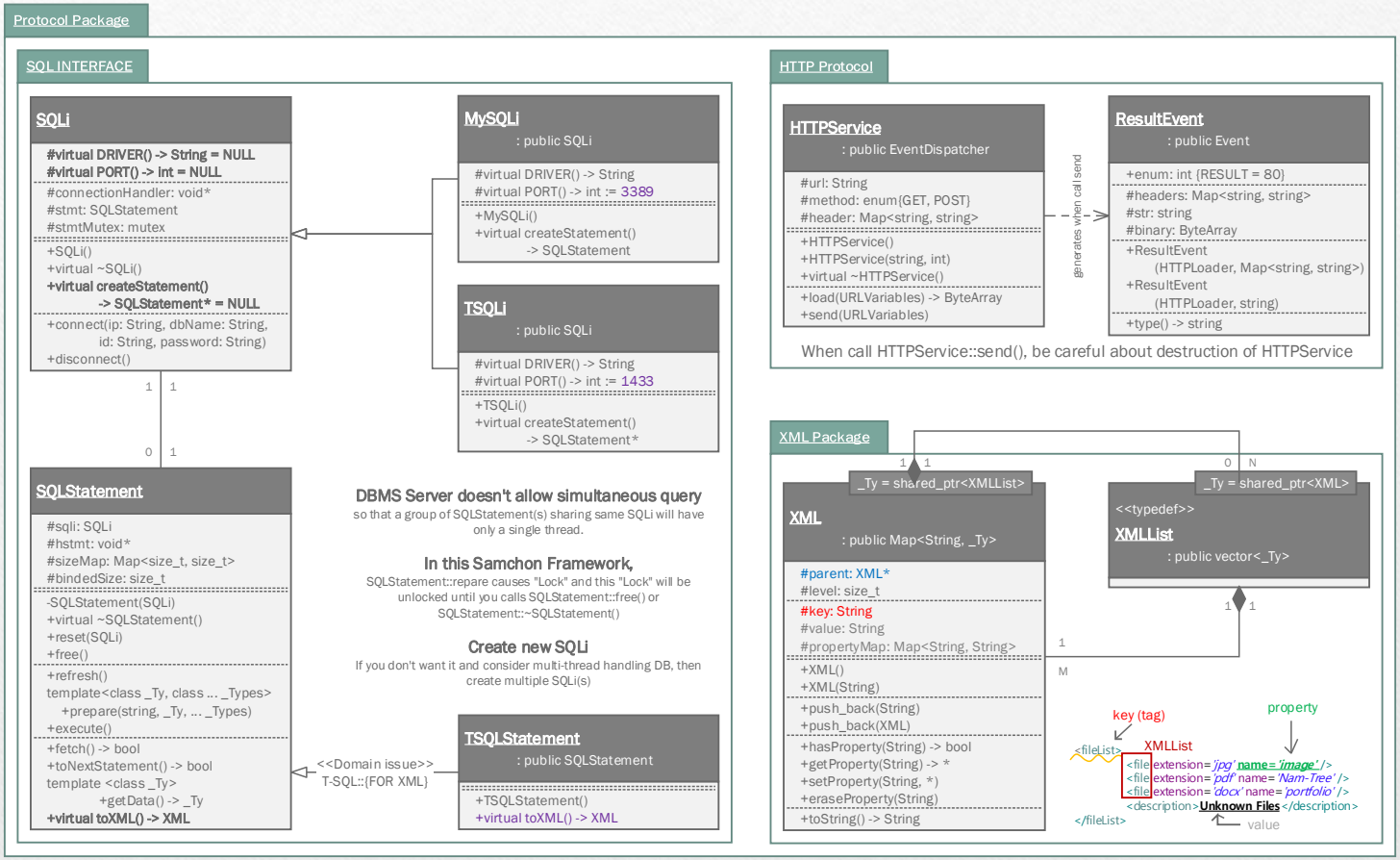
Don't archive the member variable of atomic value to XML::value causing enormouse creation of XML objects to number of member variables. An Entity must be represented by only an XML instance (tag).

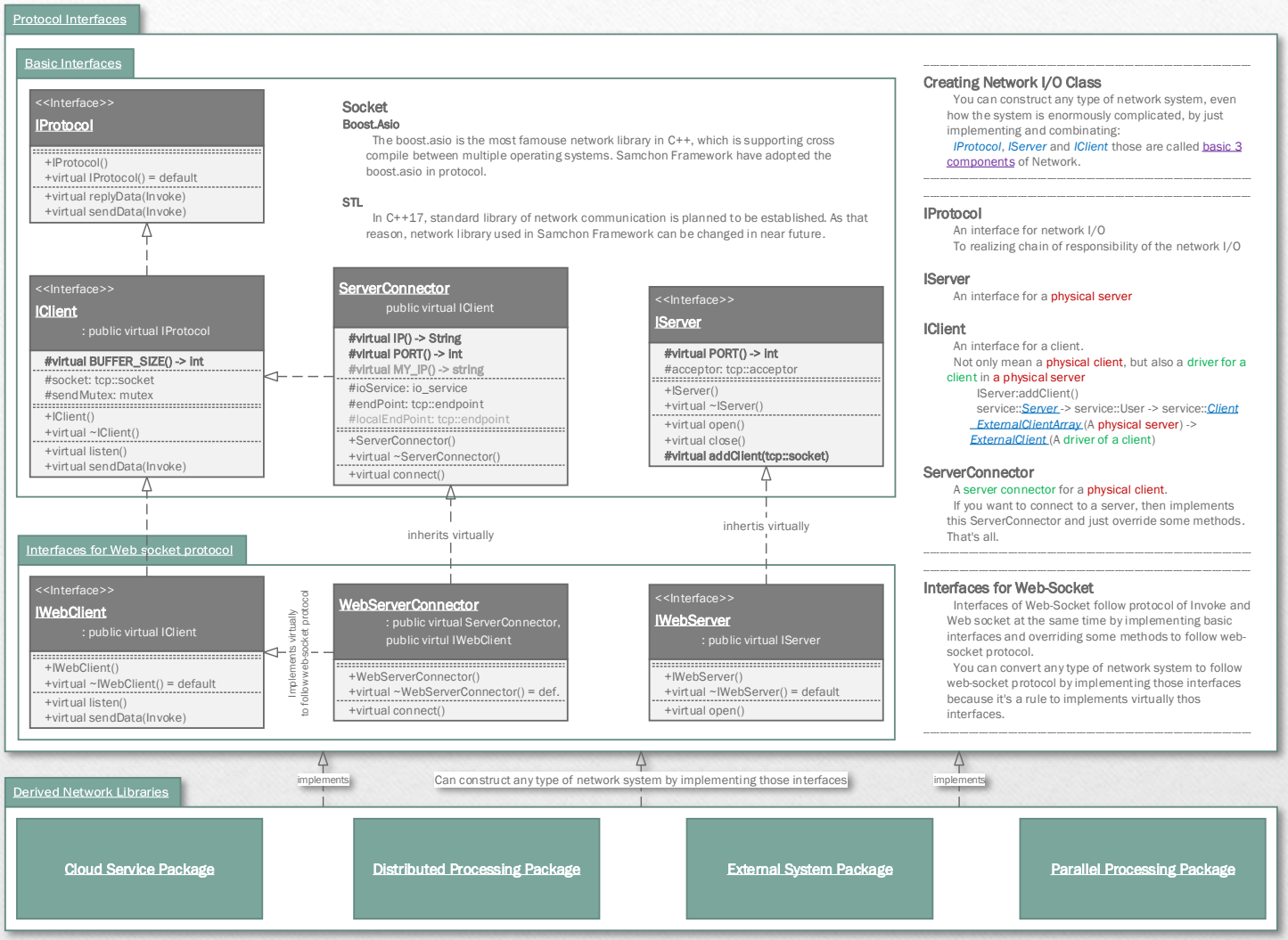
Standard Usage	Non-standard usage abusing value
<memberList> <member id='jhnam88' name='Jeongho Nam' birthdate='1988-03-11' /> <member id='master' name='Administartor' birthdate='2011-07-28' /> </memberList>	<member> <id>jhnam88</id> <name>Jeongho Nam</name> <birthdate>1988-03-11</birthdate> </member>

