Architectural Patterns and Problem Frames

Christine Choppy (LIPN, Université Paris XIII, France)
Denis Hatebur (Universität Duisburg-Essen)
Maritta Heisel (Universität Duisburg-Essen)
Overview

Motivation

Patterns

Problem Frames

Architectural Styles

Composite Structure Diagrams

Conclusions
Motivation

Problem frames provide a characterisation and classification of software development problems.
Motivation

- Problem frames provide a characterisation and classification of software development problems. Fitting a problem to an appropriate problem frame should not only help to understand it, but also to solve the problem (the idea being that, once the adequate problem frame is identified, then the associated development method should be available).
Motivation

- Problem frames provide a characterisation and classification of software development problems. Fitting a problem to an appropriate problem frame should not only help to understand it, but also to solve the problem (the idea being that, once the adequate problem frame is identified, then the associated development method should be available).
- We propose software architectural patterns corresponding to the different problem frames that may serve as a starting point for the construction of the software solving the given problem.
Motivation

Problem frames provide a characterisation and classification of software development problems. Fitting a problem to an appropriate problem frame should not only help to understand it, but also to solve the problem (the idea being that, once the adequate problem frame is identified, then the associated development method should be available).

We propose software architectural patterns corresponding to the different problem frames that may serve as a starting point for the construction of the software solving the given problem.

We show that these architectural patterns exactly reflect the properties of the problems fitting to a given frame, and that they can be combined in a modular way to solve multi-frame problems.
Motivation

- Problem frames provide a characterisation and classification of software development problems. Fitting a problem to an appropriate problem frame should not only help to understand it, but also to solve the problem (the idea being that, once the adequate problem frame is identified, then the associated development method should be available).
- We propose software architectural patterns corresponding to the different problem frames that may serve as a starting point for the construction of the software solving the given problem.
- We show that these architectural patterns exactly reflect the properties of the problems fitting to a given frame, and that they can be combined in a modular way to solve multi-frame problems.
- We also provide alternative architectures to cope with specific system characteristics (e.g. distribution).
Overview

Motivation

Patterns

Problem Frames

Architectural Styles

Composite Structure Diagrams

Conclusions
Patterns

- a promising approach to software development, a means to reuse software development knowledge on different levels of abstraction, classify sets of software development problems or solutions that share the same structure
Patterns

- a promising approach to software development, a means to reuse software development knowledge on different levels of abstraction, classify sets of software development problems or solutions that share the same structure
- introduced on the level of detailed object oriented design, now defined for different activities.
Patterns

- a promising approach to software development, a means to reuse software development knowledge on different levels of abstraction, classify sets of software development problems or solutions that share the same structure
- introduced on the level of detailed object oriented design, now defined for different activities.
- **Problem Frames** (Jackson) classify software development problems,
Patterns

▶ a promising approach to software development, a means to reuse software development knowledge on different levels of abstraction, classify sets of software development problems or solutions that share the same structure
▶ introduced on the level of detailed object oriented design, now defined for different activities.
▶ Problem Frames (Jackson) classify software development problems,
Architectural styles/ “architectural patterns” characterise software architectures
Patterns

- A promising approach to software development, a means to reuse software development knowledge on different levels of abstraction, classify sets of software development problems or solutions that share the same structure.
- Introduced on the level of detailed object oriented design, now defined for different activities.
- **Problem Frames** (Jackson) classify software development problems, **Architectural styles**/“architectural patterns” characterise software architectures. **Design Patterns** for finer-grained software design, **frameworks** less abstract, more specialised. **Idioms/“code patterns”** : low-level patterns related to specific programming languages.
Patterns

- a promising approach to software development, a means to reuse software development knowledge on different levels of abstraction, classify sets of software development problems or solutions that share the same structure
- introduced on the level of detailed object oriented design, now defined for different activities.
- **Problem Frames** (Jackson) classify software development problems,
  Architectural styles/ “architectural patterns” characterise software architectures
  Design Patterns for finer-grained software design, frameworks less abstract, more specialised.
  idioms/ “code patterns” : low-level patterns related to specific programming languages
- construct software in a systematic way, body of accumulated knowledge, not starting from scratch
Overview

Motivation

Patterns

Problem Frames

Architectural Styles

Composite Structure Diagrams

Conclusions
Problem Frames (M. Jackson)

- problem frames concept: present, classify, understand software development problems
Problem Frames (M. Jackson)

