

Single Responsibility Principle (SRP)

Produced
by:

Eamonn de Leastar (edelestar@wit.ie)

Dr. Siobhán Drohan (sdrohan@wit.ie)



Waterford Institute *of* Technology
INSTITIÚID TEICNEOLAÍOCHTA PHORT LÁIRGE

Department of Computing and Mathematics
<http://www.wit.ie/>

“S” in SOLID - Single Responsibility Principle

Every object should have a single responsibility and all of its services should be aligned with that responsibility.

“Responsibility” is defined as “a reason to change”

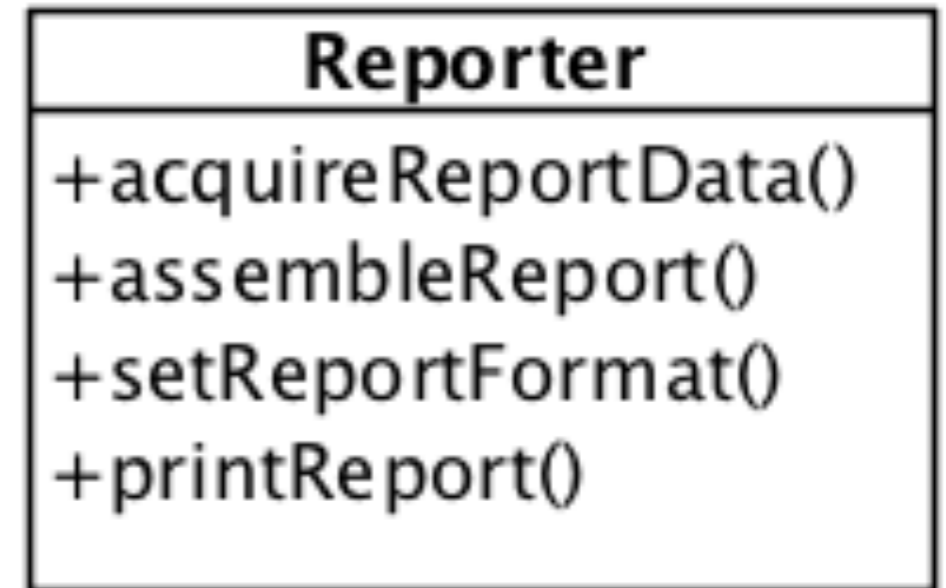
SRP: The Single Responsibility Principle

THERE SHOULD NEVER BE MORE THAN ONE REASON FOR A CLASS TO CHANGE.

- Each responsibility is an axis of change.
- When requirements change
 - a change in responsibility amongst the classes.
- If a class assumes more than one responsibility
 - more than one reason for it to change.
 - changes to one responsibility may impair or inhibit the class' ability to meet the others.

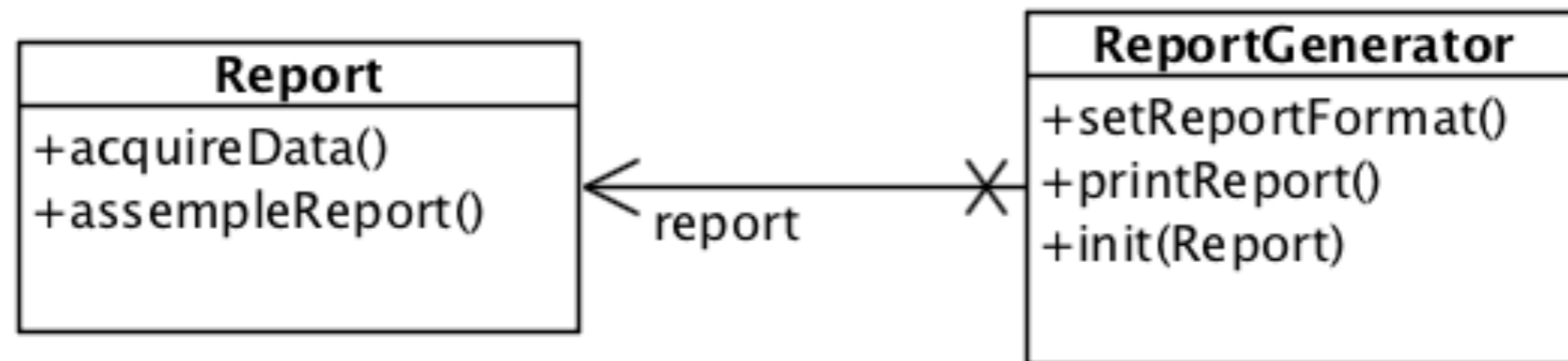
Single Responsibility Principle – Example 1

- Consider a Class that assembles and prints a report.
- The class can be changed for two reasons.
 - the content of the report can change.
 - the format of the report can change.
- These two things change for very different causes; one perhaps substantive, and one cosmetic.



Single Responsibility Principle – Example 1

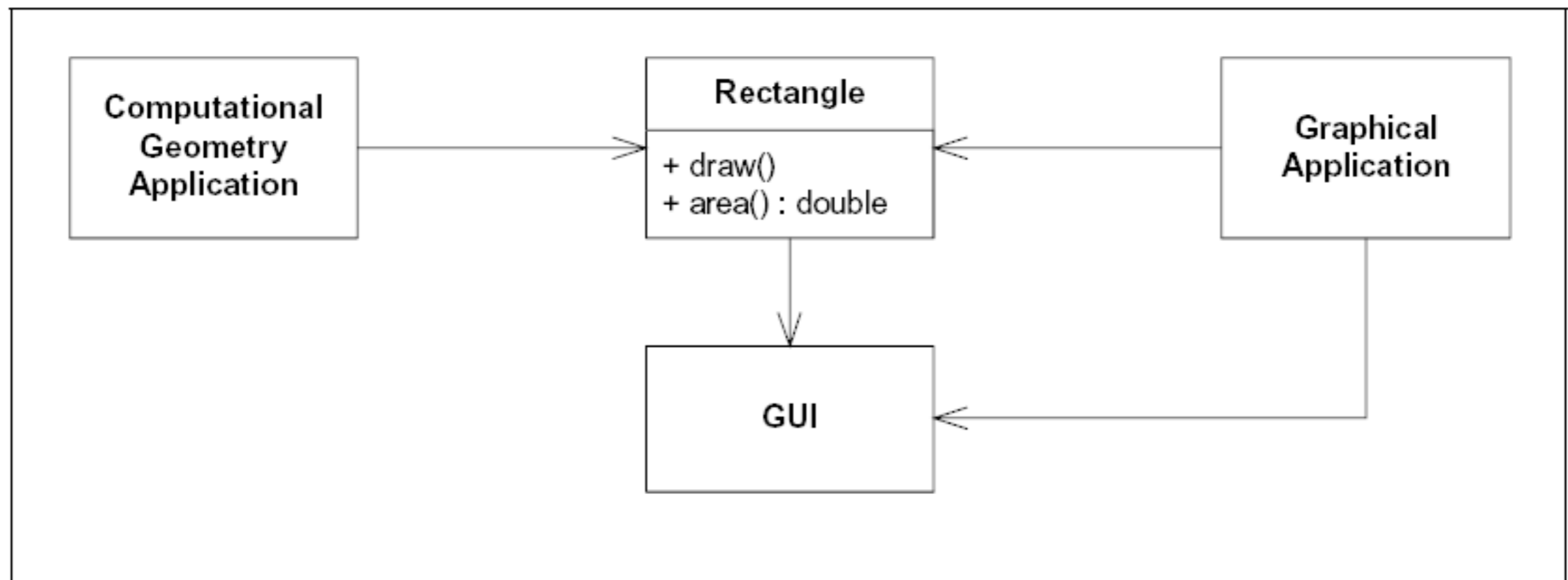
- SRP: these two aspects of the problem are really two separate responsibilities, and should therefore be in separate classes:



