

---

# **Energy Management System—A Case Study**

Wolfgang Pree

C. Doppler Laboratory for Software Engineering  
University of Linz

---

# **Introduction**

## **Methods & tools**

## **System architecture**

Framework design with hot spot cards

## **Experience**

---

# Introduction

# Introduction

---

The Energy Management System (EMS) project was carried out in cooperation between VA-Stahl (the Austrian VOEST Alpine Steel Company) and the C. Doppler Laboratory for Software Engineering.

Goal of the C. Doppler Laboratory:

Support of funding partners (such as VA-Stahl) in the application of state-of-the-art technology.

Starting point of the EMS project: steel production is characterized by several production activities with huge energy consumption. The energy demand varies significantly.

# Introduction

---

EMS project goals:

- control of energy flows by collecting relevant data regarding production activities and energy production
- forecast of future energy demands

=> optimization of energy production and distribution

=> cost reductions

The implemented EMS focuses on the core part of energy management, i.e., providing all the information to accomplish energy forecasts.

# Introduction

---

Project duration: 1992 – April 1996

‘92–’93: analysis of steel plants regarding workflows and energy consumption

‘94: design; selection of methods and tools; definition of guidelines for programming and documentation; training

‘95: implementation

‘96: test & tuning

Team size: 6-8 members

The analysis was carried out by different people (about 4-5).

---

# Methods & Tools

# Methods & tools

---

The **Object Modeling Technique (OMT)** was used for depicting the design of EMS. Select OMT was used as tool for creating OMT diagrams.

The **ami-process model** formed the framework of the EMS project (→ EU support).

Relational database system: Oracle

Programming language: C++ (with DEC Fuse as development environment)



# Methods & tools

---

Class library: Osiris Class Library (OCL), the commercial version of the NIH class library.

OCL provides basic data structures and data types, and supports garbage collection, shared objects, and object transfer.

The OCL database extension SqlGate was used to generate C++ classes according to the database schemata.