- problem frames concept: present, classify, understand software development problems
- characterisation of a class of problems in terms of their main components and the connections between these components
Problem Frames (M. Jackson)

- problem frames concept: present, classify, understand software development problems
- characterisation of a class of problems in terms of their main components and the connections between these components
- Once a problem is successfully fitted to a problem frame, its most important characteristics are known
Problem Frames (M. Jackson)

- problem frames concept: present, classify, understand software development problems
- characterisation of a class of problems in terms of their main components and the connections between these components
- Once a problem is successfully fitted to a problem frame, its most important characteristics are known
- diagram: involved domains, requirements, design, interfaces
Problem Frames (M. Jackson)

- **problem frames** concept: present, classify, understand software development problems
- characterisation of a class of problems in terms of their main components and the connections between these components
- Once a problem is successfully fitted to a problem frame, its most important characteristics are known
- diagram: involved domains, requirements, design, interfaces
- five basic problem frames, variants
A basic problem frame: required behaviour

There is some part of the physical world whose behaviour is to be controlled so that it satisfies certain conditions. The problem is to build a machine that will impose that control.
A basic problem frame: required behaviour

There is some part of the physical world whose behaviour is to be controlled so that it satisfies certain conditions. The problem is to build a machine that will impose that control.

- embedded systems
- sensors (C2) and actuators (C1)
A basic problem frame: required behaviour

- embedded systems
- sensors (C2) and actuators (C1)

Example: A machine that keeps the temperature of some liquid between given bounds. Then, the temperature of the liquid would be a shared phenomenon controlled by the environment. The corresponding sensor would be a thermometer. Another shared phenomenon would be the state of a burner. That state would be controlled by the machine, i.e., the machine is able to switch the burner on or off.
A basic problem frame: commanded behaviour

Controlled domain

Operator

Controlled
behaviour

Control machine

Commanded
operator

E4

C3

C

CM!C1
CD!C2

O!E4

B

Conclusions
There is some part of the physical world about whose states and behaviour information is continually needed. The problem is to build a machine that will obtain this information from the world and present it at the required place in the required form.
Overview

Motivation

Patterns

Problem Frames

Architectural Styles

Composite Structure Diagrams

Conclusions
Architectural Styles

Bass, Clements, and Kazman:

the software architecture of a program or computing system is the structure or structures of the system, which comprise software components, the externally visible properties of those components, and the relationships among them.
Architectural Styles

Bass, Clements, and Kazman:

the software architecture of a program or computing system is the structure or structures of the system, which comprise software components, the externally visible properties of those components, and the relationships among them.

Patterns for software architectures
Architectural Styles

Bass, Clements, and Kazman:

*the software architecture of a program or computing system is the structure or structures of the system, which comprise software components, the externally visible properties of those components, and the relationships among them.*

Patterns for software architectures
A style is characterised by:
- a set of component types that perform some function at runtime,
Architectural Styles

Bass, Clements, and Kazman:

*the software architecture of a program or computing system is the structure or structures of the system, which comprise software components, the externally visible properties of those components, and the relationships among them.*

Patterns for software architectures

A style is characterised by:

- a set of component types that perform some function at runtime,
- a topological layout indicating their runtime interrelationships,
Architectural Styles

Bass, Clements, and Kazman:

the software architecture of a program or computing system is the structure or structures of the system, which comprise software components, the externally visible properties of those components, and the relationships among them.

Patterns for software architectures
A style is characterised by:
- a set of component types that perform some function at runtime,
- a topological layout indicating their runtime interrelationships,
- a set of semantic constraints.
Architectural Styles

Bass, Clements, and Kazman:

_the software architecture of a program or computing system is the structure or structures of the system, which comprise software components, the externally visible properties of those components, and the relationships among them._

Patterns for software architectures

A style is characterised by:

- a set of component types that perform some function at runtime,
- a topological layout indicating their runtime interrelationships,
- a set of semantic constraints
- a set of connectors that mediate communication, coordination, or cooperation among components.
Architectural Styles for Problem Frames

- When choosing an architecture for a system, usually several architectural styles are possible, which means that all of them could be used to implement the functional requirements.
Architectural Styles for Problem Frames

► When choosing an architecture for a system, usually several architectural styles are possible, which means that all of them could be used to implement the functional requirements.
► we propose concrete architectural patterns for each basic problem frame in order to provide a concrete starting point for the further development of the machine.
we propose concrete architectural patterns for each basic problem frame in order to provide a concrete starting point for the further development of the machine.