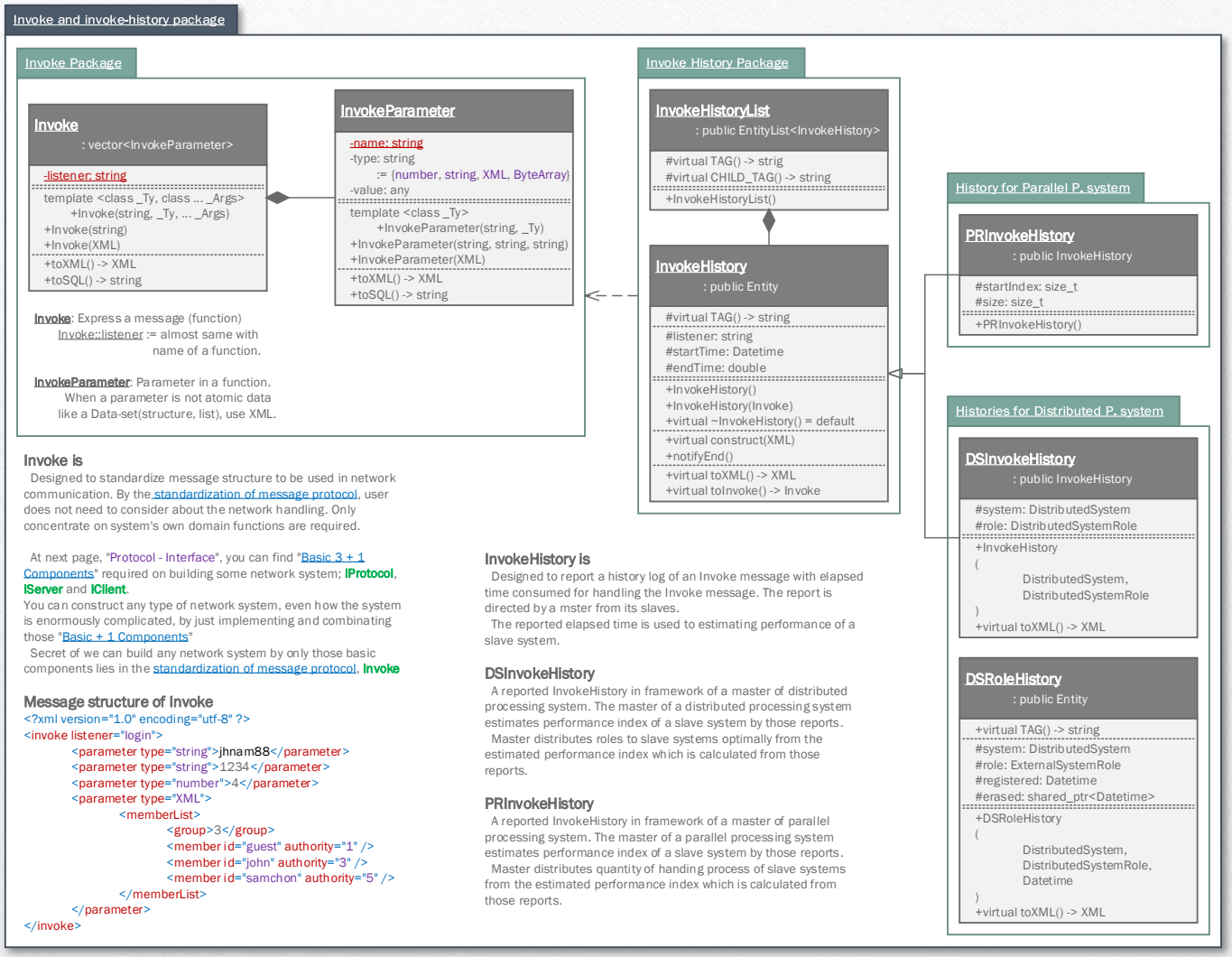
1. Documents - API

- 총 1,400 여 페이지 추정
  - C++ 약 900 여 페이지
  - TypeScript, Flex는 모름
- 미시적인 도움을 제공
  - 가장 상세한 수준의 문서
  - 모든 주요 모듈, 클래스 및 멤버에 관하여 서술함


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
_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()
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()
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.
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()


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
1
1
_Ty = shared_ptr<XMLList>
0
N
_Ty = shared_ptr<XML>
XML
: public Map<String, _Ty>
#parent: XML*
#level: size_t
#key: String
#value: String
#propertyMap: Map<String, String>
+XML()
+XML(String)
+push_back(String)
+push_back(XML)
+hasProperty(String) -> bool
+getProperty(String) -> *
+setProperty(String, *)
+eraseProperty(String)
+toString() -> String
<<typedef>>
XMLList
: public vector<_Ty>
1
1
1
M
key (tag)
property
<fileList>
XMLList
<file extension='jpg' name='image' />
<file extension='pdf' name='Nam-Tree' />
<file extension='docx' name='portfolio' />
<description>Unknown Files</description>
</fileList>
value


Protocol Interfaces
Basic Interfaces
<<Interface>>
IProtocol
+IProtocol()
+virtual IProtocol() = default
+virtual replyData(Invoke)
+virtual sendData(Invoke)
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.
<<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)
inherits virtually
inherits 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 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

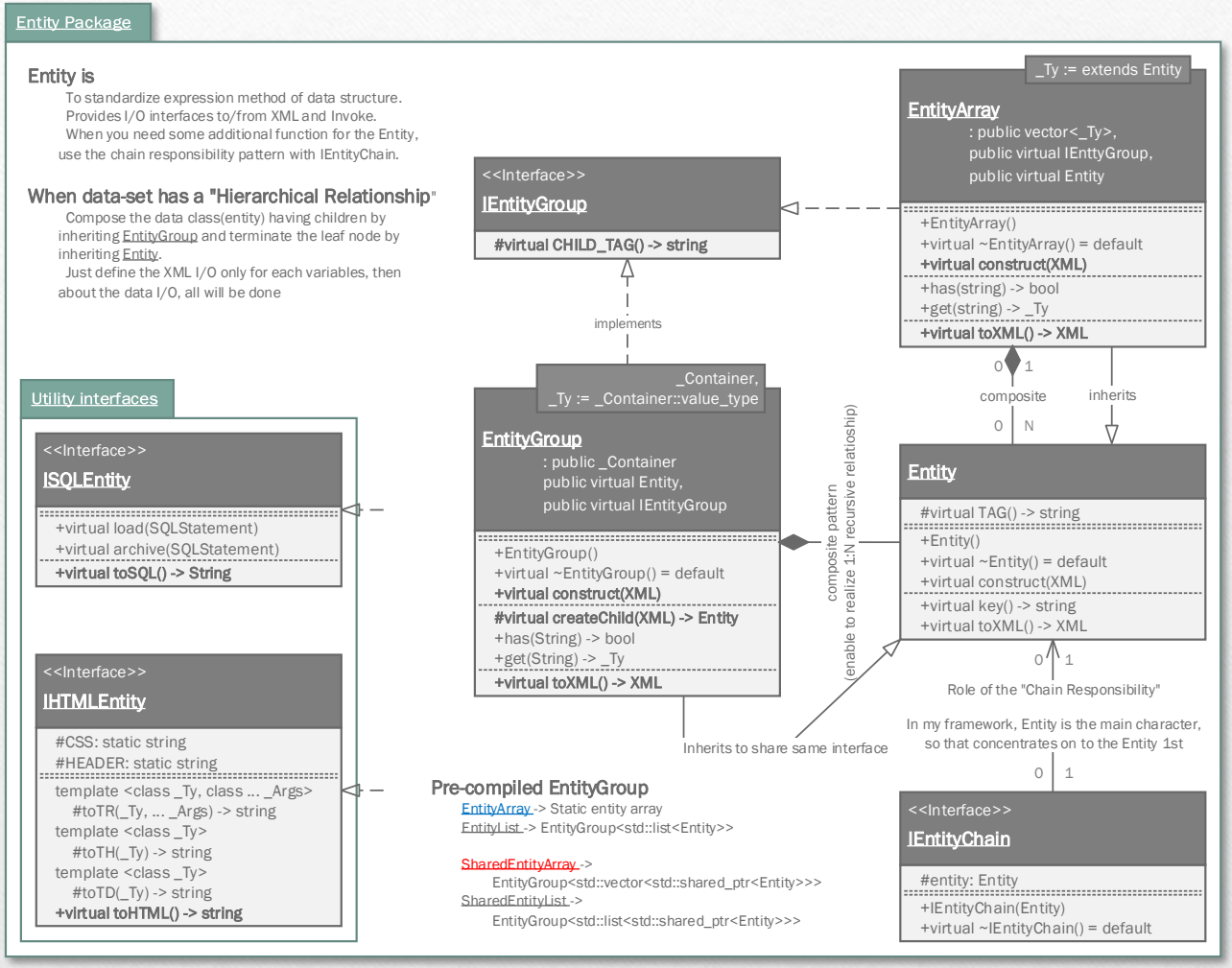
Designed to report a history log of an Invoke message with elapsed time consumed for handling the Invoke message. The report is directed by a mster from its slaves.
The reported elapsed time is used to estimating performance of a slave system.

