The Interceptor Pattern

Bogdan Dumitriu

Department of Computer Science
University of Utrecht

28th May 2006

Outline

The Problem in a Nutshell

Given a system, how do we

- allow others to monitor what goes on inside it
- and (optionally) change/extend some of its behaviour
- without making them understand our code
- and without making them change our code
- and without affecting the system?

Naturally, by using the Interceptor Pattern!

The Problem in a Nutshell

Given a system, how do we

- allow others to monitor what goes on inside it
- and (optionally) change/extend some of its behaviour
- without making them understand our code
- and without making them change our code
- and without affecting the system?

Naturally, by using the Interceptor Pattern!

The Problem in a Nutshell

Given a system, how do we

- allow others to monitor what goes on inside it
- and (optionally) change/extend some of its behaviour
- without making them understand our code
- and without making them change our code
- and without affecting the system?

Naturally, by using the Interceptor Pattern!

The Problem in a Nutshell

Given a system, how do we

- allow others to monitor what goes on inside it
- and (optionally) change/extend some of its behaviour
- without making them understand our code
- and without making them change our code
- and without affecting the system?

Naturally, by using the Interceptor Pattern!

The Problem in a Nutshell

Given a system, how do we

- allow others to monitor what goes on inside it
- and (optionally) change/extend some of its behaviour
- without making them understand our code
- and without making them change our code
- and without affecting the system?

Naturally, by using the Interceptor Pattern!

The Problem in a Nutshell

Given a system, how do we

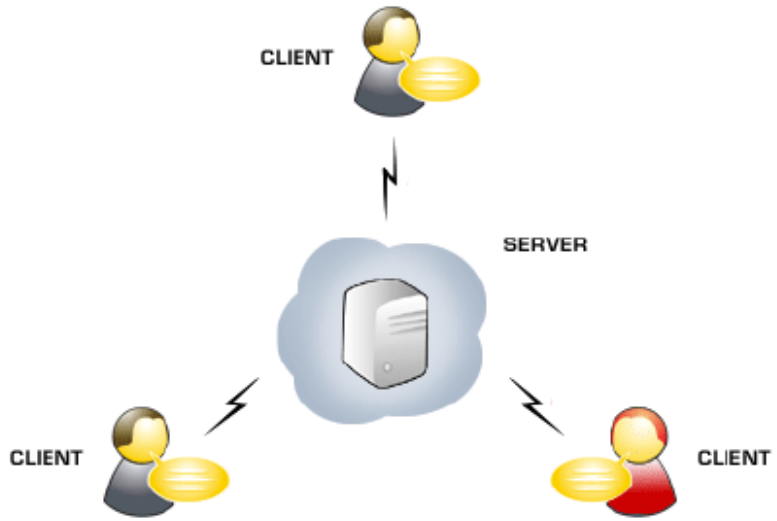
- allow others to monitor what goes on inside it
- and (optionally) change/extend some of its behaviour
- without making them understand our code
- and without making them change our code
- and without affecting the system?

Naturally, by using the Interceptor Pattern!

