Agile Software Development

Produced by

Eamonn de Leastar (edeleastar@wit.ie)

Department of Computing, Maths & Physics
Waterford Institute of Technology

http://www.wit.ie

http://elearning.wit.ie

Waterford Institute *of* Technology
INSTITIÚID TEICNEOLAÍOCHTA PHORT LÁIRGE

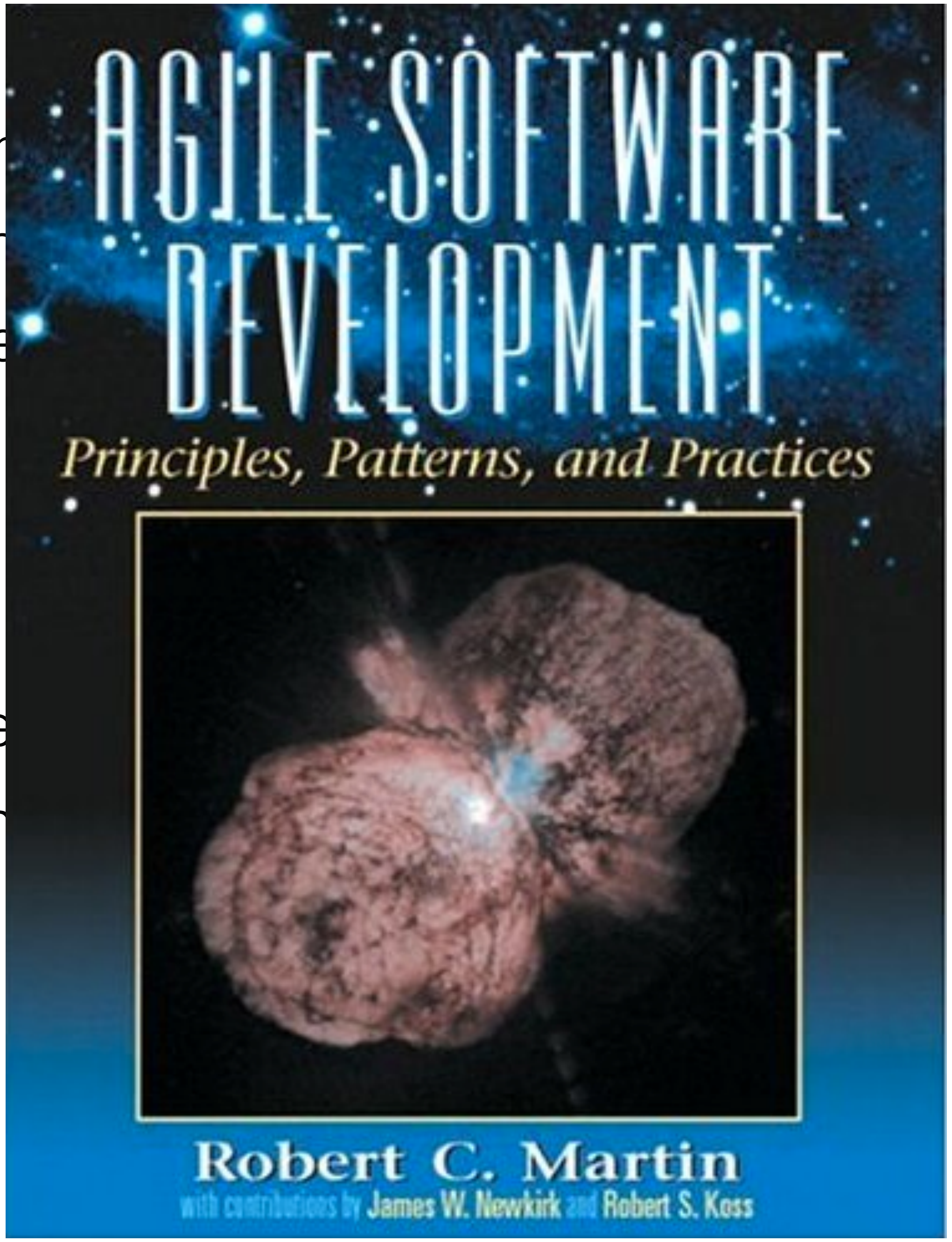
eLearning
support unit

SOLID Principles

- *The Single Responsibility Principle*
  - A class should have one, and only one, reason to change.
- *The Open Closed Principle*
  - You should be able to extend a classes behavior, without modifying it.
- *The Liskov Substitution Principle*
  - Derived classes must be substitutable for their base classes.
- *The Interface Segregation Principle*
  - Make fine grained interfaces that are client specific.
- *The Dependency Inversion Principle*
  - Depend on abstractions, not on concretions.

Source

- Agile principles, and the fourteen practices of Extrer
- Spiking, splitting, velocity, and planning iterations ar
- Test-driven development, test-first design, and acce
- Refactoring with unit testing
- Pair programming
- Agile design and design smells
- The five types of UML diagrams and how to use the
- Object-oriented package design and design pattern
- How to put all of it together for a real-world project



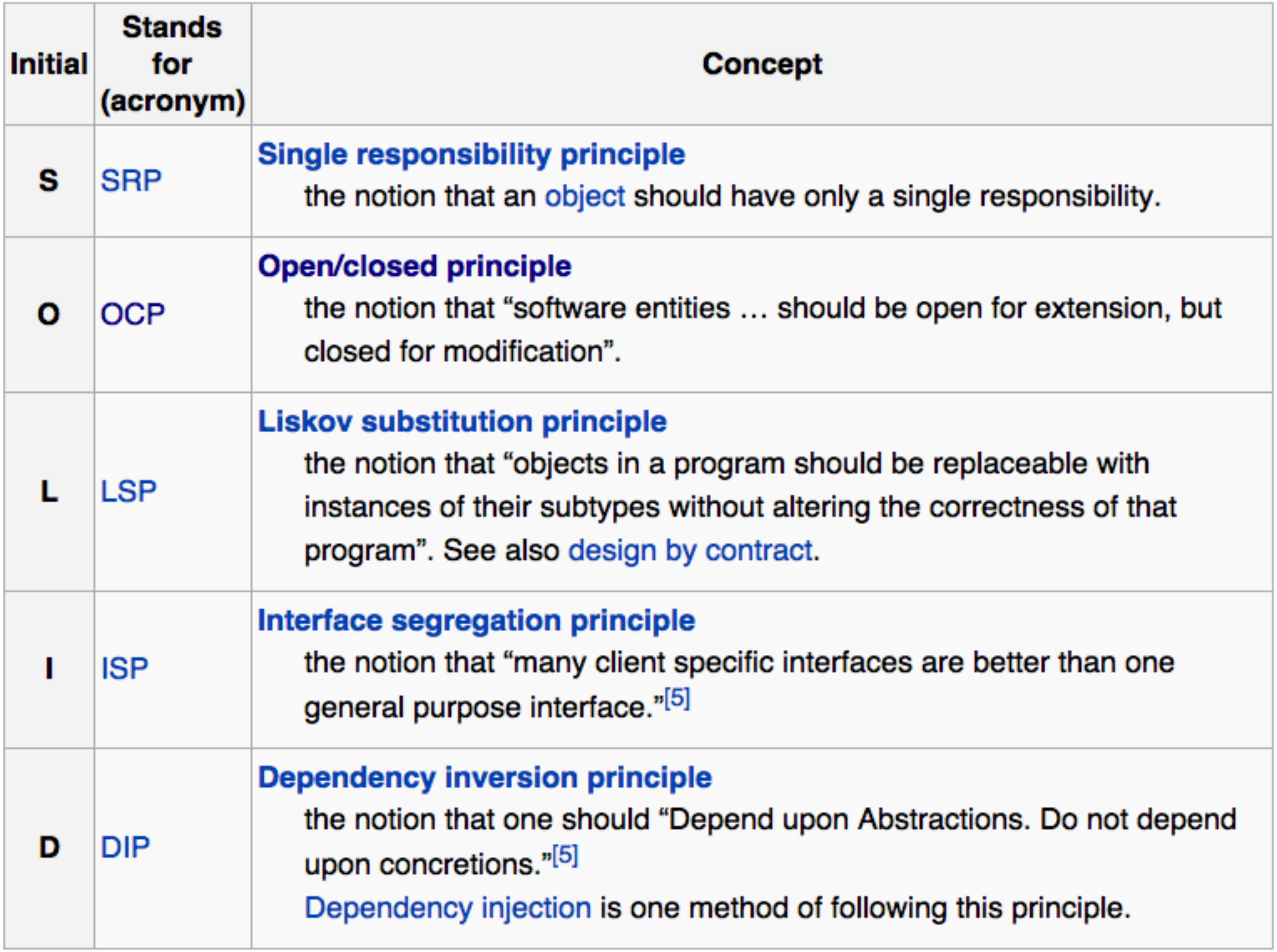
AGILE SOFTWARE DEVELOPMENT

Principles, Patterns, and Practices

Robert C. Martin

with contributions by James W. Newkirk and Robert S. Koss

Source



Initial	Stands for (acronym)	Concept
S	SRP	**Single responsibility principle** the notion that an object should have only a single responsibility.
O	OCP	**Open/closed principle** the notion that “software entities … should be open for extension, but closed for modification”.
L	LSP	**Liskov substitution principle** the notion that “objects in a program should be replaceable with instances of their subtypes without altering the correctness of that program”. See also design by contract.
I	ISP	**Interface segregation principle** the notion that “many client specific interfaces are better than one general purpose interface.”[5]
D	DIP	**Dependency inversion principle** the notion that one should “Depend upon Abstractions. Do not depend upon concretions.”[5] Dependency injection is one method of following this principle.

http://en.wikipedia.org/wiki/Solid_(object-oriented_design)

http://blog.objectmentor.com/articles/2009/02/12/getting-a-solid-start

Solid Principles in Poster Form...

