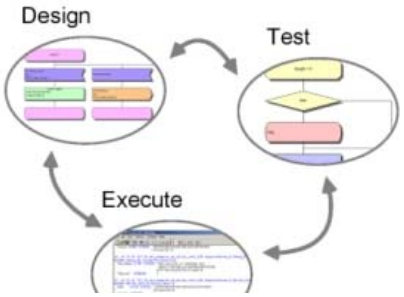
REMUNE
Advanced REal-time Multi-media and Networking Execution platform and Development Environment
IMS Project #01007

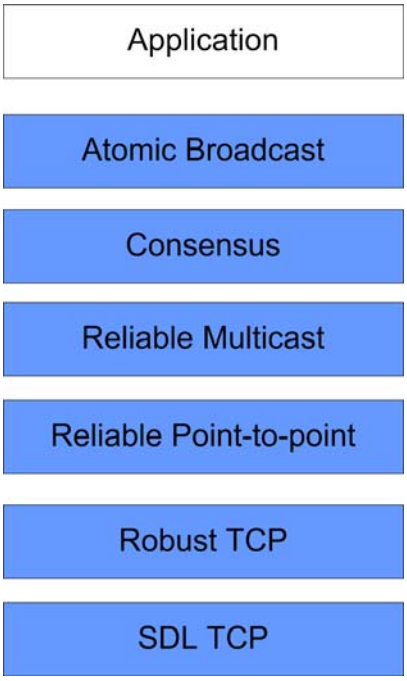
Context and Task
This work was performed in the context of the IST/IMS project **REMUNE**. REMUNE stands for Advanced Real-Time Multi-media and Networking Execution Platform and Development Environment and is a joint effort with several industrial partners. Our task was to specify and develop a group communication protocol stack in SDL, the **Specification and Definition Language**.



Motivation for Group Communication
Group communication provides primitives to communicate with a group of processes. These primitives implement higher-level abstractions that facilitate the development of complex, fault-tolerant applications. As an example, database replication can be built using group communication primitives.

Implementation
Group communication is an instrumental building block for the development of replicated fault-tolerant applications. The implementation is based on the REMUNE platform and uses the subset of SDL defined in the context of REMUNE. The group communication protocol stack provides reliable and totally ordered message delivery to all the participants in a particular group. It consists of a layered structure that composes multiple microprotocols that together provide this quality of service in message delivery. To achieve high modularity, each layer n should rely only on layers n-1 or lower (lower neighbouring layers) to provide the services it needs. Its services should only be used by layer n+1 or higher (upper neighbouring layers). The protocol is event driven. Events map directly onto the signals defined in SDL. Whenever an event or message is received by one of the blocks, an event is triggered and a handler will take some appropriate action. In the following, we briefly describe each layer of the GC protocol stack separately, starting from the bottom layer.

At the bottom of the GC protocol stack is the Unreliable Transport layer which represents the transport medium for the messages that are sent. No assumptions are made about the reliability of this medium: messages can be lost, duplicated or arrive out of order. The probability of delivery of a message is some value greater than zero that is not known in advance. This layer is provided by the REMUNE platform and will not be implemented by EPFL. Moreover, the REMUNE platform provides a TCP implementation, which can be used to built a Reliable Transport layer on top. For this purpose, we extend the TCP implementation to achieve Robust TCP, which handles temporary connection breaks. The Reliable Point-to-Point layer introduces the abstraction of a reliable peer-to-peer



communication. This layer uses the Reliable Transport layer to send messages reliably and then guarantees that the other party will eventually receive the messages that were sent. This layer can be imagined as similar to TCP. The main difference is that TCP is stream oriented (i.e. data is seen as a continuous stream) whereas the Reliable Point-to-Point layer is message oriented (i.e. data is sent and received in chunks, called messages, that will not necessarily arrive in order). The Reliable Point-to-Point layer is used by the Reliable Multicast layer or directly by the application running on top of the GC protocol stack.

The Reliable Multicast layer introduces the abstraction of reliable multicast. A process can send a message to a group of processes with the guarantee that either all the recipients will receive the message, or none at all. The latter case occurs if the sender fails while it is in the process of broadcasting the message. Reliable multicast thus ensures atomicity (i.e., the all-or-nothing property) on message delivery. The Reliable Multicast layer uses the services of the Reliable Point-to-Point layer to send the messages to the group of processes.

The Consensus block provides the services for distributed consensus among a set of processes. These services are used whenever a group of processes needs to agree on a particular value, such as the order of message delivery, as described below. More specifically, all processes start with their own initial value, and after running Consensus, they will all decide on the same resulting value, which is part of the set of initial values. The Consensus layer uses services from the Reliable Multicast block to send and receive messages from the processes taking part in the consensus.

Finally, the Atomic Multicast layer ensures total order in the delivery of messages on the group members. In other words, all the participants in the group deliver the messages in the same order to the application. Atomic Multicast uses Consensus to agree on a common order of message delivery as well as Reliable Multicast. It provides a well-defined API to the application. The implementation provided by EPFL is, to our knowledge, a first-of-a-kind, prototype implementation. It will provide results that can be used for publications and should allow to gain new insight on the composition of Group Communication protocol stacks in the context of finite-state-machine-based specification languages and their corresponding development platforms.

Publications

R. Ekwall and A. Schiper. Solving Atomic Broadcast with Indirect Consensus. In *2006 IEEE International Conference on Dependable Systems and Networks (DSN 2006)*, 2006.
[**Details** | **Full Text**]

R. Ekwall and A. Schiper. Replication: Understanding the Advantage of Atomic Broadcast over Quorum Systems. *Journal of Universal Computer Science*, 11(5):703-711, 2005.
[**Details** | **Full Text**]

A. Kupsys and R. Ekwall. Architectural Issues of JMS Compliant Group Communication. In *4th IEEE International Symposium on Network Computing and Applications (IEEE NCA 2005)*, 2005.
[**Details** | **Full Text**]

R. Ekwall, A. Schiper, and P. Urbán. Token-based Atomic Broadcast using Unreliable Failure Detectors. In *Proceedings of the 23rd Symposium on Reliable Distributed Systems (SRDS 2004)*, 2004.
[**Details** | **Full Text**]

R. Ekwall, S. Pleisch, and A. Schiper. Implementing Group Communication Protocols using SDL. In *Proceedings of the International IMS Forum 2004*, pages 333-340, 2004.
[**Details**]

R. Ekwall, S. Mena, S. Pleisch, and A. Schiper. Towards Flexible Finite-State-Machine-Based Protocol Composition. In *International Symposium on Network Computing and Applications (IEEE NCA04)*, 2004.
[**Details** | **Full Text**]

R. Ekwall, P. Urbán, and A. Schiper. Robust TCP Connections for Fault Tolerant Computing. *Journal of Information Science and Engineering*, 19(3):503-516, 2003.
[**Details** | **Full Text**]

R. Ekwall, P. Urbán, and A. Schiper. Robust TCP Connections for Fault Tolerant Computing. In *Proc. 9th International Conference on Parallel and Distributed Systems (ICPADS)*, 2002.
[**Details** | **Full Text**]