Instant Messenger

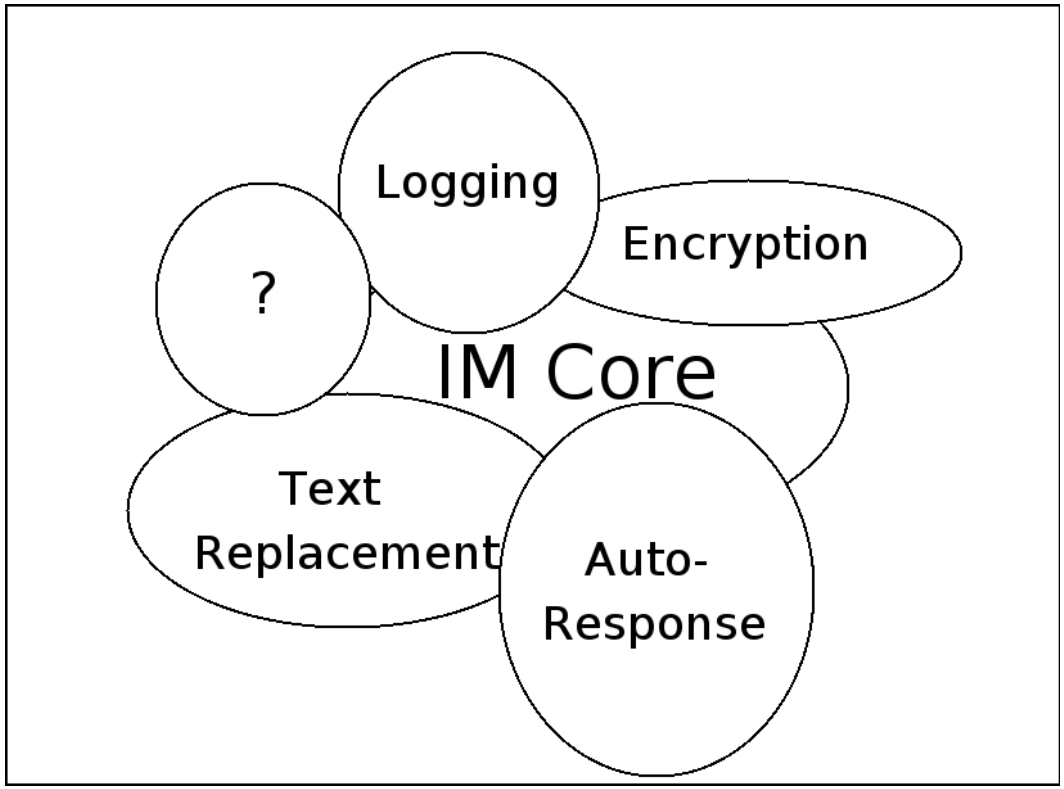
Design an IM system with:

- basic IM functionality
- logging of activity (history)
- encryption of messages
- auto response
- text replacement
- . . . what not?



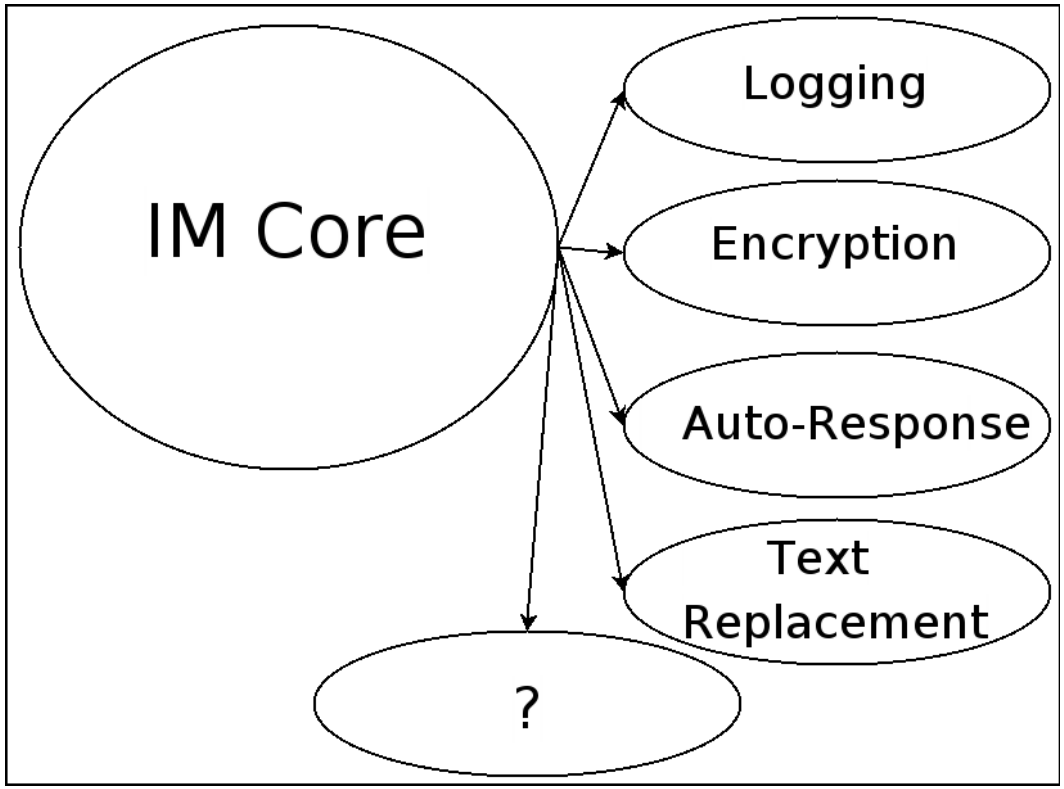
Solution (BAD)

