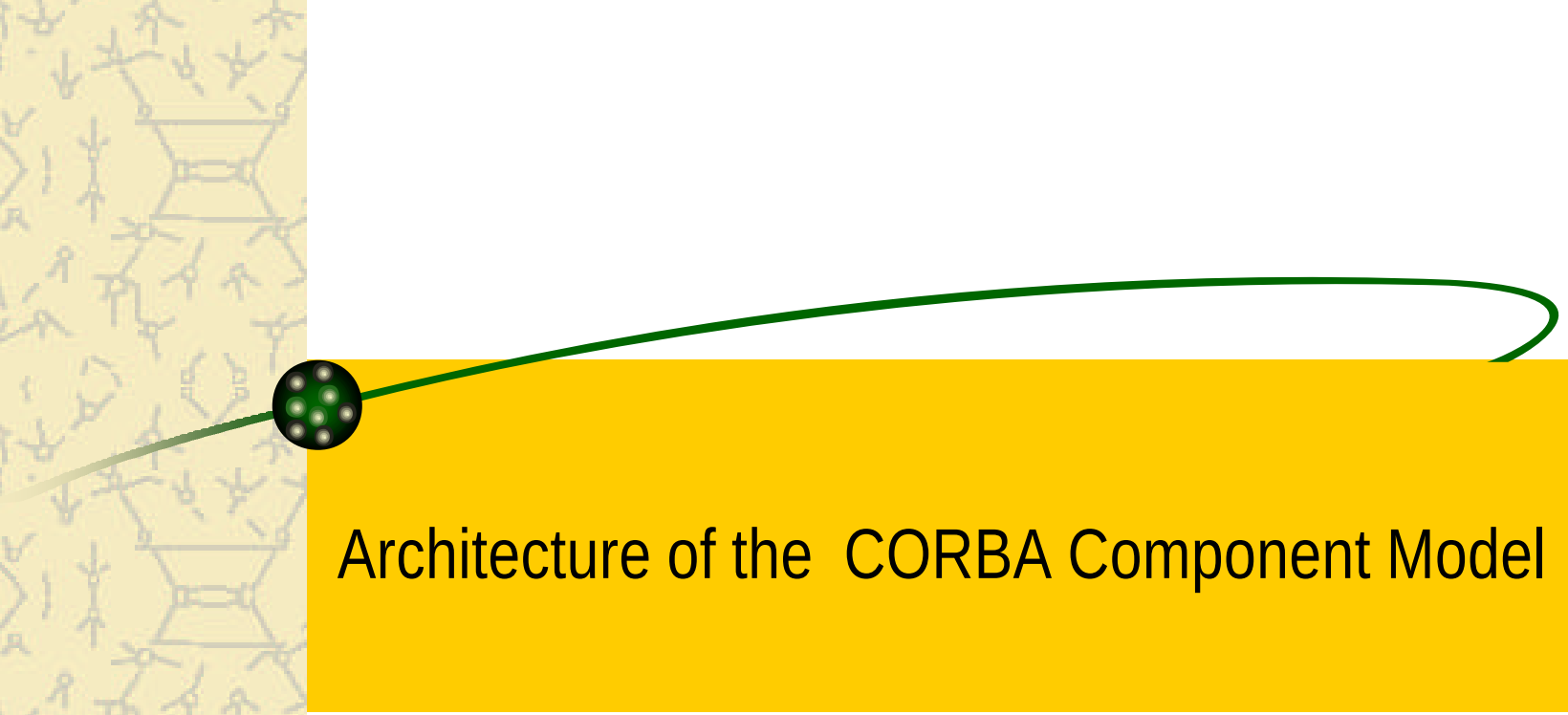


Architecture of the CORBA Component Model

Comparing CORBA and .NET: IIOP and SOAP

Interoperability

... the ability for a client on ORB A to invoke an OMG-IDL defined operation on an object on ORB B, where ORB A and ORB B are independently developed. [Interoperability RFP, 93-09-15]

Problems:

- Information model and validity of object references
- transport protocol
- Extensions (security, transactions)

Interoperability Domain

- Set of nodes/CORBA installations which can „directly“ exchange operation calls
- Domain Boundaries:
  - different networking technology (Internet vs. X.25)
  - network islands (private networks/firewalls)
  - different ORB protocol
- ORB domains

Interoperability means to overcome domain boundaries.