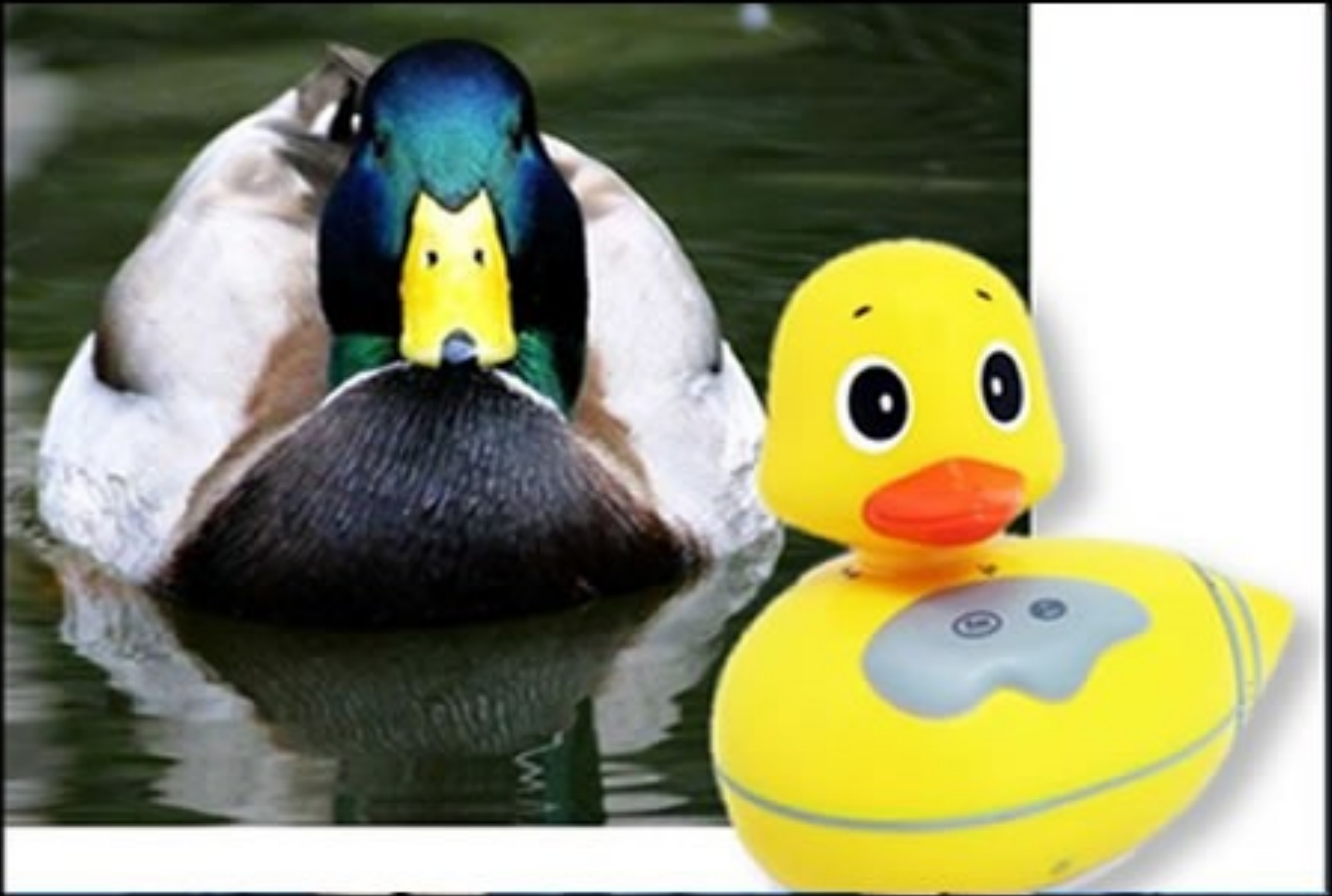
http://blogs.msdn.com/b/cdndevs/archive/2009/07/15/the-solid-principles-explained-with-motivational-posters.aspx

Single Responsibility Principle

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

Open-Closed Principle

Open-chest surgery isn’t needed when putting on a coat.

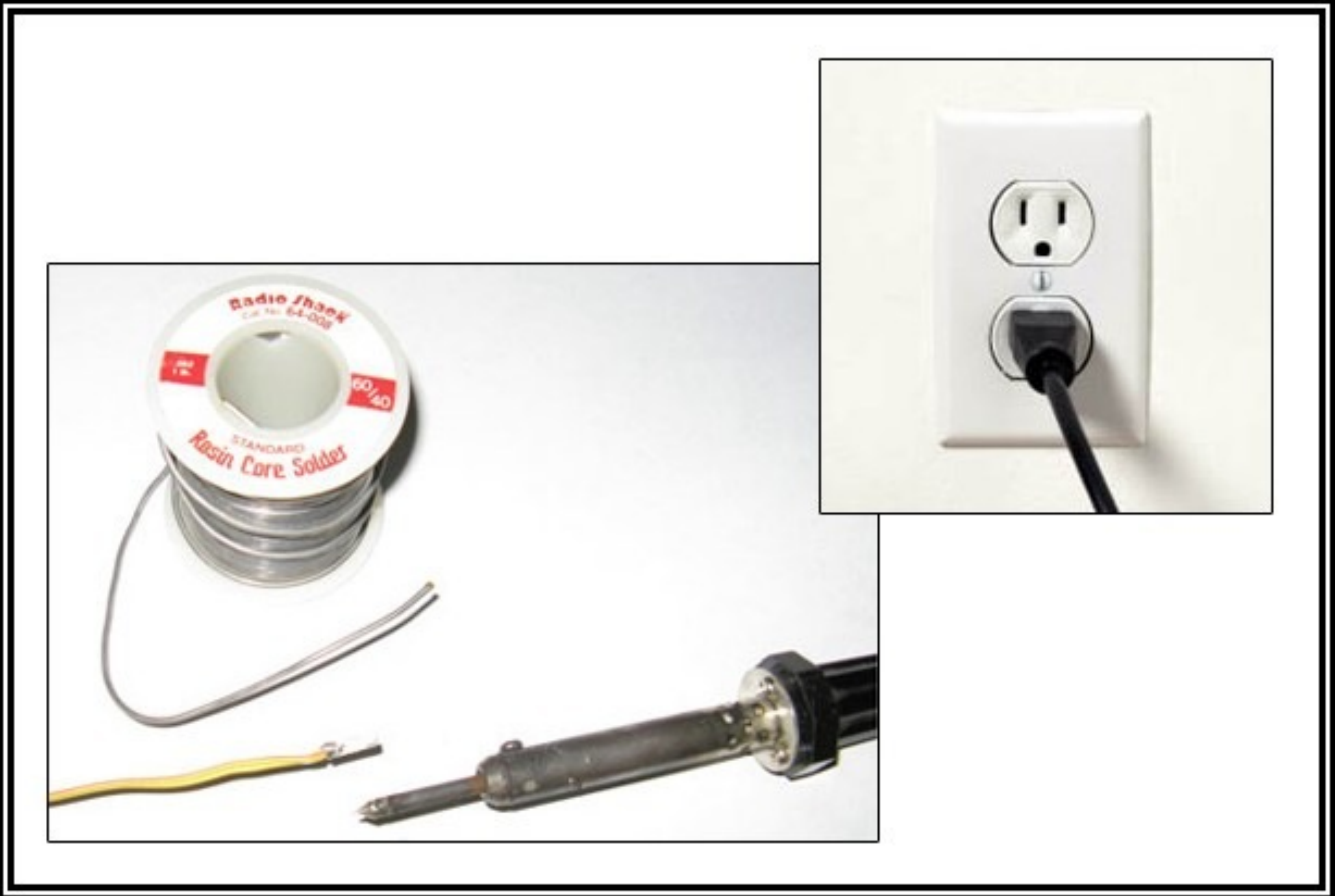


Liskov Substitution Principle

If it looks like a duck and quacks like a duck but needs batteries, you probably have the wrong abstraction.

