Chapter 1

Announcements: an Object Notification Framework

It is often necessary to get notified when an action has been performed. This can be simply done by having an instance variable pointing to the objects to be notified. However such solution is not optimal because of the coupling it introduces between the objects. In this Chapter, we present Announcements, a framework to notify objects based on a registration mechanism and first class announcements. Indeed contrary to the traditional Smalltalk dependency mechanism which broadcast simple symbol, Announcement notification are full objects. Announcement can be the basis to Observer Design pattern.

1.1 A word about object coupling

Stéf ►*should rewrite to get it more about objects and not components*◄ Here is an interesting question that often comes up often when writing components. It is one that we faced when embedding our components. How do the components communicate with each other in a way that doesn't bind them together explicitly? That is, how does a child component send a message to its parent component without explicitly knowing who the parent is? Designing a component to refer to its parent is just a part of the solution since the interfaces of different parents may be different which would prevent the component from being reused in different contexts.

There is a solution based on explicit dependencies also called the change/update mechanism. Since early versions of Smalltalk, a dependency mechanism based on a change/update protocol is available and it is the foundation of the MVC framework itself. A component registers its interest in

some event and that event triggers a notification. This is the basis of the Observer design pattern.

1.2 Announcements

Announcement developed originally by Vassili Bykov is a new framework that offers notifications with full objects. While the original dependency framework relied on symbols for the event registration and notification, Announcements promotes an object-oriented solution. Events are plain objects.

The main idea behind the framework is to set up announcers, define or reuse some announcements, let clients register interest in events and signal events. An announcement is an object representing an occurrence of a specific event. It is the place to define all the information related to the event occurrence. An announcer is responsible for registering interested clients and announcing events, see Figure **??**.

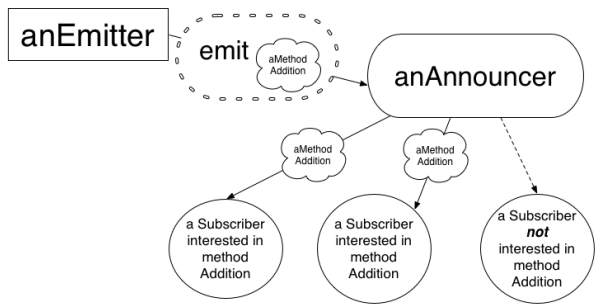

Figure 1.1: Announcement flow

API

Any object interested in an announcement registers its interest using the method on: anAnnouncementClass do: aBlock or on: anAnnouncementClass send: aSelector to: anObject. The messages on:do: and on:send:to: are strictly equivalent to the messages subscribe: anAnnouncementClass do: aValuable and subscribe: anAnnouncementClass send: aSelector to: anObject. You can also ask an announcer to unsubcribe: an object.

```
anAnnouncer on: MyAnnouncement do: [:announcement | announcement
    doSomething ]
```

```
anAnnouncer announce: (MyAnnouncement new)
```

Method 1.1: *Unsubscribing from the announcer*

```
anAnnouncer unsubscribe: self
```

1.3 Weak Announcements

Even if you do not bind the emitter with the receiver, you create a dependency between to the announcer by registering yourself. And because announcers keep a reference to subscribers, this dependency prevent your registered objects to be garbage collected.

To solve this problem, weak announcements have been implemented. A weak announcement acts like regular announcements but keeps only a weak reference to subscribers, allowing them to be garbage collected.

How to subscribe to weak announcement

This protocol is really simple, because it's almost the same than for regular announcements.

When you use to do

```
self session announcer on: RemoveChild send: #removeChild: to: self
```

you just do

```
self session announcer weak on: RemoveChild send: #removeChild: to: self
```

The major difference is that you can't use the method **when:send:** anymore. Indeed if a weak announcement store a block, it will also store the block context and due to that, keep a strong reference to the subscriber.

System Announcements

We saw that you can create your own announcements to communicate between objects, but there are already existing announcements. All subclasses of SystemAnnouncement are managed by the system to inform that it changed.

```
SystemAnnouncement
        SystemCategoryAddedAnnouncement
        SystemCategoryRemovedAnnouncement
        SystemCategoryRenamedAnnouncement
        SystemClassAddedAnnouncement
        SystemClassCommentedAnnouncement
        SystemClassRecategorizedAnnouncement
        SystemClassRemovedAnnouncement
```

```
SystemClassRenamedAnnouncement
SystemClassReorganizedAnnouncement
SystemMethodAddedAnnouncement
SystemMethodModifiedAnnouncement
SystemMethodRecategorizedAnnouncement
SystemMethodRemovedAnnouncement
SystemProtocolAddedAnnouncement
SystemProtocolRemovedAnnouncement
SystemUnknownAnnouncement
```

1.4 Discussions

Announcements are like exceptions because they notify about events, but they are different in the nature of relationship between the announcer and the handler. Exceptions are for communication along the sender stack chain. There is no explicit connection between the announcer and the handler apart from the fact that the handler calls the announcer, directly or indirectly. Announcements are for communications across the object network, with a connection between the announcer and the handler pre-established by subscribing.