Lecture 13: Event-driven programming

Event-driven programming

PUBLISHERS trigger events

EVENTS

SUBSCRIBERS handle events

ROUTINE

ROUTINE

ROUTINE
### Avoiding glue code

*Direct subscription*

- Event producer (e.g. GUI)
- Connection object
- Business model (application logic)

### Internally

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

**Event type** | **Action**
--- | ---
Left_click | Save_file

### Event-driven programming

**PUBLISHERS**
trigger events

**EVENTS**

**SUBSCRIBERS**
handle events

- ROUTINE
- ROUTINE
- ROUTINE
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:
    ```
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)

In **PUBLISHER**:

- `subscribed: LIST [SUBSCRIBER]` -- Clients subscribed to this publisher
  - `attach (s: SUBSCRIBER)`
    - do `subscribed.extend (s)`
  - `publish`
    - do `from subscribed.start until subscribed.after loop` `subscribed.item.update subscribed.forth` end

In **SUBSCRIBER**:

- `subscribe (p: PUBLISHER)`
  - do `p.attach (Current)`
    - `update*`
  - `detach`
  - `update+`
  - Inherited from Client (uses)
  - Deferred (abstract)
  - Effective (implemented)
Observer pattern

In `SUBSCRIBER`:

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

Observer pattern

In basic scheme:
- Publishers know about subscribers
- Subscriber may subscribe to at most one publisher
- May subscribe at most one operation
- Not reusable — must be coded anew for each application

Event library

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

### .NET event-delegate mechanism

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

### Observer pattern (C++, Java)

- Diagram illustrating the observer pattern with classes `SUBSCRIBER`, `PUBLISHER`, `APPCLASS`, and `LIBCLASS`.
- Key points:
  - `*` Deferred (abstract)
  - `+` Effective (implemented)
  - Inherits from Client (uses)
  - Methods: `attach`, `detach`, `update`
**.NET delegates: publisher**

D3 • Declare new event type `Click` based on the type `ClickDelegate`. (Adds a type.)

```csharp
public event ClickDelegate Click;
```

D4 • Write procedure `OnClick` to wrap handling. (Adds a routine.)

```csharp
protected void OnClick(ClickArgs e)
{
    if (Click != null)
        Click(this, e);
}
```

D5 • 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);
```

D6 • For every occurrence, trigger event

```csharp
OnClick(myClickArgs);
```

---

**.NET delegates: subscriber**

D7 • To subscribe a routine `myProcedure`:

D8 • Declare a delegate `myDelegate` of type `ClickDelegate`. (Can be combined with following step as shown next.)

```csharp
ClickDelegate myDelegate =
    new ClickDelegate(myProcedure);
```

D9 • Add it to the delegate list for the event.

```csharp
yourButton.Click += myDelegate;
```

---

**.NET delegates**

- `event` is a keyword of the language (special features of a class). But event types should be treated as ordinary objects.

- Cannot have closed arguments: for equivalent of

  ```csharp
  r(a, ?, ?, b)
  ```

  must write routine wrapper to be used for delegate.

- Cannot have open target: for equivalent of

  ```csharp
  {TYPE}.r(…)
  ```

  must write routine wrapper.
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)
    ```

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)
- Direct subscription
- Connection object
- 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