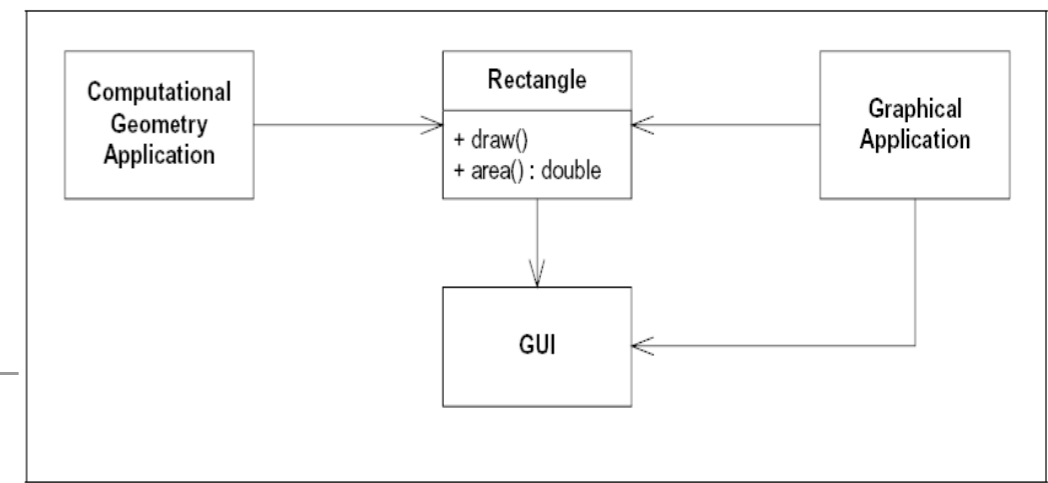
- SRP: Avoid coupling two things that change for different reasons at different times.

Single Responsibility Principle – Example 2

- The Rectangle class has two methods:
 - one draws the rectangle on the screen
 - the other computes the area of the rectangle.
- Two applications use this class:
 - one application uses Rectangle to help it with the mathematics of geometric shapes.
 - the other uses the class to render a Rectangle on a window.



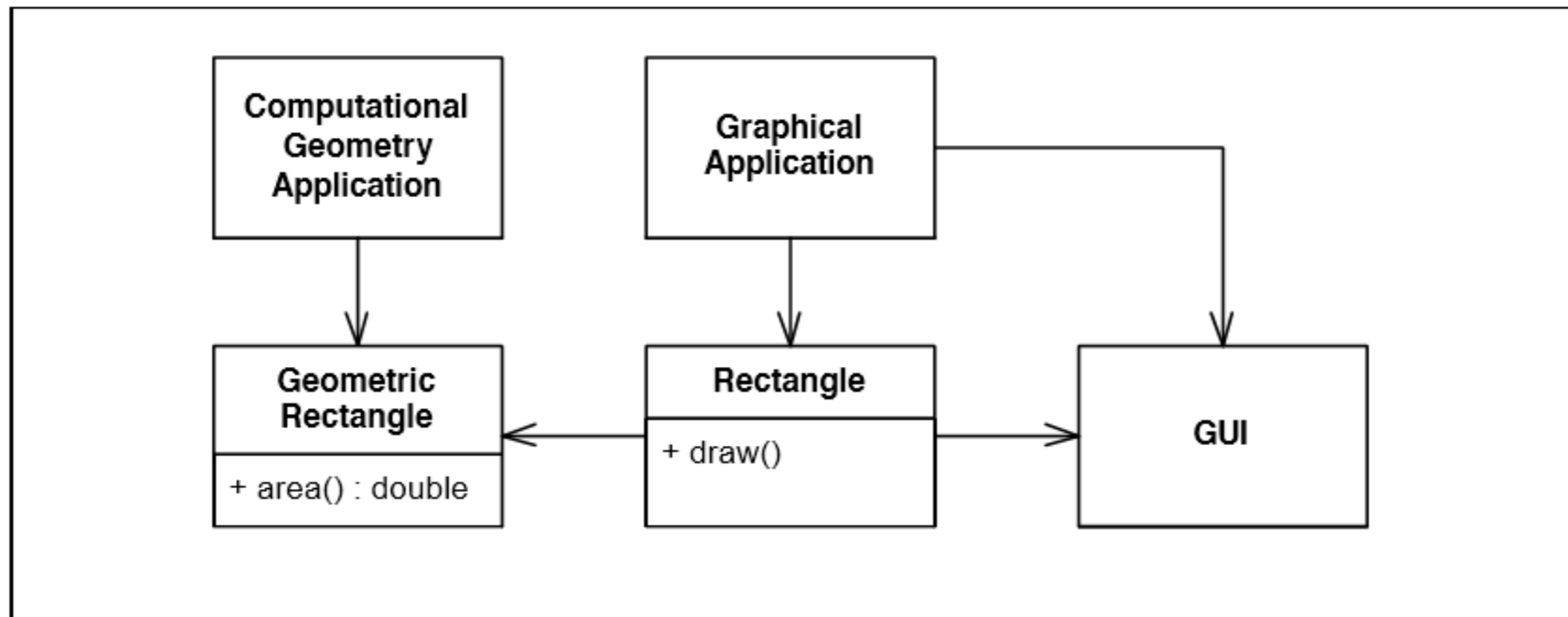
SRP Violation



- Rectangle has two responsibilities:
 - provide a mathematical model of the geometry of a rectangle.
 - render the rectangle on a graphical user interface.
- Violation of SRP:
 - the GUI must be included in the in the computational geometry application.
 - the class files for the GUI have to be deployed to the target platform.
 - if a change to the Graphical Application causes the Rectangle to change for some reason, that change may force us to rebuild, retest, and redeploy the Computational Geometry Application.

Single Responsibility Principle – Example 2

- Separate the two responsibilities into two separate classes
 - Moves the computational portions of Rectangle into the GeometricRectangle class.
- Now changes made to the way rectangles are rendered cannot affect the ComputationalGeometry Application.



What is a Responsibility?

- “A reason for change.”
- If you can think of more than one motive for changing a class, then that class has more than one responsibility.

```
interface Modem  
{  
    void dial(String pno) ;  
    void hangup() ;  
    void send(char c) ;  
    char rcv() ;  
}
```

Modem Responsibilities

```
interface Modem
{
    void dial(String pno) ;
    void hangup() ;
    void send(char c) ;
    char recv() ;
}
```

- Two responsibilities:
 - connection management (dial and hangup functions)
 - data communication (send and recv functions)
- They have little in common
 - may change for different reason
 - will be called from different parts of the applications
- They will change for different reasons.

