Lecture 13: Event-driven programming

Event-driven programming

PUBLISHERS
trigger events

SUBSCRIBERS
handle events

EVENTS

EVENT TYPE

ACTION

Left_click

Save_file

Avoiding glue code

Event producer
(e.g. GUI)

Direct subscription

Connection object

Business model (application logic)

Internally

- Event-Action table
- (More precisely: Event_type-Action-Table)
- More precisely: Context-Event_type-Action-Table)

Event-driven programming
Event Library

- Class EVENT_TYPE
- Publisher side, e.g. GUI library:
  - (Once) declare event type:
    ```
    click: EVENT_TYPE [TUPLE [INTEGER, INTEGER]]
    ```
  - (Once) create event type object:
    ```
    create click
    ```
  - Each time the event occurs:
    ```
    click.publish ([x_coordinate, y_coordinate])
    ```
- Subscriber side:
  ```
  click.subscribe (agent my_procedure)
  ```

Subscriber variants

```
click.subscribe (agent my_procedure)
my_button.click.subscribe (agent my_procedure)
```

```
click.subscribe (agent your_procedure (a, ?, ?, b))
```

```
click.subscribe(agent other_object,other_procedure)
```

EiffelVision style

```
my_button.click.action_list.extend (agent my_procedure)
```

Observer pattern (C++, Java)

```
PUBLISHER

attach detach

LIBCLASS

APPCLASS

update*

update+

Deferred (abstract)

* Effective (implemented)

Inherits from Client (uses)

```

Observer pattern

```In PUBLISHER:

 subscribed: LIST [SUBSCRIBER]

-- Clients subscribed to this publisher

attach (s: SUBSCRIBER) is

-- Record subscription of s.

do

 subscribed.extend (s)

end

publish is

-- Trigger event.

do

 from subscribed.start until subscribed.after loop

 subscribed.item.update

 subscribed.forth

end

```

```In SUBSCRIBER:

 subscribe (p: PUBLISHER) is

-- Subscribe to p's event.

 do

 p.attach (Current)

end
```
Observer pattern

In SUBSCRIBER:

```
subscribe (p: PUBLISHER) is
  -- Subscribe to p's event.
  require
    publisher_exists: p /= Void
  do
    p.attach (Current)
  end
```

Event library

- Publisher, e.g. GUI library:
  - Declare and create:
    ```
    click: EVENT_TYPE [TUPLE [INTEGER, INTEGER]]
    ```
  - Trigger each event with arguments.
    ```
    click.publish ([x, y])
    ```
- Subscriber (to subscribe a routine r):
  ```
  my_button.click.subscribe (agent r)
  ```

Background: .NET

- Basis for future development of Windows
- Introduced in 2000
- Layer on top of the operating system
- Supports advanced Web technologies, especially through ASP.NET
- Based on an object model
- Microsoft languages: C#, Visual Basic .NET
- Multi-language, e.g. Eiffel, Cobol, Oberon
- Numerous libraries of reusable components
- International standard (Common Language Interface) through ECMA and ISO
- Non-Windows implementation: Mono

Observer pattern (C++, Java)

- Deferred (abstract)
- Effective (implemented)

.NET event-delegate mechanism

- Publisher or subscriber:
  D1. Introduce descendant ClickArgs of EventArgs repeating types of arguments of myProcedure. (Adds a class.)
  ```
  public class ClickArgs {
    int x, y;
  }
  ```
  D2. Declare delegate type ClickDelegate based on that class. (Adds a type.)
  ```
  public void delegate ClickDelegate (Object sender, ClickArgs e);
  ```
**.NET delegates: publisher**

- Declare new event type `Click` based on the type `ClickDelegate`. (Adds a type.)
  ```csharp
  public event ClickDelegate Click;
  ```
- Write procedure `OnClick` to wrap handling. (Adds a routine.)
  ```csharp
  protected void OnClick (ClickArgs e)
  {
    if (Click != null)
      Click (this, e);
  }
  ```
- For every event occurrence, create instance of `ClickArgs`, passing arg values to constructor. (Adds a run-time object.)
  ```csharp
  ClickArgs myClickArgs = new ClickArgs (h, v);
  ```
- For every occurrence, trigger event `OnClick (myClickArgs);`

**.NET delegates: subscriber**

- To subscribe a routine `myProcedure`:
  ```csharp
  Declare a delegate `myDelegate` of type `ClickDelegate`. (Can be combined with following step as shown next.)
  ```
- Instantiate it with `myProcedure` as constructor’s argument.
  ```csharp
  ClickDelegate myDelegate = new ClickDelegate (myProcedure);
  ```
- Add it to the delegate list for the event.
  ```csharp
  yourButton.Click += myDelegate
  ```

**Event library**

- **Publisher**, e.g. GUI library:
  ```csharp
  Declare and create:
  ```
  ```csharp
  EVENT_TYPE [TUPLE [INTEGER, INTEGER]]
  ```
- Trigger each event with arguments.
  ```csharp
  click.publish ({x, y})
  ```
- **Subscriber** (to subscribe a routine `r`):
  ```csharp
  my_button.click.subscribe (agent r)
  ```

**Lessons**

- Avoid magic: what’s available to the language designer should be available to the programmer.
- Role of language mechanisms: genericity, constrained genericity, tuples.
- Importance of choosing the right abstractions
  - Observer Pattern: `PUBLISHER, SUBSCRIBER`
  - .NET: event, delegate, event type, delegate type?
  - Eiffel Event Library: `EVENT_TYPE`

**Avoiding glue code**

- Event producer (e.g. GUI)
  ```csharp
  Direct subscription
  ```
  ```csharp
  Business model (application logic)
  ```
Complementary material

- Eiffel: The Language, 3rd edition (draft), chapter 25
  - Available online at:
    (User name: Talkitover; password: etl3)

- Paper on Event-driven programming
  - Available online at:
    http://www.inf.ethz.ch/~meyer/ongoing/events.pdf

End of lecture 13