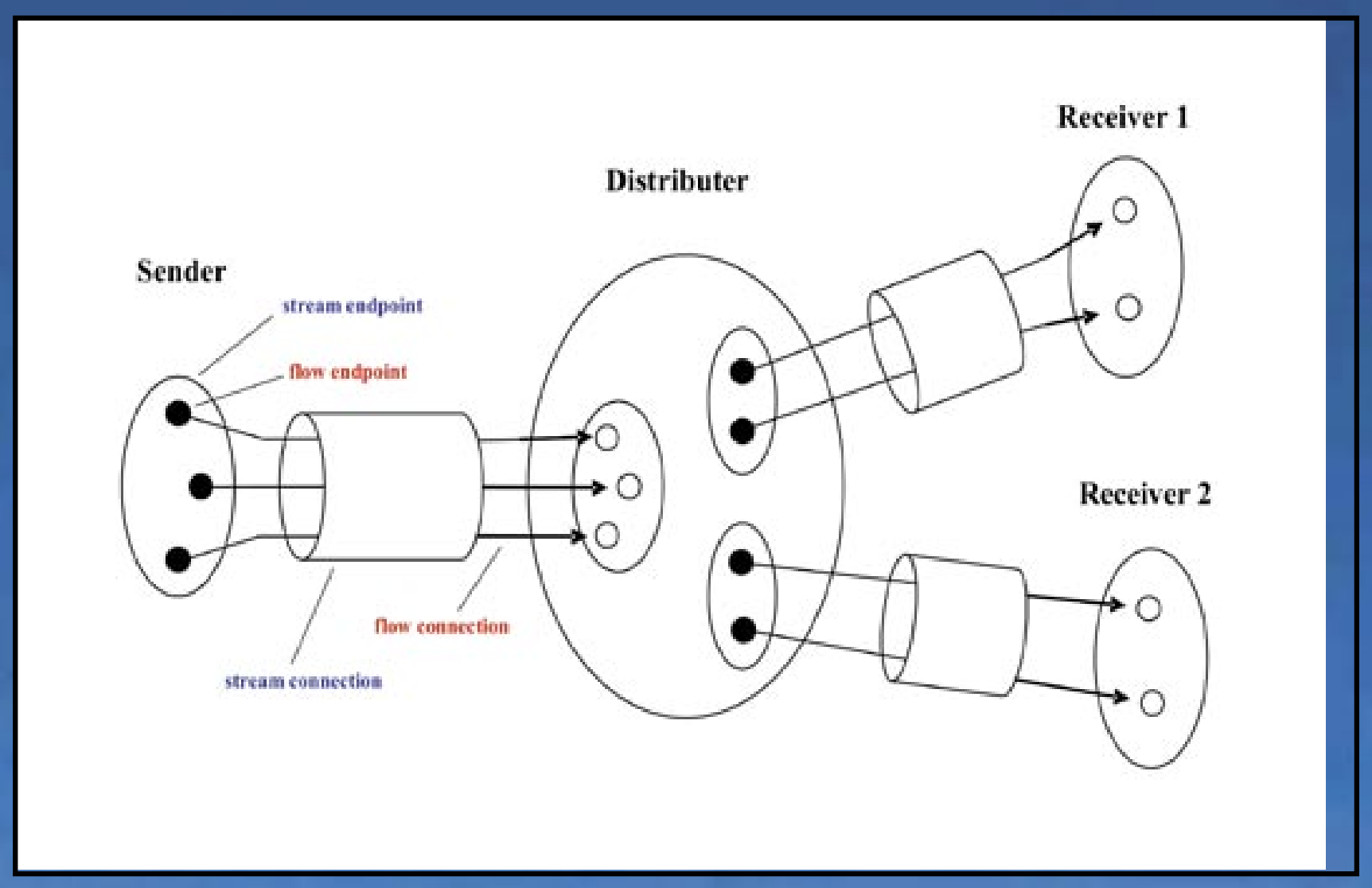
BULK DATA TRANSFER DISTRIBUTER: A HIGH PERFORMANCE MULTICAST MODEL IN ALMA ACS

R. Cirami[1], P. Di Marcantonio[1], G. Chiozzi[2], B. Jeram[2]

[1]INAF- Osservatorio Astronomico di Trieste, via G. B. Tiepolo 11, I-34131 Trieste, Italy
[2]European Southern Observatory, Karl-Schwarzschildstr. 2, D-85748 Garching, Germany