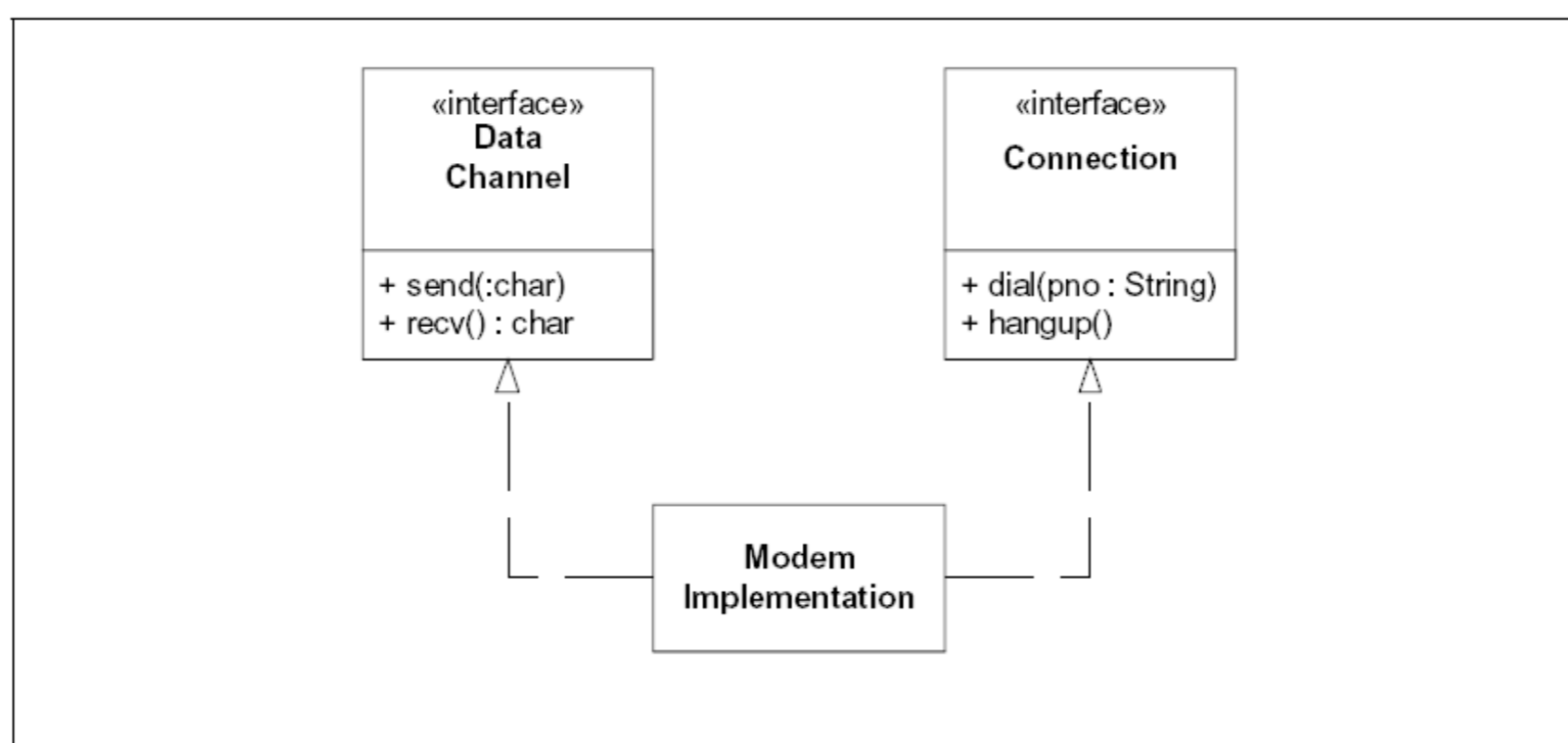
Should the responsibilities be separated?

- It depends!
- How do you foresee the application changing?
 - e.g. could the signature of the connection methods potentially change, without any change to the send/receive mechanism?

```
interface Modem
{
    void dial(String pno);
    void hangup();
    void send(char c);
    char recv();
}
```

Should the responsibilities be separated?

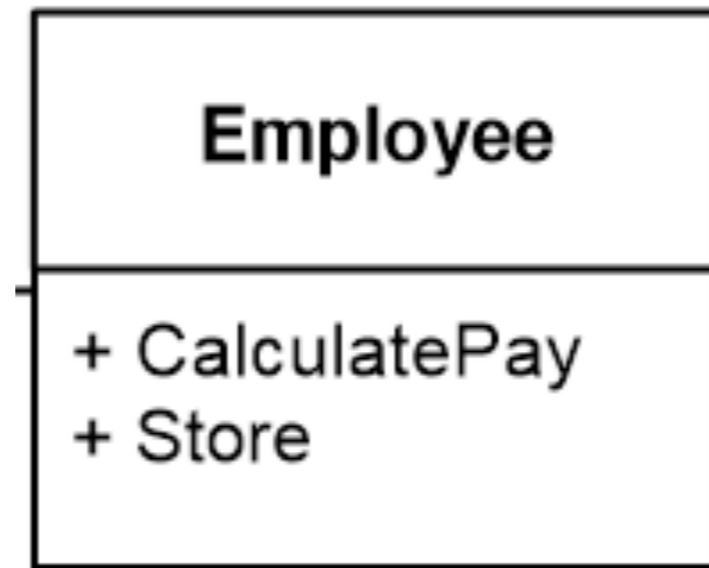
- If the application can change in ways that cause the two responsibilities to change at different times → separate the responsibilities.
- Separation here is at interface level and not class level.



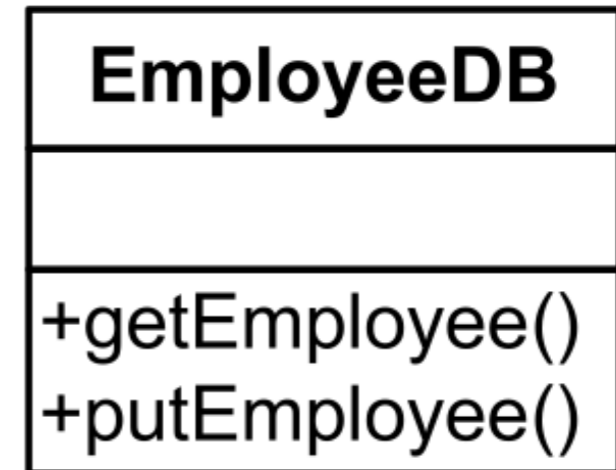
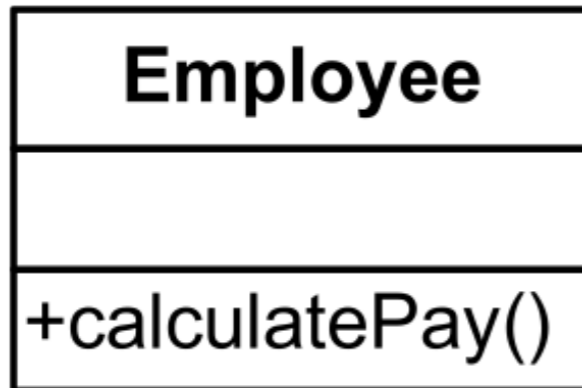
- **CAUTION: Needless complexity** can occur when there is no **foreseeable** need to separate the responsibilities.

Single Responsibility Principle – Example 4

- Coupling persistence services (store) with business rules (calculatePay) could violate SRP.



