Global Event Model Ideas
Adam Rifkin — October 27, 1997

- Two Examples.
  - Room/Floor/Building Security.
  - An “Active Document” for a Budget.
- Local Event Models.
  - JavaBeans.
  - Compare with Infospheres, CORBA, Multicast.
- Global Event Model.
  - Compositional Structure.
  - Specification Notation.
  - Predicates.
  - Soft Real Time.
  - Fault Tolerance.
  - Exceptions.

Example: Room/Floor/Building Security

- Predicates for rooms.
- Predicates for floors composed by the predicates for the rooms.
- Predicates for the building composed by the predicates for the floors.

Example: An "Active Document" for a Budget

- A document where different people have different views based on relative importance to the document.
- Each view allows different manipulations of the document.

Components

- Components of Components.
  - Properties, which are broadcastable variables.
  - Events, which are messages that announce property changes.
  - Like objects, attributes and methods for doing stuff.
- JavaBeans is a component model implemented in Java. JavaBeans represents the culmination of years of event work, from operating systems and GUIs to CORBA and OpenDoc. And yet, even now events are distributed only locally within a single beanbox on a single machine — in other words, they are not distributed at all.
- Operational model: Components consist of types, variables, properties, event targets, and event sources. Communication happens through events. Delete, add, listeners, methods, beans.

Properties

- A *property* is an attribute name (and corresponding value) that represents a single piece of data that affects its component's appearance or behavior.

- When the value of a *bound property* changes, the component owning the property sends out a notification message to all of the components that have expressed interest in the property. This is called a *change notification service*.

- A *constrained property* is similar to a bound property, except that other components can veto a change to that property value.

- Conceptually, an event is any meaningful occurrence. For example, `PropertyChangeevent` occurs whenever a property value changes (as in, a mouse moves).

- A *property editor* allows developers to change the value of a component property through a nice visual interface.

- A *property sheet* visually displays all of a component's properties, each of which is supported by a corresponding property editor.

Events

- *Events* are the mechanism by which a component informs other components that something interesting has happened. Think of them as active messages.
- For example, a button is pressed at a source, and channeled through an event listener, to automatically trigger a method in an event observer.
- An *event* is immutable and does not usually change as it propogates from an event source through an event listener (also called an event notifier) to one or more event observers (also called event targets).
- A *framework* (such as JavaStudio) can associate the events that a component fides with methods in other components.
- *Introspection* is the process of discovering a component's properties, methods, and events at runtime.
- *Persistence* is the saving of internal state.

Event Sources

- An *event source* triggers an event state object when its internal state changes, usually as a result of some external stimulus.
- An event source delivers an event to a list of listeners sequentially using its own thread. It has methods such as `addListener` and `removeListener`.
- An event source must catch and handle all of the exceptions raised when it calls listeners.
- JavaBeans permits event sources to define their own delivery-order semantics to event listeners.

Event Listeners

- An *event listener* is a callback object that serves as a bridge between an event source and a consumer of events. A listener may encapsulate a number of semantically related methods.
- An event listener allows the application-specific code to be localized in one or more *event observers*.
- Potential uses of event listeners:
  1. Implement event queues for observers.
  2. Filter events before they reach observers.
  3. Demultiplex multiple event sources into a single notifier.
- JavaBeans cleanly separates GUI code from the event handling code, which resides in the event observer.

Event Observers

- An event observer contains application-specific event handling methods that respond to events triggered in an event source.
- The *multicast event registration* allows any number of event listeners to be associated with an event source under the same event types.
- *Event delivery* is the process of flowing a triggered event from an event source to an event observer through an event listener.
- An *event-driven system* runs in a multithreaded environment and has an asynchronous nature. Such environments are subject to races and deadlocks. To reduce risks, we synchronize the registration methods and call event listeners from within unsynchronized code.

Publish and Subscribe

- Sources publish events. Listeners subscribe.
- A source component needs:
  - A method call to fire off an event notification upon occurrence.
  - Declarations for `add---EventListener` and `remove---EventListener` that other components call to get on and off event notification lists.
- A listener component needs:
  - To call the source component's `add---EventListener` to register interest in listening (or call `remove---EventListener` when it loses interest).
  - A method to perform some action after receiving the event from the source component.
- Creating new events:
  - New event support class (e.g., `PropertyChangeSupport`).

- New type of event objects (e.g., `PropertyChangeEvent`).
- New type of event listener (e.g., `PropertyChangeListener`).
- Pair of register/cancel methods (e.g., `addPropertyChangeListener` and `removePropertyChangeListener`).

Comparison with Infospheres and Multicast

- Infospheres 1.
  - Outbox can bind to any inbox.
  - Subscribers must beg listener to add or remove, but the publisher has all of the control.
  - However, subscriber can move so the publisher never really knows where subscriber is.
- Multicast.
  - Subscribers listen on a multicast address.
  - Publisher posts to multicast address without having list of listeners.
  - Separate registry program handles add and remove.
- Other comparisons? CORBA events?

From a Local Event Model to a Globel Event Model

- Specification Notation.
- Predicates.
- Soft Real Time.
- Fault Tolerance.
- Exceptions.
- Compositional Structure.

Compositional Structure

- *Problem*: how to deal with multiple notifiers? Study compositional structure of event model. Notifiers that can call other notifiers. In the middle could have several intermediate stages. If you think about this as one big system, what can you prove?
- Need to investigate design choice of how to structure applications. Perhaps a hierarchical structure (buildings, floors, and rooms). Perhaps a more flat structure (buyers, sellers, markets, companies).

Specification Notation

- *Problem*: how to specify components, events, and event flow?
- Timed temporal logic. State transition diagrams.

Predicates

- *Problem*: what to put in event listeners to make them easier to program and reason about? Dealing with predicates on the attribute-value pairs bundled with events. Especially predicates that have time in them. How to deal with in a compositional way?

- A predicate comprises ands and nots on attribute/value pairs in event headers. A predicate may include an access control list. A predicate may include soft real time.

- Use of XML message headers. Attribute-value pairs allow arbitrary predicates in event notifiers. Personalization of response to events in a customizable GUI. Customization is key.

Soft Real Time

- *Problem*: how to deal with time in predicates?
- Using a clock not guaranteed to be accurate. For example, wait 5 minutes, then monitor again. Keep local control of constraints.
- From my understanding, Paul only has leads-to in his certificates, so adding time to them seems arbitrary because you don't know when leads-to actually happens. This is where soft real time could be useful.
- For example, give a timed certificate to the runtime checker. The checker can then itself throw events. This is a good use of a safety property, with no need to modify progress conditions.

Fault Tolerance

- *Problem*: a global event model must have facilities for handling faults and excessive communication delays. How to do this?
- Example: fire alarm goes off, no response in 5 minutes. Response: diagnose the line as dead and try something else. In beans, the applications are usually GUIs that are not safety-critical or mission-critical, so faults and excessive delays are all right.
- Feedback cycles are necessary. For example, what we're programming the thermostat? Now, what if there is a fire? Feedback cycles detect different types of heat and do diagnostics to determine whether turning on the air conditioning makes sense.

Exceptional Behavior

- *Problem*: A global event model must have facilities for handling exceptions such as late responses or nonresponsive communication channels. How to do this?
- Component behavior. If a component is not behaving correctly, raise an exception. For example, if it violates one of Paul's certificates, all bets are off. Can do as event exception behaviors as runtime checks, intuition says.