We could do it like this:



Solution (GOOD)

Or like this:



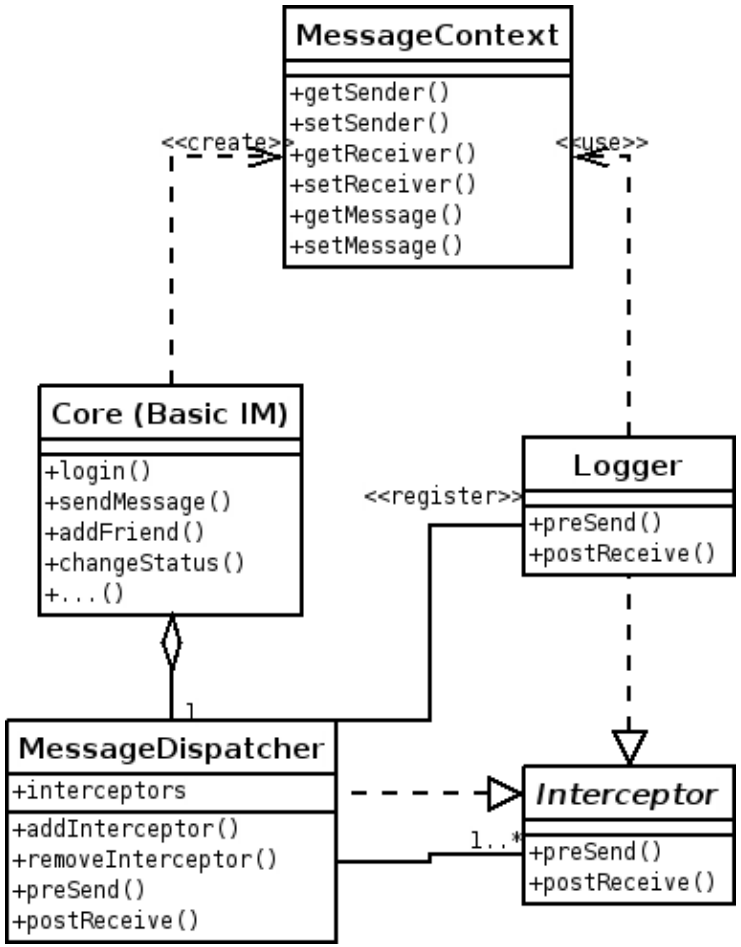
Solution (in words)

- build core with basic IM functionality
- allow services to register with the core
- make sure core triggers these services
- let others do the hard work

