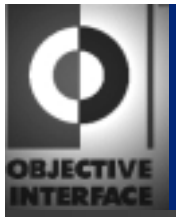
Notification Service Tutorial

bill.beckwith@ois.com

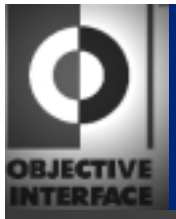
orbos – 2001-04-15

ORBexpress™



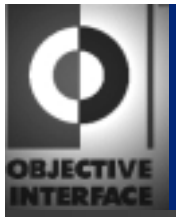
Outline

- Goals
- Relationship to Event Service
- Overview
- Conformance Points
- Addressing of RFP Requirements
- Summary



Goals

- **Goals of Notification Service are**
  - Satisfies real industry requirements
  - Natural evolution from the OMG Event Service
- **Result is based on real-world experience prototyping, developing and deploying related products**



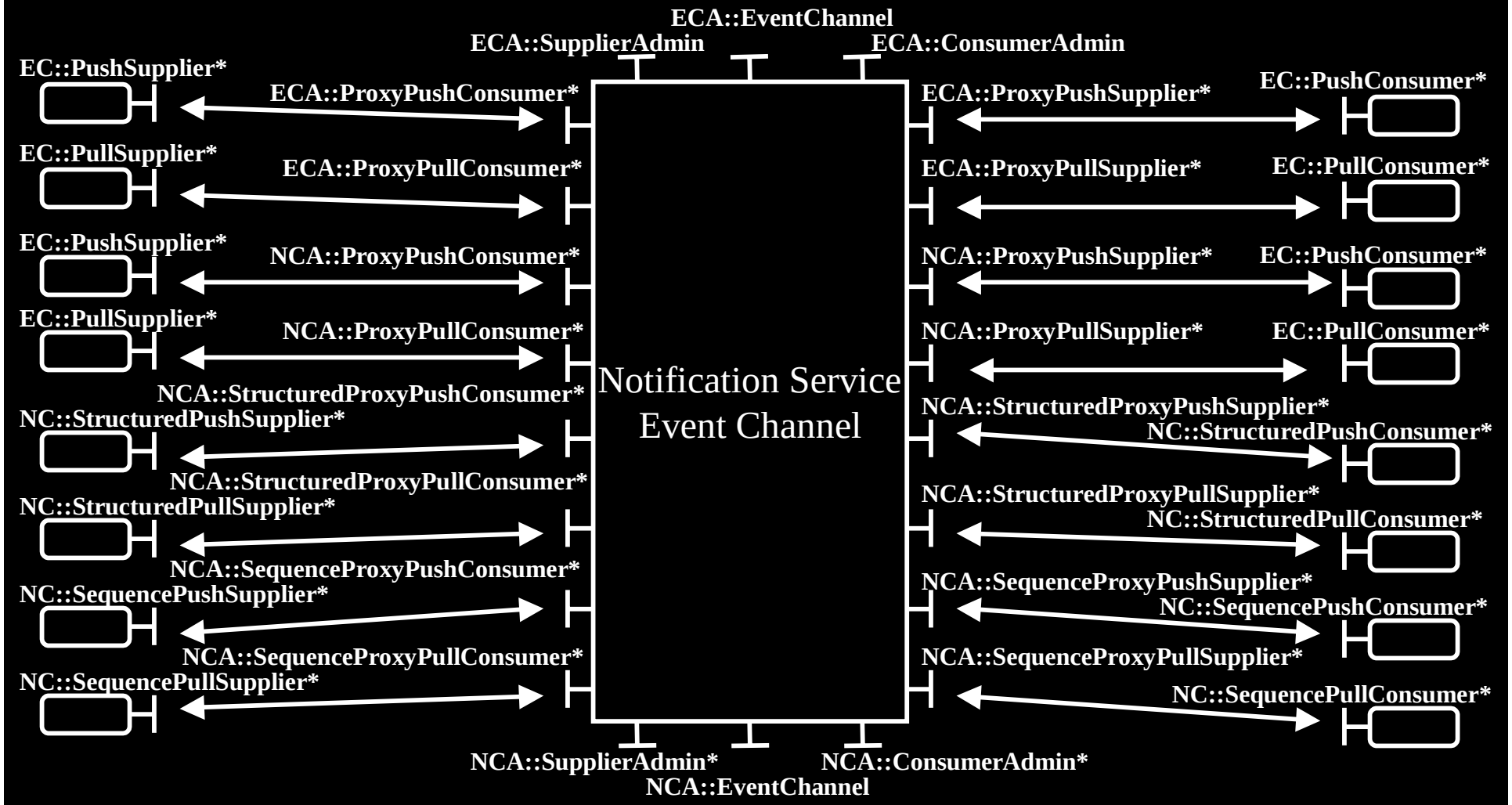
Outline

- Goals
- Relationship to Event Service
- Overview
- Conformance Points
- Addressing of RFP Requirements
- Summary

New Features

- **Notification extends the OMG Event Service, adding the following features**
  - Structured events
  - Event filtering using filter objects
    - Works with Any events, Structured events, and Typed events
  - QoS configurability on per-channel, per-proxy, or per-message basis
  - Event subscription information sharing between channels and clients
  - An optional event type repository
- **Our proposal also includes a mapping of commonly used Telecom event types into Structured events**

Notification Channel Architecture



Relationship to Event Service

- **Complete backward compatibility to OMG Event Service maintained**
  - Due to interface inheritance, Notification Service event channel can be treated exactly like an Event Service event channel
    - This inheritance exists at the Event Channel, Admin, and Proxy levels supporting backward compatibility and gradual migration to Notification
