HUMANOBS – Humanoids That Learn Socio-Communicative Skills By Observation
EU 7th Framework Project #231453

Document Title:

mBrane Release Doc

mBrane Version 1.0

Authors

Thor List, CMLabs
Eric Nivel, RU-CADIA
Kristinn R. Thórisson, RU-CADIA

PART 2 OF 6 OF DELIVERABLE **D5 Platform Software Release 1**					
BELONGS TO WP:	WP4 - Realtime Experimental Platform				
WP LEAD:	CMLabs				
WP PARTICIPANTS:	CMLabs, UNIPA-DINFO, SUPSI-IDSIA, RU-CADIA				
	Del. ID #	**WP**	**Orig. Date**	**Actual Date**	**Docum. Vers.**
DATA	5	4	Mo12	Mo12	1
Remarks:					

Introduction

This document covers the technical details of the version 1.0 release of the mBrane framework. It will describe the version history, the current state of the software, the technical issue encountered and which ones were solved, the performance and the further work.

mBrane Version History

Version 1.0 is the first official release of mBrane. This targets 32 and 64 bit versions of Microsoft Windows 7 (will work on Windows XP and Windows Vista) and 32 and 64 bit versions of Linux distributions (should work on any Unix system with PThreads and GCC version 4.3 and above).

Version 1.0 has a full implementation of the space-based subscription system working across platforms and uses UDP and non-blocking TCP connections between the nodes. It does not yet include any caching of data inside the nodes, nor does it support Infiniband as a transport mechanism (see mBrane Roadmap for more details).

Current State

Version 1.0 of mBrane includes the following major features:

mBrane Core

The mBrane Core contains the implementation of all the basic components for handling memory, messages, smart pointers, payloads and communication. It also provides the OS independent functions for working with queues and pipes, XML parsing, semaphores, mutexes, threading and sockets.

mBrane Node

The mBrane Node provides the main user application interface which users need to create mBrane modules. It contains the main implementation of the mBrane Network Node including initialisation, reference node selection, subsequent node registration, inter-node synchronisation and node deregistration. It provides the execution model using thread pools and registers all subscriptions for data, modules and spaces.

mBrane TCP Library

The mBrane TCP library provides an mBrane Node with the ability to select and use TCP seamlessly as a communication protocol between two nodes. This can be used for control, data and streaming message types.

mBrane UDP Library

The mBrane UDP library provides an mBrane Node with the ability to select and use UDP seamlessly as a communication protocol between two nodes. This can be used for control, data and streaming message types.

mBrane PingPong User Library

The PingPong user library was the first implementation of user modules in mBrane and has been left there to illustrate to users by example how to implement their own modules. It shows how to create simple modules and custom message types and how a module is entered into the registration library using the configuration files.

mBrane Performance User Library

The performance user library was created to test the overall performance of mBrane to see if the performance targets were met and monitor these during the further development of mBrane. To see more details on the performance measurements, please refer to the Performance chapter.

Technical Issues

During the development of mBrane a number of issues of a technical nature were encountered. Some of these were resolved, some bypassed as no appropriate and satisfying solution was found and some are still remaining.

Issues Resolved

On Linux, the TCP implementation offered by the OS does not offer significant buffering in case of vast amounts of data being read by a single thread. This was resolved by converting to non-blocking threads and reading into a software defined buffer instead. Also, disabling the NAGLE routine helped on the speed of the data transfers.

The differences in compiler technologies offered a whole set of issues, as the initial version was created for Windows using Microsoft's Visual Studio. GCC does things somewhat differently, some easily fixed using static casts, but others needed more elaborate work-arounds. One example of this was high precision time, which on Windows can be implemented using performance counters, offering time differences down to the nearest 100ns, while the Linux kernel offers more direct, but less accurate time functions, accurate down to the nearest millisecond, but only on some systems.

Finally, on Linux a lot of the atomic operations such as adding or substitution are offered, but only as more or less undocumented GCC features – Google was very helpful in resolving such matters.

Issues Bypassed

One issue that could not be satisfactorily resolved was the use of the __COUNTER__ macro. In both compilers this is supposed to be replaced by 0 on the first encounter, 1 on the next and so on, but the order in which this happens on Linux is not the order in which the Visual Studio does this. As this was used to assign automatic ids to system classes, this had to be redesigned from scratch using another assignment method as no solution to the __COUNTER__ macro issue could be found.

Issues Remaining

The memory handling is still an issue, especially regarding a mysterious double deletion on shutdown. We have tried changing the way memory is being allocated and assigned, but there are still issues remaining. However, as these only manifests themselves on system shutdown they were deemed to be less important for the moment.

Performance

The ultimate goal of the system is to be able to have 100 modules sending 10 messages to each other in 10 milliseconds. The performance measurements presented in this section were designed to measure exactly this.

We initially designed 10 modules sending round robin messages to each other. Module 1 sent out a message which module 2 was subscribed to. It in return sent out a message which module 3 was subscribed to, and so on. We ran two tests, one called the Single Node Test where all modules reside on a single node and a second test called Dual Node Test where every other module in the chain resides on a different node.

The results can be seen in Table 1.

Test	Number of messages	Average transmission time	Standard Deviation
Single Node Test	1,000,000	21.37482 usec	0.54616
Dual Node Test	1,000,000	87.58788 usec	5.497727

Table 1: Performance tests on the two node configurations

From this we can immediately see (unsurprisingly) that message transmission time is much faster when transmitted locally instead of across the network. We can also see that the network introduces more of a spread in the results as more uncertainty is introduced.

The interpretation of these results is offered in Table 2.

Test	Messages per 10 ms	Target per 10 ms	Result vs target
Single Node Test	467.84	100	+367%
Dual Node Test	114.17	100	+14%

Table 2: Interpretation of the performance results

As we can see, for the single node test we are already way over the target performance goal and for the dual node test we have beat this with 14%.

These results may be affected by the fact that only two nodes were used in the test. Scaling up to 8 nodes should provide better CPU performance, but may reveal further limitations in the networking.

mBrane Roadmap

The mBrane development does not end with version 1.0. Several features and issue fixes are planned, including some already mentioned in Technical Issues.