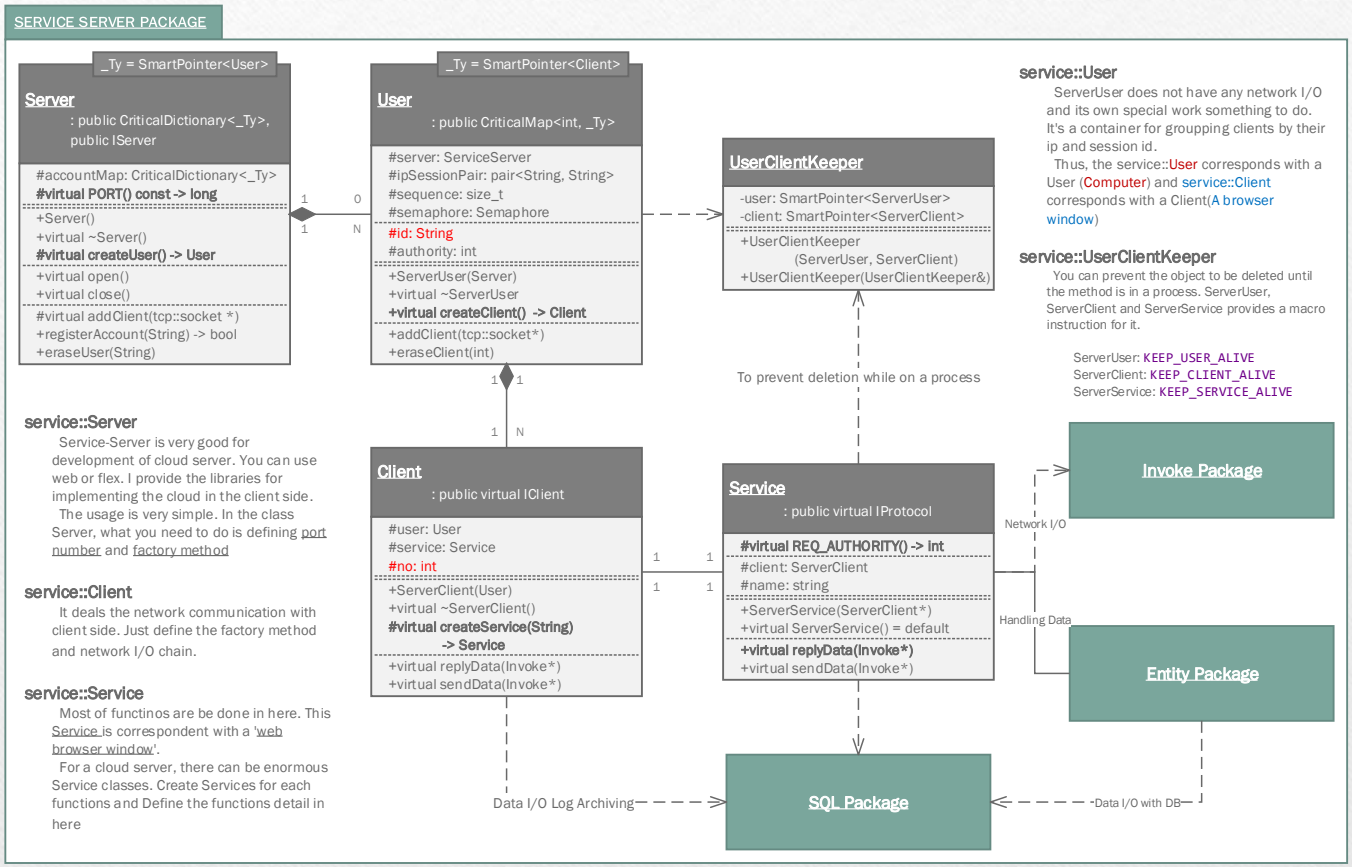
DSInvokeHistory

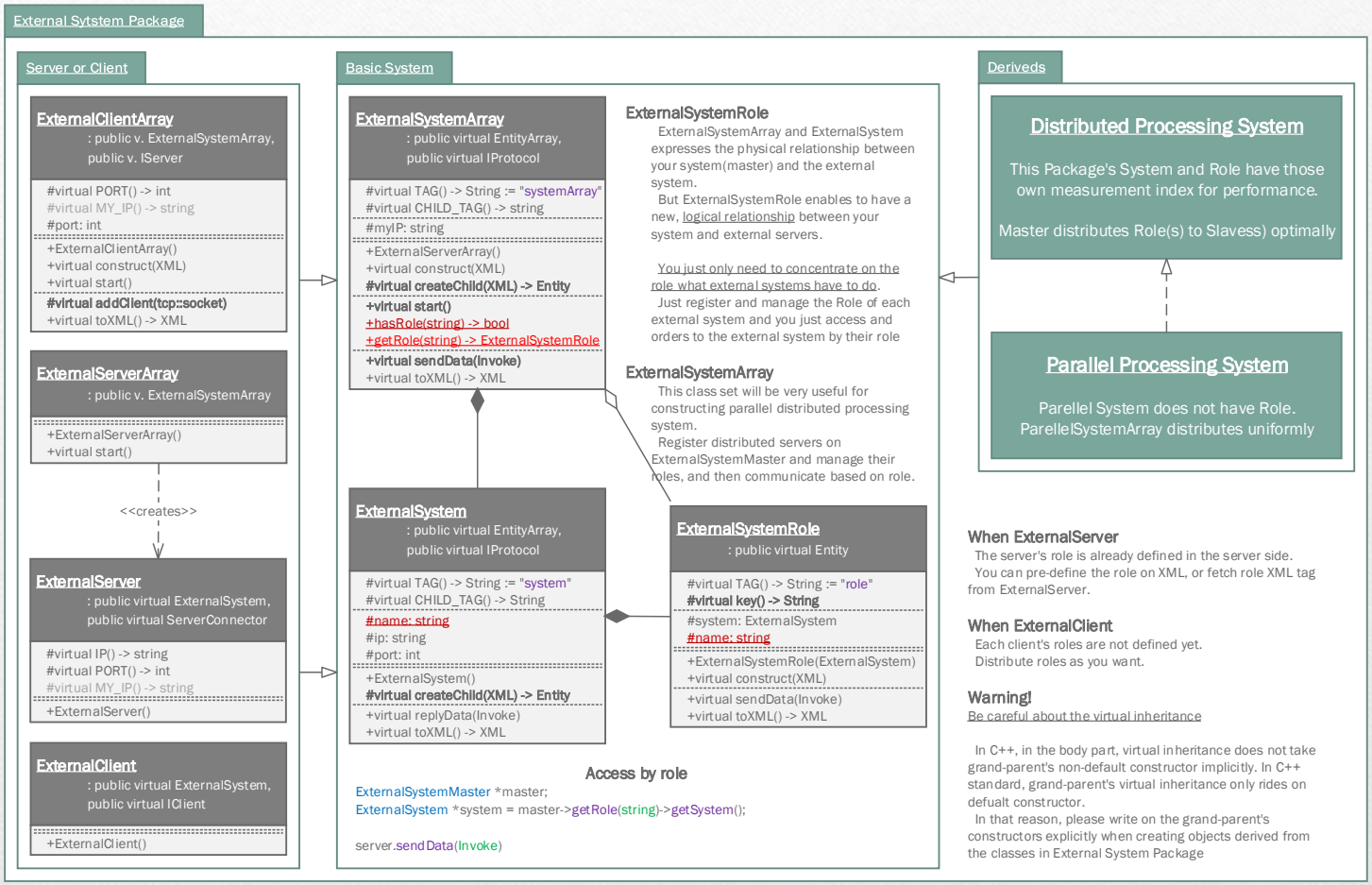
A reported InvokeHistory in framework of a master of distributed processing system. The master of a distributed processing system estimates performance index of a slave system by those reports.
Master distributes roles to slave systems optimally from the estimated performance index which is calculated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