  - Event Service client APs can connect either to Event Service or Notification Service style proxies
    - Only Notification Service style proxies support filtering and QoS configurability defined by Notification

Outline

- Goals
- Relationship to Event Service
- Overview
- Conformance Points
- Addressing of RFP Requirements
- Summary

Native Event Formats

- **Three native forms of event transmission**
  - Anys, Structured Events, Sequences of Structured Events
  - New client interfaces defined for Structured Events and Sequences of Structured Events
    - Clients using Any events use OMG Event Service client interfaces
  - Different Notification Service style proxy interfaces defined for each style of client
    - Client AP developers can select the form of events they wish to deal with by implementing appropriate client interfaces and connecting to appropriate proxies

Native Event Formats (cont.)

- **Consumer APs decision of which style of event to deal with completely separate from Supplier APs decision**
  - Channel performs appropriate translation when consumer and supplier use different style of events
    - Any -> Structured Event: Channel places event into remainder_of_body portion of Structured Event, sets event_type in header to %ANY
    - Structured Event -> Any: Channel copies Structured Event into an Any, setting the Typecode field appropriately
    - Sequence -> singleton event: Channel delivers each event in the sequence individually, performing translation if consumer deals with Anys
    - Singleton events -> Sequence: Channel delivers multiple events in a sequence, performing translation if supplier deals with Anys