Interface Segregation Principle

You want me to plug this in *where?*



DEPENDENCY INVERSION PRINCIPLE

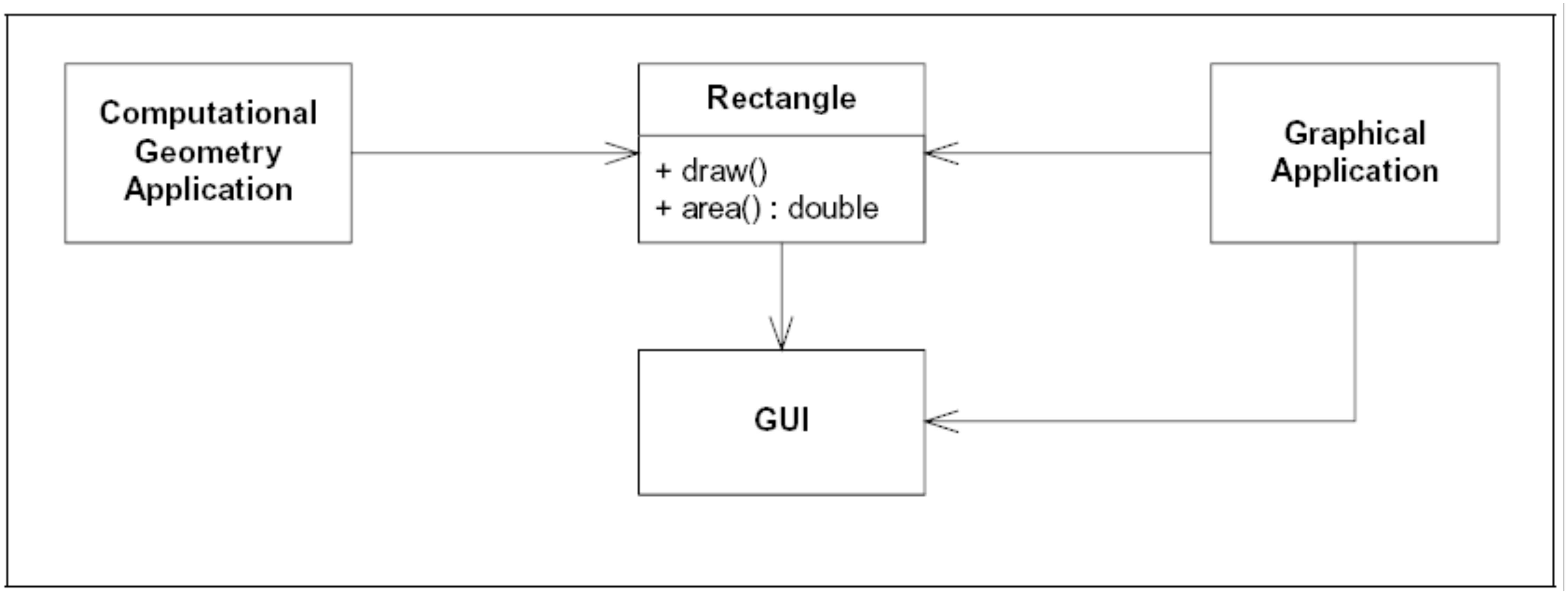
Would You Solder A Lamp Directly To The Electrical Wiring In A Wall?

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 the requirements change, that change will be manifested through a change in responsibility amongst the classes.
  - If a class assumes more than one responsibility, then there will be more than one reason for it to change.
  - Changes to one responsibility may impair or inhibit the class’ ability to meet the others.

Example

- 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.



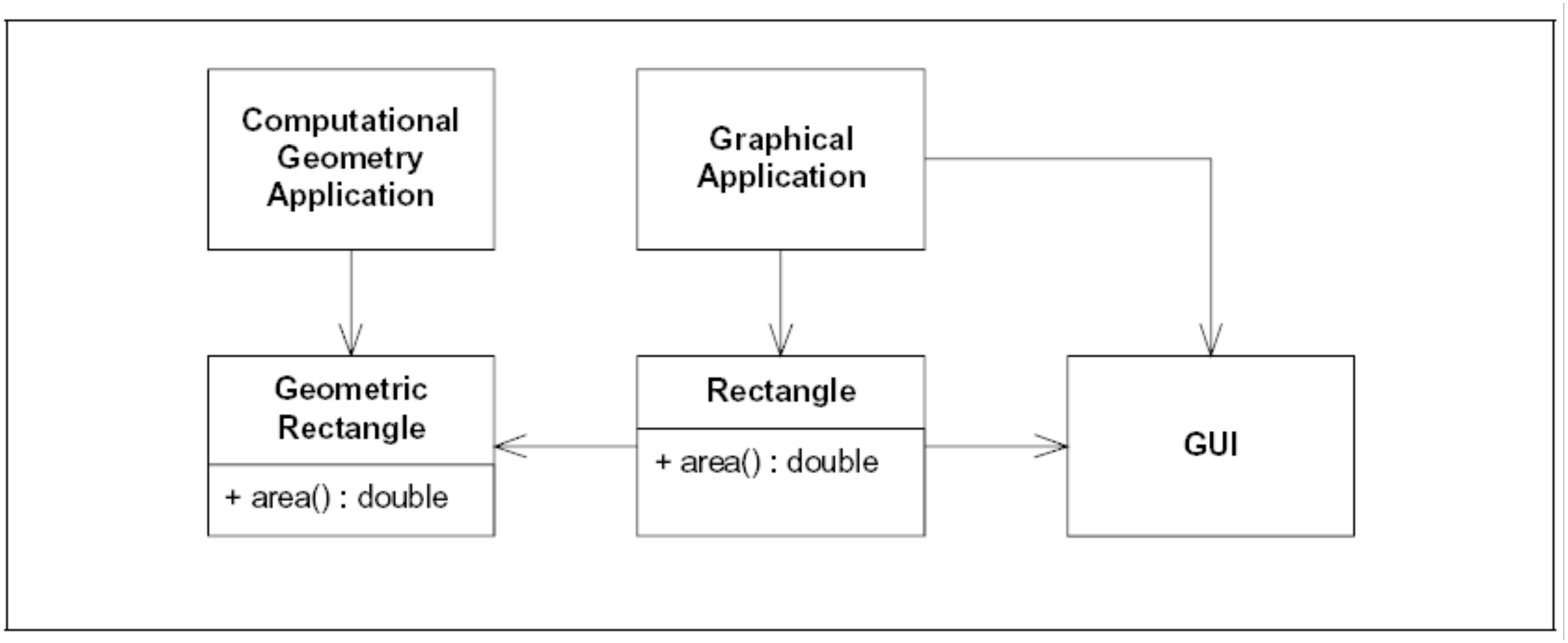
SRP Violation

- 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.

SRP Fix

- 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 recv();
}
```

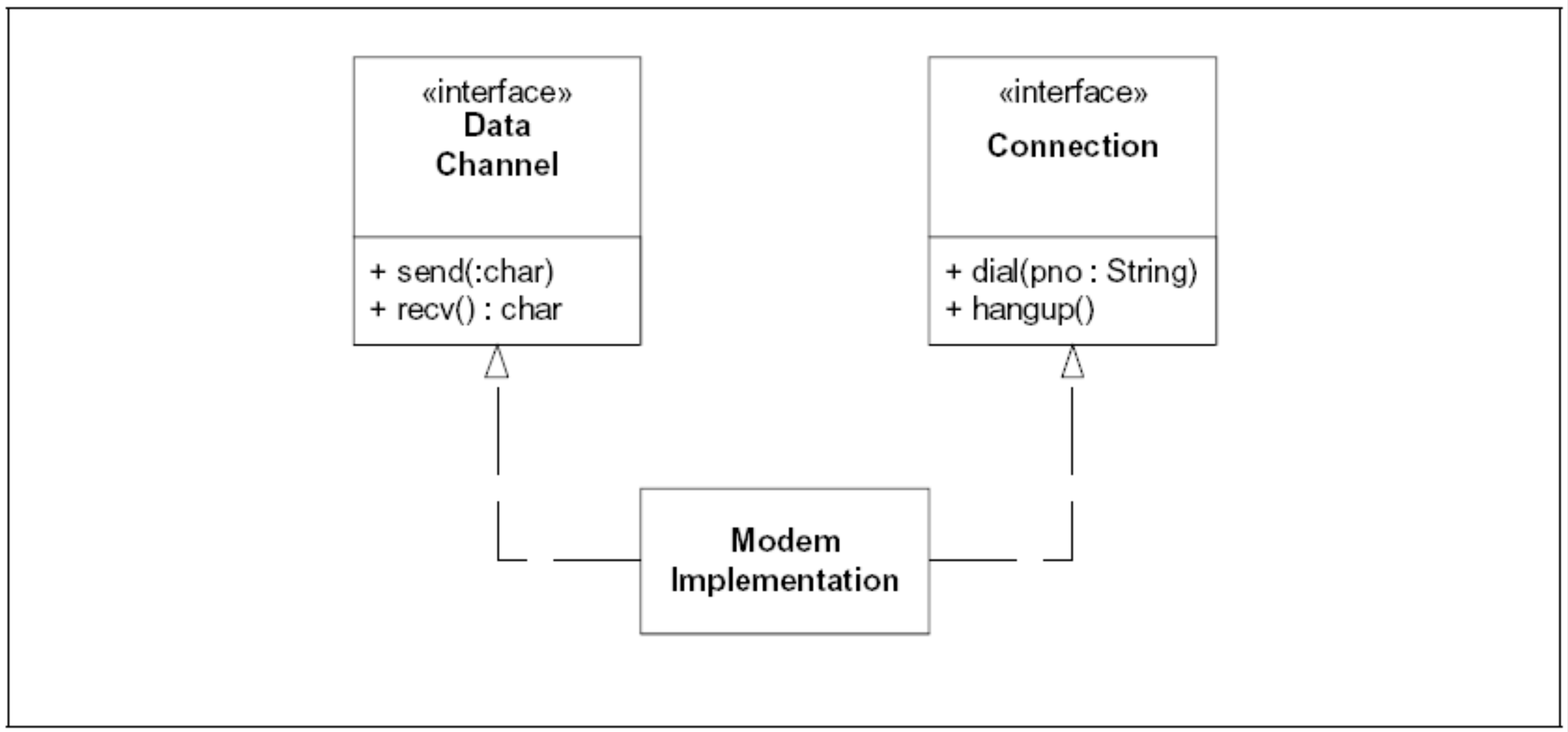
Modem Responsibilities

```
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.

