

Purpose of this example

The purpose of this example is to provide a template for distributed systems.

The system described is not exactly distributed, but actually comes in two parts, modelled in 2 different projects via two different systems: a server and a client, where the client sends requests for simple computations to the server and the server sends back the result.

The server and the client must actually run on the same host, but different languages are used: the client is generated in C++ and the server in C.

This project contains the overall description for the whole system and is only here for documentation. It also includes the references for the shared parts, i.e the definition of the messages exchanged by the 2 systems. The communication between the server and the client is done via a socket.

Code generation

For each project, 3 generation profiles are defined:

- One named *<platform>Target*, allowing to generate the final code for the target;
- One named *<platform>DebugWith<other part>*, allowing to debug the part, supposing the other part is there;
- One named *<platform>DebugNo<other part>*, allowing to debug the part alone, meaning there will actually be no communication done with anything.

For the first two, the server must be started first, either in a terminal for the target profile, or in the model debugger for the other one. Then the client can be started, again either in a terminal for the target profile, or in the model debugger for the other one. The client automatically sends requests to the server, no action is required from the user.

Debugging the parts

For the standalone debug profiles (*<platform>DebugNo<other part>*), each part can be debugged by itself. The messages that are supposed to come from the other part must then be sent manually via the debugger GUI ("Send" button). In this case, messages sent and received by the debugged part will actually appear as messages in a MSC trace.

When debugging with a profile for which the 2 parts are required (*<platform>DebugWith<other part>* or *<platform>Target*), messages will actually be encoded and sent through a socket as explained above. So the messages will not appear as messages in a MSC trace. Instead, the trace will show a take and a give on a semaphore used to protect the socket against concurrent access. Remember to always run the server before the client. You will also have to stop the server "manually" at the end of the session, either with control+c if the server is run in a terminal, or with the 'stop' button if the server is run in the debugger.