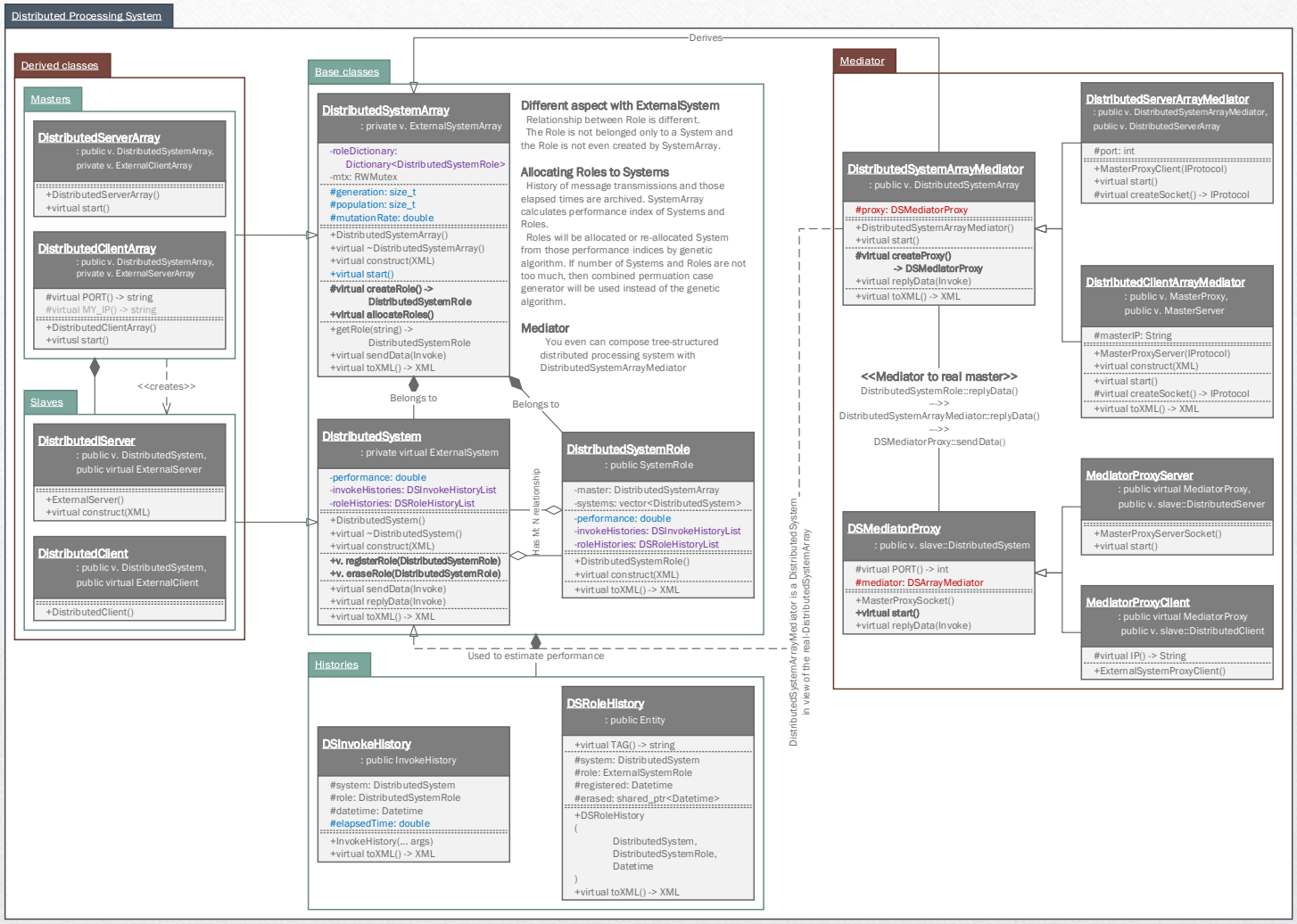
PRInvokeHistory

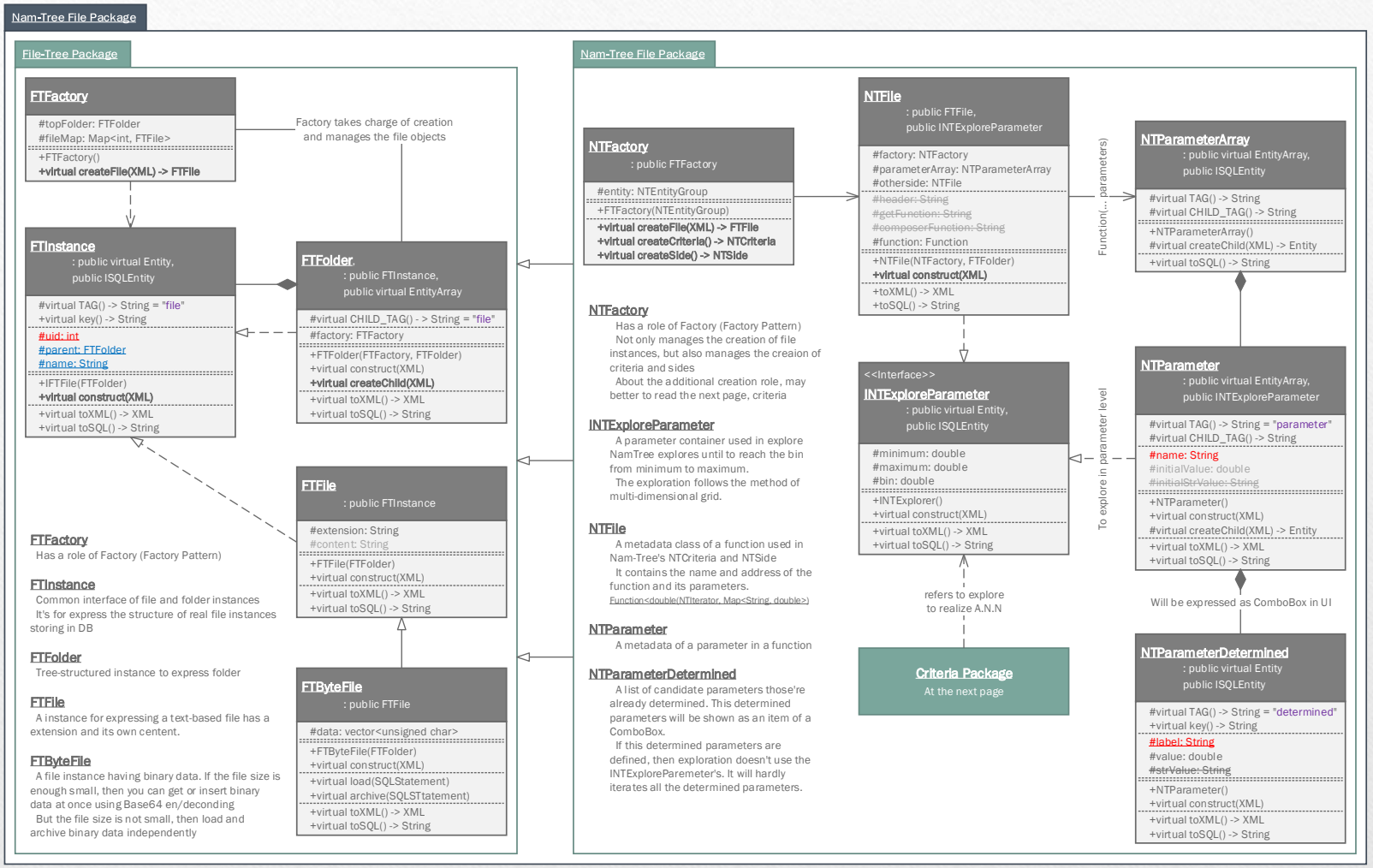
A reported InvokeHistory in framework of a master of parallel processing system. The master of a parallel processing system estimates performance index of a slave system by those reports.
Master distributes quantity of handing process of slave systems from the estimated performance index which is calculated from those reports.