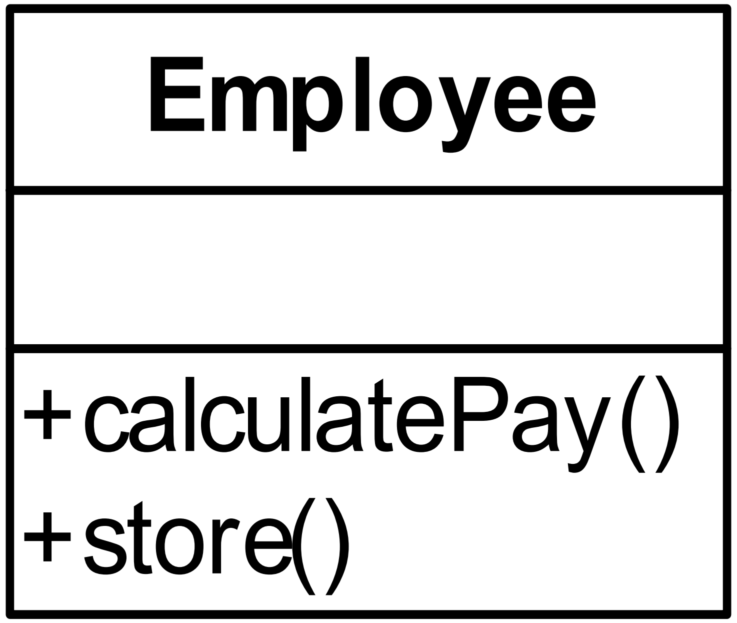
Separation of Responsibilities

- Separate the two responsibilities into two separate interfaces.
- However, we may couple the two responsibilities into a single Modem Implementation class.
- This is not necessarily desirable, but it may be necessary. (for implementation purposes)



SRP Violation?

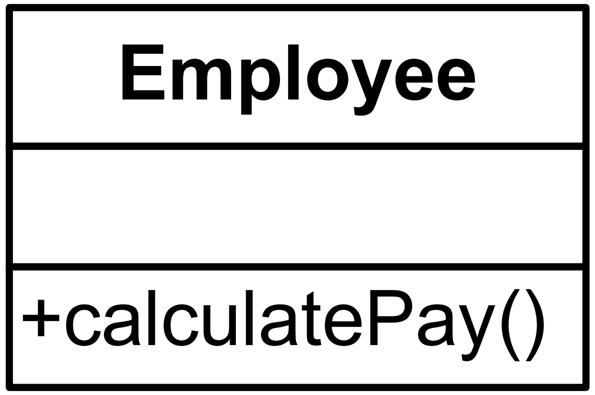
- Coupling persistence services (store) with business rules (calculatePay) violates SRP



Employee

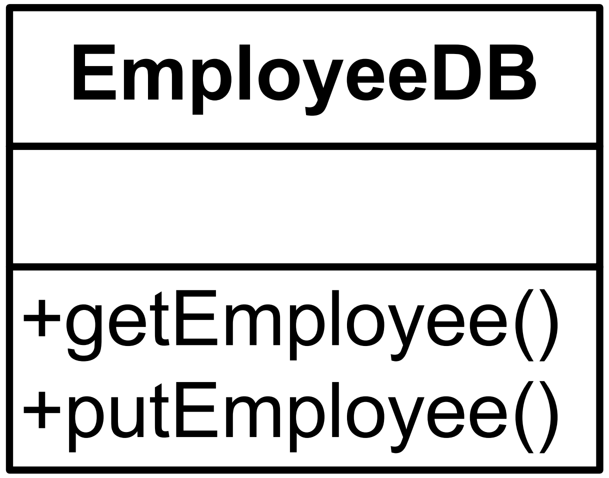
+calculatePay()
+store()

Separate Concerns



Employee

+calculatePay()



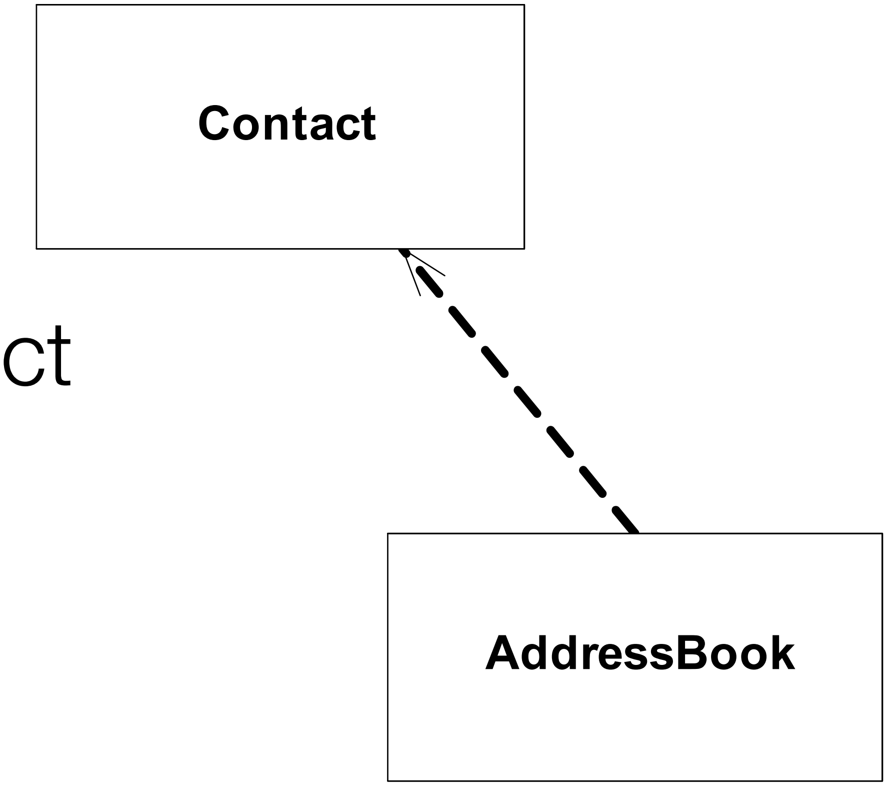
EmployeeDB

+getEmployee()
+putEmployee()

Example - Personal Information Manager

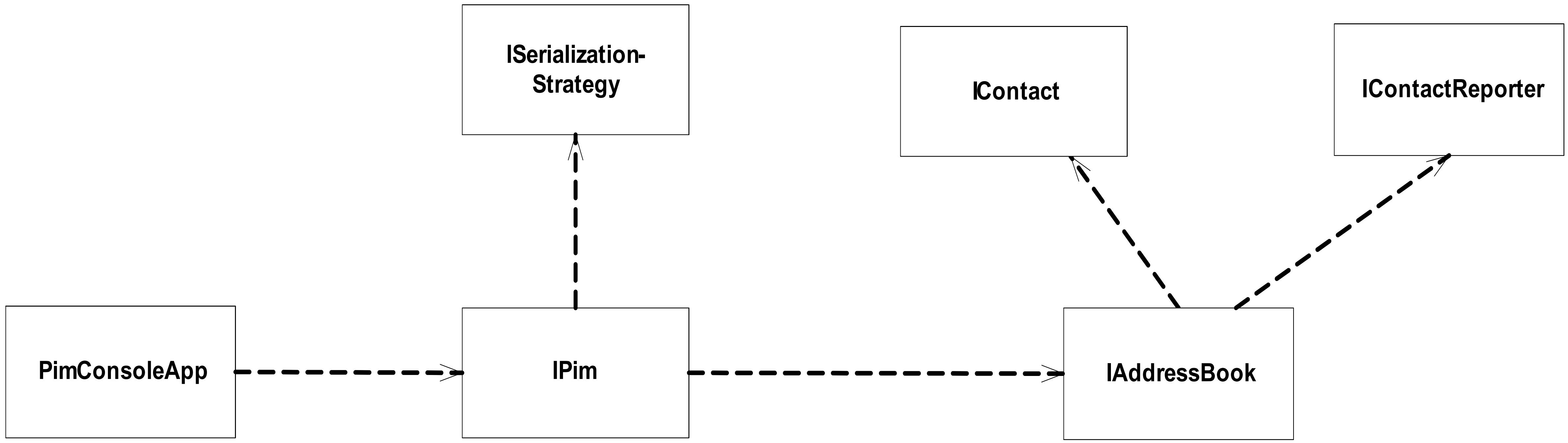
- Design an Application to manage a contact list.
- It should support:
  - Console based UI
  - Load/save to/from a file on disk
  - Simple reports and search functions.