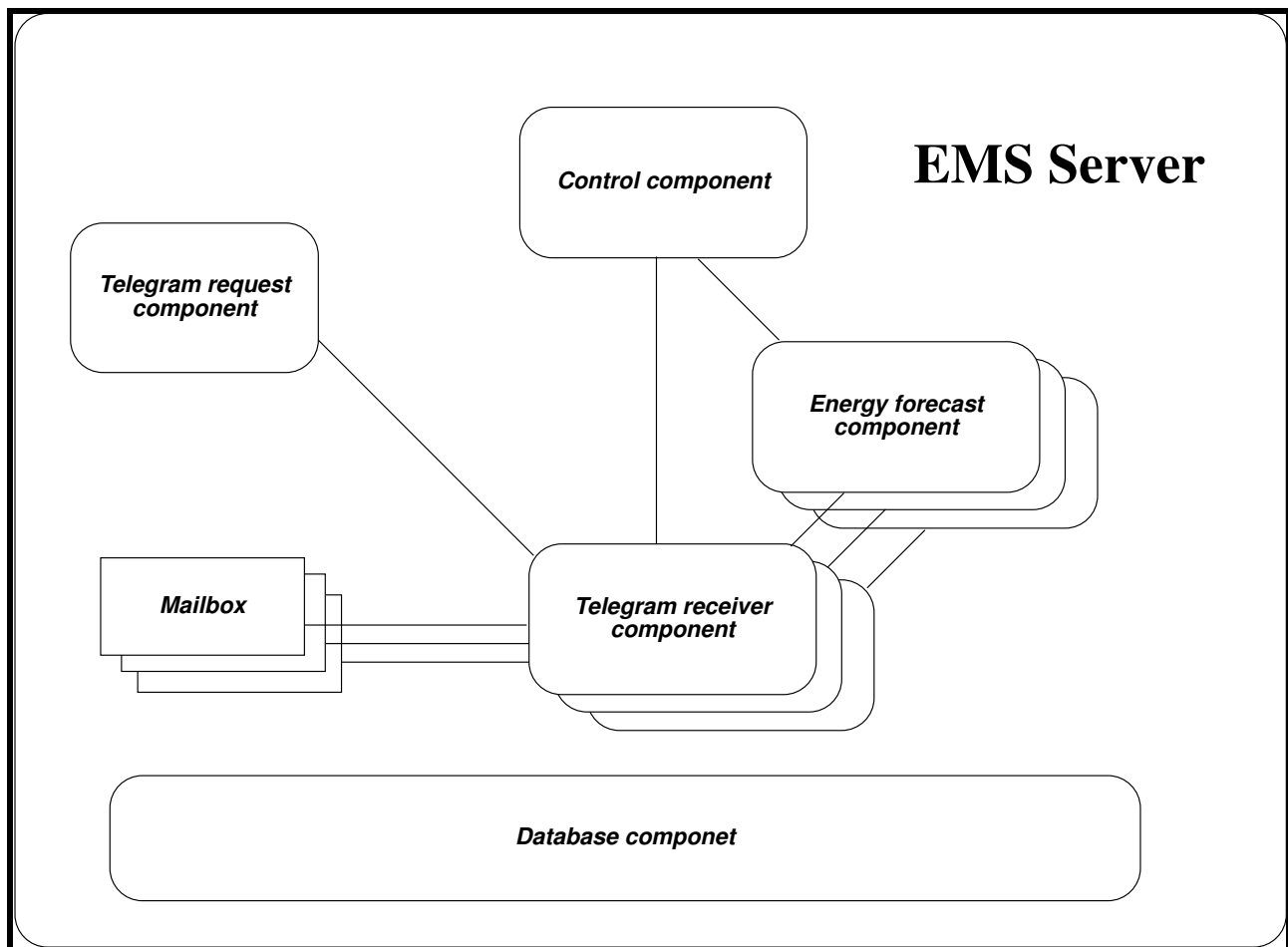
PowerBuilder formed the tool for implementing the GUI and the database access.

---

# System Architecture

# System architecture

---



# System architecture

---

## Database:

Number of relations:

for the core system and applications: 32

for data received as telegrams: 14

for forecasts: 1

for GUI of clients: 16

$\Sigma$ : 63

Number of constant data records:

description of plants: 157

telegram descriptions: 46

process/workflow descriptions: 9

event descriptions: 17

# System architecture

---

Number of new data records (daily):

measurement values: 20,000 – 30,000

forecast values: 7,000 – 15,000

## **Telegram receiver component:**

Requirements:

### **Flexibility**

- => structure and contents is described in the database
- => generic telegram receiving processes (frameworks) that are adapted on demand
- => connection to forecast processes via events

# System architecture

---

## Security

- => syntactical and semantical checks
- => protocol of the overall telegram traffic as plain text
- => time-dependent control

## Efficiency

- => only relevant data are stored
- => grouping of data in various different categories

## Portability (regarding net and protocol)

- => encapsulation of net and protocol services in abstract classes
- => communication via telegram objects

# System architecture

---

## Forecast component:

The component implementation relies on discrete event simulation.

Let us discuss the design and redesign of the time-dependent event notification system as **example how to use socalled hot spot cards in object modeling**:

How can one assess the **quality of a framework**?

