Managing Sensor Groups and Registering New Temperature Sensors

Jason Roberts
.NET MVP

@robertsjason dontcodetired.com

Overview

Design overview

The importance of message immutability

Implementing message immutability

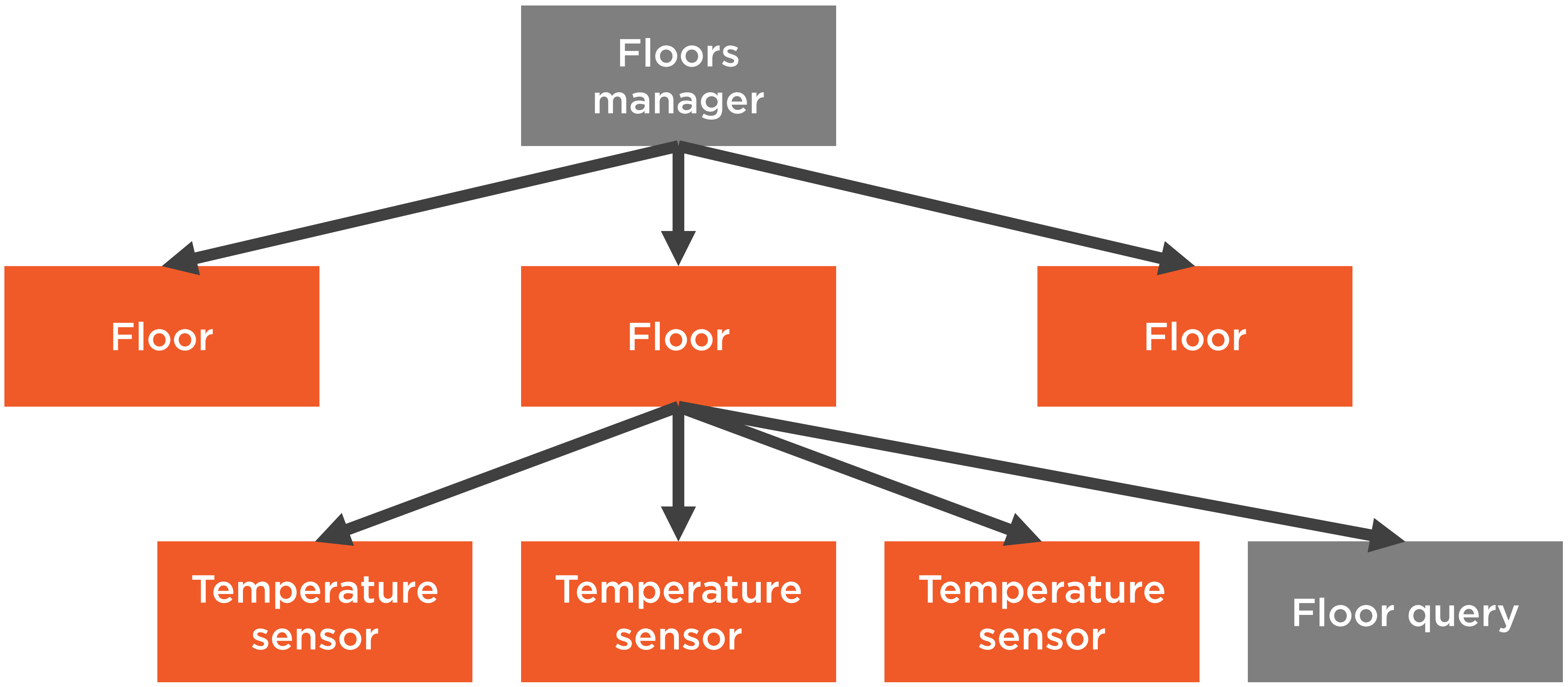
Add new message classes

Add new FloorsManager class

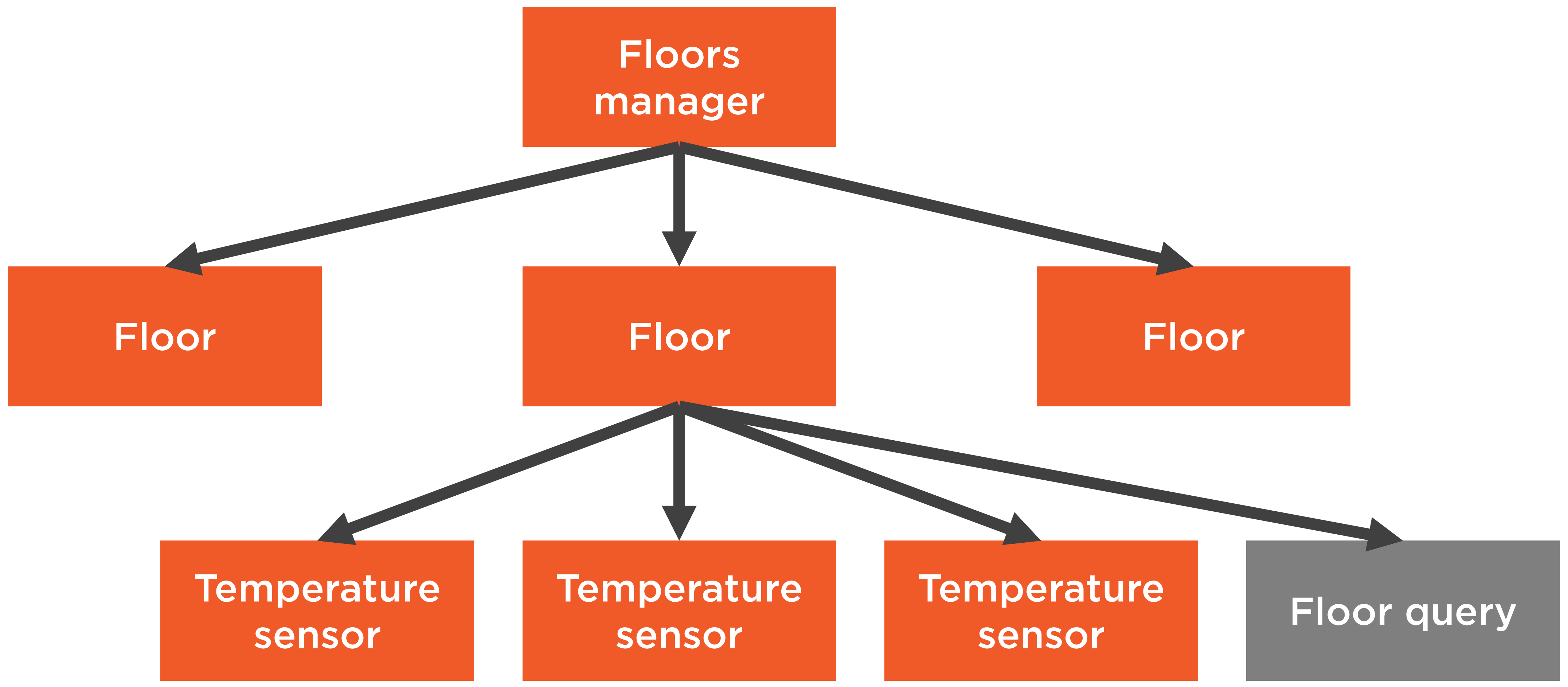
- List floor IDs
- Register new floor actors
- Reuse existing floor actors
- Monitoring child actors for terminated messages