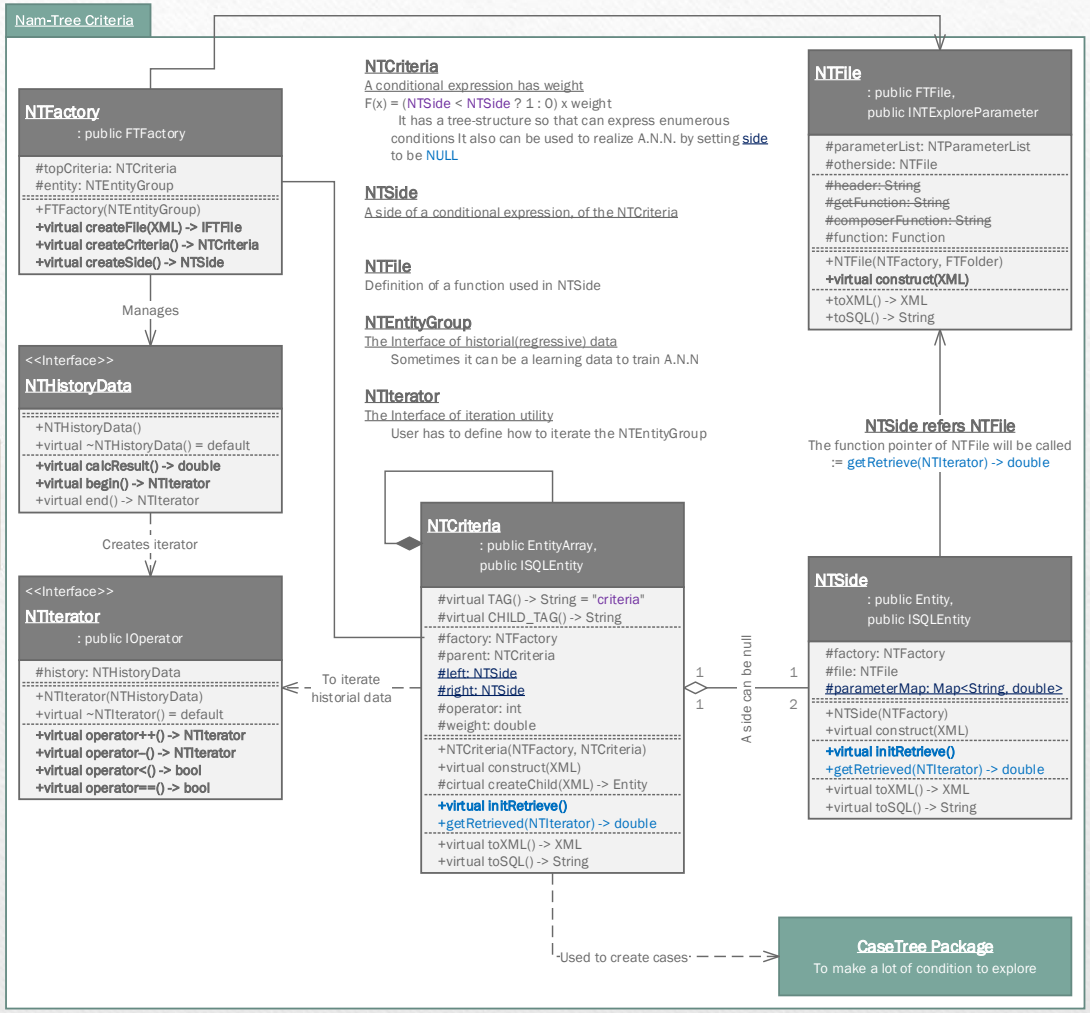

Entity Package
Entity is
To standardize expression method of data structure.
Provides I/O interfaces to/from XML and Invoke.
When you need some additional function for the Entity, use the chain responsibility pattern with IEntityChain.
When data-set has a "Hierarchical Relationship"
Compose the data 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
<<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
_Container,
_Ty := _Container::value_type
EntityGroup
: public _Container
public virtual Entity,
public virtual IEntityGroup
+EntityGroup()
+virtual ~EntityGroup() = default
+virtual construct(XML)
#virtual createChild(XML) -> Entity
+has(String) -> bool
+get(String) -> _Ty
+virtual toXML() -> XML
Inherits to share same interface
composite pattern
(enable to realize 1:N recursive relatioship)
_Ty := extends Entity
EntityArray
: public vector<_Ty>,
public virtual IEnttyGroup,
public virtual Entity
+EntityArray()
+virtual ~EntityArray() = default
+virtual construct(XML)
+has(string) -> bool
+get(string) -> _Ty
+virtual toXML() -> XML
0
1
composite
inherits
0
N
Entity
#virtual TAG() -> string
+Entity()
+virtual ~Entity() = default
+virtual construct(XML)
+virtual key() -> 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
<<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>>>


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
1
0
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
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*)
Service
: public virtual IProtocol
#virtual REQ_AUTHORITY() -> int
#client: ServerClient
#name: string
+ServerService(ServerClient*)
+virtual ServerService() = default
+virtual replyData(Invoke*)
+virtual sendData(Invoke*)
1
1
1
1
Network I/O
Handling Data
Invoke Package
Entity Package
SQL Package
Data I/O Log Archiving
Data I/O with DB
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
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


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


Distributed Processing System
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()
+virtual start()
<<creates>>
Slaves
DistributedIServer
: public v. DistributedSystem,
public virtual ExternalServer
+ExternalServer()
+virtual construct(XML)
DistributedClient
: public v. DistributedSystem,
public virtual ExternalClient
+DistributedClient()
Derives
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()


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 content.
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
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
refers to explore to realize A.N.N
Criteria Package
At the next page
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.



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 relationship

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)

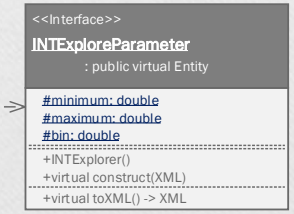
Make a **NTSide** to be nullptr, then NTCriteria will explore the best value
Nam-Tree will calculate the conditions from mininmum to maximum in INTExploreParameter reach to the bin, by the method of multi-dimensional grid.