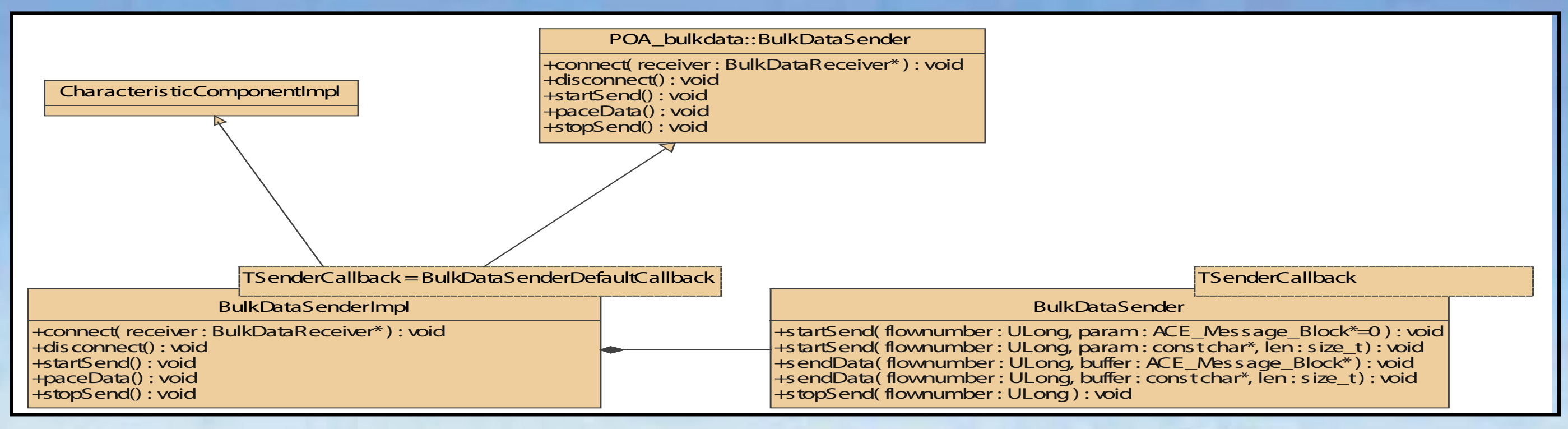
The ALMA astronomical interferometer will consist of at least 50 12-m antennas operating at millimeter wavelength. The whole software infrastructure for ALMA is based on ACS, which is a set of application frameworks built on top of CORBA. To cope with the very strong requirements for the amount of data that needs to be transported by the software communication channels of the ALMA subsystems (a typical output data rate expected from the Correlator is of the order of 64 MB per second) and with the potential CORBA bottleneck due to parameter marshalling/de-marshalling, usage of IIOP protocol, etc., a transfer mechanism based on the ACE/TAO CORBA Audio/Video (A/V) Streaming Service has been developed. The ACS Bulk Data Transfer architecture bypasses the CORBA protocol with an out-of-bound connection for the data streams (transmitting data directly in TCP format), using at the same time CORBA for handshaking and leveraging the benefits of ACS middleware. Such a mechanism has proven to be capable of high performances, with a *measured efficiency comparable to that of a raw socket connection*.
Besides a point-to-point communication model, the ACS Bulk Data Transfer provides a multicast model. Since the TCP protocol does not support multicasting and all the data must be correctly delivered to all ALMA subsystems, a distributer mechanism has been developed. The ACS Bulk Data Distributer mimics a multicast behaviour managing data dispatching to all receivers willing to get data from the same sender.