Inter-ORB-Bridges

- Mediate between ORB domains
- Client in domain A contacts bridge, which contacts domain B
- Different functions:
  - conversion of object references
  - conversion of message formats
  - validation of message authenticity
- Allows to integrate non-CORBA systems
  - COM/CORBA interworking
- Source of interoperability: GIOP

GIOP: General Inter-ORB Protocol

Prerequisites:

- connection-oriented transport
- full-duplex connections
- connection is symmetric w.r.t. shutdown
- transport is reliable
- transport transmits byte streams
- transport informs about connection loss (disorderly release)

GIOP (2)

- GIOP defines
  - Message format: Common Data Representation (CDR)
  - Message types
  - Structure of object references: Interoperable Object Reference (IOR)
- GIOP message format and IOR format are defined in IDL
  - will be transmitted through CDR
- IIOP
  - Transport is TCP
  - IOR format is specialized (IOR profile)

Common Data Representation

- Bi-endian
  - Endianness is typically „native“ for sender (JDK: always big-endian)
- Data are encoded as a sequence of primitive values
  - structures, parameter boundaries are not represented
- Each primitive type has a fixed size
  - char, (wchar), octet, boolean: 1
  - short, unsigned short: 2
  - long, unsigned long, float, enum: 4
  - long long, unsigned long long, double: 8
  - long double: 16

Common Data Representation (2)

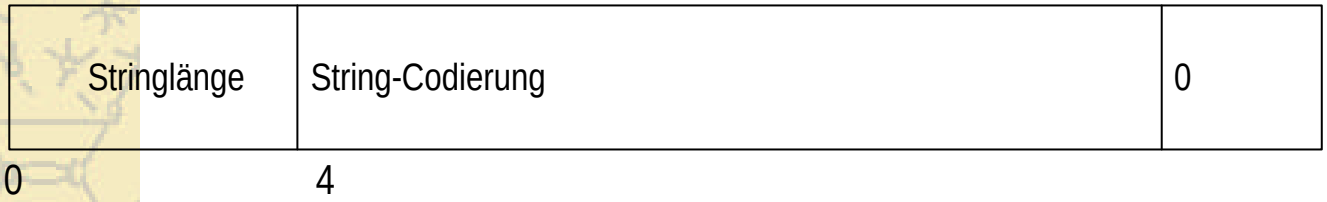
- Alignment: Each primitive value is aligned relative to the message start
  - usually a multiple of its size
  - long double: 8
  - Idea: allow direct copying from transport buffer into C structures

Encoding of Primitive Types

- Integral types: binary, signed types in two‘s complement
- Floating point types: IEEE-754
- octets: „as-is“
- boolean: TRUE==1, FALSE==0
- characters: character set according to „character set negotiation“

Encoding of complex types

- Strings: Length, Contents, null-termination



- Structures: Element for element, padding according to alignment

Encoding of Complex Types (2)

- Unions: Discriminator, union branch
- Array: Element for element
  - multi-dimensional: last index grows fastest
- Sequence: length, elements
- Enum: like unsigned long, enumerators are numbered starting with 0

Encoding of Complex Types (3)

- Any: typecode(type of value), value encoding
- Context: sequence<string>
- Exception: string(repos-id), struct(exception-members)

Interoperable Object References

```
module IOP {
   typedef unsigned long ProfileId;
   struct TaggedProfile {
        ProfileId tag;
        sequence <octet> profile_data;
   };
   struct IOR {
        string type_id;
        sequence <TaggedProfile> profiles;
   };
}
```

- IOR-String: hexified version of an encapsulation containing an IOP::IOR

IOR Profiles

- Define protocol independent contact information
- Defined in IDL
- encoded as an encapsulation

```
module IOP {
   const ProfileId TAG_INTERNET_IOP = 0;
   const ProfileId TAG_MULTIPLE_COMPONENTS = 1;
   const ProfileId TAG_SCCP_IOP = 2;
};
```

IIOP Profile

```
module IIOP {
   struct Version {
        octet major;
        octet minor;
   };
   struct ProfileBody_1_1 {// also used for 1.2
        Version iiop_version;
        string host;
        unsigned short port;
        sequence <octet> object_key;
        sequence <IOP::TaggedComponent> components;
   };
};
```

Tagged Components

- Define protocol-independent contact information
- represented as TAG_MULTIPLE_COMPONENTS or in the IIOP profile

```
module IOP{
typedef unsigned long ComponentId;
struct TaggedComponent {
   ComponentId tag;
   sequence <octet> component_data;
};
typedef sequence<TaggedComponent> TaggedComponentSeq;
};
```

Standard IOR Components

```
module IOP {
    const ComponentId TAG_ORB_TYPE = 0;
    const ComponentId TAG_CODE_SETS = 1;
    const ComponentId TAG_POLICIES = 2;
    const ComponentId TAG_ALTERNATE_IIOP_ADDRESS = 3;
    const ComponentId TAG_ASSOCIATION_OPTIONS = 13;
    const ComponentId TAG_SEC_NAME = 14;
    const ComponentId TAG_SPKM_1_SEC_MECH = 15;
    const ComponentId TAG_SPKM_2_SEC_MECH = 16;
    const ComponentId TAG_KerberosV5_SEC_MECH = 17;
    const ComponentId TAG_CSI_ECMA_Secret_SEC_MECH = 18;
    const ComponentId TAG_CSI_ECMA_Hybrid_SEC_MECH = 19;
    const ComponentId TAG_SSL_SEC_TRANS = 20;
    const ComponentId TAG_JAVA_CODEBASE = 25;
    ...
```