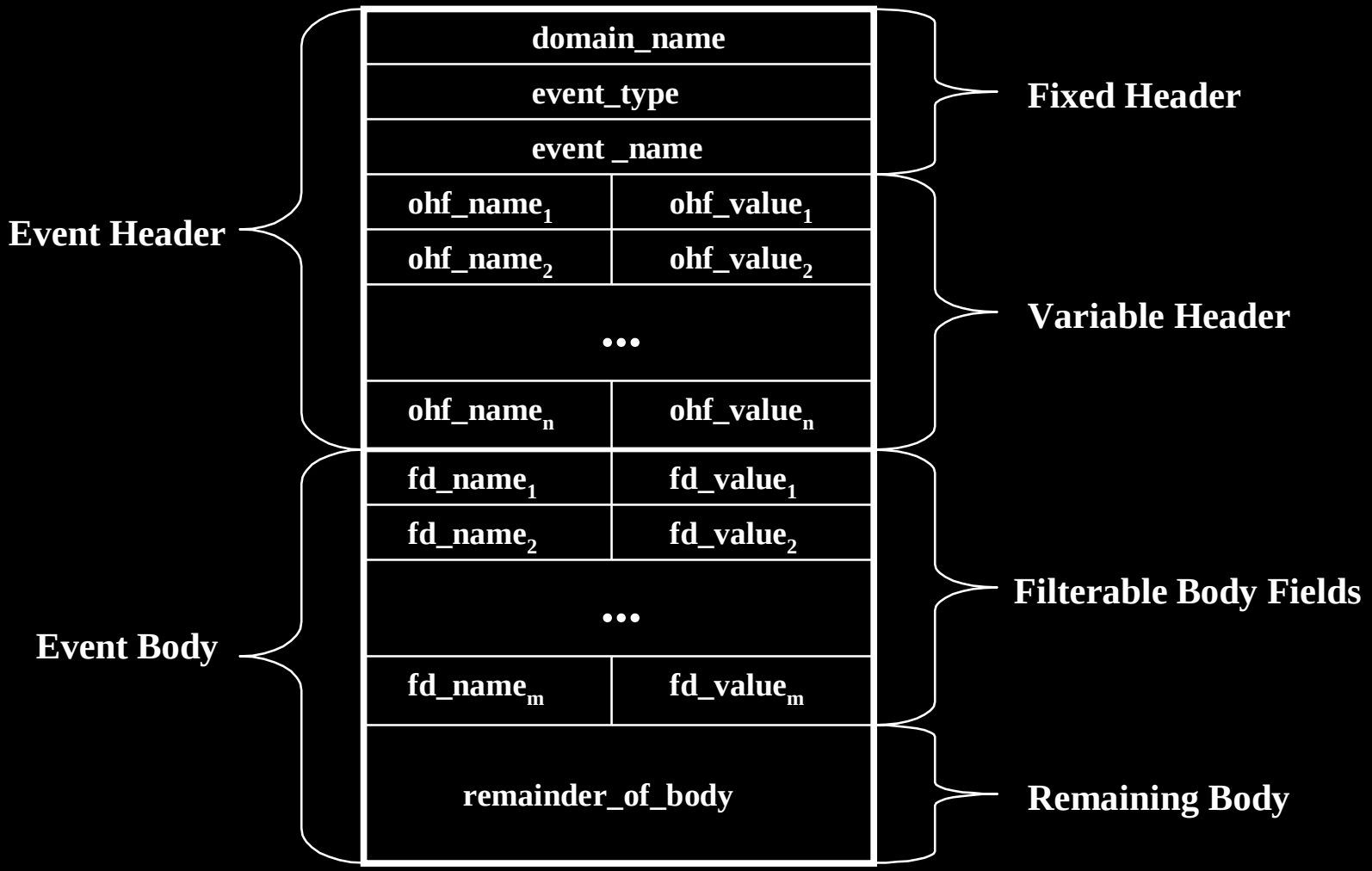
New Basic Architectural Features

- **Channel can support any number of Notification Service style ConsumerAdmin or SupplierAdmin instances**
  - Each instance logically manages the group of proxies it creates
  - Filter objects can be associated with an Admin object, and are applied to all proxies created by that Admin
- **Ease of channel administration greatly improved**
  - Explicit factory interfaces defined for every type of instantiable object within the service
  - Factories associate unique IDs with all objects they create, and can respond to queries to obtain instance references by ID
  - Objects within the channel maintain back references to their

Structured Events

- **New native event type defined**
  - To ease mapping of commonly used event types defined using other technologies (e.g., CMIP, SNMP) into CORBA events
  - To provide a framework for defining standard event header fields which contain event type and per-message QoS information
  - To provide a standard CORBA event format which is more predictable and thus easier to use for AP developers than CORBA::Any, and easier to implement than OMG Typed Events

Structure of a Structured Event



Contents of a Structured Event

- Minimal fixed header containing event type info
  - domain_name identifies a vertical industry within which event_type is scoped (e.g., Telecoms)
  - event_type identifies the name of a standard event type within the domain (e.g., CommunicationsAlarm)
  - event_name is a user defined field provided to hold the name of an individual event
- Variable length header containing QoS properties
  - EventReliability, Priority, StartTime, StopTime, TimeOut
- Filterable body fields intended to contain interesting event fields upon which filtering likely to be performed
- remainder_of_body, likely to contain bulk data (e.g., binary file)

Filter Objects

- **Notification Service clients have fine-grained control over which events are forwarded by a channel, and which are discarded**
  - This control is supported in the form of Filter objects
  - A list of Filter objects can be associated with each Admin and Proxy object within each channel
  - Each Filter object encapsulates a set of constraints which specify events that should be forwarded
    - Each constraint consists of a sequence of structures indicating event types, along with a boolean expression over the fields of an event
    - Event instances which are of one of the types indicated in the sequence of structures, and whose contents satisfy the boolean expression, essentially match the constraint

Filter Objects (cont.)