Basic terminology



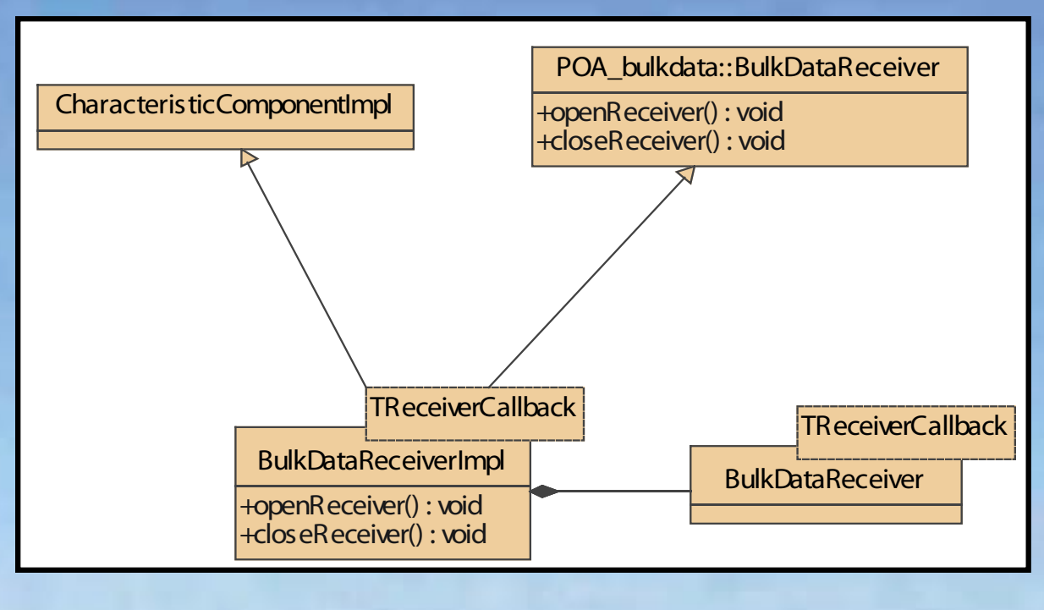
The CORBA A/V Streaming Service specification defines a flow as a continuous sequence of frames in a clearly identified direction. A stream is defined as a set of flows between two objects, and is terminated by a stream endpoint. A stream endpoint can have multiple flow endpoints, acting as a source or as a sink of data.
The ACS Bulk Data Transfer provides C++ classes and ACS Characteristic Components which implement the features described. In the peer-to-peer communication model, it allows to connect a ***sender*** component (the producer of data) with a ***receiver*** component (the consumer), creating dynamically as many flows as required. In the multicast model a sender component sends data to the ***distributer*** component which, in turn, delivers the data to one or more connected receiver components.

The ACS Sender Component


CharacteristicComponentImpl
POA_bulkdata::BulkDataSender
+connect(receiver : BulkDataReceiver*) : void
+disconnect() : void
+startSend() : void
+paceData() : void
+stopSend() : void
TSenderCallback = BulkDataSenderDefaultCallback
BulkDataSenderImpl
+connect(receiver : BulkDataReceiver*) : void
+disconnect() : void
+startSend() : void
+paceData() : void
+stopSend() : void
TSenderCallback
BulkDataSender
+startSend(flownumber : ULong, param : ACE_Message_Block*=0) : void
+startSend(flownumber : ULong, param : const char*, len : size_t) : void
+sendData(flownumber : ULong, buffer : ACE_Message_Block*) : void
+sendData(flownumber : ULong, buffer : const char*, len : size_t) : void
+stopSend(flownumber : ULong) : void