Separate the Responsibilities

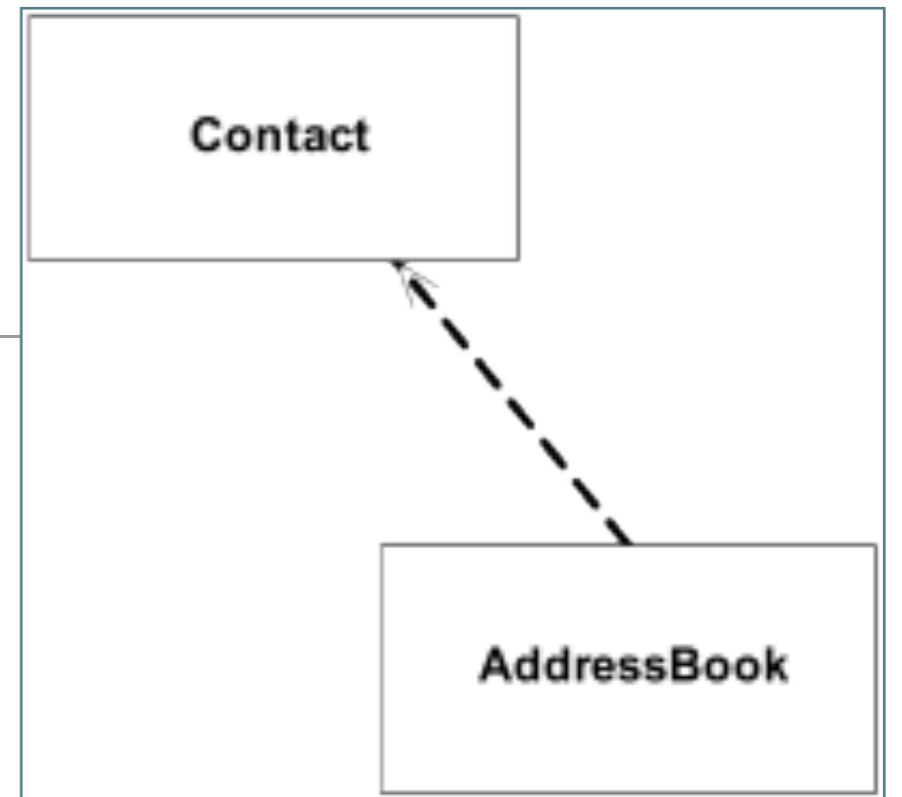


Single Responsibility Principle – Example 5

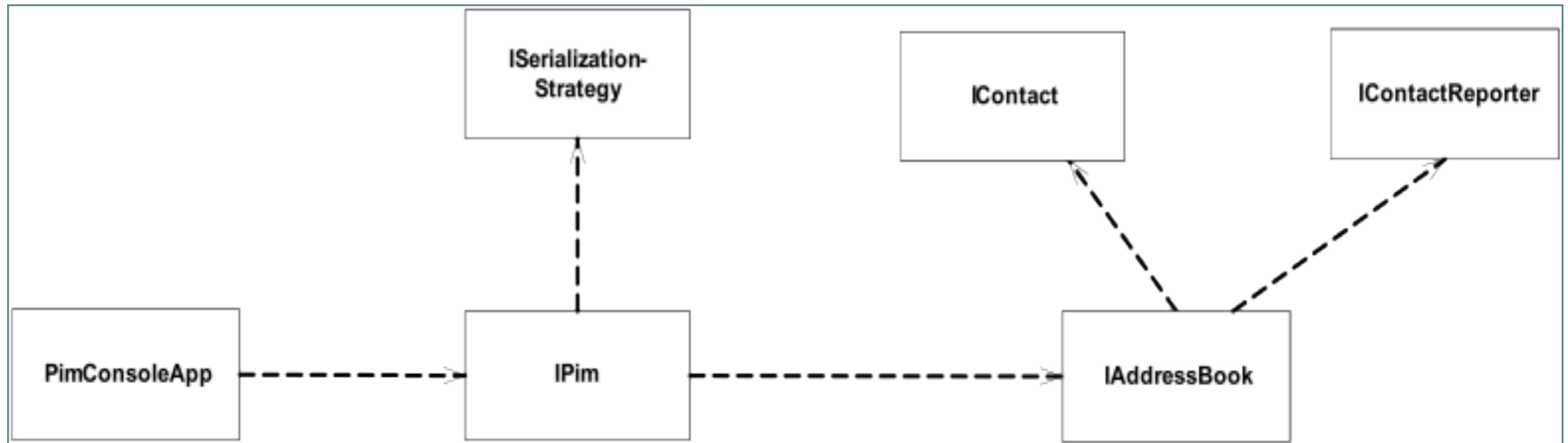
- Design an Application to manage a contact list.
- It should support:
 - Console based UI
 - Load/save to/from a file on disk
 - Simple reports and search functions

AddressBook

- Propose two classes:
 - Contact - to represent each contact
 - AddressBook - to incorporate
 - serialization
 - reporting
 - UI
 - etc...
- Violates SRP as AddressBook has multiple reasons to change
 - Data structure change (e.g. HashMap to TreeMap)
 - Serialization mechanism (e.g. binary to XML)
 - Alternative reports (e.g. different formats and content)
 - Command line syntax changes



