Agent Patterns

11/21/99

Click here to start

Table of Contents

Author: Dwight Deugo, Liz Kendall, Michael Weiss

Agent Patterns

Prof. Dwight Deugo
School Of Computer Science, Carleton University, Ottawa, Canada
deugo@scs.carleton.ca

Prof. Elizabeth Kendall
Sun Microsystems Chair of Network Computing
Monash University, Australia
kendall@rmit.ed.au

Dr. Michael Weiss
Strategic Technology, Mitel Corporation, Ottawa, Canada
michael_weiss@mitel.com

1

Outline

- Introduction
  - Background
  - Philosophy
  - Requirements
- Current Research
  - Architectural, Communication, Travelling, Coordination Patterns
- Writing Patterns
  - Approach
  - Formats
  - Patterns for Writing Patterns
- Concluding Remarks

2

Introduction

Background
Philosophy
Requirements

3

Why the Comments?

- MAC3
  - "... gaining more widespread acceptance and recognition as a useful abstraction and technology.."
  - "..we are uninterested in papers that describe yet another mobile agent system..."

- Starting to understand research elements:
  - principles
  - facts
  - fundamental concepts
  - techniques
  - architectures

4

Slide 4 of 126

Introduction

- Roots in Alexander's work on urban design and building architecture

  - Alexander: Each pattern is a three-part rule, which expresses a relation between a certain context, a problem and a solution... The pattern is, in short, at the same time a thing which happens in the world and the rule which tells us how to create the that thing, and when we must create it. It is both a process and a thing; both a description of a think which is alive, and a description of the process which will generate that thing (Timeless p. 247)

5

Slide 5 of 126

Pattern Definition

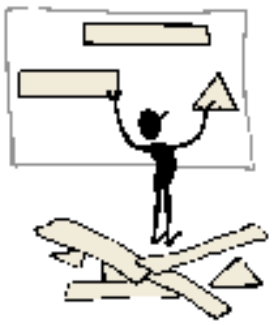
- A pattern is a piece of literature that describes a design problem and a general solution for the problem in a context
  - more than rules of thumbs, like recipes not plans
  - plans can be reverse engineered but recipes can not be (easily) documentation
  - driven by principles
  - serve human and social needs
  - capture important practices and existing methods and practices uncodified by conventional methods form of communication
  - capture structure not immediately apparent

6

Slide 6 of 126

Pattern Language

- Collection of patterns that build on each other to generate a system
- A software family
- A collection of rules that build all members of a family and only members of a family, e.g. Farmhouses, Cap Cod houses, Testing
- Place individual patterns in context
- Not a decision tree of patterns



7

Slide 7 of 126

Pattern Language Cont'd

- Alexander (Timeless, p 312-317)
  - Each pattern then, depends both on the smaller patterns it contains, and on the larger patterns within which it is contained
  - And it is the network of these connections between patterns which creates the language
  - In this network, the links between the patterns are almost as much a part of the language as the patterns themselves.
  - It is, indeed the structure of the network which makes sense of individual patterns, because it anchors them, and helps make them complete.