- **Run-time behavior of Filter Objects**
  - Logically, event instances arrive at the Proxy objects within a channel
  - If a given Proxy has no Filter objects associated with it, it simply forwards all events
  - If a given Proxy has one or more Filter objects associated with it, it forwards only those event instances which match at least one constraint encapsulated by at least one of its Filter objects
    - Within each Filter object, constraints are logically ORed together
    - OR semantics also applied between the Filter objects associated with a given Proxy

Filter Objects Associated with Admins

- **Filter objects can be associated directly with an individual Proxy object, or with an Admin object**
  - All Filter objects associated with an Admin object apply to all Proxy objects created by the Admin
  - Any modification to Filter objects associated with an Admin affect all Proxy objects in that Admin's group
    - This applies to modifications to the constraints associated with individual Filter objects, as well as additions and deletions to the list of Filter objects
- **Advantage of this feature**
  - Each Admin can represent a group of clients with similar requirements
    - These clients share a set of event subscriptions encapsulated by a single set of Filter objects

Mapping Filter Objects

- **Supplier Proxy objects (and their managing ConsumerAdmin objects) can also have Mapping Filter objects associated with them**
  - Similar to Filter objects, except that instead of affecting event forwarding decisions, affect the way a given Supplier Proxy treats each event with respect to certain properties
    - Have a value associated with each constraint, which is returned upon a successful match
    - Supplier Proxy uses this value instead of the property value explicitly associated with the event
  - Mapping Filter objects explicitly defined to affect event priority and lifetime
- **Enable clients to control how events are treated by the channel with respect to certain properties**

The Default Constraint Grammar

- **Clients of Notification subscribe to events of interest using boolean expressions over event type and contents**
  - These expressions must be valid in terms of some well-defined constraint grammar
  - Filter interface defined generically so that Notification Service implementers and end-users can provide implementations that use some proprietary constraint grammar
- **Notification Service specifies a default constraint grammar which must be supported by all conformant implementations**
  - Default grammar is based on the Trader Constraint Language, with extensions defined to remove ambiguities

Default Constraint Grammar (cont.)

- **Extensions to TCL defined to enable referencing components of complex structures**
  - '$' token introduced to denote the current event, or run-time variable
  - '.' operator defined to enable referencing members of structural data types
  - '[n]' operator defined to reference nth element of an array
- **Other extensions defined to enhance convenience of defining constraints over events**
  - To reference the 'value' member of an element of a name-value pair list nv, expression may contain nv(value)
    - E.g, can write $.EventHeader.variable_header(priority) > 2 instead of ($.EventHeader.variable_header[1].name == 'priority') and ($.EventHeader.variable_header[1].value > 2)

Default Constraint Grammar (cont.)

- **Support for run-time variables**
  - $curtime denotes current time of day
  - If field is any well-defined header field, $field references the value member of that field
    - E.g., $priority can be used instead of $.EventHeader.variable_header(priority)
  - Note that this makes it convenient to write a single constraint that can apply to either a structured event or an Any event
    - Filter engine can walk through an instance of either one and extract the value portion of the appropriate fields
- **Constraints can be specified using positional notation instead of specifying names of structure members**
  - Important since CORBA does not require names of IDL type members to be marshalled into the Typecode of an Any

Quality of Service Administration

- Notification is defined to enable implementations to be configurable by end-users to support a variety of qualities of service
- QoS properties can be set at various levels
  - On a per-channel basis, defining default QoS property settings for all Admin and Proxy interfaces within the channel
  - On a per-Admin basis, defining QoS properties which will override those set at the Channel level
  - On a per-Proxy basis, defining QoS properties which will override those set at the Channel or Admin level
  - On a per-message basis, which override all other settings

QoS Administration (cont.)