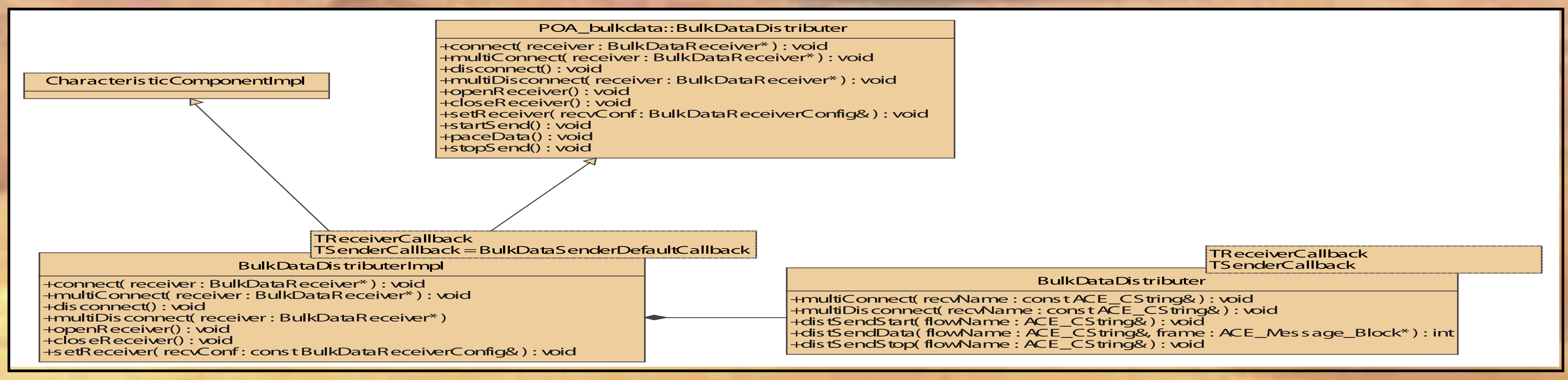
The ACS Characteristic Component relative to the Sender is implemented as a C++ template class. The template parameter is a callback which can be used for sending asynchronous data. This callback class provides methods for sending data at predetermined user-configurable time intervals. For the Distributer such asynchronous mechanism has not been implemented yet, and only the synchronous mechanism is provided.
As shown in the figure, the *BulkDataSenderImpl<TSenderCallback>* template class realizes a component providing the implementation for the *BulkDataSender* IDL interface (represented in the diagram by the CORBA-generated *POA_bulkdata::BulkDataSender* skeleton class). *BulkDataSenderImpl<TSenderCallback>* provides a concrete implementation for the ***connect()*** and ***disconnect()*** methods using the contained C++ wrapper class (*BulkDataSender<TSenderCallback>*). The *connect()* method is responsible for the connection establishment with the Distributer component, passed as a parameter.