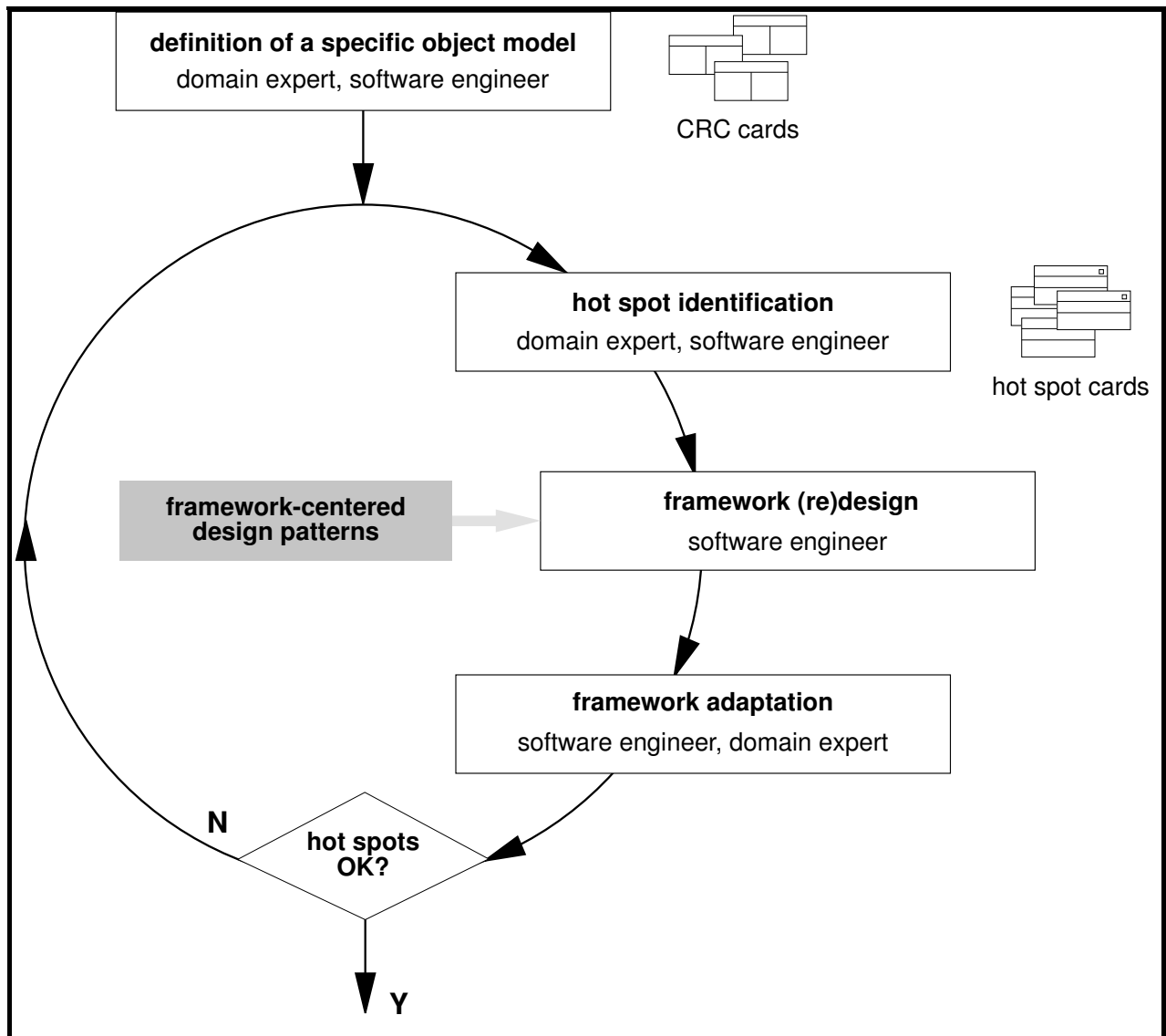
Striving for **flexibility for the flexibility's sake**—by incorporating as many patterns as possible—**does not result in a good framework**.

**=> Hot spot identification has to become an explicit activity in the framework development process**

# System architecture

---

Hot-spot-driven OO design:





# System architecture

---

domain experts:

**identify flexibility requirements &**  
capture them on hot spot cards

software engineers:

**modify the object model** in order to incorporate the  
desired hot spots

- in practice a clear distinction between both roles  
is not possible
  - essential patterns describe how to design  
flexibility into a framework
  - explicit hot spot identification pinpoints the places  
where flexibility is indeed crucial
- => **hot spot identification forms the precondition to  
exploit the potential of design patterns**

# System architecture

---

**Hot spot cards form the common denominator of communication** between domain experts and software engineers.

Domain experts often do not know about object technology.