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 (HashMap to TreeMap)
  - Serialization mechanism (binary to XML)
  - Alternative reports (different formats and content)
  - Command line syntax changes

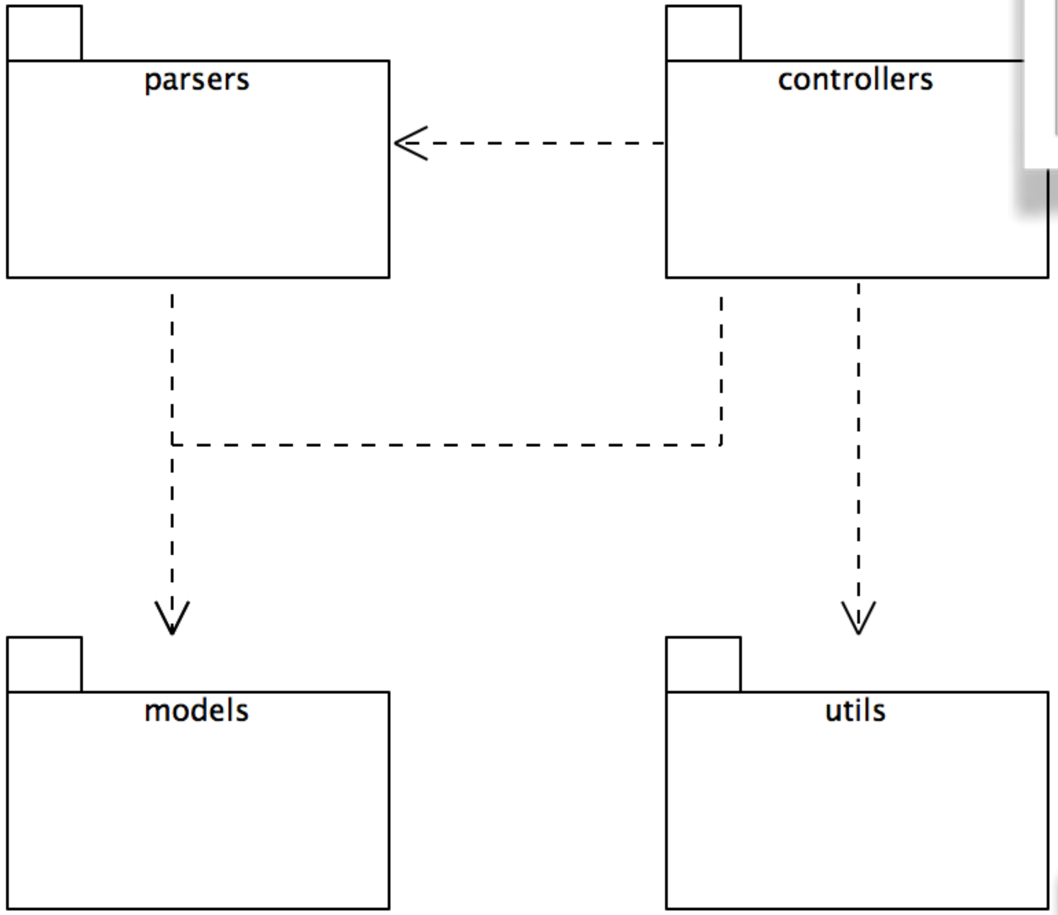
Refactor Addressbook



- AddressBook responsible for contact data structure
- ContactReporter responsible for format and content of reports
- SerializationStrategy responsible for persistence
- Pim 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 - package responsibilities

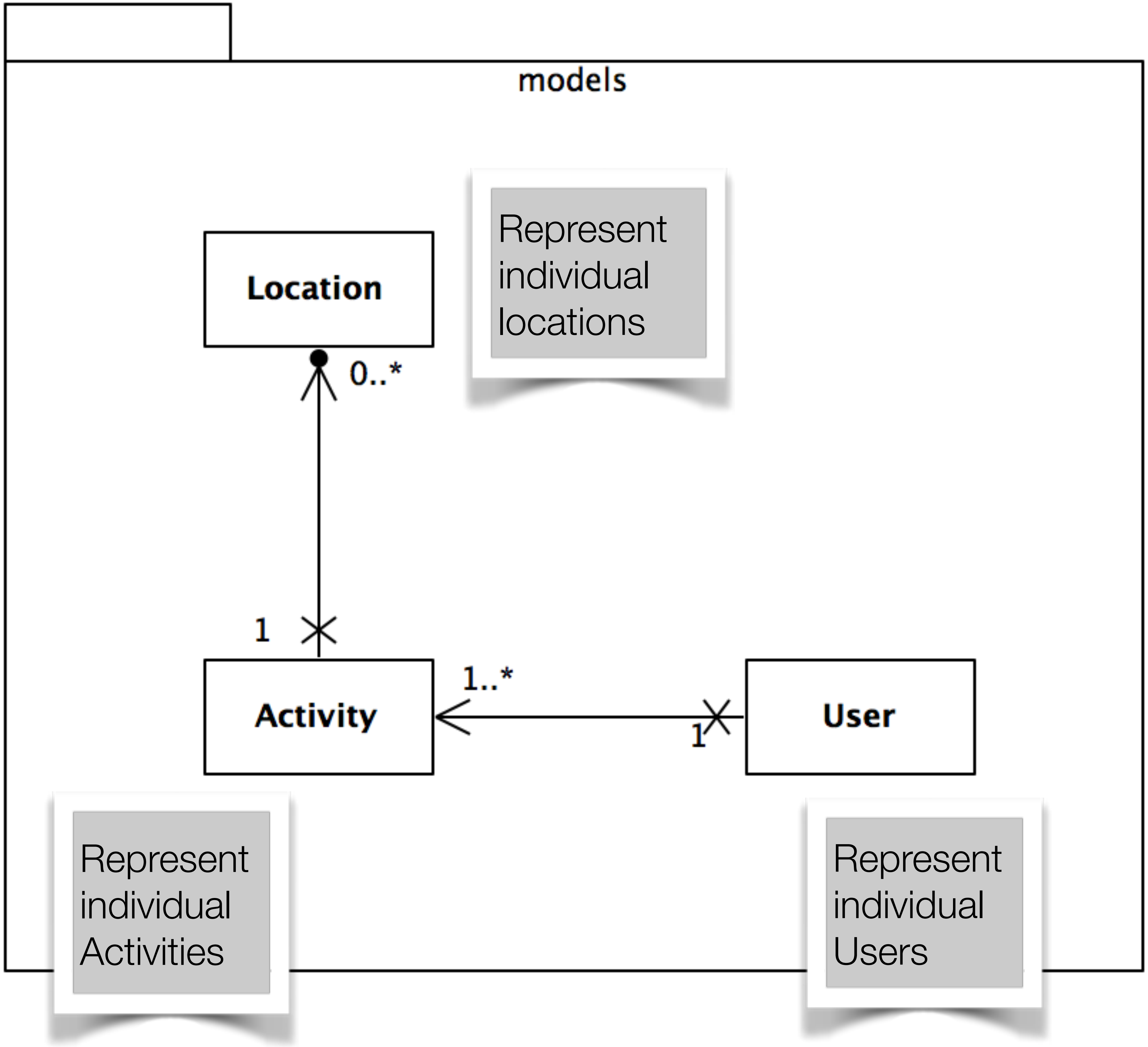
transform the model into various formats



Application services + user interface

information model for the app

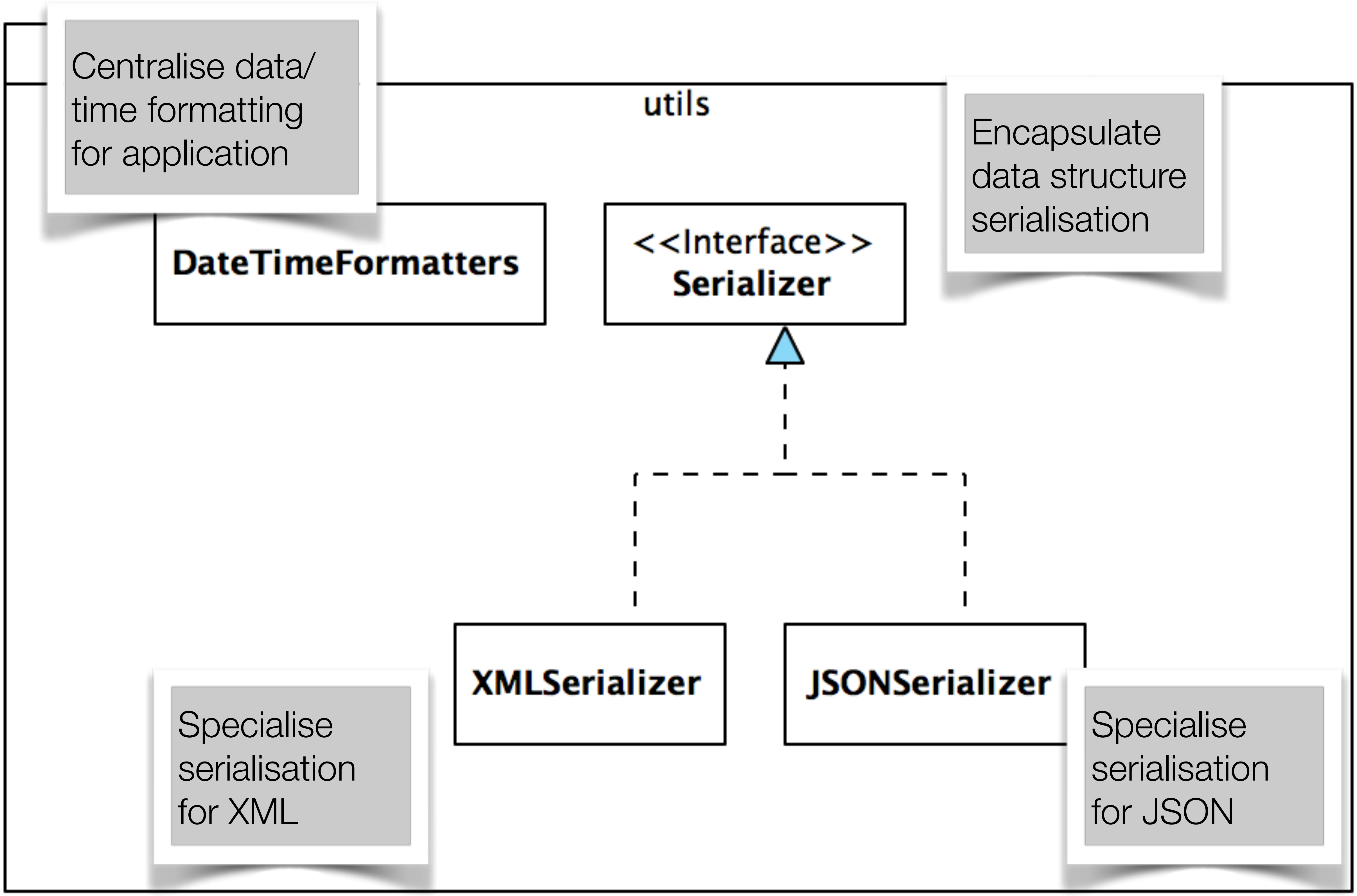
general purpose application independent utilities