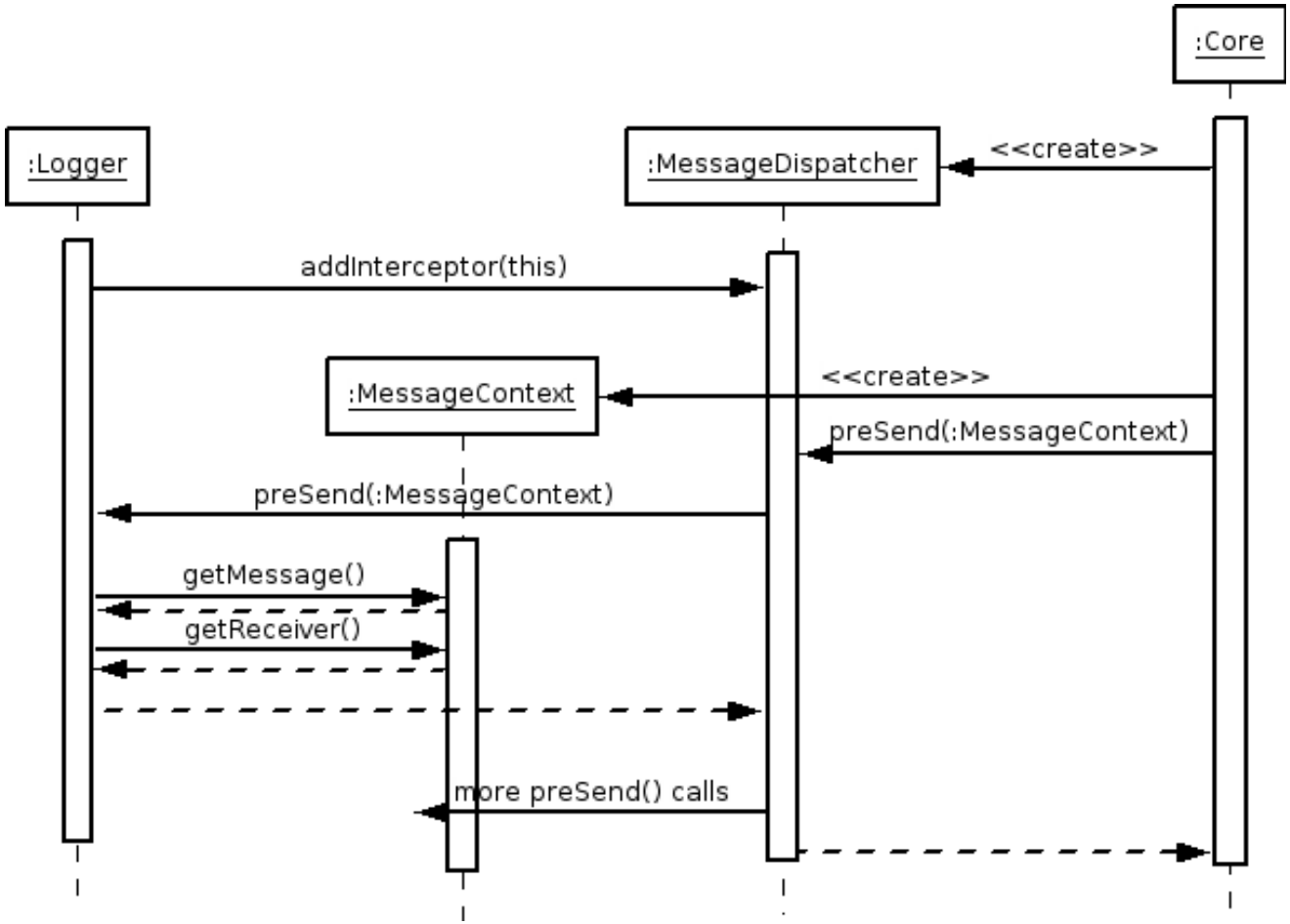
Participants

- Framework (or core system)
- Dispatcher(s)
- Context Object(s)
- Interceptor Interface(s)
- Concrete Interceptor(s)
- Application (using the Concrete Interceptor(s))

Solution (class diagram)



Solution (class diagram)