8

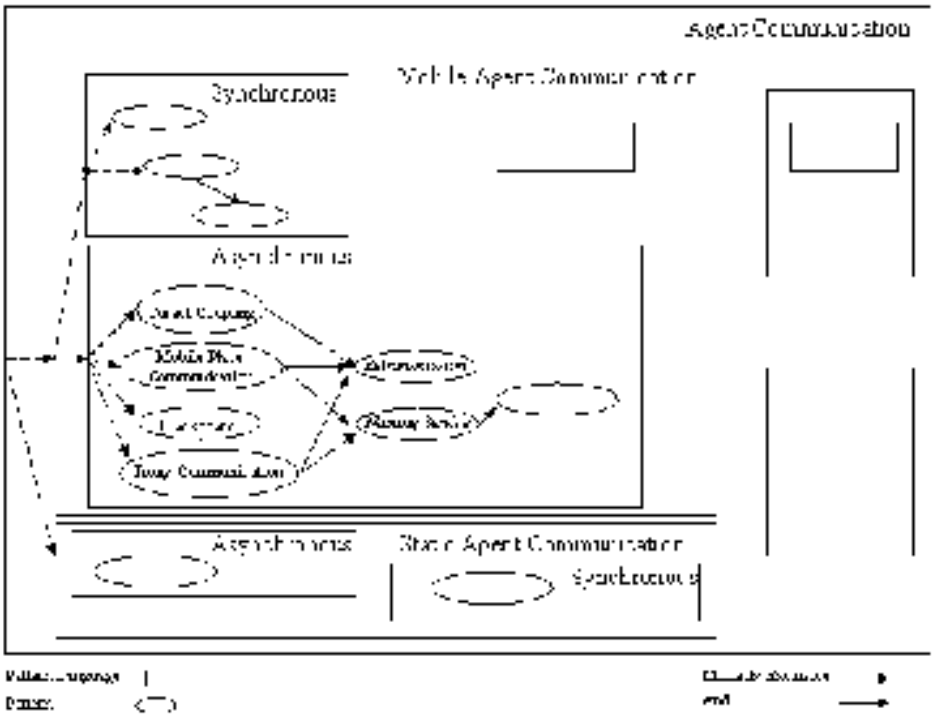
Pattern Language Cont'd

- The language is a good one, capable of making something whole, when it is morphologically and functionally complete.
- The language is morphologically complete when I can visualize the kind of buildings which it generates very concretely.
- And the language is functionally complete when the system of patterns it defines is fully capable of allowing all its inner forces to resolve themselves,

9

Slide 9 of 126

Communication Pattern Language



10

Generativity

- The structure of patterns are not themselves solutions, but they generate solutions
- Patterns that work this way are called generative
  - Alexander: This quality in buildings and in towns cannot be made, but only generated, indirectly by the ordinary actions of the people, just as a flower cannot be made, but only generated from a seed, (1979, p. xi)
- This generativity is an elusive quality, so elusive he calls it 'the quality without a name'

11

Slide 11 of 126

Why is Generativity Important?

- Real problems go deeper than surface symptoms
- Address problems with emergent behavior
- Good pattern fruit of intense review and refinement
- Understand the principles and values of lasting solutions and long-term emergent behavior
- Go beyond quick fix

12

Slide 12 of 126

Why Bother?

- Patterns capture obscure but important practices:
  - Design
  - Coding
  - Analysis
- Patterns capture hidden structure and decisions
  - Deep components of architecture and design are larger than any architectural build block such as procedures and objects

13

Pragmatics: What Patterns Can Do?

- Productivity
  - short-circuit discovery
  - avoid rework

- Development Interval
  - reduce time
  - road-maps

- Cost
  - Customer Satisfaction

14

Slide 14 of 126

Pragmatics: What Patterns Can't Do?

- Don't cover all situations
  - each new domain begs new patterns
  - patterns only come with experience
- For humans, not machines
  - no case-tools, yet
- Still need experts
  - to develop patterns
  - to determine when to apply them

15

Why are Patterns Important?

- Much of intelligent software development to date, especially involving agents, has been done ad hoc, creating many problems
  - Lack of agreed definitions
  - Duplicated effort
  - Inability to satisfy industrial strength requirements
  - Difficulty identifying and specifying common abstractions above the level of single agents
  - Lack of common vocabulary
  - Complexity
  - Only goals and solutions presented

- Problems limit the extent to which "industrial applications" can be built using technology

16

Classifying Patterns

- Idioms
  - Low-level Patterns that depend on specific implementation technologies (Lazy optimization in Smalltalk)
- Design Patterns
  - Problems, forces and solution are language independent so they stand as general design practices for common classes of software problems (Singleton)
- Framework Patterns
  - A partially completed body of code designed to be extended built on may system level patterns that tie together system parts and mechanisms (Client-Server)

