Architectural Benefits of Ice

The Ice architecture provides a number of benefits to application developers:

- Object-oriented semantics
  Ice fully preserves the object-oriented paradigm "across the wire." All operation invocations use late binding, so the implementation of an operation is chosen depending on the actual run-time (not static) type of an object.

- Support for synchronous and asynchronous messaging
  Ice provides both synchronous and asynchronous operation invocation and dispatch, as well as publish-subscribe messaging via IceStorm. This allows you to choose a communication model according to the needs of your application instead of having to shoe-horn the application to fit a single model.

- Support for multiple interfaces
  With facets, objects can provide multiple, unrelated interfaces while retaining a single object identity across these interfaces. This provides great flexibility, particularly as an application evolves but needs to remain compatible with older, already deployed clients.

- Machine independence
  Clients and servers are shielded form idiosyncrasies of the underlying machine architecture. Issues such as byte ordering and padding are hidden from application code.

- Language independence
  Client and server can be developed independently and in different programming languages. The Slice definition used by both client and server establishes the interface contract between them and is the only thing they need to agree on.

- Implementation independence
  Clients are unaware of how servers implement their objects. This means that the implementation of a server can be changed after clients are deployed, for example, to use a different persistence mechanism or even a different programming language.

- Operating system independence
  The Ice APIs are fully portable, so the same source code compiles and runs under both Windows and Unix.

- Threading support
  The Ice run time is fully threaded and APIs are thread-safe. No effort (beyond synchronizing access to shared data) is required on part of the application developer to develop threaded, high-performance clients and servers.

- Transport independence
  Ice currently offers both TCP/IP and UDP as transport protocols. Neither client nor server code are aware of the underlying transport. (The desired transport can be chosen by a configuration parameter.)

- Location and server transparency
  The Ice run time takes care of locating objects and managing the underlying transport mechanism, such as opening and closing connections. Interactions between client and server appear connection-less. Via IceGrid, you can arrange for servers to be started on demand if they are not running at the time a client invokes an operation. Servers can be migrated to different physical addresses without breaking proxies held by clients, and clients are completely unaware how object implementations are distributed over server processes.

- Security
  Communications between client and server can be fully secured with strong encryption over SSL, so applications can use unsecured public networks to communicate securely. Via Glacier2, you can implement secure forwarding of requests through a firewall, with full support for callbacks.

- Built-in persistence
  With Freeze, creating persistent object implementations becomes trivial. Ice comes with built-in support for Berkeley DB, which is a high-performance database.

- Source code availability
  The source code for Ice is available. While it is not necessary to have access to the source code to use the platform, it allows you to see how things are implemented or port the code to a new operating system.

Overall, Ice provides a state-of-the art development and deployment environment for distributed computing that is more complete than any other platform we are aware of.

See Also

- Ice Architecture
- Ice Services