**=> object-oriented jargon has to be excluded**

Domain experts...

- can think in terms of **software functionality**
  - => function hot spots
- **understand the principal entities** in a domain
  - => data hot spots

# System architecture

---

Function hot spot card:

<b>Hot spot name</b> specify degree of flexibility: <input type="checkbox"/> adaptation without restart <input type="checkbox"/> adaptation by end user
general description of semantics
sketch hot spot behavior in at least two specific situations

Data hot spot card:

<b>Hot spot name</b> specify the importance: <input type="checkbox"/> principal domain abstraction <input type="checkbox"/> subordinate abstraction
general description of semantics
outline at least two specific incarnations of the abstraction

# System architecture

---

- data hot spots are those entities in a domain where generalization should take place, typically **by introducing an abstract class**
- expect only a very few data hot spot cards
- function hot spots **closely correspond to hook methods**
- expect a punch of function hot spot cards

# System architecture

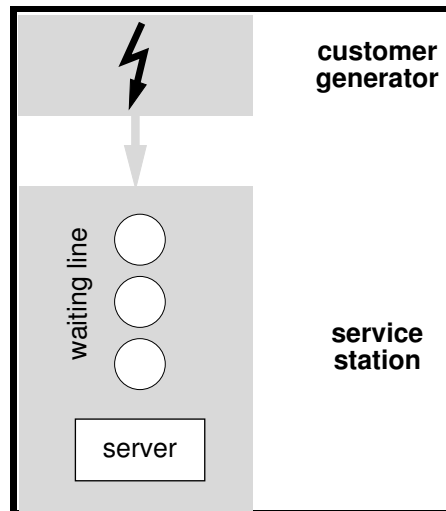
---

<i>without restart</i>	<i>by end user</i>	<i>object model transformation</i>
—	—	additional hook method <b>(Unification pattern)</b>
✓	—	additional hook method in separate hook class <b>(Separation pattern)</b>
—	✓	additional hook method + configuration tool
✓	✓	additional hook method in separate hook class + configuration tool

# System architecture

---

Simulation domain:



Design of the time-dependent notification mechanism:

Notifier		Actor <i>abstract</i>
time: long actors: SortedQueue	0 <i>manages</i> ► *	time: long
Notifier() schedule(a: Actor, time: long) notify(duration: long) reset()		Actor(t: long) commit() { <i>abstract</i> }

# System architecture

---

Workflow management systems coordinate users who work together in order to accomplish a task.

Thus the **core of a workflow system** is a time-dependent event notification mechanism, **similar to the one in discrete event simulation**.

Simulation systems deal with **simulated time**, workflow systems with **real time**.