Refactor Addressbook



IAddressBook

responsible for contact data structure

IContactReporter

responsible for format and content of reports

ISerializationStrategy

responsible for persistence

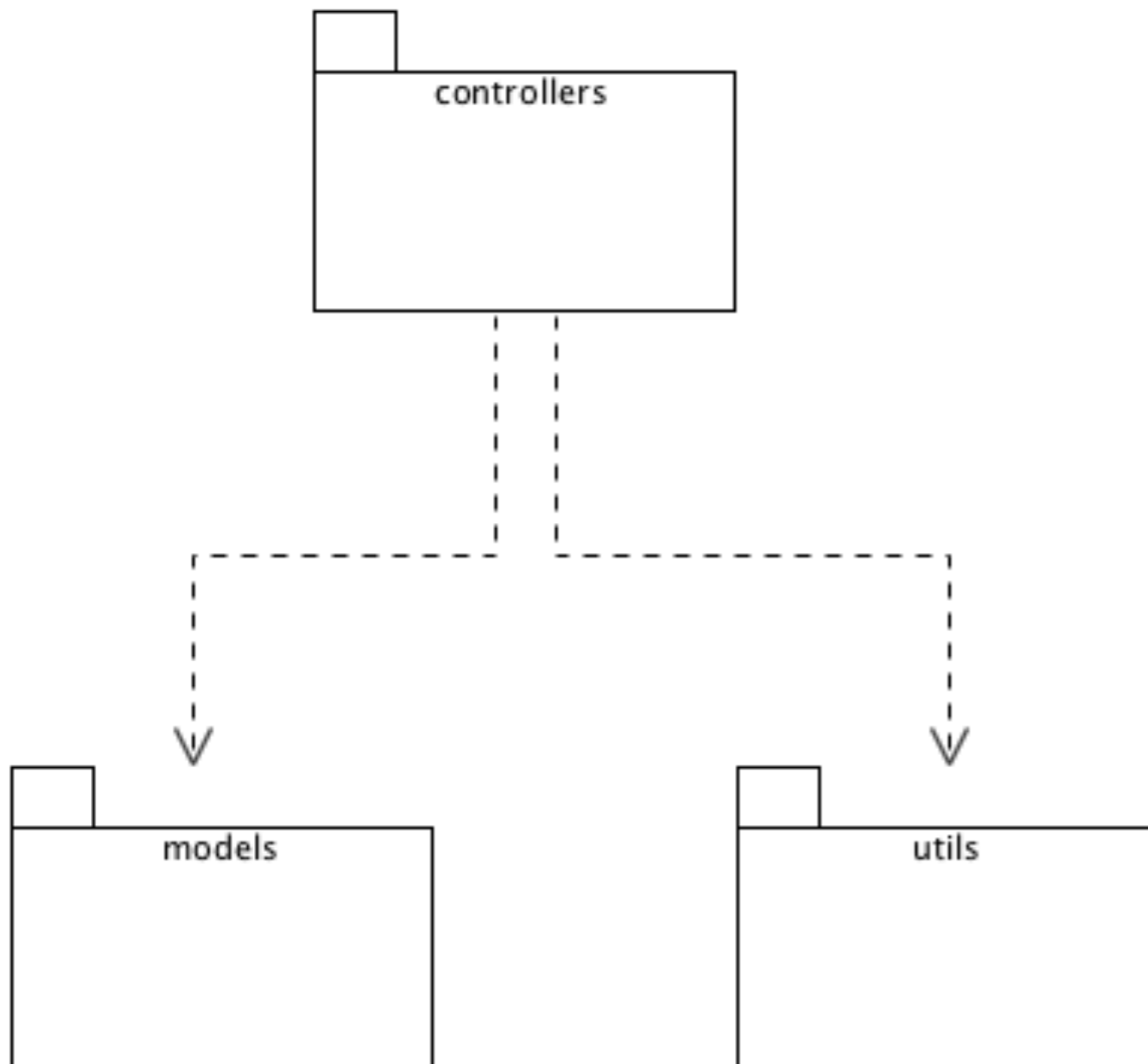
IPim

responsible for binding address book to serialization mechanism – and for exposing coherent top level functionality

PimConsoleApp

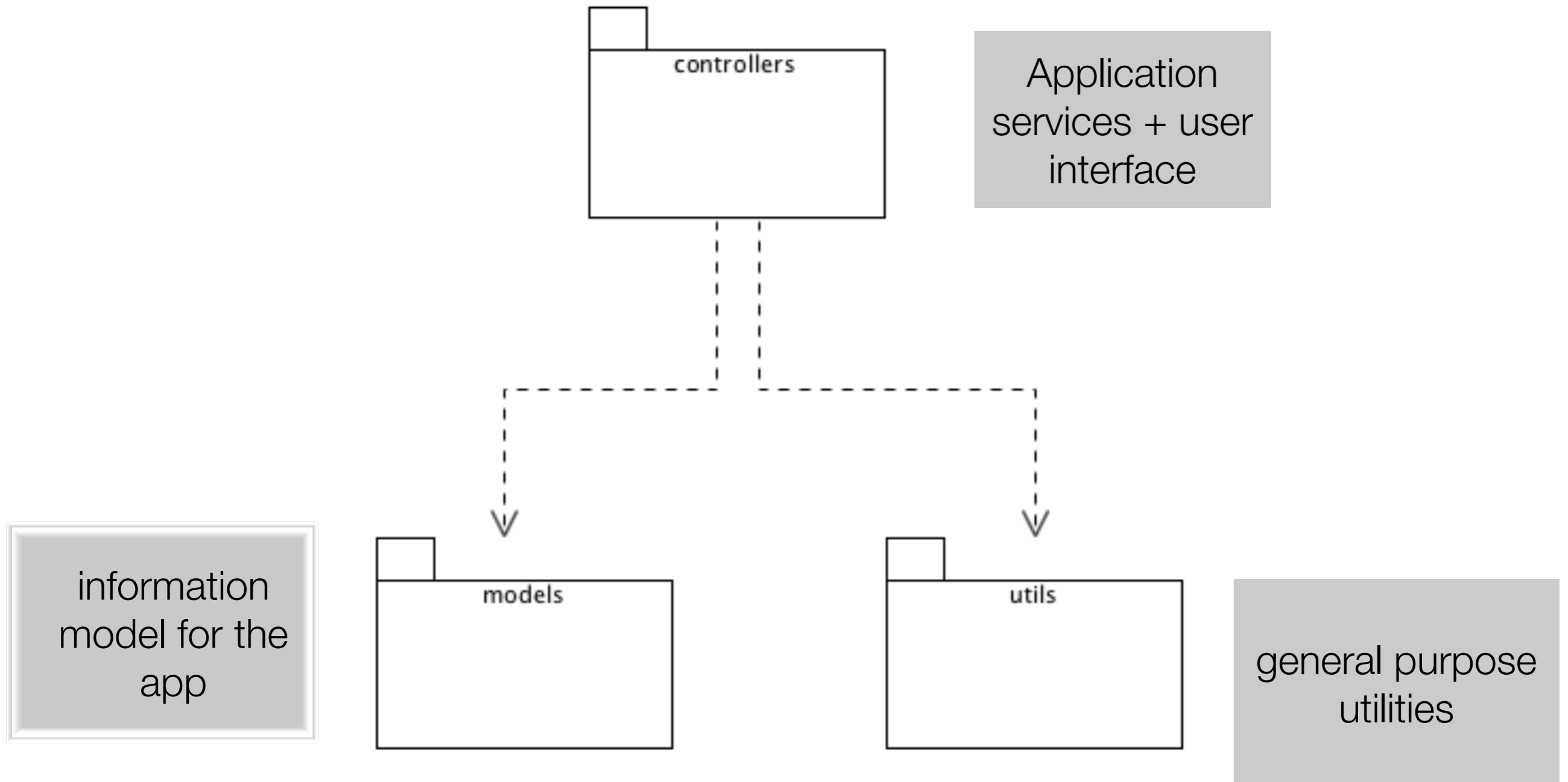
responsible binding an running application to an IPim.