These architectural patterns are based on a **Layered architecture**.
we propose concrete architectural patterns for each basic problem frame in order to provide a concrete starting point for the further development of the machine.

These architectural patterns are based on a Layered architecture.

The components in this layered architecture are either Communicating Processes (active components) or used with a Call-and-Return mechanism (passive components). That design decision is taken in a later step of the development.
we propose concrete architectural patterns for each basic problem frame in order to provide a concrete starting point for the further development of the machine.

These architectural patterns are based on a Layered architecture

The components in this layered architecture are either Communicating Processes (active components) or used with a Call-and-Return mechanism (passive components). That design decision is taken in a later step of the development.

the Repository and the Pipe-and-Filter architectural styles can be mapped to the layered architecture
we propose concrete architectural patterns for each basic problem frame in order to provide a concrete starting point for the further development of the machine.

These architectural patterns are based on a Layered architecture.

The components in this layered architecture are either Communicating Processes (active components) or used with a Call-and-Return mechanism (passive components). That design decision is taken in a later step of the development.

the Repository and the Pipe-and-Filter architectural styles can be mapped to the layered architecture.

We use UML 2.0 composite structure diagrams to represent architectural patterns as well as concrete architectures.
Overview

Motivation

Patterns

Problem Frames

Architectural Styles

Composite Structure Diagrams

Conclusions
Composite Structure Diagrams

> Means to describe architectures
Composite Structure Diagrams

- Means to describe architectures
- They contain named rectangles, called **parts**. These parts are components of the software.
Composite Structure Diagrams

- Means to describe architectures
- They contain named rectangles, called parts. These parts are components of the software.
- In an object-oriented implementation components are instantiated classes.

Means to describe architectures
- They contain named rectangles, called parts. These parts are components of the software.
- In an object-oriented implementation components are instantiated classes.
Composite Structure Diagrams

- Means to describe architectures
- They contain named rectangles, called parts. These parts are components of the software.
- In an object-oriented implementation components are instantiated classes.
- Each component may contain other (sub-) components. Atomic components can be described by state machines and operations for accessing internal data.
Composite Structure Diagrams

► Means to describe architectures
► They contain named rectangles, called parts. These parts are components of the software.
► In an object-oriented implementation components are instantiated classes.
► Each component may contain other (sub-) components. Atomic components can be described by state machines and operations for accessing internal data.
► In our architectures, components for data storage are only included if the data is stored persistently. Otherwise they are assumed to be part of some other component.
Means to describe architectures

- They contain named rectangles, called parts. These parts are components of the software.
- In an object-oriented implementation components are instantiated classes.
- Each component may contain other (sub-) components. Atomic components can be described by state machines and operations for accessing internal data.
- In our architectures, components for data storage are only included if the data is stored persistently. Otherwise they are assumed to be part of some other component.
- Parts may have ports, denoted by small rectangles. Ports may have interfaces associated to them. Provided interfaces are denoted using the “lollipop” notation, and required interfaces using the “socket” notation.
Notation for Architectures
Required Behaviour Frame Diagram and Architecture

Control machine

Controlled domain

Required behaviour

Controlled Domain (C2)

Controlled Domain (C1)

Sensor IAL

Sensor HAL

Actuator IAL

Actuator HAL

Application

Control Machine

Architectural Patterns
and Problem Frames

Christine Choppy
(LIPN, Université Paris XIII, France)

Denis Hatebur
(Universität Duisburg-Essen)

Maritta Heisel
(Universität Duisburg-Essen)

Motivation

Patterns

Problem Frames

Architectural Styles

Composite Structure Diagrams

Conclusions
A pattern-based software development process

- a context diagram showing the problem context
A pattern-based software development process

- a context diagram showing the problem context
- the overall problem is decomposed into subproblems that fit to problem frames → a set of problem diagrams that are instantiated frame diagrams

For each subproblem, a specification for the machine domain must be derived, thus addressing the frame concern. Each machine domain corresponding to a subproblem is then structured by instantiating the proposed architectural patterns.

The instantiated patterns must afterwards be merged to obtain the architecture of the machine solving the overall problem.