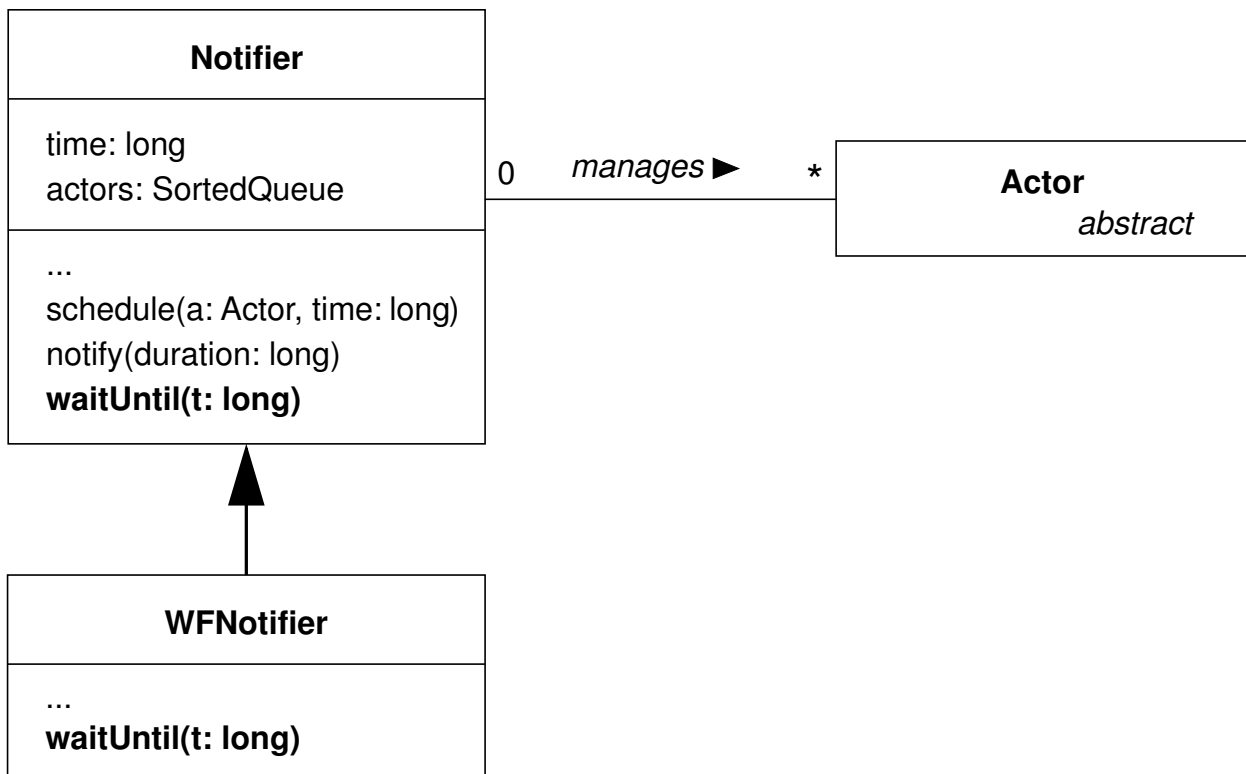
=> additional flexibility is required in the notification mechanism:

<b>Dealing with time delays</b> specify degree of flexibility: <input type="checkbox"/> adaptation without restart <input type="checkbox"/> adaptation by end user
how to proceed on the time axis
simulation: no delay  workflow management system: real-time delay

# System architecture

---

Resulting redesign of the notification mechanism:





# System architecture

---

Java implementation:

```
public class Notifier extends Thread {  
    ...  
    public void notify(long duration) {  
        Actor actor;  
        long endOfNotificationPeriod= time + duration;  
        do {  
            if (!actors.isEmpty()) {  
                actor= actors.dequeue();  
                time= actor.time;  
                waitUntil(time);    // additional hook  
                actor.commit();  
            } else    // no more actors enqueued  
                time= endOfNotificationPeriod+1;    // exit loop  
        } while (time <= endOfNotificationPeriod);  
    }  
    public void waitUntil(long t) { // default: no delay  
    }  
    ...  
}
```

# System architecture

---

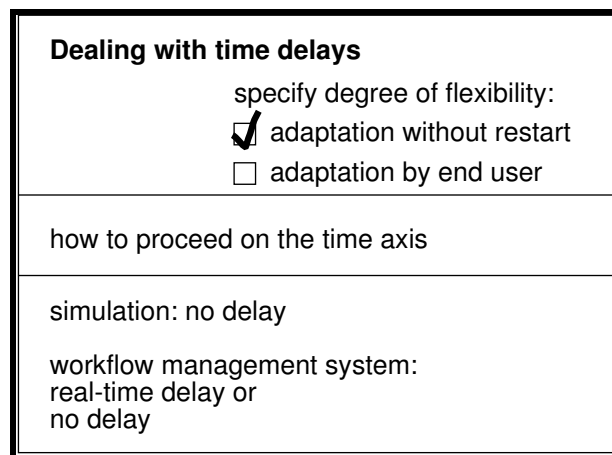
The workflow system modifies the behavior by overriding the hook method `waitUntil(...)`:

```
public class WFNotifier extends Notifier {  
    ...  
    public void waitUntil(long t) {  
        long timePeriod= ... // calculate 't – current time'  
        sleep(timePeriod);  
    }  
    ...  
}
```