Pacemaker-console-solution

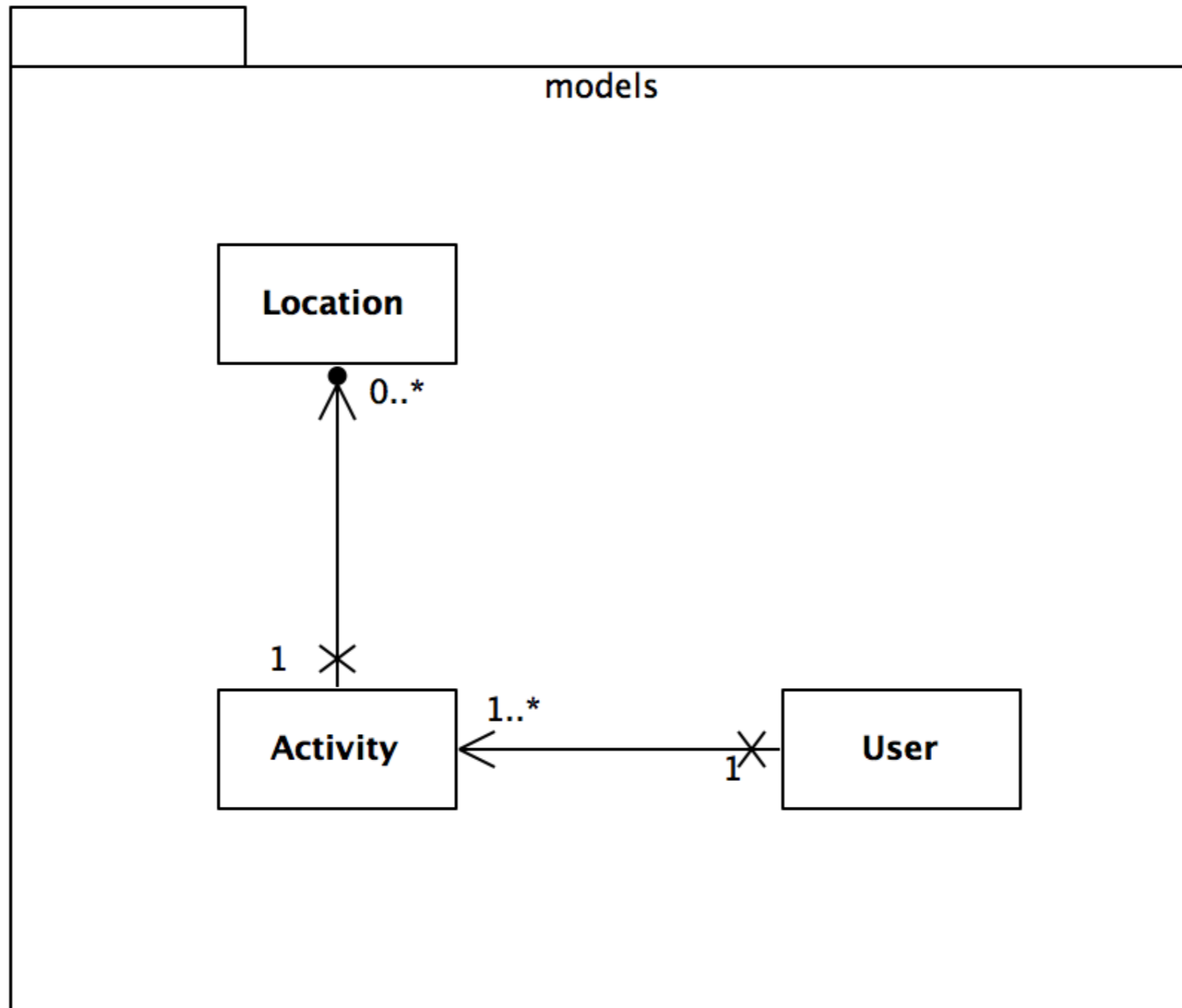


- ▼ **controllers**
 - ◉ Main
 - ◉ PacemakerAPI
 - ◉ PacemakerConsoleService
- ▼ **models**
 - ◉ Activity
 - ◉ Location
 - ◉ User
- ▼ **utils**
 - ◉ AsciiTableParser
 - ◉ Console
 - ◉ JsonSerializer
 - ◉ Serializer
 - ◉ TimeFormatters
 - ◉ XMLSerializer