Represent individual locations

Represent individual Activities

Represent individual Users

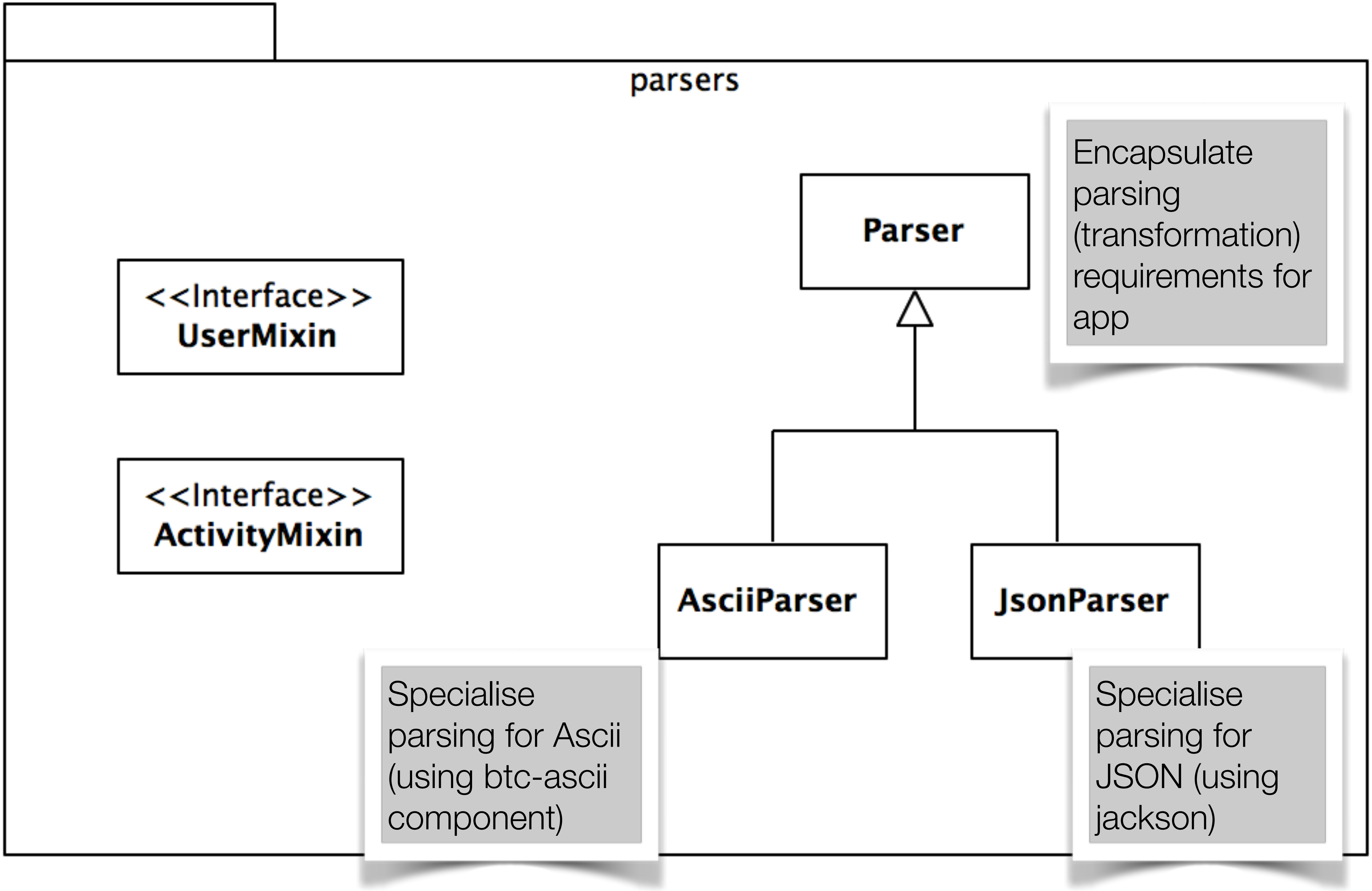


Centralise data/ time formatting for application

Encapsulate data structure serialisation

Specialise serialisation for XML

Specialise serialisation for JSON



parsers

Parser

<<Interface>>
UserMixin

<<Interface>>
ActivityMixin

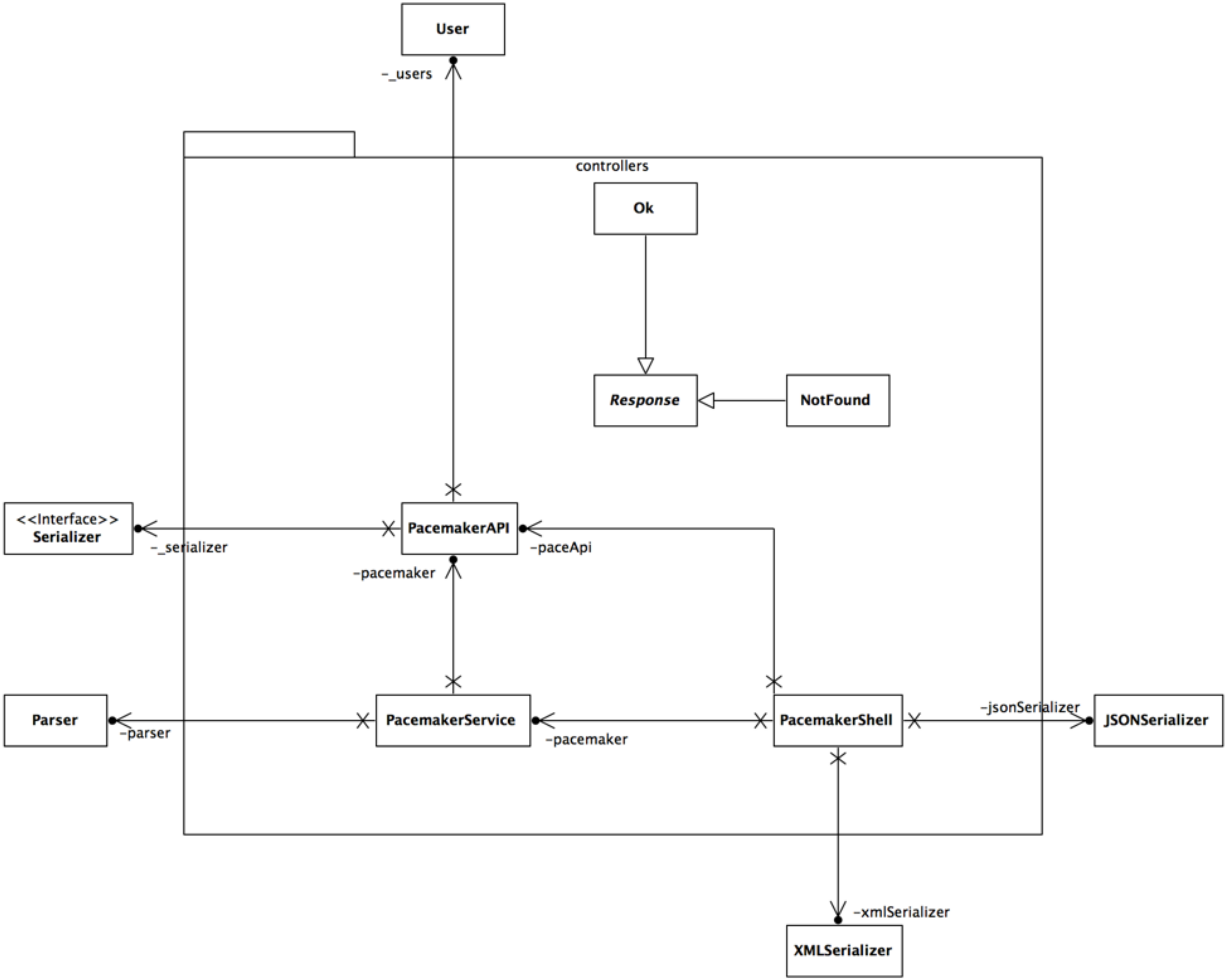
AsciiParser

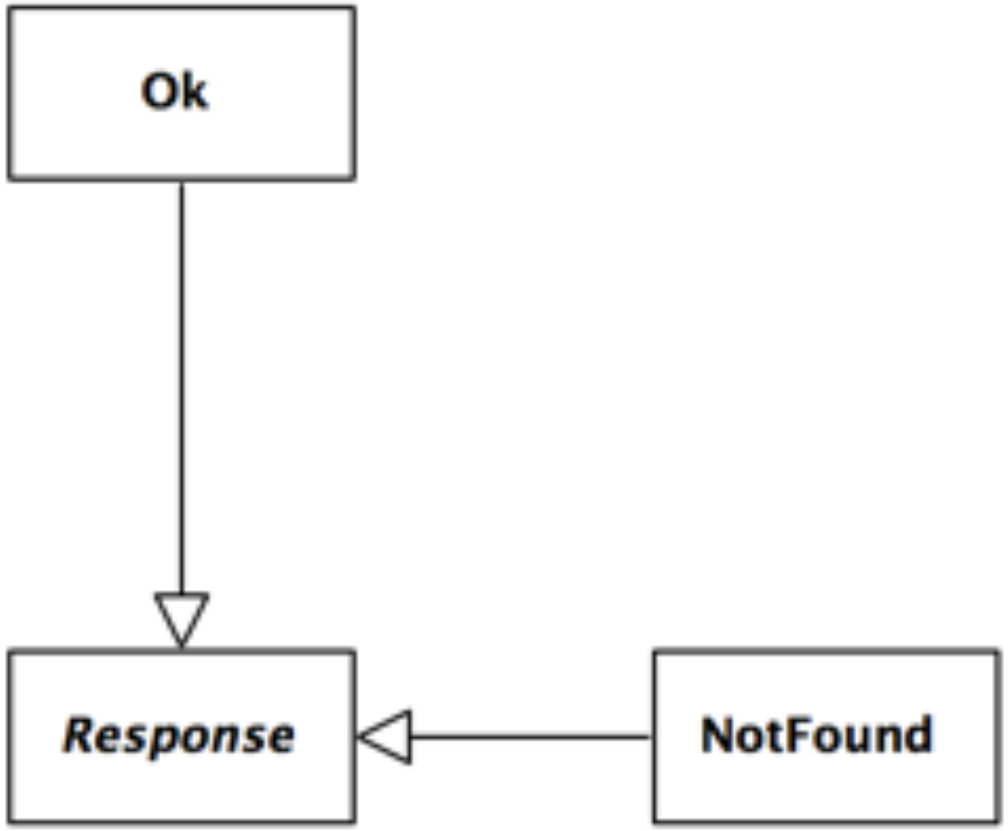
JsonParser