17

Slide 17 of 126

Classifying Patterns Cont'd

- Anti
  - encode practices that don't work or are destructive (Blob Object)
- Meta
  - Nothing much meta, a set of patterns that describes how to construct frameworks independent of specific domains
- Elementary
  - Building block from which other patterns can be built.
- Others
  - Many patterns cut across all boundaries
  - Agent

18

Formats

- Alexander's Form

IF you find yourself in CONTEXT
for example EXAMPLES,
with PROBLEM,
entailing FORCES
THEN for some REASONS,
apply DESIGN FORM AND/OR RULE
to construct SOLUTION
leading to NEW CONTEXT and
OTHER PATTERNS

19

Slide 19 of 126

Modern Alexander Format

- Name of Pattern
- Aliases
- Problem
- Context
- Forces
- Solution
- Resulting Context
- Rationale
- Known Uses
- Related Patterns
- Sketch
- References
- Example

Gang of Four Format (Gamma et al.)

- Pattern Name (Scope, Purpose)
- Intent
- Also Known As
- Motivation
- Applicability
- Structure
- Participants
- Collaborations
- Consequences
- Implementation
- Sample Code and Usage
- Known Uses
- Related Patterns

21

Slide 21 of 126

Portland Form

- Portland Form makes a statement that goes something like: "such and so forces create this or that problem, therefore, build a thing-a-ma-jig to deal with them." The pattern takes its name from the thing-a-ma-jig, the solution.

- The total paragraph structure ends up looking like:
  - Having done so and so you now face this problem..
  - Here is why the problem exists and what forces must be resolved...
  - Therefore:
    - Make something along the following lines. I'll give you the help I can...
    - Now you are ready to move on to one of the following problems...

22

Slide 22 of 126

Common Requirements

- Description of best practices
  - Or at least generally accepted practices. Some people see patterns as a step toward construction of definitive Software Engineering Handbooks.

- Appropriate generality
  - Evidence that the pattern recurs. This almost always requires that you abstract over several known uses. This may require mention of situations in which the pattern does not apply, along with references to alternative patterns.

23

Common Requirements Cont'd

- Scope
  - The context in which someone may want to apply the pattern is fully described. When appropriate, including references to other patterns that typically lead to application of this pattern.

- Constructiveness
  - The pattern is phrased in a way that allows people to build an instance of the solution. In some cases, this may entail a set of diagrams showing essential relations. In others, it may entail a series of design steps that pattern users should follow, along with a description and/or example of the solution form as it should appear.

24

Common Requirements Cont’d

- Completeness
  - All relevant forces are described
- Utility
  - Evidence that the solution successfully resolves the forces, or when they are only partially resolved, and/or when they introduce new forces, references to related patterns that may apply.

25

Current Research

Architectural patterns
Communication patterns
Travelling patterns
Coordination patterns

26

Architectural Patterns

- Layered Agent
- Mobile Agent
- Resource Manager
- Jurisdiction

27

Slide 27 of 126

Forces

- The architecture must be flexible to address simple and sophisticated agent behavior.
- The architecture must support reuse of design and code through extension for specific applications.
- The architecture must address security, scalability and integration issues.
- The architecture must hide low-level details such as distribution and security aspects.

Correspondence to Patterns

	Flexiblity	Reuse	Security, scalability, integration	Hide details
Layered Agent	X	X		
Mobile Agent			X	X
Resource Manager		X	X	
Jurisdiction			X	