GIOP Messages

- Protocol assumes connection between client and server
- Connection management is invisible to the application (implicit binding)
- different protocol versions:
  - Version 1.0 (CORBA 2.0)
  - Version 1.1 (CORBA 2.1): Fragmentation
  - Version 1.2 (CORBA 2.3): bidirectional communication
- downwards compatible: old clients can talk to new servers

Message Types

Message Type	Initiator	Value	GIOP Version
Request	Client	0	1.0, 1.1, 1.2
Reply	Server	1	1.0, 1.1, 1.2
CancelRequest	Client	2	1.0, 1.1, 1.2
LocateRequest	Client	3	1.0, 1.1, 1.2
LocateReply	Server	4	1.0, 1.1, 1.2
CloseConnection	Server	5	1.0, 1.1, 1.2
MessageError	beide	6	1.0, 1.1, 1.2
Fragment	beide	7	1.1, 1.2

Structure of a GIOP Message

- Basic Structure:
  - GIOP message header
  - message-specific header
  - message-specific body

```
module GIOP {
   struct Version {octet major; octet minor;        };
   enum MsgType_1_1 { Request, Reply, CancelRequest, ... };

   struct MessageHeader_1_1 {
        char magic [4]; // GIOP
        Version GIOP_version;
        octet flags;        // Bit 0: Endianness (0: big)
                            // Bit 1: more fragments
        octet message_type;
        unsigned long message_size;
};
```

Request

```
struct RequestHeader_1_1 {
   IOP::ServiceContextList service_context;
   unsigned long request_id;
   boolean response_expected;
   octet reserved[3];
   sequence <octet> object_key;
   string operation;
   CORBA::OctetSeq requesting_principal;
};
```

- followed by parameters (GIOP 1.2: 8-aligned)

Service Context

- Additional information transmitted from ORB to ORB

```
module IOP {
    typedef unsigned long ServiceId;
    struct ServiceContext {
          ServiceId context_id;
          sequence <octet> context_data;
    };
    typedef sequence <ServiceContext>ServiceContextList;
    const ServiceId TransactionService = 0;
    const ServiceId CodeSets = 1;
    const ServiceId ChainBypassCheck = 2;
    const ServiceId ChainBypassInfo = 3;
    const ServiceId LogicalThreadId = 4;
    const ServiceId BI_DIR_IIOP = 5;
    ...
```

Reply

```
enum ReplyStatusType_1_0 {
   NO_EXCEPTION,
   USER_EXCEPTION,
   SYSTEM_EXCEPTION,
   LOCATION_FORWARD
};
struct ReplyHeader_1_0 {
   IOP::ServiceContextList service_context;
   unsigned long request_id;
   ReplyStatusType_1_0 reply_status;
};
```

Reply (2)

- NO_EXCEPTION: followed by result, out-parameters
- USER_EXCEPTION: according to CDR
- SYSTEM_EXCEPTION:

```
module GIOP {
   struct SystemExceptionReplyBody {
        string exception_id;
        unsigned long minor_code_value;
        unsigned long completion_status;
   };
};
```

- LOCATION_FORWARD: IOR of the new location

Location Forwarding

- Permanent Object References: Client knows host/port
- Idea: Operator can move object
- Two procedures:
  - Client sends normal request, gets LOCATION_FORWARD, retries
  - Client sends LOCATE_REQUEST, gets LOCATE_REPLY

Location Forwarding (2)

```
struct LocateRequestHeader_1_0 {
   // Renamed LocationRequestHeader
   unsigned long request_id;
   sequence <octet> object_key;
};

enum LocateStatusType_1_0 {
   UNKNOWN_OBJECT,
   OBJECT_HERE,
   OBJECT_FORWARD
};
struct LocateReplyHeader_1_0 {
   unsigned long request_id;
   LocateStatusType_1_0 locate_status;
};
```

CancelRequest

```
module GIOP { // IDL
    struct CancelRequestHeader {
        unsigned long request_id;
    };
};
```

CloseConnection

- Client can close connection at any time
- server only if there are no pending replies
- CloseConnection consists only of MessageHeader

MessageError

- sent for erroneous or unrecognized message
- consists only of message header
- sender closes connection after transmission

Fragment

- Allows to split message into smaller chunks

```
module GIOP {
    struct FragmentHeader_1_2 {
        unsigned long request_id;
    };
};
```