- **Standard QoS properties defined for**
  - Reliablility
    - Separate properties defined for connection and event instance reliability
    - Both properties can be set to either BestEffort or Persistent
  - Priority
    - Can be set within the header of a Structured Event to indicate the priority of an event instance
    - Consumers can override per-event setting using Mapping Filters
  - Order Policy
    - Proxy can be configured to order event delivery based on priority, earliest expiration time, or FIFO
  - Discard Policy
    - Proxy can be configured to discard events due to buffer overflow based on priority, earliest expiration time, FIFO, or LIFO

QoS Administration (cont.)

- **Standard QoS properties related to time**
  - StartTime
    - On per-message basis, specifies an absolute time after which channel should begin delivering an event
    - StartTimeSupported at the Proxy level indicates whether or not a given Proxy supports per-message StartTime setting
  - StopTime
    - On per-message basis, specifies an absolute time after which channel should discard the event if not yet delivered
    - StopTimeSupported at the Proxy level indicates whether or not a given Proxy supports per-message StopTime setting
  - Timeout
    - Specifies a relative time after which the undelivered event should be discarded
    - Can be overridden by a Mapping Filter

Setting and Validating QoS

- **set_QoS supported by Event Channel, Proxies, and Admins can be used to change QoS settings**
  - Operation raises UnsupportedQoS exception in cases when request cannot be satisfied by target object
    - Exception indicates which requested property setting(s) could not be satisfied, and for each what range of values could be satisfied
- **validate_QoS tests whether a set of desired QoS settings could be satisfied by a target object, without actually setting them**
  - This operation may also raise UnsupportedQoS
- **validate_event_QoS tests whether a Proxy could satisfy a set of QoS property settings on a per-message basis**

Sharing Subscription Info

- **Facilities provided to share subscription information between channel and suppliers**
  - Enables suppliers to keep track of which event types are of interest to consumers connected to the channel
    - Supplying of events of interest to no consumer can be avoided
  - Supplier interfaces inherit NotifySubscribe interface, which supports subscription_change operation
    - Operation intended to be invoked whenever there is a change in the set of event types which a consumer wants to receive
    - Proxy suppliers are registered as “callbacks” with filter objects, and are automatically notified whenever constraints are modified
    - Channel can propagate this info to its suppliers by invoke their subscription_change operation
  - Proxy consumers also support obtain_subscription_types which can be invoked at any time by suppliers