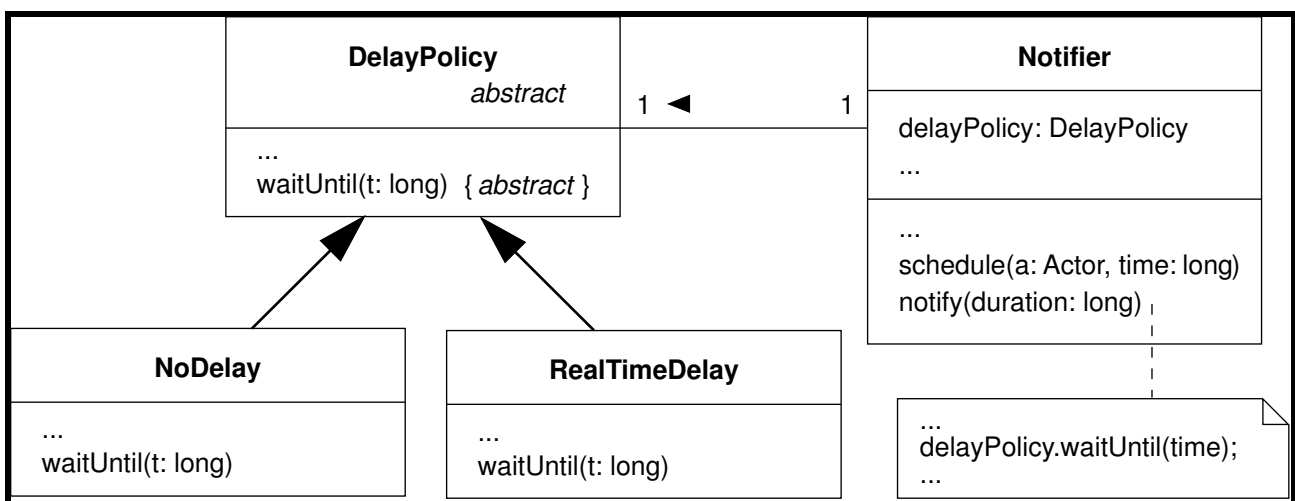
# System architecture

---

If the workflow system should also allow simulations of workflows, **run-time changes of the notification mechanism** would be required:



=> Hook method in separate hook class:



# System architecture

---

The assumption is rather idealistic to have perfect domain experts at hand.

In practice, **domain experts are absolutely not used to answering questions regarding a generic solution.**

Ways to overcome this obstacle:

- take a look at maintenance
- investigate scenarios/use cases
- ask the right people

# System architecture

---

## Control component:

### Requirements:

- maximum availability

- intuitive GUI

### Functionality:

- start and stop of processes

- control of all processes

- automatic restart of processes

- display of system state

- protocol generation of all process activities

The component implementation was based on Shared Objects as provided by OCL.

---

# Experience

# Experience

---

- + OMT proved to be a valuable communication means  
(Note: OMT Select was not used for code generation.)
- + DEC hardware and software (UNIX, C/C++ compiler)  
were very reliable
- + PowerBuilder met our expectations regarding  
advertised possibilities and efficiency
- OMT requires tool support; Select OMT was not  
adequate to support our needs
- DEC's C++ development environment Fuse is  
significantly worse compared to state-of-the-art  
environments such as SNIFF+

# Experience

---

## Organizational aspects:

- + satisfying quality management based on accompanying documentation and standardization measures
- coordination overhead due to (geographically) separated teams