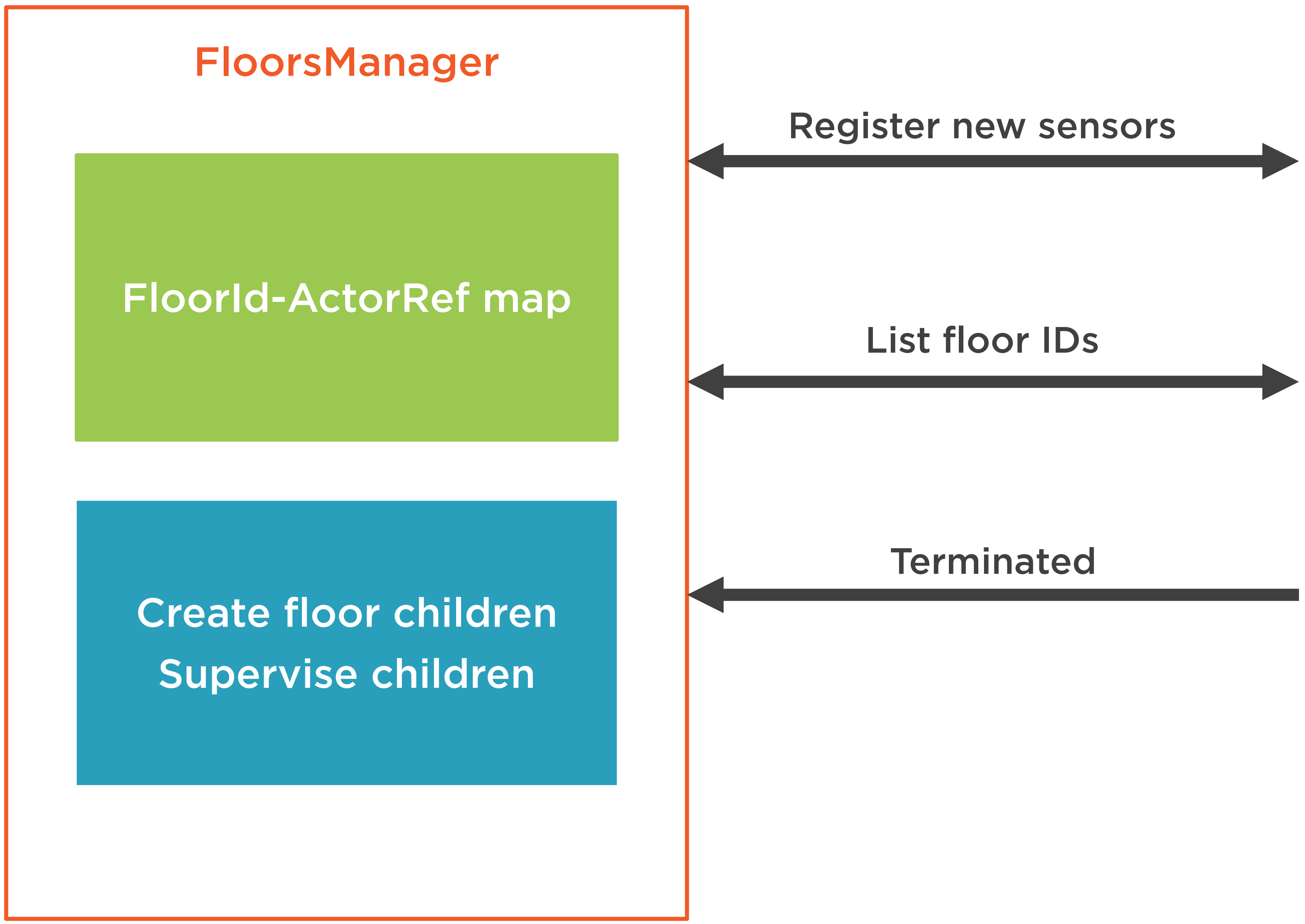
Working FloorsManager actor & tests

Supervision Hierarchy



Supervision Hierarchy




FloorsManager
FloorId-ActorRef map
Create floor children
Supervise children
Register new sensors
List floor IDs
Terminated

Messages

Simple POCO classes

Should be immutable

Sent from:

- Actors
- Outside actor system

Change actor state

Execute actor functionality

Messages

Simple POCO classes

Should be immutable

Sent from:

- Actors
- Outside actor system

Change actor state

Execute actor functionality

One of the main benefits of the actor model is its built-in concurrency management.

“Do not pass mutable objects between actors... prefer immutable messages. If the encapsulation of actors is broken by exposing their mutable state to the outside, you are back in normal .NET concurrency land with all the drawbacks.”

Akka.NET documentation
[http://getakka.net/articles/concepts/actor-systems.html]

Message Immutabality

```
public sealed class RespondTemperatureSensorIds
{
   public long RequestId { get; }
   public ISet<string> Ids { get; }

   public RespondTemperatureSensorIds(long requestId,
                                      ISet<string> ids)
   {
      RequestId = requestId;
      Ids = ids;
   }
}
```

Message Immutabality

```
public sealed class RespondTemperatureSensorIds
{
    public long RequestId { get; }
    public ISet<string> Ids { get; }

    public RespondTemperatureSensorIds(long requestId,
                                       ISet<string> ids)
    {
        RequestId = requestId;
        Ids = ids;
    }
}
```

Message Immutabality

```
public sealed class RespondTemperatureSensorIds
{
    public long RequestId { get; }
    public ISet<string> Ids { get; } // Ids.Add("xyz");

    public RespondTemperatureSensorIds(long requestId,
                                       ISet<string> ids)
    {
        RequestId = requestId;
        Ids = ids;
    }
}
```

Summary

Design overview

The importance of message immutability

Implementing message immutability

- IImmutableSet<string>

Added new FloorsManager class

Added new message classes

Added tests

- List floor IDs
- Register new floor actors
- Reuse existing floor actors
- Monitoring child actors for terminated messages

Next:

Querying Temperature Sensor Actor Data