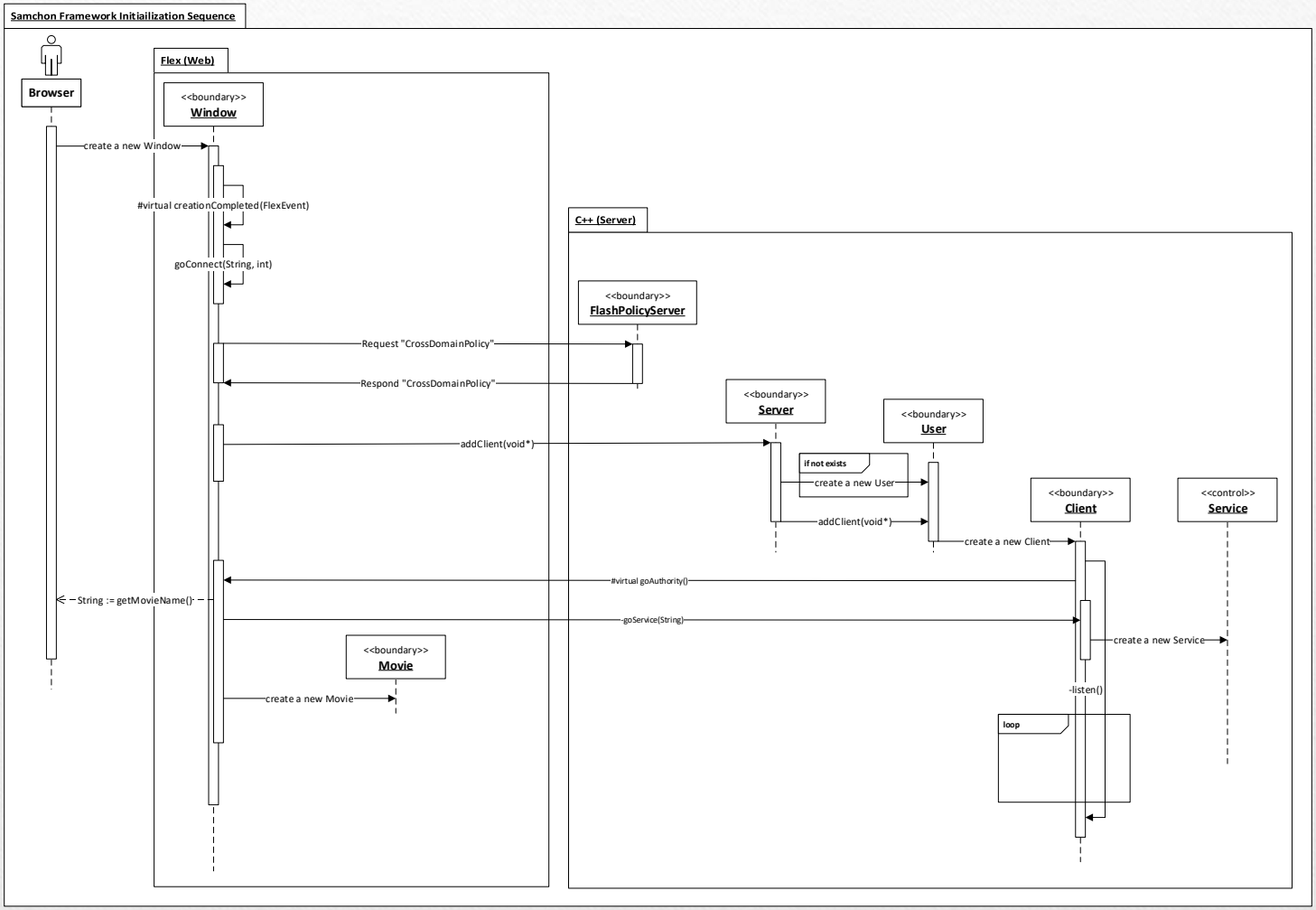
4-2. Exploring parameter in NTSide

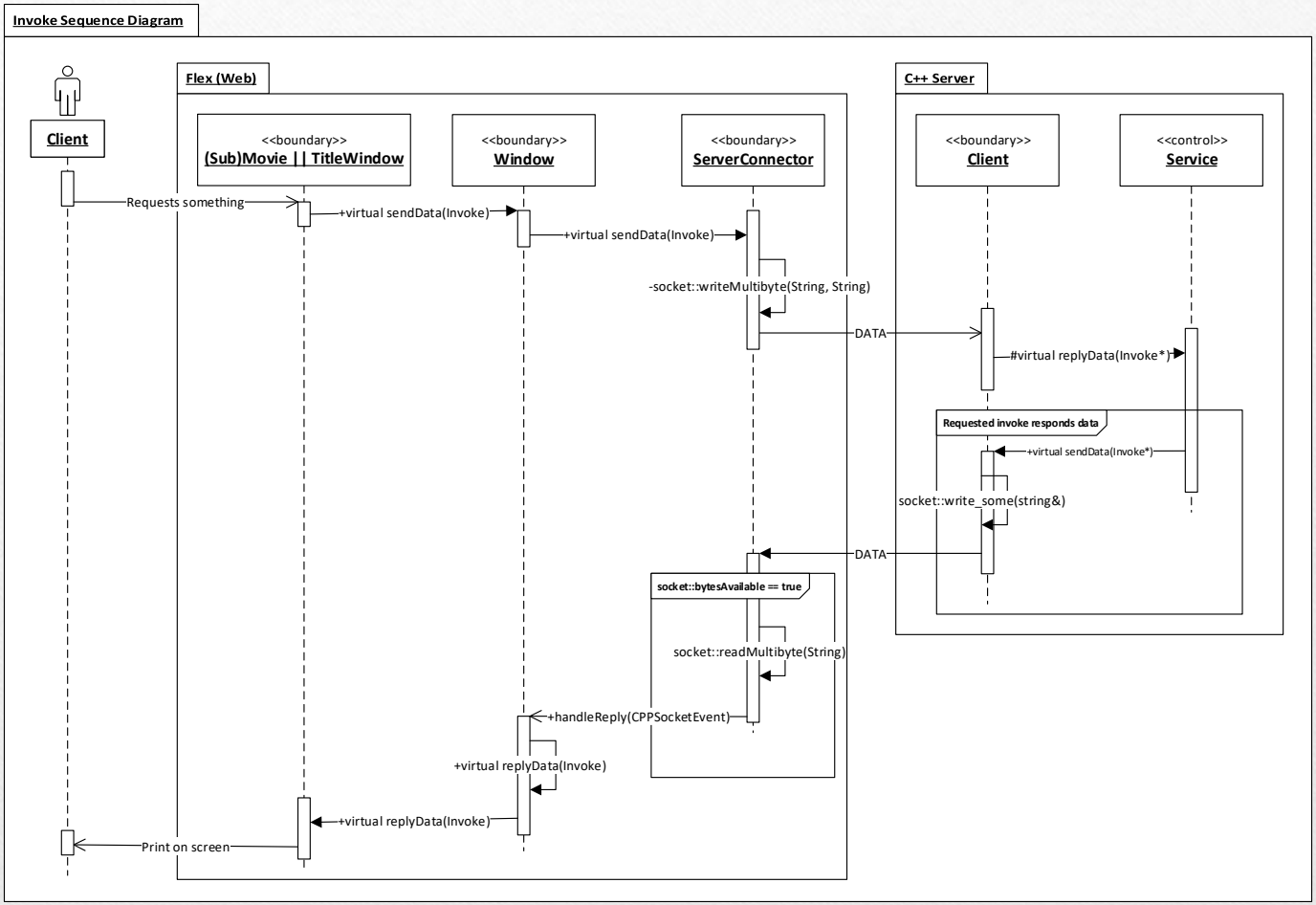
If you set the parameterMap to be empty, Nam-Tree will explore the best parameter until reach to the bin in INTExploreParameter from minimum to maximum

4-3. Exploring by creating NTCriteria (create conditions)

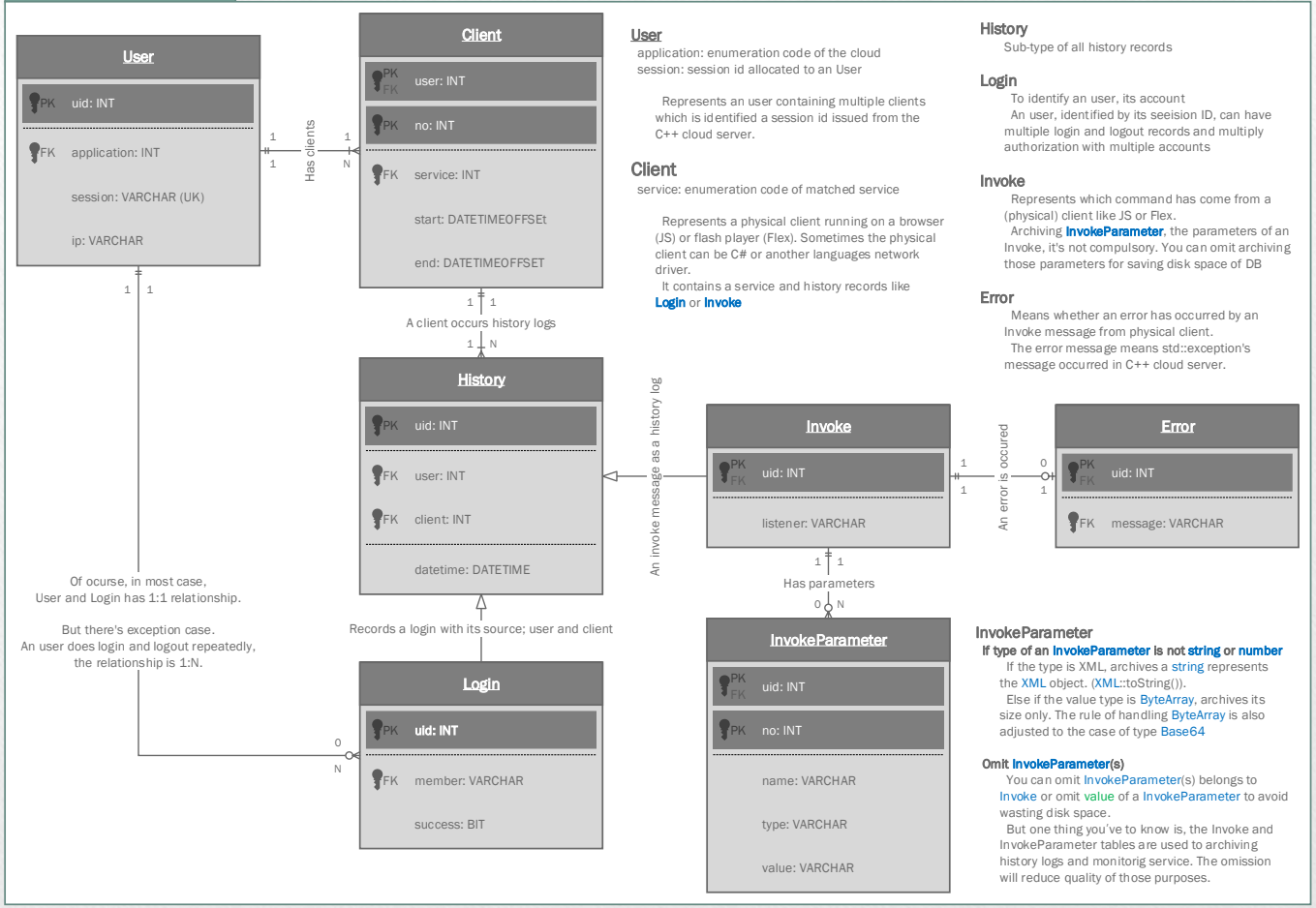
If all the side (left and right) in a NTCriteria are nullptr, then the NTCriteria will make a lot of children NTCriteria(s) to test a lot of cases that can be, so that best condition will be made up.
This process will ride on same routines of 4-1, and 4-2, for each created cases. Of course, this process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