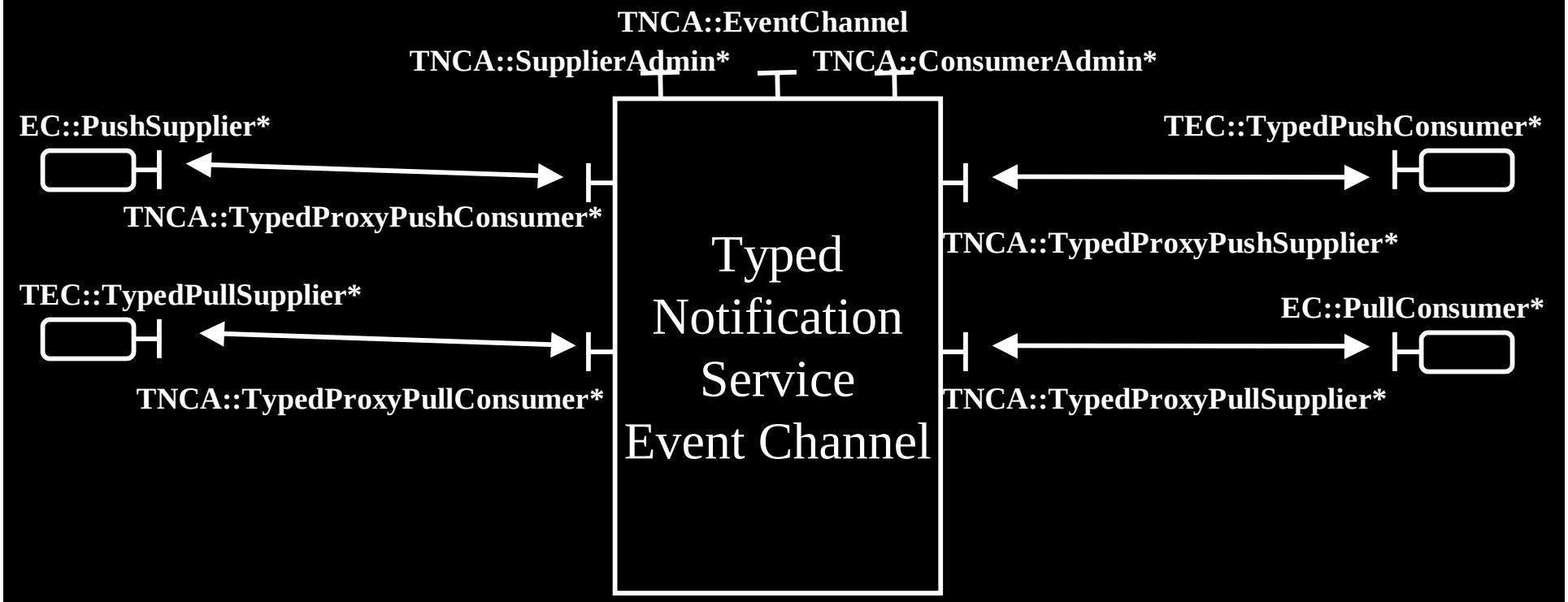
Sharing Offer Info

- **Facilities provided to share info between channel and consumers about what event types are being supplied**
  - Enables consumers to dynamically discover the types of events being offered to the channel by suppliers
    - Consumers can use this info to construct constraints to receive these new types of events
  - Consumer interfaces inherit NotifyPublish interface, which supports offer_change operation
    - Suppliers can inform the channel of changes to the types of events they will supply by invoking this operation on their Proxy consumer
    - Channel will propagate this info by invoking this operation on its consumers
  - Proxy suppliers also support obtain_offered_types which can be invoked at any time by consumers

The Typed Notification Channel

- **A hybrid of the typed event channel defined by OMG Event Service, and the Notification Service event channel**
  - Enables typed event service clients to take advantage of event filtering and QoS configurability features provided by Notification
    - Filter objects support “match_typed” operation, which accepts a sequence of name/value pairs formed by the operation name and input parameters of a typed push or pull operation
  - Typed event channel supports connections from untyped clients, clients that supply/consume Structured Events (single or sequence), and clients that supply/consume typed events
    - Proxies with explicit connect operations are defined for all clients that want to supply or consume typed events, to make usage model more intuitive

Typed Notification Channel Interfaces



NOTE: This are in addition to all interfaces supported by untyped Notification Service Event Channel

Optional Event Type Repository

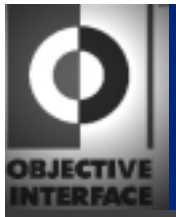
- **Spec defines an Event Type Repository which can be optionally supported by an implementation**
  - Defined in terms of IDL interfaces generated from a meta-model using the mapping specified in the MOF standard
  - ETR associates names with Event Types which are meaningful and unique within a particular domain
    - Default domain, “”, supports a flat name space
  - Each Event Type has associated with it one or more Properties, each consisting of a property name and a Typecode
    - These properties define the fields of an event instance which conforms to the associated Event Type
    - Properties define either the filterable_data portion of a Structured Event, or the fields of a structure transmitted within an Any or Typed event

Event Type Repository (cont.)

- **Event Type hierarchies**
  - Event Types may inherit from exactly one other Event Type
  - Full Event Type name formed by composing the names of all parent types in an inheritance hierarchy, according to some domain specific naming scheme
  - Alternatively, Event Types may import other event types, in which case they acquire properties of the types they import from but bear no semantic relationship to those types
- **Advantages of the Event Type Repository**
  - Suppliers can discover the properties of an event they must populate to ensure conformance to a given type
  - Consumers can dynamically discover the structure of event types, enabling them to write well-formed subscriptions

Telecoms Specific Event Types

- **Specification defines a mapping of X.721 OSI Telecommunication Events to Structured Events**
  - Attributes of an X.721 Notification template are mapped to name/value pairs that will be placed in the filterable_data portion of a Structured Event
  - IDL types of these attributes are defined in TelecomNotification module, following XoJIDM translation of the corresponding ASN.1 type
  - Standard mappings defined for several OSI event types
    - Alarm Events
    - Security Alarm Events
    - Relationship Management Events
    - State Management Events
    - Object Management Events

