Abstract

READY is an event notification service that provides efficient, decoupled, and asynchronous event notifications. READY supports consumer specifications that match over single and compound event patterns, communication sessions that manage quality of service for event delivery, grouping constructs for sessions and specifications, event zones and boundary routers that bound the scope of event distribution and control the mapping of events across zones.

1. Introduction

Event-based communication is a powerful communication model that offers a decoupled communication infrastructure, supports dynamic changes, and preserves autonomy. In this model, information producers announce information on one or more communication channels and information consumers subscribe to channels of interest. An event notification service is a key enabling technology for building new event-based services on distributed platforms. Several commercial products are available where event consumers (subscribers) accept events from suppliers (publishers) often via a mediator. Publish-subscribe mechanisms have been retro-fitted to existing middleware products such as persistent message queue products.

In this demonstration, we present our work on the READY event notification system [2, 3]. READY facilitates cross-system information flow in an enterprise by providing the middleware for: (i) integrating applications that can run in a distributed and, possibly, heterogeneous environment and (ii) supporting rapid development of network-ready value-added services. READY supports consumer specifications that match over single and compound event patterns; communication sessions that manage quality of service (QoS) and ordering properties for event delivery; grouping constructs for both specifications and sessions; event zones and boundary routers that bound the scope of event distribution and control the mapping of events across zones, among others.

2. High-level event constructs

READY clients interact with READY using admin, supplier and consumer sessions. Admin sessions are used for creating and destroying supplier and consumer sessions and session groups and for other administrative operations. Supplier sessions supply events while consumer sessions register specifications that describe event patterns and actions to take when matches are found for patterns – the most common action, notify, causes the delivery of a notification with the matched event(s). One can suspend and resume a specification, enabling and disabling matching, respectively. Specifications can be grouped together, enabling efficient suspend and resume of many specifications.

A specification is composed of matching expressions and actions that are triggered by successful matches. Matching expressions bind event variables to events that match a given event pattern. A simple matching expression is a pattern that matches a single event, having the form:

\[ \text{event}_\text{var} : \text{event\_type} | \text{expression} \]

The expression is any legal expression from the filter grammar specified in [1]. Simple matching expressions can be combined to form compound matching expressions that describe a combination of events using the operators & & (and), || (or), ; (then), and butnot. The & &, ||, and butnot operators begin matching both LHS and RHS at the same time, and they match successfully once both sides match, either side matches, or the LHS matches first, respectively. In contrast, the sequencing (;) operator does not attempt to match the RHS until the LHS has matched. Another form of compound matching is to specify sequences of events that match the same “element pattern” using:

\[ \text{event\_var}[j..k] : \text{event\_type} | \text{expression} \]

The event\_var[j..k] indicates that the event variable will be bound to a sequence of events, with j and k being the minimum and maximum number of events, respectively.
Note that \( j \) can be zero, indicating no lower bound, and \( k \) can be an asterisk, indicating no upper bound. As a top-level expression, sequence matching completes when \( j \) successful element matches occur. When contained within a larger expression, a sequence match is successful once \( j \) element matches occur, but elements continue to be added to the sequence until either \( k \) element matches occur or a containing expression successfully matches.

Actions are imperative statements executed at specified points during matching. An assignment action binds an event variable to a newly-created event. This is most often used for aggregation, where a set of matching events is summarized by a single new event. Actions that operate over event variables include the notify action (mentioned above) and the announce action, which acts as a supplier of a new event. Actions that do not require contacting suppliers or consumers are called local actions; these actions only involve changing the internal state of READY servers.

READY uses CORBA's Notification Service [1] structured events, which consist of header, filterable body, and remainder of body. The header contains event type, event name, and an optional variable part with QoS-specific attributes, timestamps, and so on. READY event type definitions specify a set of required and optional fields, where each field contains a field name, a type identifier, and a value. READY types can have subtypes, and a subtype declaration simply adds additional required or optional field specifications to those of its parent type. The filterable body contains attribute-value pairs, and the remainder of the body is an opaque value.

Event consumers/suppliers can be partitioned into event zones based on various boundaries, e.g., administrative, logical, and so on. Event routing between zones is done by boundary routers, specialized READY servers that connect READY zones. Boundary routers support event translations (zones can differ in the conventions adopted for use of particular fields within events, event types, etc.) using mapping specifications that match against events in one zone and announce, possibly translated, events in another zone. In addition, boundary routers address information flow constraints that may exist across zone boundaries by restricting event throughput used by inter-zone mapping specifications.

3. Implementation notes

The initial READY prototype [2, 3] used its own Application Programming Interface (API) to support filters which match over both single and compound event patterns, QoS properties for event delivery, grouping constructs, and event zones and boundary routers that bound the scope of event distribution and control the mapping of events across zones. The standardization of the CORBA Notification Service and its support for parts of the READY functionality motivated us to re-implement READY on top of the omniORB CORBA ORB [4].

We chose a CORBA ORB and not an existing CORBA Notification Service implementation because: (i) compound matching is not supported by the CORBA Notification Service, and (ii) existing implementations do not offer the necessary hooks to discover the registered filters. Currently, we are finishing the implementation of the complete CORBA Notification Service specification. Once this is done, we plan to offer compound matching by using the READY filter language in the place of the constraint language specified in the Notification Service specification. Since this change does not affect the interfaces of the Notification Service, the resulting implementation would be fully backwards compatible with the CORBA Notification Service.

4. Demonstration details

Some of the functionality supported by READY will be demonstrated using WEB-based event supplier and consumer applications. These applications are either generic or specialized. Generic suppliers are used to announce events containing attributes and values that are entered by the user using HTML forms. Generic consumers are used to subscribe to receive notifications that match the specifications that are provided by the user using HTML forms. On the other hand, specialized suppliers and consumers are READY clients that have been developed for specific application domains. In particular, we will show an email consumer that registers to receive notifications when state changes occur at a messaging server. These state changes are triggered by operations carried out by the SMTP, POP, and IMAP components of the messaging server. In addition, we will show an alert supplier that triggers delivery of user alerts via phone calls. More information about READY and available WEB-based demonstrations can be found at the READY home page: http://www.research.att.com/~ready.

References