29

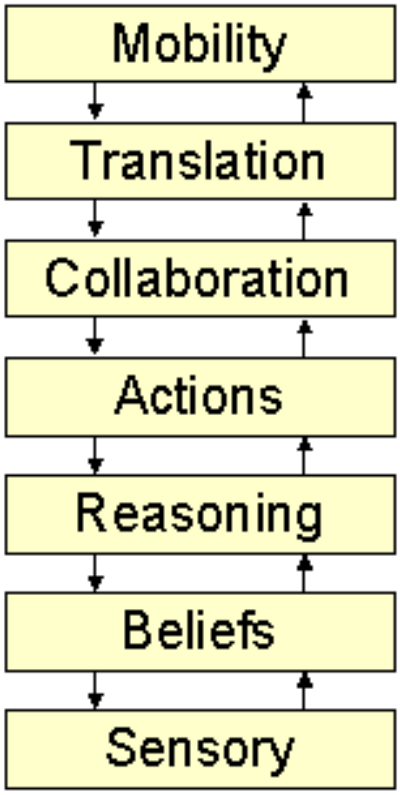
Slide 29 of 126

Layered Agent

- Problem: How do you organize and structure agent behavior into software?
- Context: You are writing an agent system.
- Solution:
  - Decompose an agent into layers addressing the different dimensions of strong agency (autonomous, social, reactive, proactive).
  - Each layer implements a sub-framework on which higher-level behavior can be built. The layers can be customized to the needs of a specific application.

30

Layered Agent



31

Slide 31 of 126

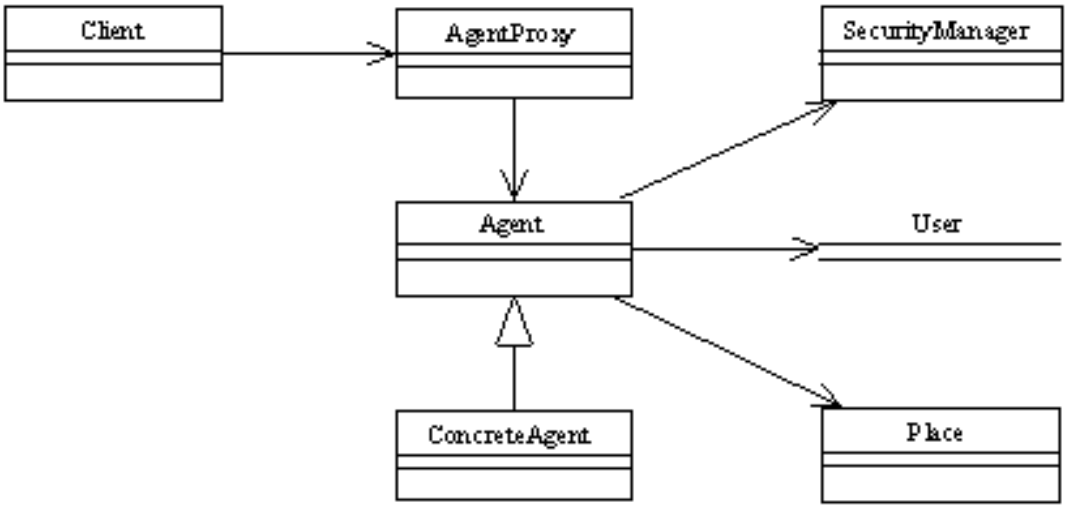
Mobile Agent

- Problem: How do you encapsulate the security and distribution aspects of agents?
- Context: You are writing a mobile agent system. You are concerned with weak agency issues.
- Solution:
  - Client access through an agent proxy.
  - Control all operations available via the proxy through a security manager.
  - Generic agent to implement mobility.
  - Represent the business-specific behavior in a subclass of the generic agent.