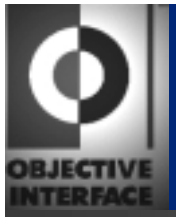


Outline

- Goals
- Relationship to Event Service
- Overview
- Conformance Points
- Addressing of RFP Requirements
- Summary

Conformance Points

- Conforming implementations will implement all interfaces from CosNotification, CosNotifyFilter, CosNotifyComm, and CosNotifyChannelAdmin
  - CosTypedNotifyChannelAdmin is optional
- Conforming implementations will provide filter objects which support the specified default constraint grammar
- All defined standard QoS properties will at least be understood by conforming implementations



Outline

- Goals
- Relationship to Event Service
- Overview
- Conformance Points
- Addressing of RFP Requirements
- Summary

Addressing of RFP Requirements

- **General Notification Interface Requirements**
  - Service supports decoupled communication between notification producers and consumers
  - Currently no explicit support for preserving identity of notification originator in case of sequential configuration of channels
    - Should be addressed in conjunction with Federated Notification Servers
  - Notification data can be filtered on or treated as additional info about the notification; can be in any IDL expressible form
  - Both push and pull models are supported

Addressing of RFP Requirements (cont.)

- **Notification Delivery Interface**
  - The notification delivery interface is of identical style to that defined in the OMG Event Service
  - Delivery QoS can be specified per-event, per-connection, or per-channel
- **Notification Subscription Interface**
  - Consumers can optionally supply filters to the proxy-suppliers they connect to
    - Filters can be arbitrarily complex, and can be associated with event type or content
    - Consumers can add/remove filter subscriptions at any time
    - Filters can be applied to typed, untyped, or “structured”events

Addressing of RFP Requirements (cont.)

- **Management Interface**
  - Notification consumers and suppliers can register or de-register with the service using the OMG Event Service style administrative interface
    - Suppliers can specify the types of events they will supply by invoking the “offer_change” operation on their associated consumer proxy
    - Consumers can supply filters describing which specific events they are interested in receiving upon registration
- **Dynamic Addition of User-Defined Types**
  - Service can dynamically handle new event types in form of typed events, or new standard mapping to structured events

Addressing of RFP Requirements (cont.)

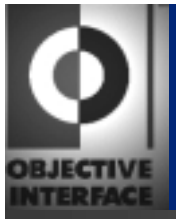
- **Security**
  - Proposed service works in conjunction with the OMG Security service to control access to individual events, or proxy interfaces and channels in case of networked channels
- **Federation**
  - This requirement was not addressed, as previously described

Addressing of RFP Requirements (cont.)

- **Qualities of Service**
  - Support for specifying QoS requirements on per-channel, per-admin, per-proxy, or per-message basis
  - Support for all QoS requirements defined in RFP
    - Assured delivery
    - Notification prioritization and assured delivery ordering
    - Aging of notifications
    - At-least-once delivery guarantee

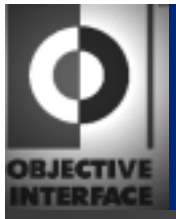
Addressing of RFP Requirements (cont.)

- **Notification Construction**
  - Defined filtering mechanism is flexible enough to be applied to any type of event, including untyped and typed OMG events
  - Support for transmission of and filtering on structured events subsumes requested support for
    - Standard header dynamic notification
      - Realized as a structured event with no filterable data fields defined for the body
    - Standard header static notification
      - Realized as a structured event with all body fields defined as filterable data



Outline

- Goals
- Relationship to Event Service
- Overview
- Conformance Points
- Addressing of RFP Requirements
- Summary



Summary

- **Direct extension of the OMG Event Service, adding many new features**
  - Ability to filter events based on their type and/or contents
  - Ability to configure the service at various levels to support a variety of QoS requirements
  - New event structures to which commonly used event types from other technologies can be conveniently mapped
    - Examples of such mappings for common Telecoms event types
  - Ability to share event subscription/offer info between the service and its clients
  - Optional Event Type Repository
- **Result of the combined efforts of several potential implementers and end-users of this service**