Encapsulate parsing (transformation) requirements for app

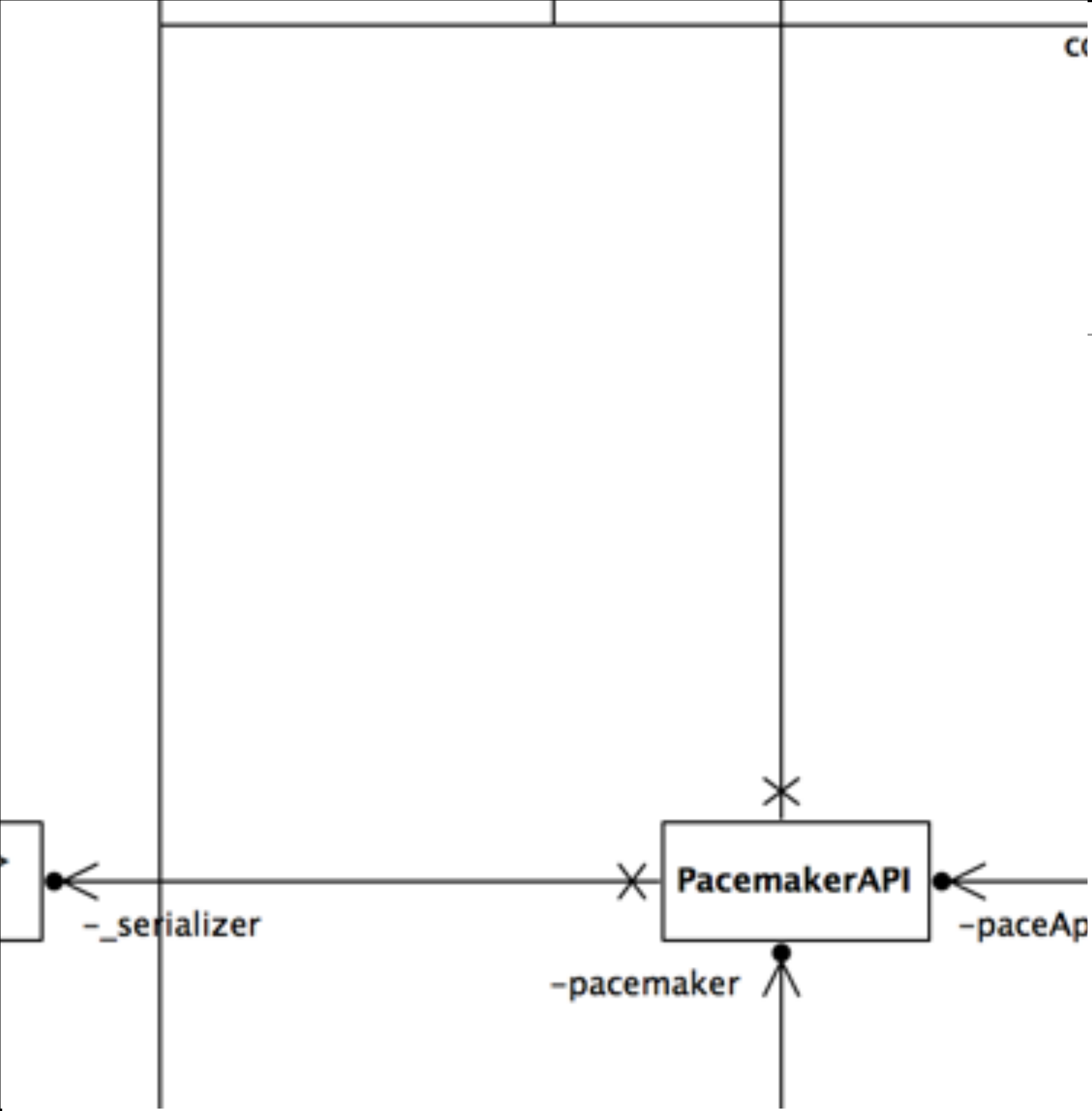
Specialise parsing for Ascii (using btc-ascii component)

Specialise parsing for JSON (using jackson)

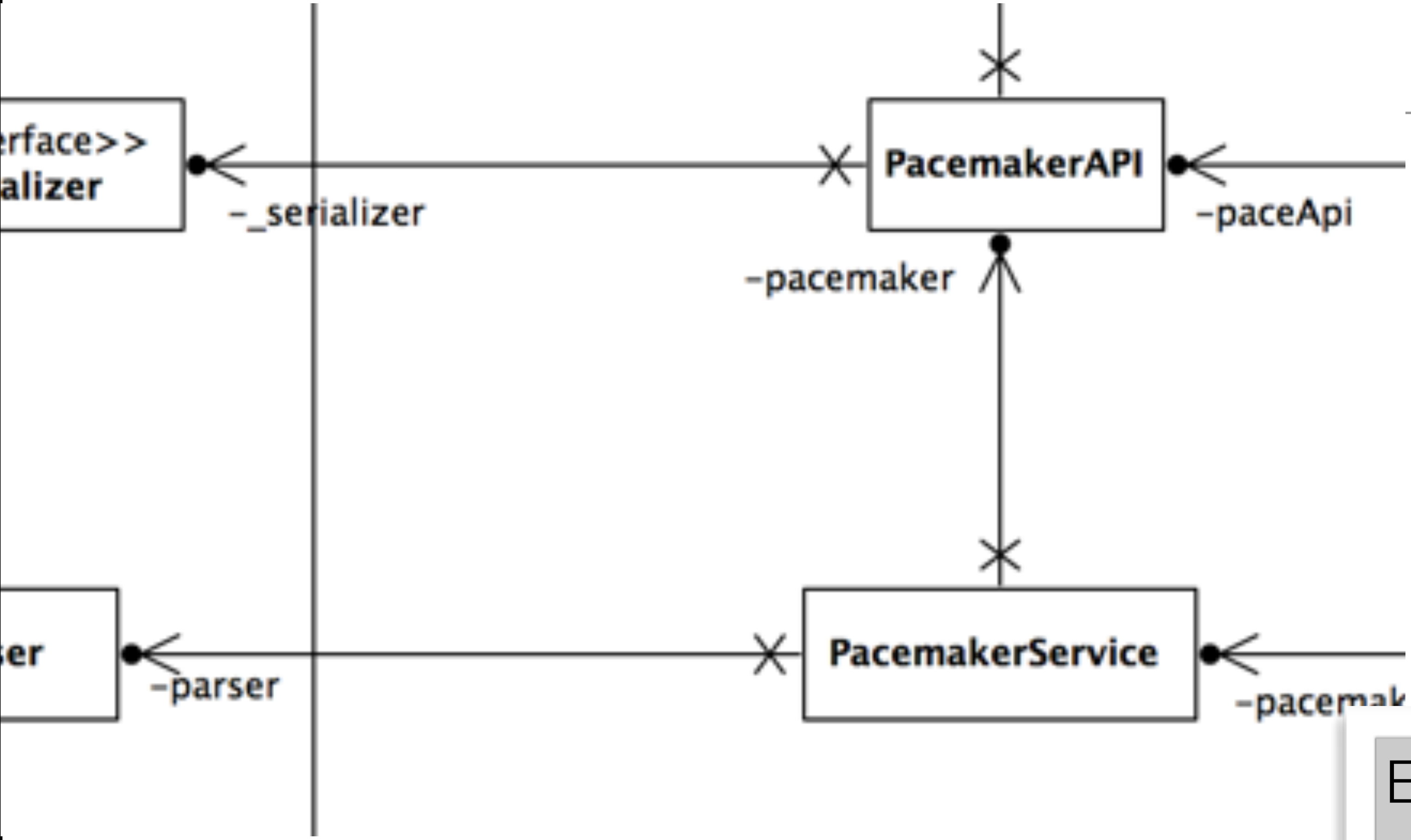

User
-_users
controllers
Ok
Response
NotFound
<<Interface>>
Serializer
-_serializer
PacemakerAPI
-paceApi
-pacemaker
Parser
-parser
PacemakerService
-pacemaker
PacemakerShell
-jsonSerializer
JSONSerializer
-xmlSerializer
XMLSerializer