The ACS Receiver Component



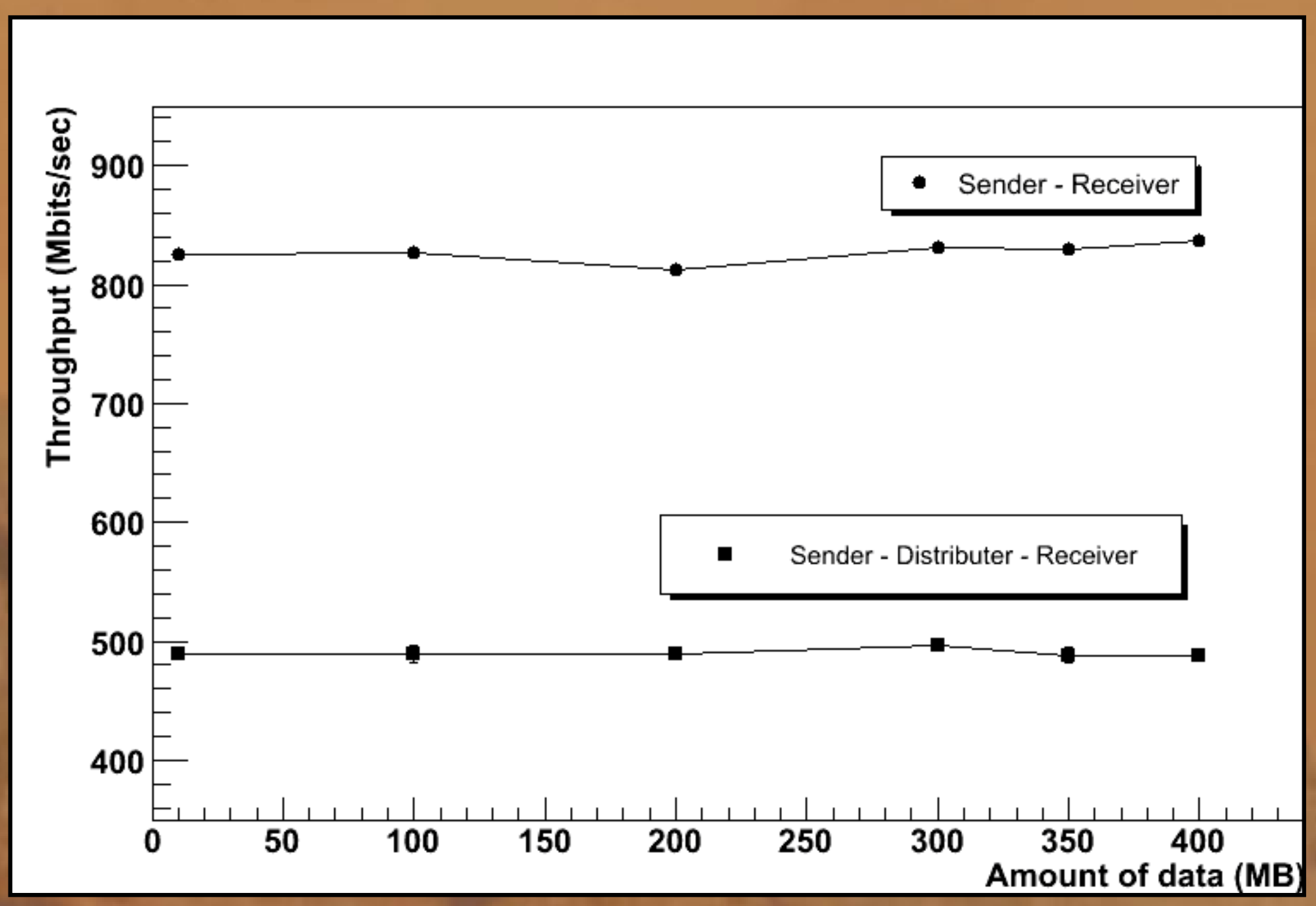
The ACS Characteristic Component relative to the Receiver is implemented also as a template class. The template parameter in this case is a callback class, which has to be provided by the user and must be used to actually retrieve and manage the received parameters and data stream.
The figure shows the class diagram for a receiver component. Two methods are implemented in this case: ***openReceiver()***, which reads e.g. from the Configuration Database all the connection parameters - as in the Sender case - and creates the required flow endpoints accordingly, and ***closeReceiver()***, used to close the connection.