32

Slide 32 of 126

Mobile Agent



33

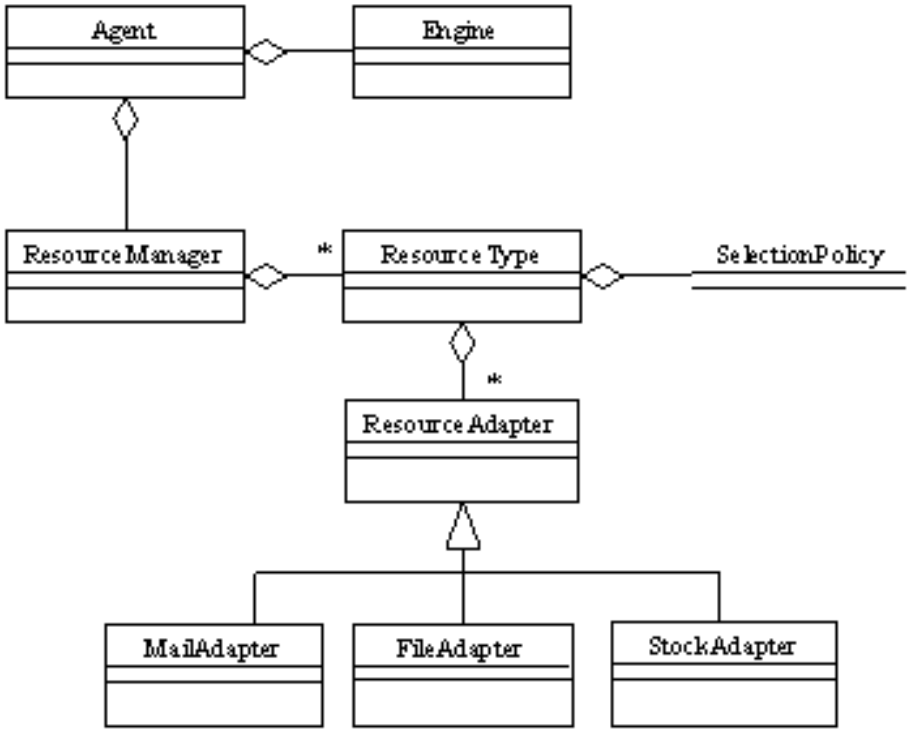
Slide 33 of 126

Resource Manager

- Problem: How can you minimize dependencies between an agent and its resources?
- Context: You are writing an agent that needs to interact with external resources.
- Solution:
  - Represent entire or part of resources (e.g. mail, files, stock quote servers) by resource adapters.
  - Adapters only relate to other adapters through the engine (via events and rules).
  - Resource manager uses selection policies to choose the most suitable adapter for each request.