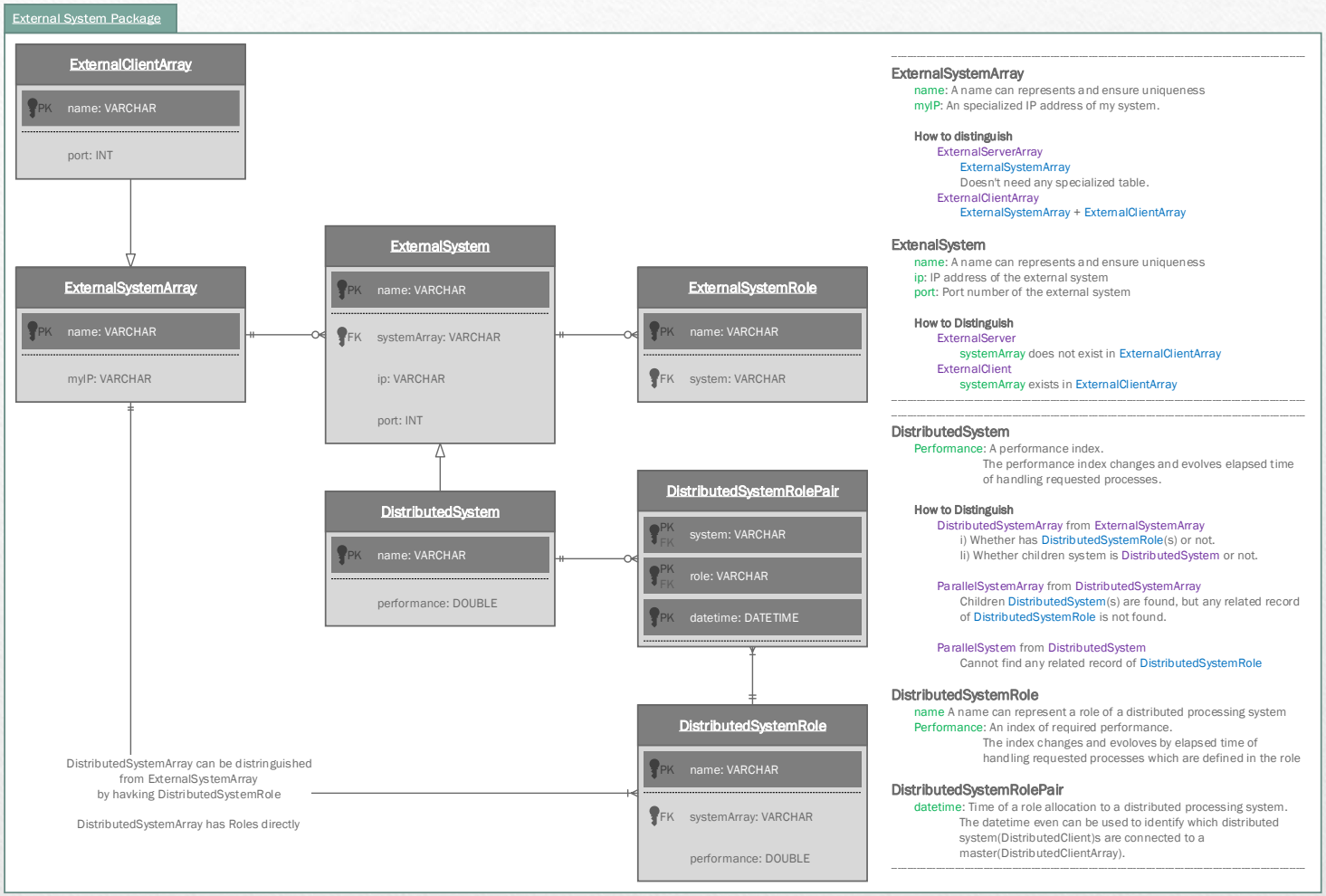



Samchon Framework Initiailization Sequence
Browser
Flex (Web)
<<boundary>>
Window
create a new Window
#virtual creationCompleted(FlexEvent)
goConnect(String, int)
C++ (Server)
<<boundary>>
FlashPolicyServer
Request "CrossDomainPolicy"
Respond "CrossDomainPolicy"
<<boundary>>
Server
<<boundary>>
User
addClient(void*)
if not exists
create a new User
addClient(void*)
<<boundary>>
Client
<<control>>
Service
create a new Client
#virtual goAuthority()
String := getMovieName()
goService(String)
create a new Service
<<boundary>>
Movie
listen()
create a new Movie
loop


Invoke Sequence Diagram
Flex (Web)
C++ Server
Client
<<boundary>>
(Sub)Movie | | TitleWindow
<<boundary>>
Window
<<boundary>>
ServerConnector
<<boundary>>
Client
<<control>>
Service
Requests something
+virtual sendData(Invoke)
+virtual sendData(Invoke)
-socket::writeMultibyte(String, String)
DATA
#virtual replyData(Invoke*)
Requested invoke responds data
+virtual sendData(Invoke*)
socket::write_some(string&)
DATA
socket::bytesAvailable == true
socket::readMultibyte(String)
+handleReply(CPPSocketEvent)
+virtual replyData(Invoke)
+virtual replyData(Invoke)
Print on screen

Historical Log of Service Package





ExternalSystemArray

name: A name can represents and ensure uniqueness
myIP: An specialized IP address of my system.

How to distinguish
- ExternalServerArray
  - ExternalSystemArray
    - Doesn't need any specialized table.
- ExternalClientArray
  - ExternalSystemArray + ExternalClientArray

ExtenalSystem

name: A name can represents and ensure uniqueness
ip: IP address of the external system
port: Port number of the external system

How to Distinguish
- ExternalServer
  - systemArray does not exist in ExternalClientArray
- ExternalClient
  - systemArray exists in ExternalClientArray

DistributedSystem

Performance: A performance index.
The performance index changes and evolves elapsed time of handling requested processes.

How to Distinguish
- DistributedSystemArray from ExternalSystemArray
  - i) Whether has DistributedSystemRole(s) or not.
  - Ii) Whether children system is DistributedSystem or not.
- ParallelSystemArray from DistributedSystemArray
  - Children DistributedSystem(s) are found, but any related record of DistributedSystemRole is not found.
- ParallelSystem from DistributedSystem
  - Cannot find any related record of DistributedSystemRole

DistributedSystemRole

name A name can represent a role of a distributed processing system
Performance: An index of required performance.
The index changes and evoloves by elapsed time of handling requested processes which are defined in the role

DistributedSystemRolePair

datetime: Time of a role allocation to a distributed processing system.
The datetime even can be used to identify which distributed system(DistributedClient)s are connected to a master(DistributedClientArray).

1. Documents - Designs

- 40 여 장의 설계도
- Architecture Designs
  - C++ Class diagram
  - JS Class Diagram
  - Sequence Diagram
  - Network Diagram
- Examples' designs

1. Documents - Guidance

- 개발 가이드 문서
- API 문서에 대한 개괄 요약본
  - 거시적인 관점에서 각 모듈별로 간략히 서술
  - 아키텍처 디자인에 대한 해설도 곁들임
- 한글로 작성됨

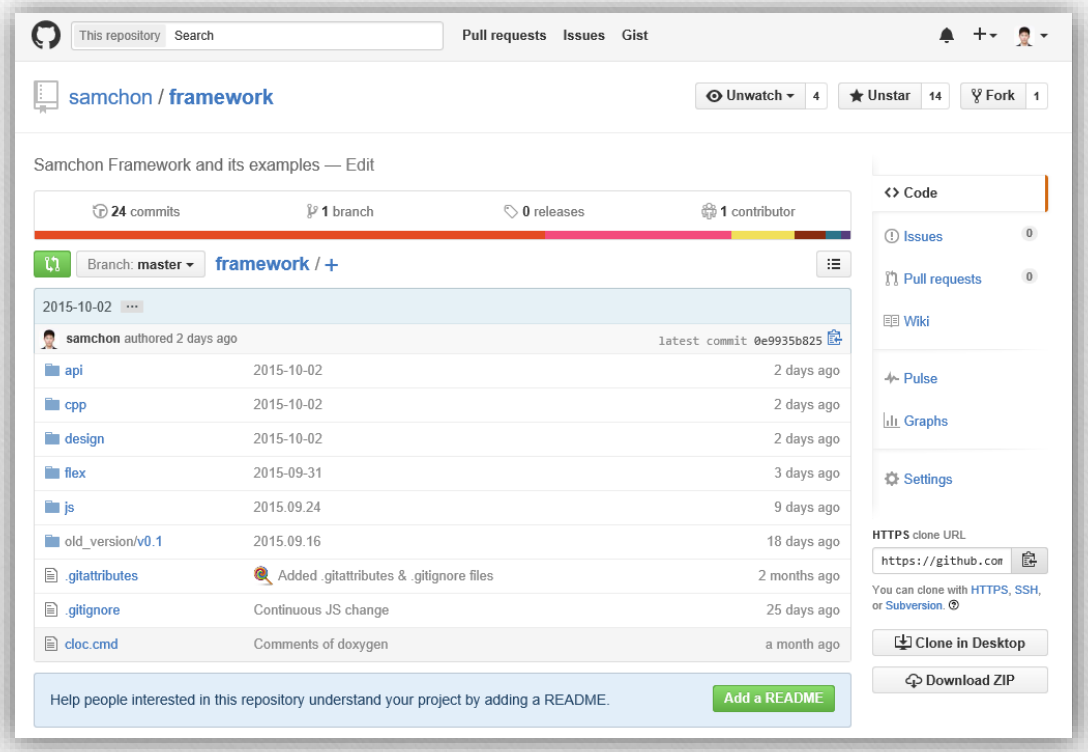
2. Examples

- 설계 문서와 가이드 문서를 통해 삼촌 프레임워크의 전반을 개략적으로 이해하고
- 예제 코드를 통해 실 활용 사례를 익히며
- API 문서로 세세한 사항들을 확인할 수 있는 것이 이상적