The ACS Distributer Component

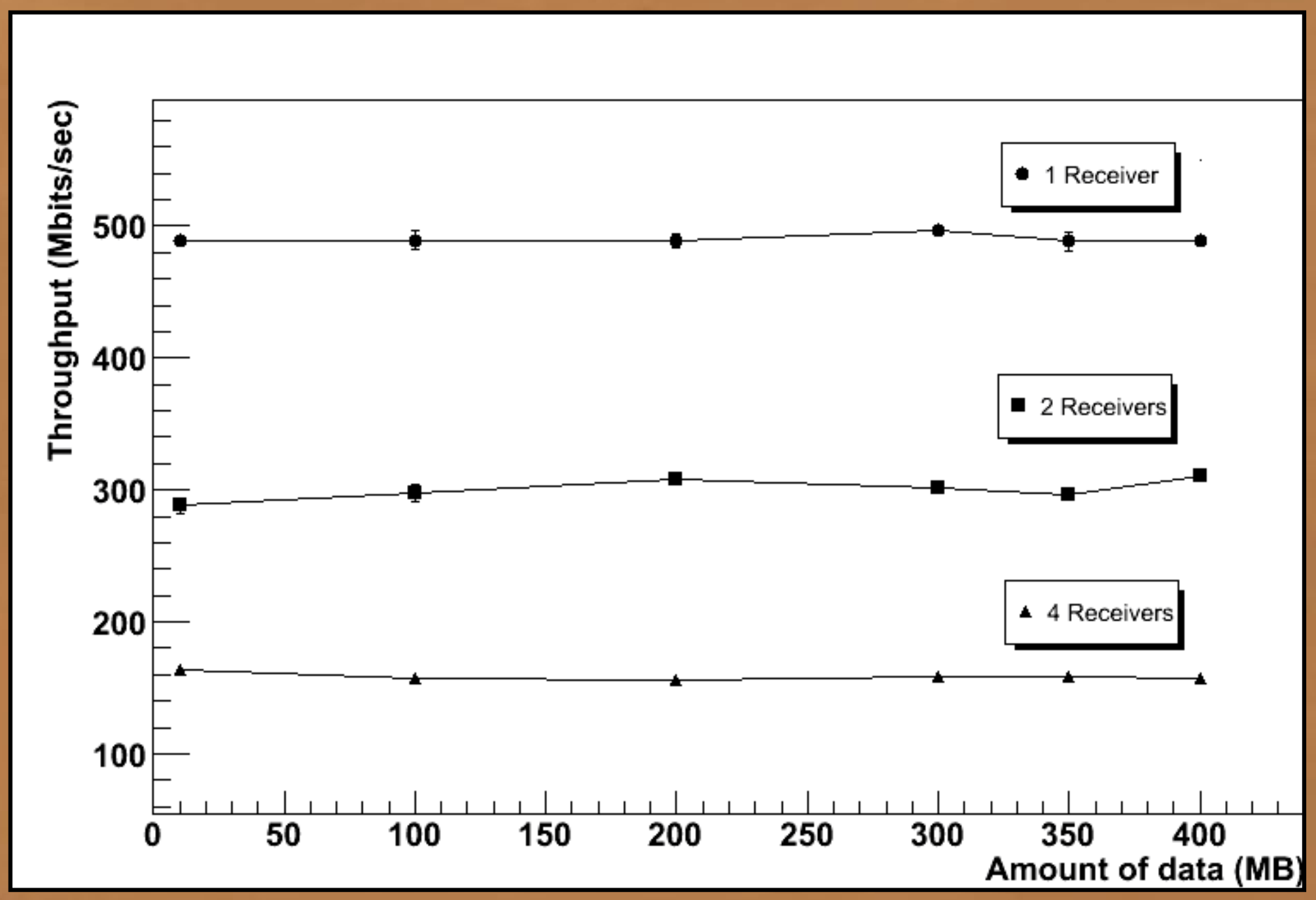

CharacteristicComponentImpl
POA_bulkdata::BulkDataDistributer
+connect(receiver : BulkDataReceiver*) : void
+multiConnect(receiver : BulkDataReceiver*) : void
+disconnect() : void
+multiDisconnect(receiver : BulkDataReceiver*) : void
+openReceiver() : void
+closeReceiver() : void
+setReceiver(recvConf : BulkDataReceiverConfig&) : void
+startSend() : void
+paceData() : void
+stopSend() : void
TReceiverCallback
TSenderCallback = BulkDataSenderDefaultCallback
BulkDataDistributerImpl
+connect(receiver : BulkDataReceiver*) : void
+multiConnect(receiver : BulkDataReceiver*) : void
+disconnect() : void
+multiDisconnect(receiver : BulkDataReceiver*)
+openReceiver() : void
+closeReceiver() : void
+setReceiver(recvConf : const BulkDataReceiverConfig&) : void
TReceiverCallback
TSenderCallback
BulkDataDistributer
+multiConnect(recvName : const ACE_CString&) : void
+multiDisconnect(recvName : const ACE_CString&) : void
+distSendStart(flowName : ACE_CString&) : void
+distSendData(flowName : ACE_CString&, frame : ACE_Message_Block*) : int
+distSendStop(flowName : ACE_CString&) : void