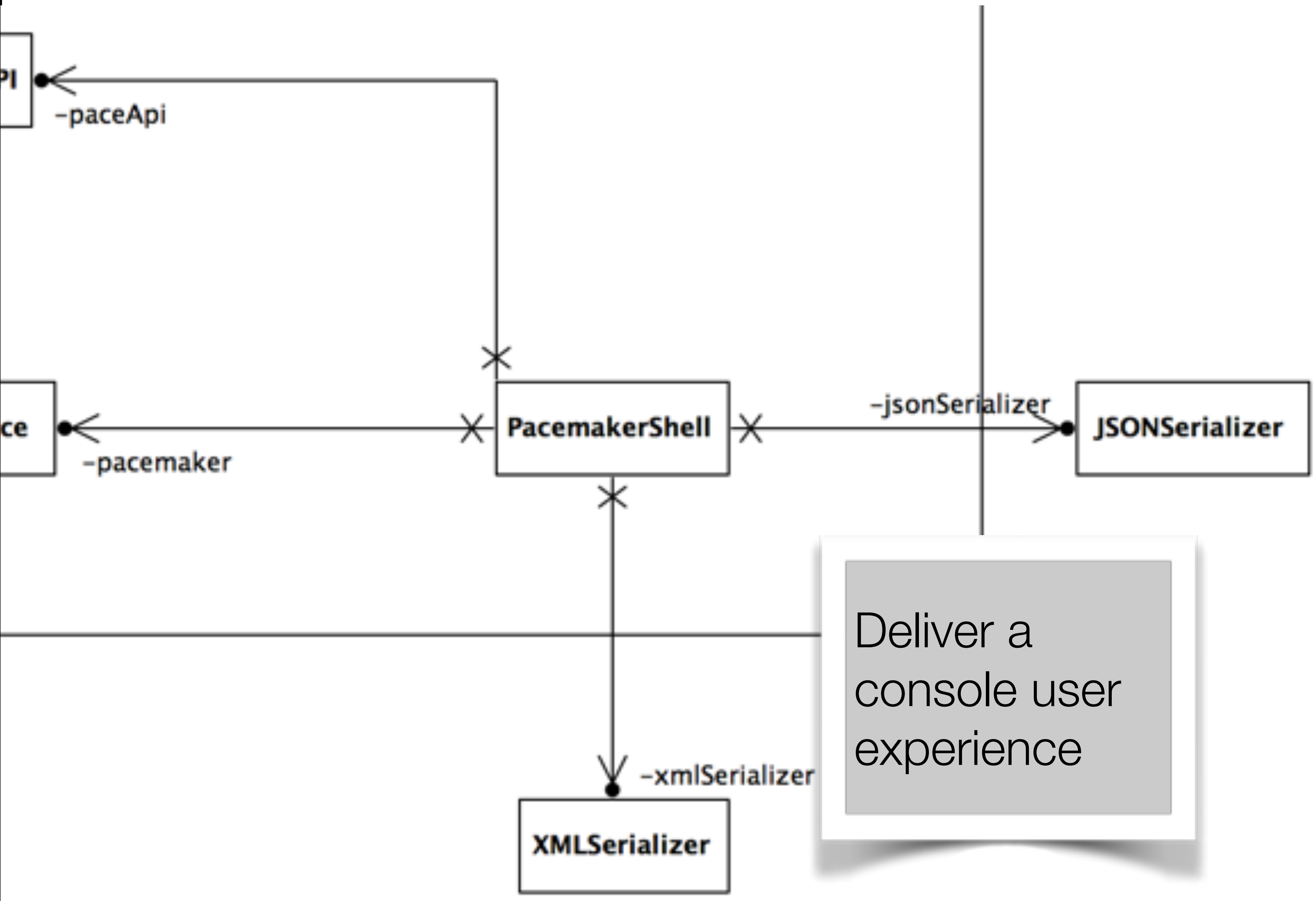
Represent **responses** from an application to **requests** from clients (use HTTP terminology)



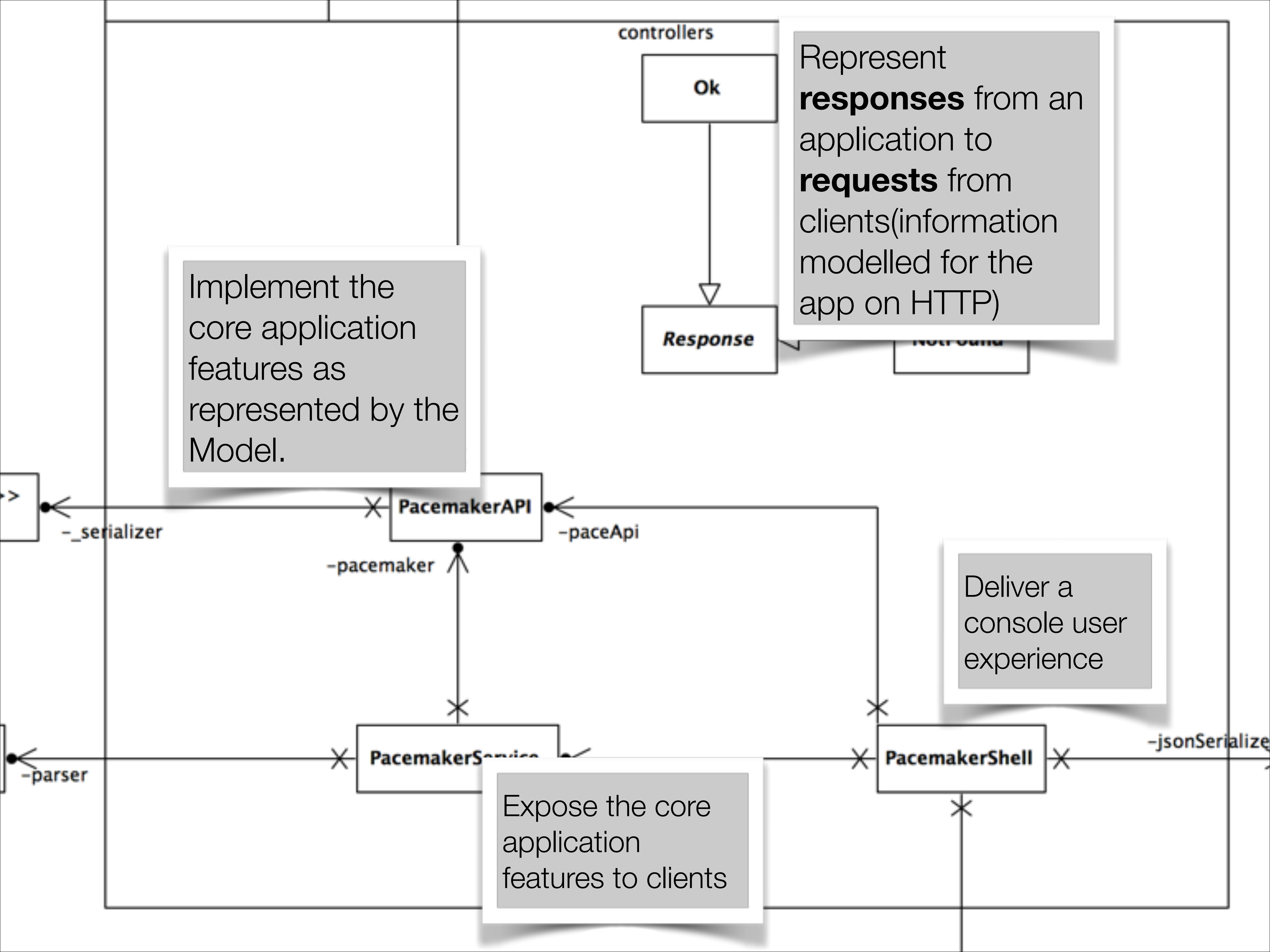
Implement the core application features as represented by the Model.



Expose the core application features to clients



Deliver a console user experience



Represent **responses** from an application to **requests** from clients(information modelled for the app on HTTP)

Implement the core application features as represented by the Model.

Deliver a console user experience

Expose the core application features to clients

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*.

Waterford Institute *of* Technology
INSTITIÚID TEICNEOLAÍOCHTA PHORT LÁIRGE

eLearning
support unit