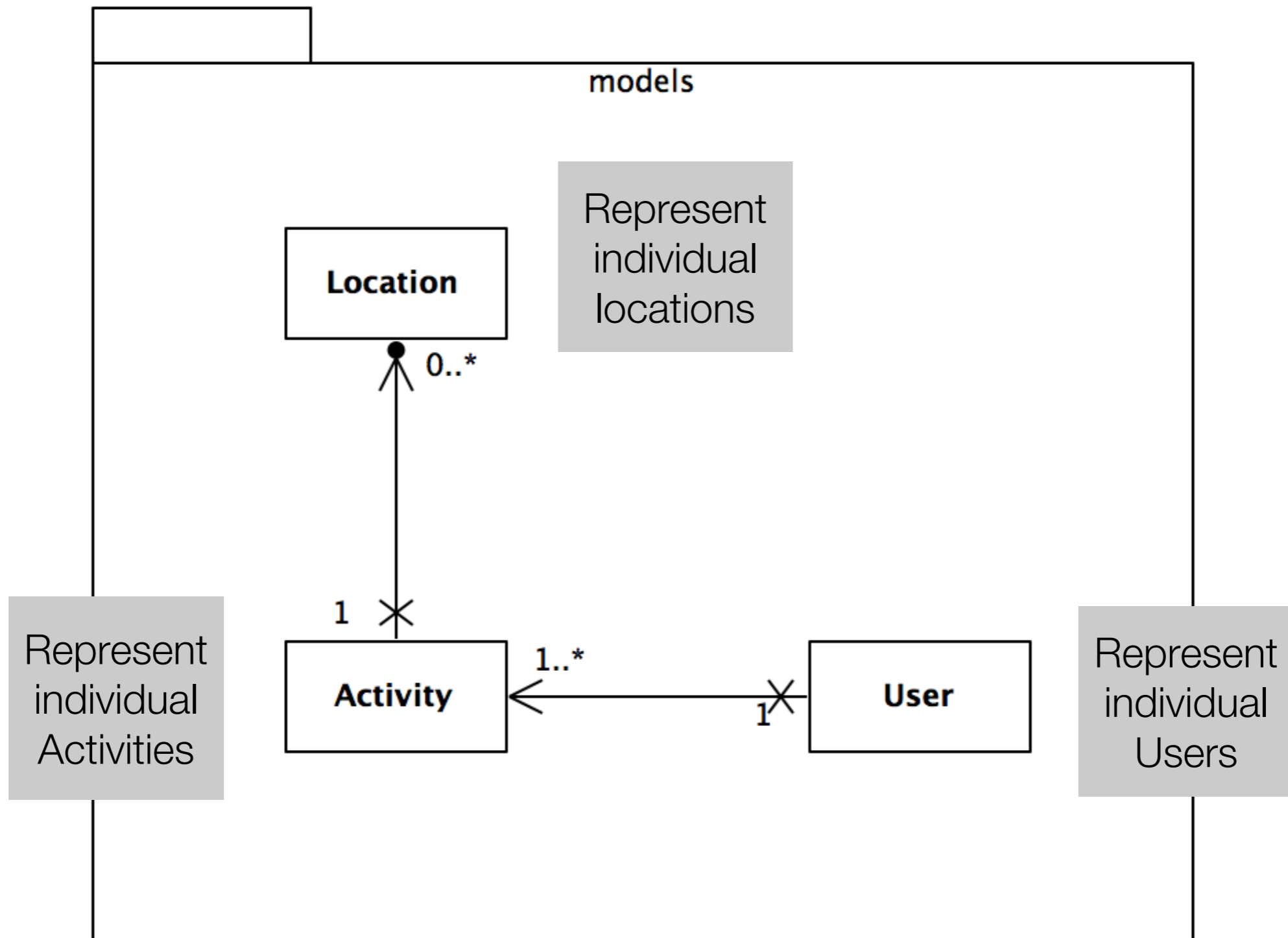
Pacemaker - package responsibilities



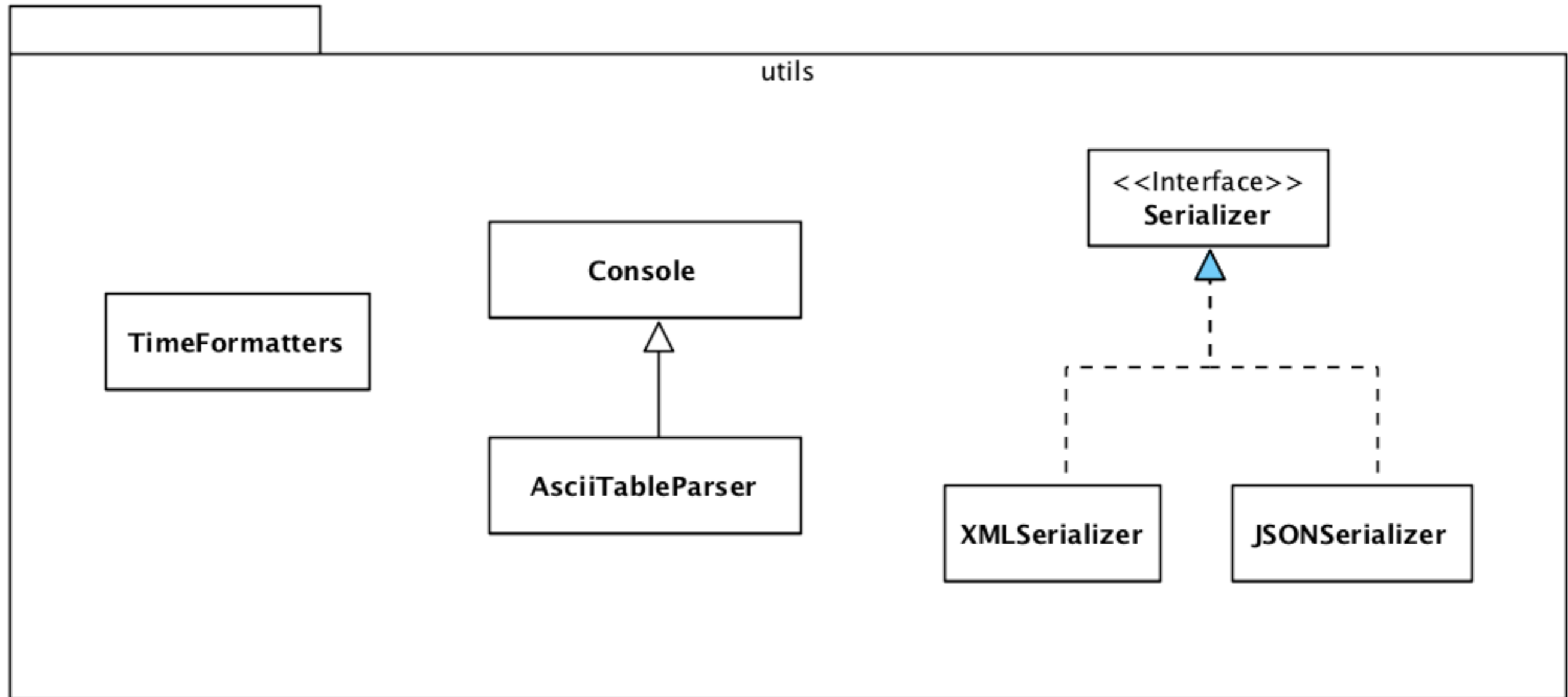
Pacemaker – Model



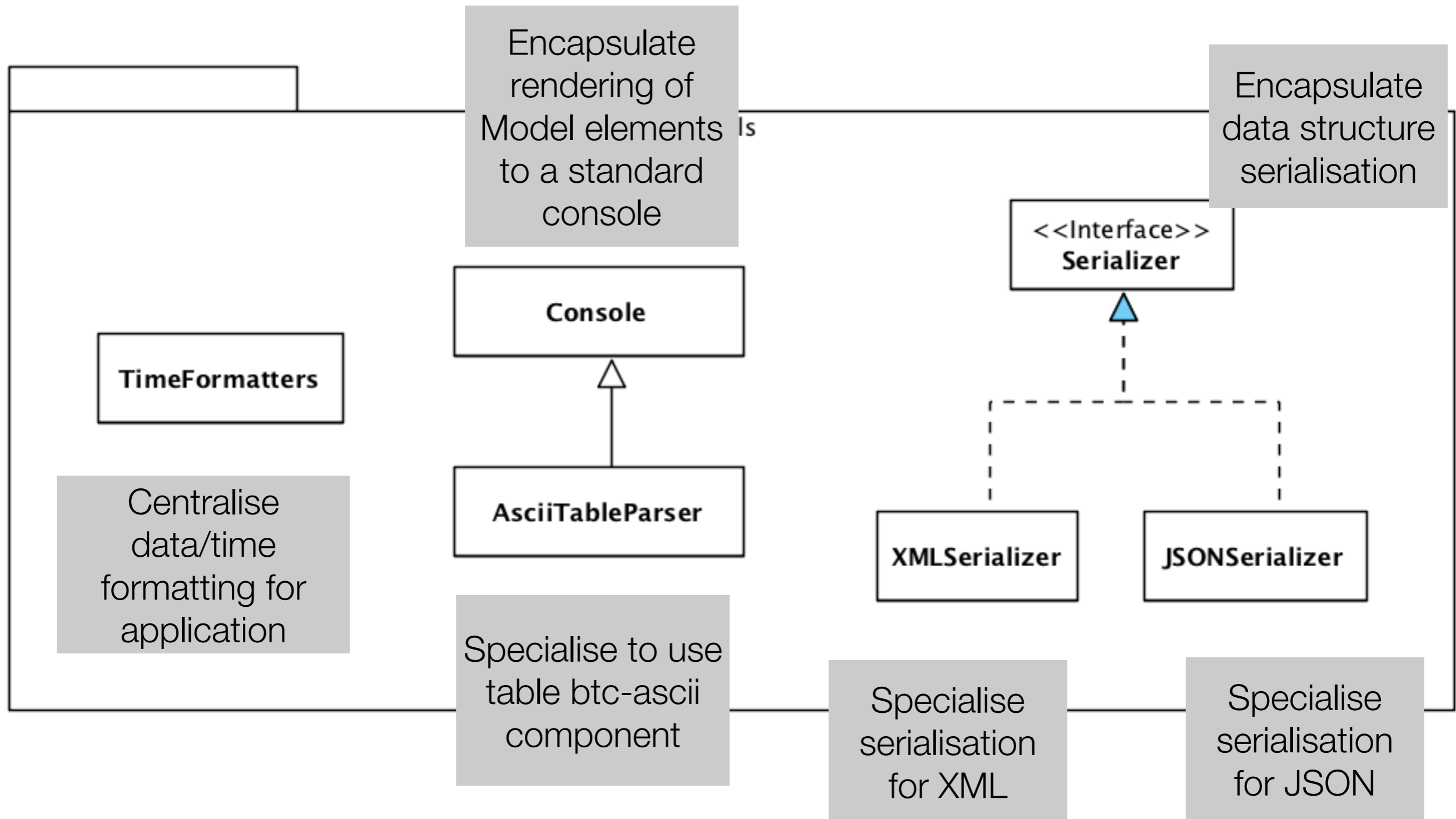
Pacemaker – Model Responsibilities

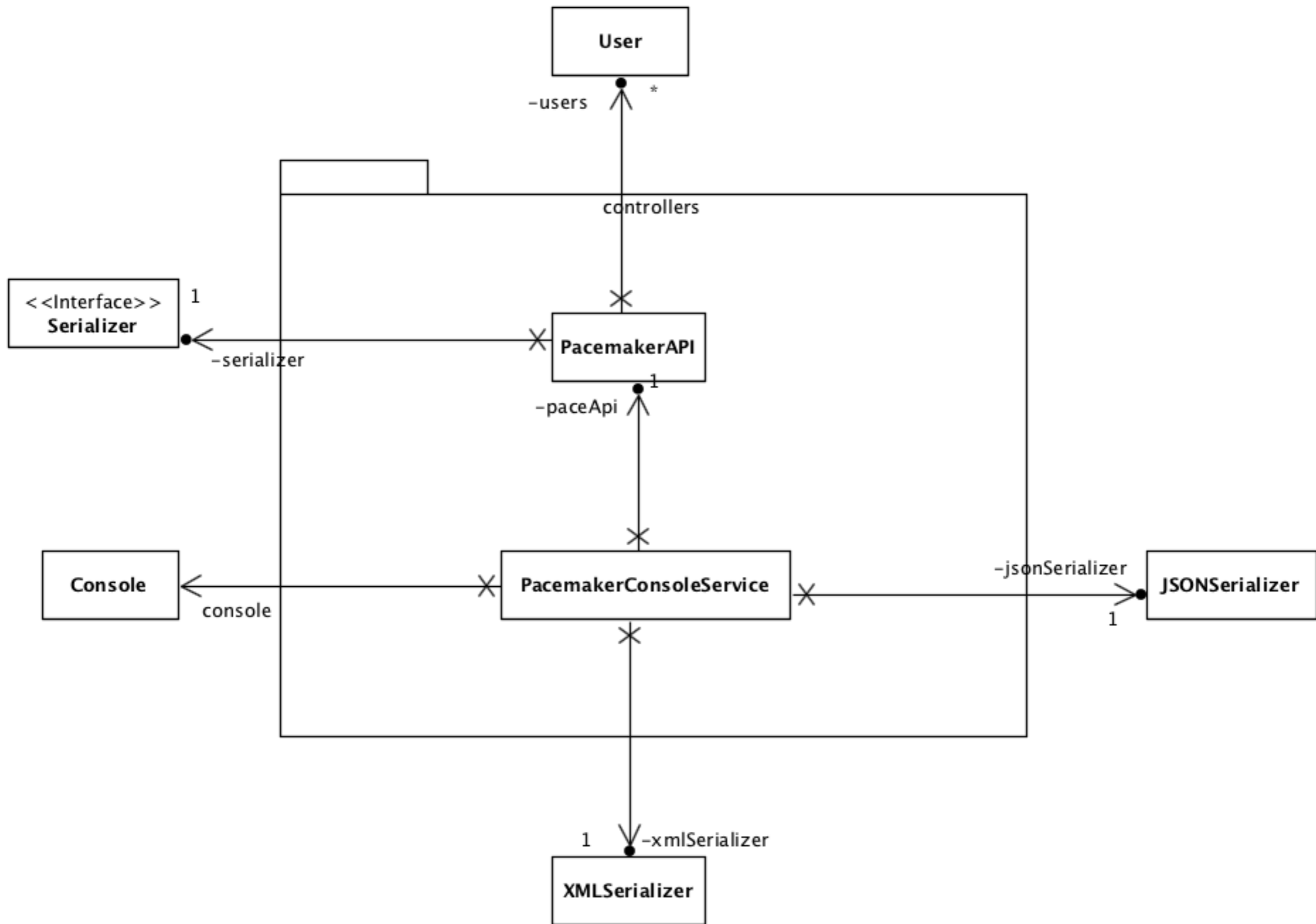


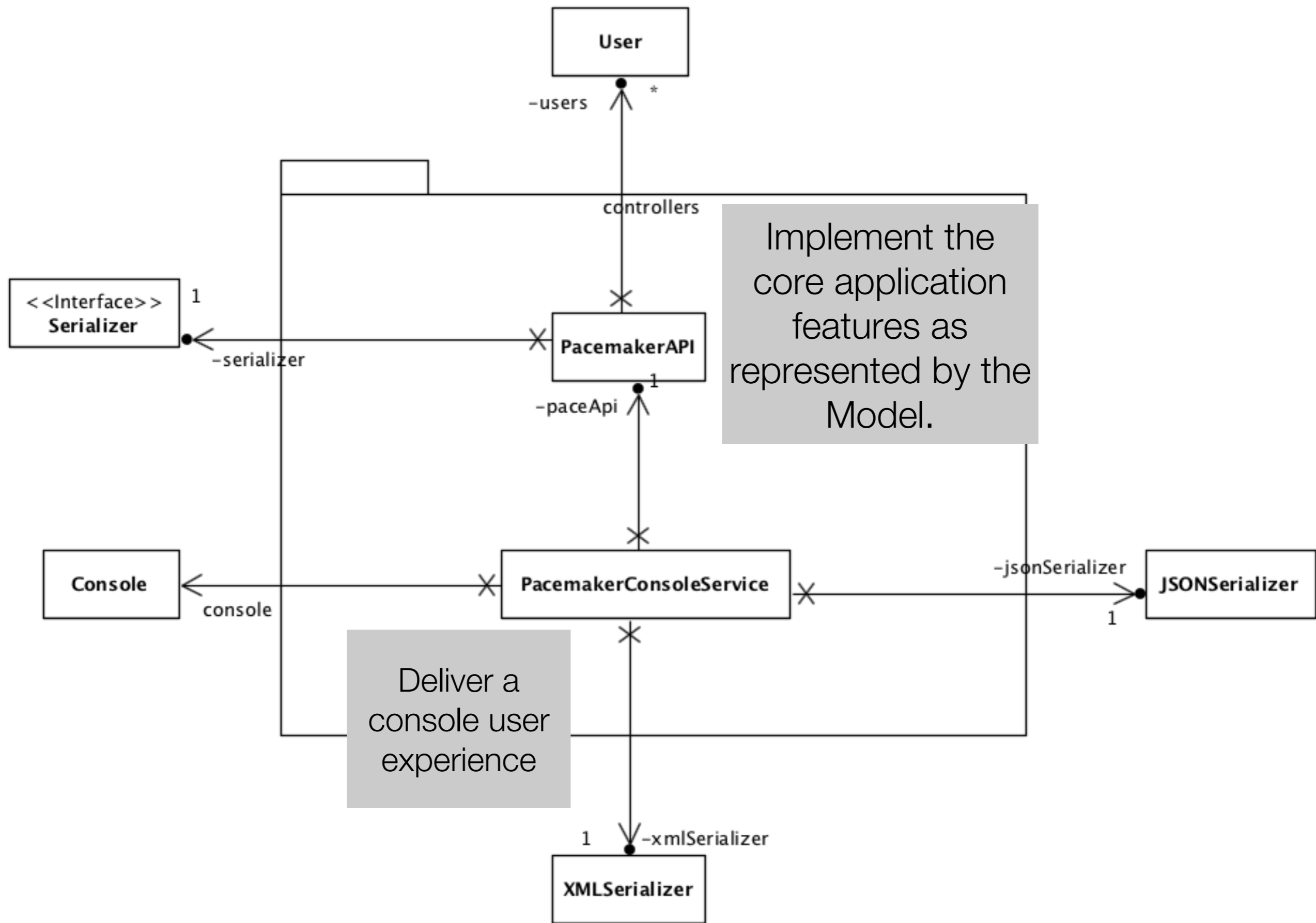
Pacemaker – Utils



Pacemaker – utils responsibilities







SRP Summary

- Changes in requirements are manifested as changes in class responsibilities.
- Therefore a ‘cohesive’ responsibility is a single axis of change – requirement changes often are restricted to a few cohesive responsibilities (in a reasonably designed system).
- Thus, to avoid coupling responsibilities that change for different reasons, a class should have only one responsibility, one reason to change.
- Violation of SRP causes spurious dependencies between modules that are hard to anticipate, in other words fragility.



Single Responsibility Principle

Just because you *can* doesn't mean you *should*.