The ACS Bulk Data Distributer acts as a receiver towards the Sender and as a sender towards the Receivers willing to get the dispatched data, so the ACS Characteristic Component relative to the Distributer inherits both from the sender and receiver interfaces. It is implemented as a template class with two template parameters: the sender callback, which takes the default value *BulkDataSenderDefaultCallback*, and the distributer receiver callback (*BulkDataDistributerCb*), used to receive the data from the Sender. When the Distributer receives parameters or data from the Sender, this callback manages and dispatches the received data to all connected Receivers (registered in an internal hash table), taking care of the possible receiver timeout and availability. At present this forwarding mechanism is implemented in a sequential way, i. e. the internal hash table which holds the information of the connected Receivers is read sequentially. This mechanism will be improved in the next Bulk Data releases.
The figure shows the class diagram for a Distributer component. The two methods ***multiConnect()*** and ***multiDisconnect()*** allow the Receivers to register and de-register from the Distributer.

Distributer Performances



In order to evaluate the performance of the ACS Bulk Data Distributer we developed and implemented a Sender, a Distributer and one (or more) Receiver(s). The aim of the experiment was to measure the throughput, i.e., the number of bits per second, sending data of different size from a Sender to one or more Receivers passing through a Distributer. In all the performance tests the data were sent on one flow.
For the measurements we used two Compaq PCs (one CPU P4, 3.0 GHz) equipped with 1GB RAM and 80 GB HD connected via a 1Gbit Ethernet network. Both PCs were isolated from the Institute LAN to avoid external network loads, but to be more realistic the two PC were connected using two Gbit switches. Scientific Linux 4.1 operating system and ACS 5.0.2 were installed on both machines.



The figures above represent the results of our performance tests. Every point is an average of 100 samples. The error bars represent the error on the mean. In all the tests, the Sender and the Distributer were located on one PC, whereas the Receiver(s) on the other one.
The figure on the left shows that the *throughput*, i.e., the number of bits per second, between a Sender and a single Receiver is more the 800 Mbits/sec, and that the Distributer introduces a penalty of performances of the order of 300 Mbits/sec. This is an expected result, since the Distributer introduces a further data processing step in the data flow. Data must be received, read and forwarded to the Receiver. The Distributer then has to wait until the Receiver has fully consumed the data in order to be synchronized. This could be considered the best performance achievable fulfilling the synchronization requirement.
The figure on the right shows the throughput when one or more Receivers are connected to the Distributer. It can be seen that for every connected Receiver the performance penalty increases and the overall throughput for four connected Receivers is of the order of 180 Mbits/sec. This is due to the sequential way of sending the data. This mechanism will be improved in the next Bulk Data releases using e.g. a thread pool, the Half-Sync/Half-Async pattern or changing the underlying protocol to one that support multicasting.

References

• P. Di Marcantonio, R. Cirami, B. Jeram, G. Chiozzi, "Transmitting huge amounts of data: design, implementation and performance of the Bulk Data Transfer mechanism in ALMA ACS", ICALEPCS 2005, Geneva, Switzerland, October 2005
• ACS Web page, http://www.eso.org/projects/alma/develop/acs/
• OMG Audio/Video Streams Specification, v.1.0, http://www.omg.org/cgi-bin/doc?formal/2000-01-03.
• N. Surendran et al., "The Design and Performance of a CORBA Audio/Video Streaming Service", Proceedings of HICSS-32 vol. 8, Hawaii 1999, 8043.