One-Page PEPt

Harold Carr
Sun Microsystems
harold.carr@sun.com

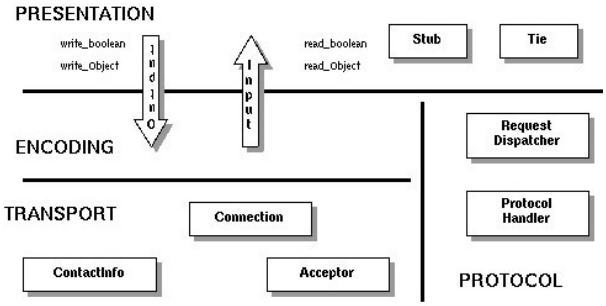
Abstract

PEPt is an architecture for implementing RPC systems. Although RPC systems seem quite varied they actually share the same fundamental building blocks: *presentation, encoding, protocol* and *transport (PEPt).* Presentation encompasses the data types and APIs available to the programmer. Encoding is the representation of those types on the wire. Protocol frames the encoded data to denote the boundaries and intent of the message. Transport moves the encoding + protocol from one location to another. The PEPt architecture *enables a single programming model to adaptively change encodings, protocols and transports.*

1. Benefits

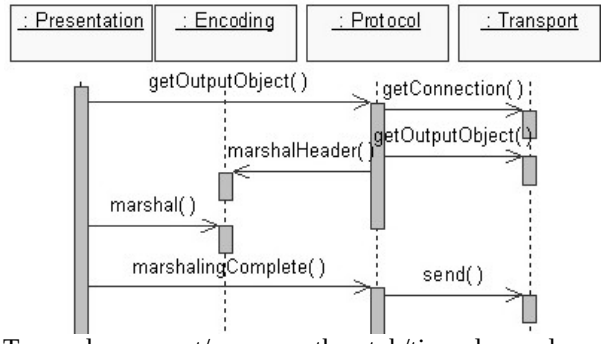
PEPt provides a simple but comprehensive framework in which to understand, implement, reuse, evolve and maintain finer-grained details of distributed communications systems. Most importantly, PEPt allows an RPC system to adaptively change its subsystems.

2. PEPt Architecture



We illustrate PEPt by showing the interaction of its main blocks in the lifecycle of an RPC request/response. We assume programmers interact with an RPC system via Presentation block stubs and ties.

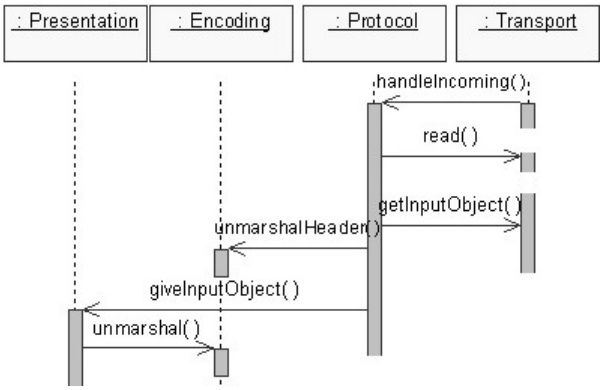
3. Send



To send a request/response the stub/tie only needs an Encoding OutputObject obtained from Protocol. On the client side, Protocol contains a list of encoding/protocol/transport combinations (EPTC) which can service the call. Transport ContactInfo of the selected EPTC is a factory for Encoding OutputObjects, Transport Connections and Protocol-specific RequestDispatchers. Two key points: Transport is the plug-in point for alternate EPTCs, and *presentation block stubs and skeletons should be independent of the EPTC.*

Adaptive Send: If the selected EPTC supports connection multiplexing, getConnection would search a ConnectionCache before creating a new Connection. The EPTC may support fragmentation requiring interaction between Encoding and Transport. After Send, a client may block on a read, suspend while another thread reads a multiplexed connection, or the client thread may continue servicing the server side of the request. These and other details are adaptively changed depending on the selected EPTC.

4. Receive



When a request/reply arrives on Transport, ProtocolHandler reads enough bits to determine the EPTC. Transport ContactInfo/Acceptor then acts as a factory for appropriate InputObjects and RequestDispatchers. Protocol directs the request to the appropriate stub/tie.

5. Conclusions and Future Work

PEPt provides a unified architecture for implementing RPC systems. PEPt handles addressing, colocation, retries, error handling, connection multiplexing, thread pooling, fragmentation, transactions, security, plugging in and adaptively changing EPTCs (e.g., CDR/IIOP, SOAP/HTML). PEPt has been used in a commercial CORBA system and to prototype a SOAP/HTML system. In the future we plan to show how PEPt applies to messaging systems.