Dear Inmind PI’s and Students:

We are pleased to inform you that we have released a new Multiuser Framework (MUF) that is available through our github repository (https://github.com/inmind-project/inmind-development.git) under directory /Server/Multiuser/. The MUF is a library that is hosted on a Maven repository so you can use it as simple as adding it as a dependency on your deployment descriptor file. On the github repo you will find some examples of how to use the MUF.

If your module performs certain processing at server side, this framework will ease you the implementation of a scalable system with the following features:

* Session Management: a Session Manager automatically creates a new session every time a client connects and then manages its lifecycle (i.e., connection, disconnection, control of inactivity, resources management, etc.)
* Low latency: a whole communication action (send a message from the client and receive a response from the server) takes around 10-13 ms. in total. This test was made using my own computer (MacBook Pro 2.5 GHz Intel core i7, 16 GB 1600 MHz DDR) but it could be improved if you use an AWS instance. These latency ranges were the same even when the framework was tested with 1,000 clients.
* Robustness and Reliability: the MUF supports error handling (e.g., crash errors trigger specific system recovery actions), disconnection and automatic reconnection (if network fails or communication freezes or gets locked), management of queue overflows and memory leaks, data loss prevention (when a client disconnects due to an unexpected reason, the system will keep messages in memory until the client reconnects), gracefully shutdown (if system crashes and cannot be restarted, all resources such as sockets, queues, shared objects, DB, etc. are closed and released before the system exits).
* Simple: all the communication complexity is hidden, that is, you don’t have to deal with details about sockets, http requests, ports, etc. You just call functions of other modules and components as if they were local objects, despite of the fact they may be services running remotely.
* Pluggable Architecture: the MUF allows you to define pluggable modules that can be added anytime and using different mechanisms, so you can easily evolve your system over time and make it more flexible to changes. There are two kind of pluggable modules: Orchestrators that are in charge of arbitrating the internal processing of each request message from a client, and PluggableComponents that are your domain-specific modules (e.g., in a dialogue system domain, PluggableComponents may be ASR, NLG, NLU, DM, etc.)
* Component state: you can define your components’ behavior as simple as adding an “annotation” to your code. Your components may be Stateful (it keeps an internal representation or model so a new component is created for each session), Stateless (no representation of the state, so a component is shared by multiple sessions) or Pool (you define a maximum number of instances of your component).
* Blackboard: the MUF uses a Blackboard system, a common knowledge base that is iteratively updated by a diverse group of specialist knowledge sources, in this case, the Pluggable modules. Blackboard updates all the listeners that are subscribed to specific messages from other components. Also, the blackboard keeps a history of all interactions between components (components never communicate directly to each other but through the Blackboard) so components may extract past messages stored in the Blackboard.
* Automatic Scalability: thanks to the MUF design and the communication patterns it uses (e.g, PUB-SUB, REQ-RESP, ROUTER-DEALER, etc.) your system can easily scale from 1 client to thousands without any extra effort. Now, if you use Amazon Lambda server (AWS) as well, then this scalability feature will be improved even further. MUF’s Philosophy: you write your app for one client and run it for thousands.
* User Model: you can conveniently create your customized user models and store them on disc. A User Model component filters messages that go through the Blackboard and extracts specific information from these messages (e.g., user preferences and interests).
* Sync and Async execution: MUF provides an API so you can define different flows of control in your orchestrator, so you can use an event-oriented approach, or a direct-invocation approach. Also, you can define how your components are executed (sync or asynchronously) and, in those cases where components run asynchronously, you can force the synchronization.
* Inversion of Control: the MUF uses dependency injection technique, so your implementation delegates the responsibility of providing its dependencies to the MUF (injector). It allows your system to be reconfigured without recompilation, make easier to unit test your components in isolation, allow concurrent or independent development, etc.
* Logging: you can define multiple levels of logging and different ways to store logs (files, json, database, etc.). All interaction that goes through the blackboard is logged by default, but you can add additional logging points.
* Contracts: the MUD defines contracts (shared libraries with classes and global constants) to avoid issues with parsing and matching messages and content.
* API ‘s: the MUF provides different API’s that encapsulate natural patterns for communication, parsing json, serialization, etc.
* Multiplatform and Multilanguage: thanks to MUF uses a potent messaging and concurrency libraries such as ZMQ, you can easily communicate the MUF implementation (Java code) with clients written in almost any programming language with low extra effort (there is an example in the github repo of how to connect Java MUF with a Python MUF, but so many other languages may be supported). Also, you can communicate your MUF with another MUF which runs on another machine, that means that your MUF may behave as a server when receiving requests from clients (phones) or as a client when it sends or forwards messages to other MUF’s and waits for responses from them, so you can get a nested architecture of MUF’s.

Please find attached an oversimplified picture of the architecture.

If you think that this framework fits your needs, do not hesitate to contact me and I will give you detailed information of how to use it.

Best,

Oscar