The components of the combined architecture must be specified in more detail, and it must be shown that the combined architecture fulfills the specifications of all subproblems.
A pattern-based software development process

- a context diagram showing the problem context
- the overall problem is decomposed into subproblems that fit to problem frames → a set of problem diagrams that are instantiated frame diagrams
- For each subproblem, a specification for the machine domain must be derived, thus addressing the frame concern. Each machine domain corresponding to a subproblem is then structured by instantiating the proposed architectural patterns.
A pattern-based software development process

- a context diagram showing the problem context
- the overall problem is decomposed into subproblems that fit to problem frames → a set of problem diagrams that are instantiated frame diagrams
- For each subproblem, a specification for the machine domain must be derived, thus addressing the frame concern. Each machine domain corresponding to a subproblem is then structured by instantiating the proposed architectural patterns.
- The instantiated patterns must afterwards be merged to obtain the architecture of the machine solving the overall problem.
A pattern-based software development process

- a context diagram showing the problem context
- the overall problem is decomposed into subproblems that fit to problem frames → a set of problem diagrams that are instantiated frame diagrams
- For each subproblem, a specification for the machine domain must be derived, thus addressing the frame concern. Each machine domain corresponding to a subproblem is then structured by instantiating the proposed architectural patterns.
- The instantiated patterns must afterwards be merged to obtain the architecture of the machine solving the overall problem.
- the components of the combined architecture must be specified in more detail, and it must be shown that the combined architecture fulfils the specifications of all subproblems.
Overview

Motivation

Patterns

Problem Frames

Architectural Styles

Composite Structure Diagrams

Conclusions
Gaining a thorough understanding of the problem to be solved is a necessary prerequisite for solving it.
Conclusions

► Gaining a thorough understanding of the problem to be solved is a necessary prerequisite for solving it.
► With problem frames, one can even hope for more than just a full comprehension of the problem at hand. Since problem frames are patterns, they represent problem structures that occur repeatedly in practice. Hence, it is worthwhile to look for solution structures that match the problem structures represented by problem frames.
The construction of the solution of a software development problem should begin with the decision on the main structure of the solution, i.e., a decision on the software architecture.
Conclusions

The construction of the solution of a software development problem should begin with the decision on the main structure of the solution, i.e., a decision on the software architecture. We want to exploit the knowledge gained in representing a problem as an instance of a problem frame in taking that decision. For each problem frame, we propose a corresponding architectural pattern that takes into account the characteristics of the problems fitting to the given problem frame.
Conclusions (follwd)

Our architectural patterns structure the software into layers. Of course, this is not the only possible way of structuring, but a very convenient one. We have chosen it because a layered architecture makes it possible to divide platform-dependent from platform-independent parts, because different layered systems can be combined in a systematic way, and because other architectural styles can be incorporated in such an architecture. That choice has been validated in several industrial projects, dealing for example with smart cards, protocol converters, web/mail-servers, and real-time operating systems.
Like problem frames, our architectural patterns must be instantiated to develop a solution for a concrete problem. The structure provided by an architectural pattern constitutes a concrete starting point for the process of constructing a solution to a problem that is represented as an instance of a problem frame.
The provided architectural patterns as well as the problem frames are quite generic and cover a wide range of problems; both can be refined to accommodate more specific kinds of problems.
The provided architectural patterns as well as the problem frames are quite generic and cover a wide range of problems; both can be refined to accommodate more specific kinds of problems.

The architectural patterns support the recomposition of solutions developed for different subproblems of multi-frame problems in a systematic way.
Context Diagram for ATM Problem

- **Admin**: insert_money
  - request_log
  - display_log

- **ATM**: card_inside, no_card_inside
  - enter_pin
  - enter_request
  - refuse_withdrawal
  - account_balance
  - withdraw_money

- **Money supply / Case**: take_banknotes
  - banknotes
  - case

- **Customer**: insert_card, remove_card
  - retract_card, eject_card

- **Card reader**: account_data

**Motivation**

**Patterns**

**Problem Frames**

**Architectural Styles**

**Composite Structure Diagrams**

**Conclusions**
Problem Diagram for Money-Case Control

C1 : \{put\_banknote\_to\_case, start/stop\_open\_case, take\_banknotes\_from\_supply, start/stop\_close\_case, retract\_banknotes\_from\_case\}

C2 : \{case\_is\_open, case\_is\_closed, banknotes\_removed\}

C3 : \{banknotes\_in\_case\}

E4 : \{enter\_request, enter\_pin\}
Architecture for the Money-Case Control Subproblem
Composed Architecture

Architectural Patterns and Problem Frames

Christine Choppy
(LIPN, Université Paris
XIII, France)

Denis Hatebur
(Universität
Duisburg-Essen)

Maritta Heisel
(Universität
Duisburg-Essen)

Motivation

Patterns

Problem Frames

Architectural Styles

Composite Structure Diagrams

Conclusions