34

Resource Manager



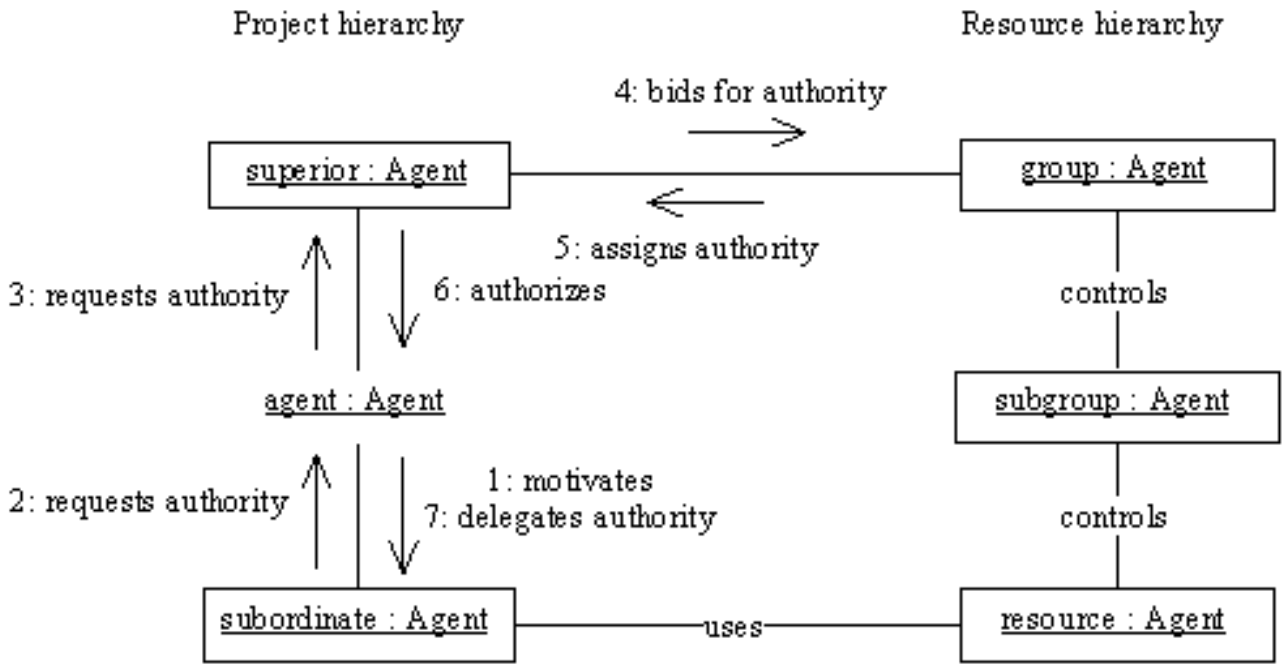
35

Jurisdiction

- Problem: How can agents share resources in a large-scale agent system?
- Context: The external resources are used by many agents. You need to ensure resources are assigned to agents working on critical tasks.
- Solution:
  - Introduce higher-level agents that manage resource-managing agents (jurisdiction).
  - Higher-level agents set policies of lower-level agents and resolve policy conflicts.
  - Higher-level agents endow agents in need of service with authority over resources.

36

Jurisdiction



37

Slide 37 of 126

Communication Patterns

- Meeting
- Ambassador
- Finder
- Proxy
- Badges
- Whiteboard
- Translator

Forces

- Mobile agents change locations frequently.
- Communication takes place on a peer-to-peer basis.
- Members of a group might change over time.
- Communication peers are not known to each other.
- Message indirection can cause additional traffic.
- Agents in different locations need to interact heavily.
- An agent needs to store data locally.
- Agents use different languages or semantics.

39

Correspondence to Patterns

	Changing agent locations	Peer-to-peer	Changing group members	Peers not known	Message indirection	Extensive interaction	Local storage	Heterogeneity
Meeting	X					X		
Ambassador			X		X			
Finder	X	X						
Proxy	X	X			X			
Badges		X		X				
Whiteboard					X		X	
Translator								X