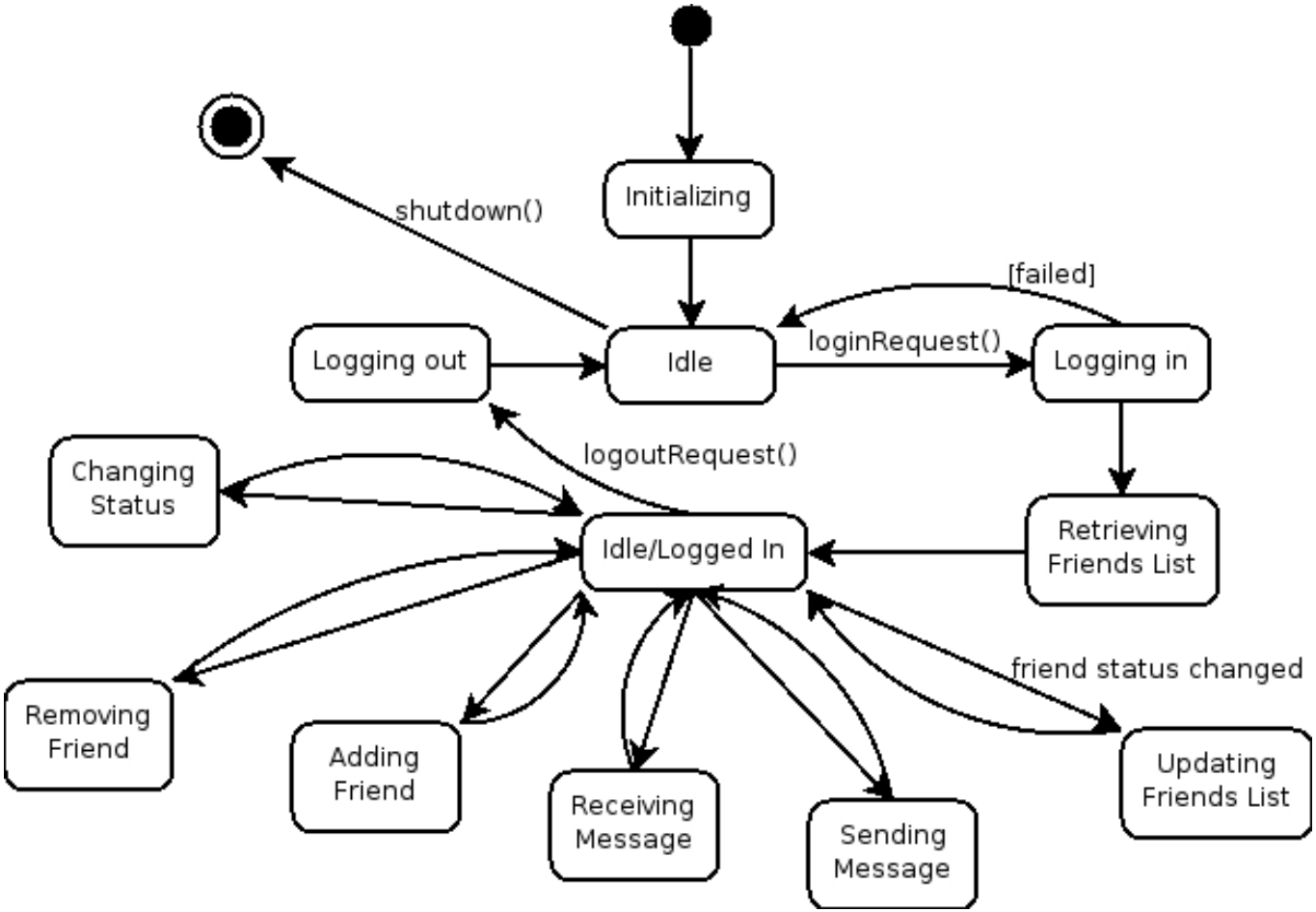


Back to our example

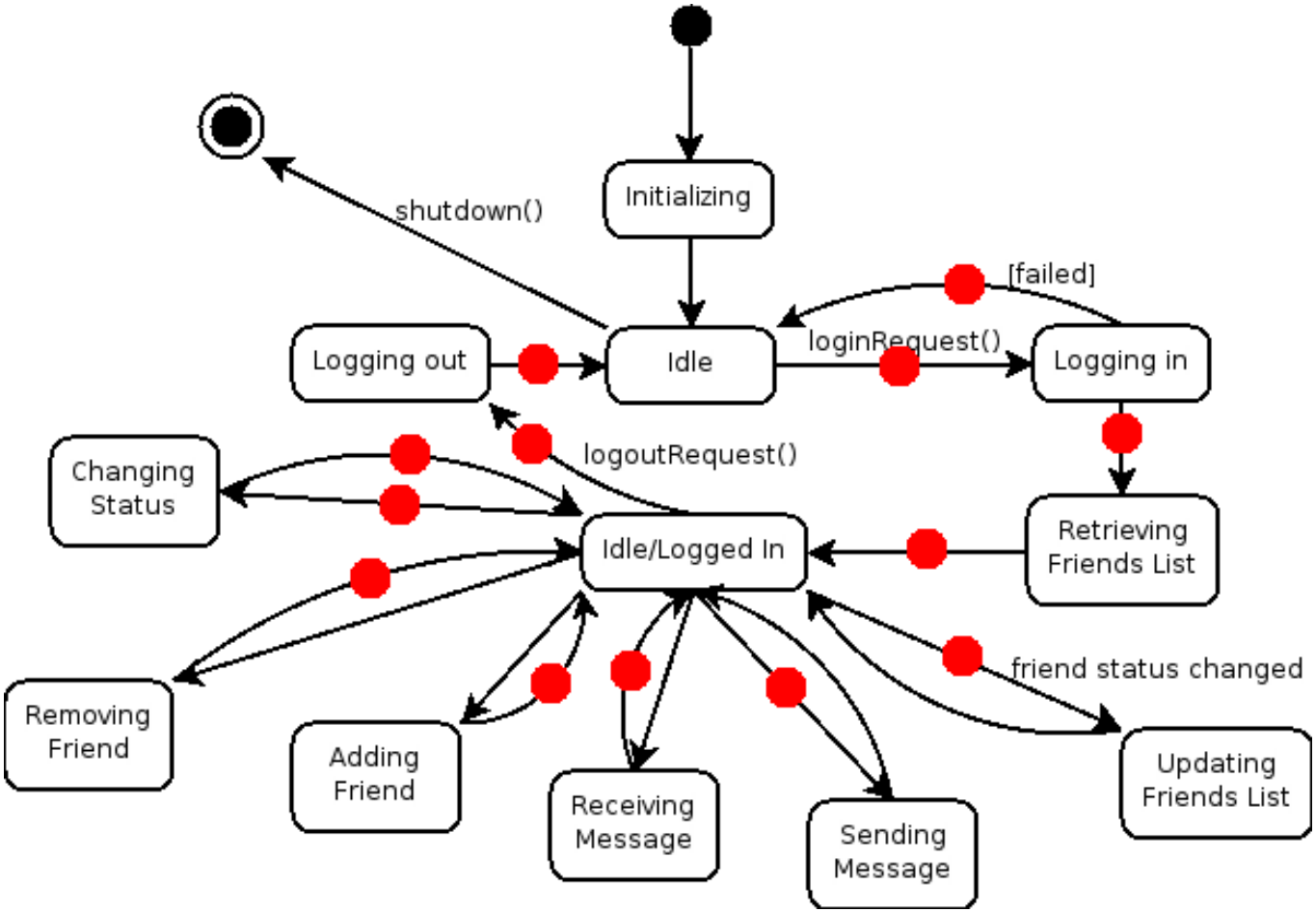
How do we go about the Instant Messenger implementation?

- draw state diagram of system
- identify and group about interception points
- define dispatcher-interceptor-context object triplets
- make sure the core calls the dispatchers
- (leave to others) write interceptors to provide services

State diagram



Identify interception points



Group interception points

Loging in/Loging out Group:

- pre-login and post-login events
- pre-logout and post-logout events

All these are read-only.

Group interception points

Friend-Related Group:

- friend added event
- friend removed event
- friend logged in event
- friend logged out event
- friends list received event

All these are read-only.

Group interception points

Message Exchange-Related Group:

- pre message send event
- post message received event

These are both read/write.

Define triplets

For each group of interception points we define:

- a Dispatcher
- an Interceptor Interface
- a Context Object