2. Examples

- 각 모듈별 활용 방법을 예제 코드로 적었으며
  - 앞서의 병렬-분선처리 네트워크 시스템도 이러한 예제들 중에 하나
- 솔루션급 예제도 두 개 제공된다
  - Samchon Simulation
  - Hansung Timetable

3. GitHub


This repository
Search
Pull requests
Issues
Gist
samchon / framework
Unwatch 4
Unstar 14
Fork 1
Samchon Framework and its examples — Edit
24 commits
1 branch
0 releases
1 contributor
Branch: master
framework / +
2015-10-02
samchon authored 2 days ago
latest commit 0e9935b825
api 2015-10-02 2 days ago
cpp 2015-10-02 2 days ago
design 2015-10-02 2 days ago
flex 2015-09-31 3 days ago
js 2015.09.24 9 days ago
old_version/v0.1 2015.09.16 18 days ago
.gitattributes Added .gitattributes & .gitignore files 2 months ago
.gitignore Continuous JS change 25 days ago
cloc.cmd Comments of doxygen a month ago
Help people interested in this repository understand your project by adding a README.
Add a README
Code
Issues 0
Pull requests 0
Wiki
Pulse
Graphs
Settings
HTTPS clone URL
https://github.com
You can clone with HTTPS, SSH, or Subversion.
Clone in Desktop
Download ZIP

3. GitHub

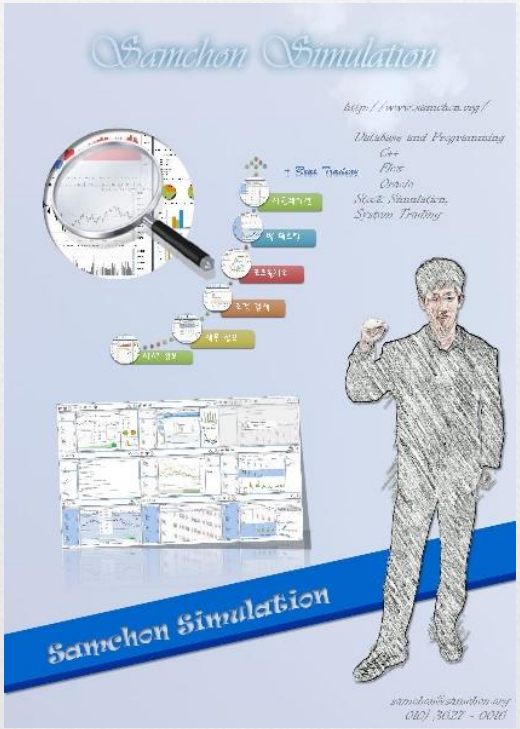
- 공식 홈페이지
  - http://samchon.org/framework
  - http://samchon.github.io/
- 깃허브
  - https://github.com/samchon
  - https://github.com/samchon/framework

Future Plan

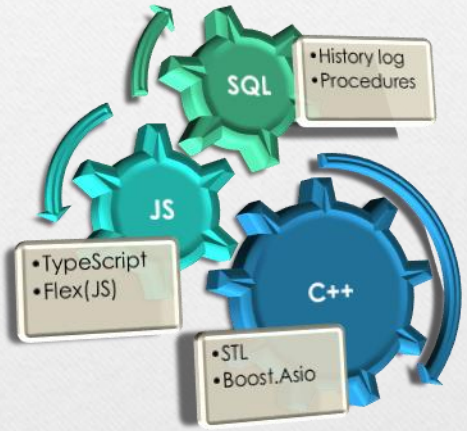
1. 공개SW개발자대회 2016
2. Samchon Framework
3. Samchon UML

1. 2016 공개SW 개발자대회

- 20대가 지나기 전,
- 우승할 때까지
- 계속 출품할 생각
- 다음 대회 때 또 뵈요



2. Samchon Framework

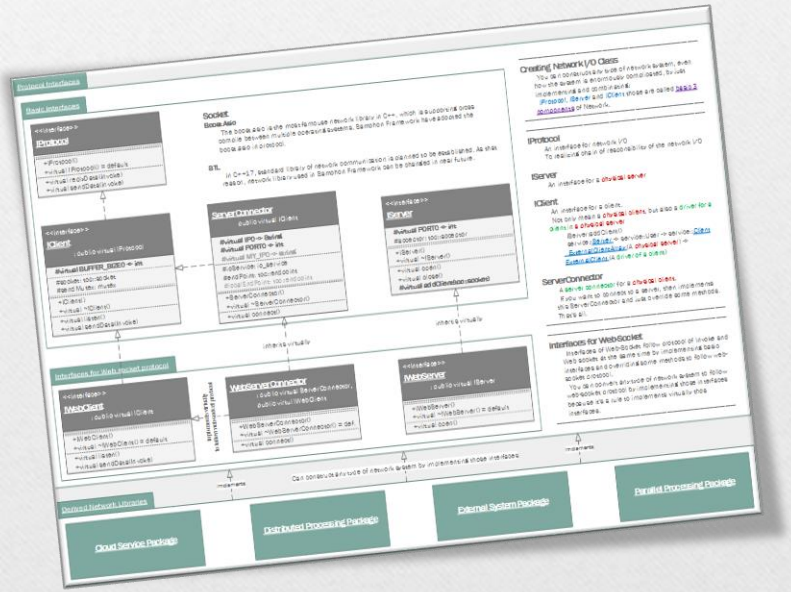


- 더 많은 예제
- 더 간결한 문서

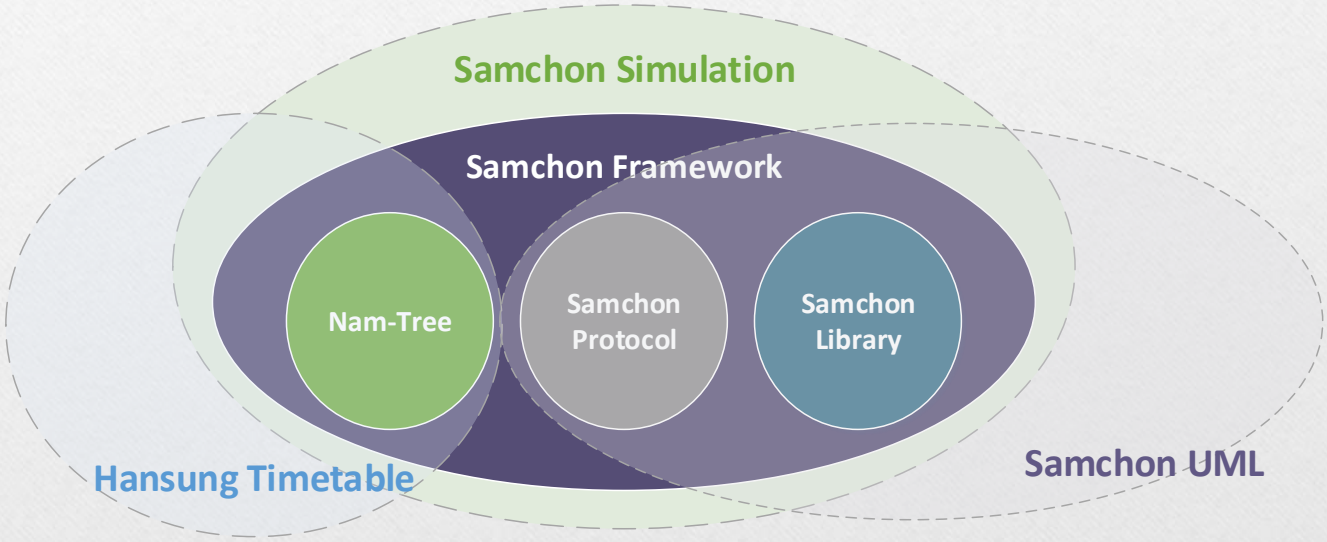
- 분산처리 강화
- 자바스크립트 강화

3. Samchon UML

- 네트워크 시스템을
  - 객체지향적으로 설계할 수 있는 프레임워크
- UML 설계 단위에서 구현할 수 있도록
- Samchon UML 제작



3. Samchon UML



마치며

Samchon Framework

남정호

http://samchon.org