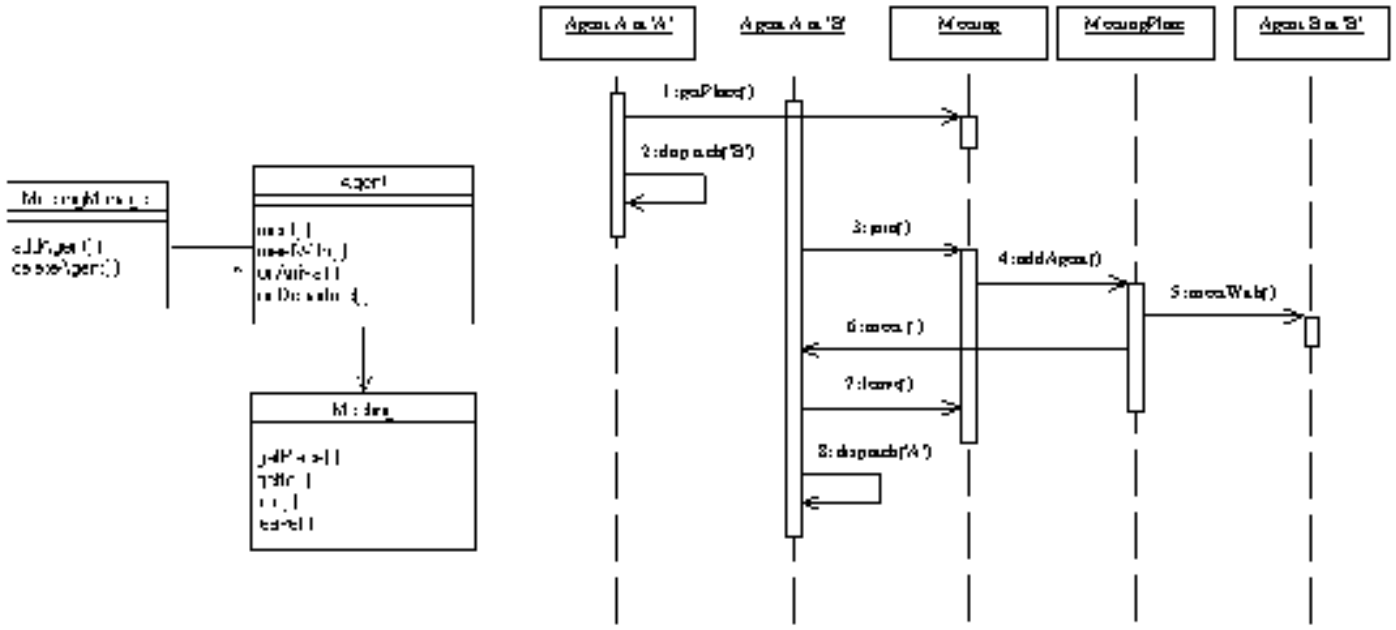
40

Slide 40 of 126

Meeting

- Problem: How do you synchronize agents in time and place?
- Context: You are writing a mobile agent system. Agents need to interact extensively.
- Solution:
  - Abstracts synchronization in time and place that is required for local interaction.
  - Agents dispatch themselves to a meeting place where they engage in local interaction.

Meeting

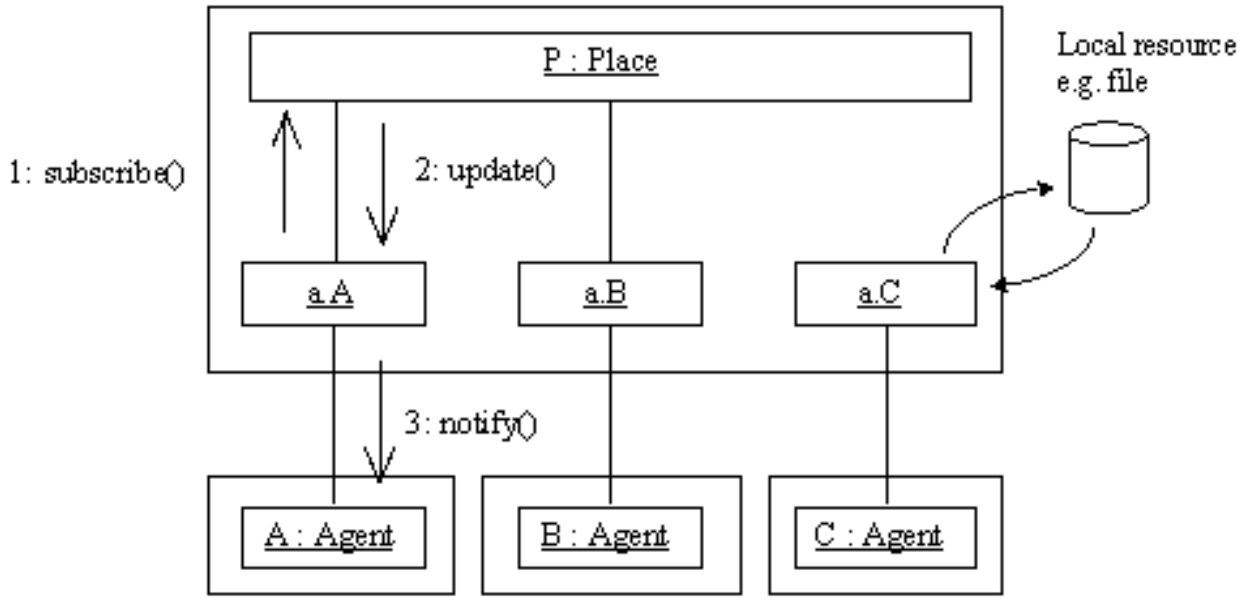


Ambassador

- Problem: How can an agent monitor a remote location for events of interest?
- Context: Agents in a remote location generate events that an agent needs to process.
- Solution:
  - An agent sends an Ambassador agent to the remote place that should be monitored.
  - An Ambassador waits for specified events upon which it should notify the delegating agent.
  - May be used in conjunction with Translator pattern.

43

Ambassador



44

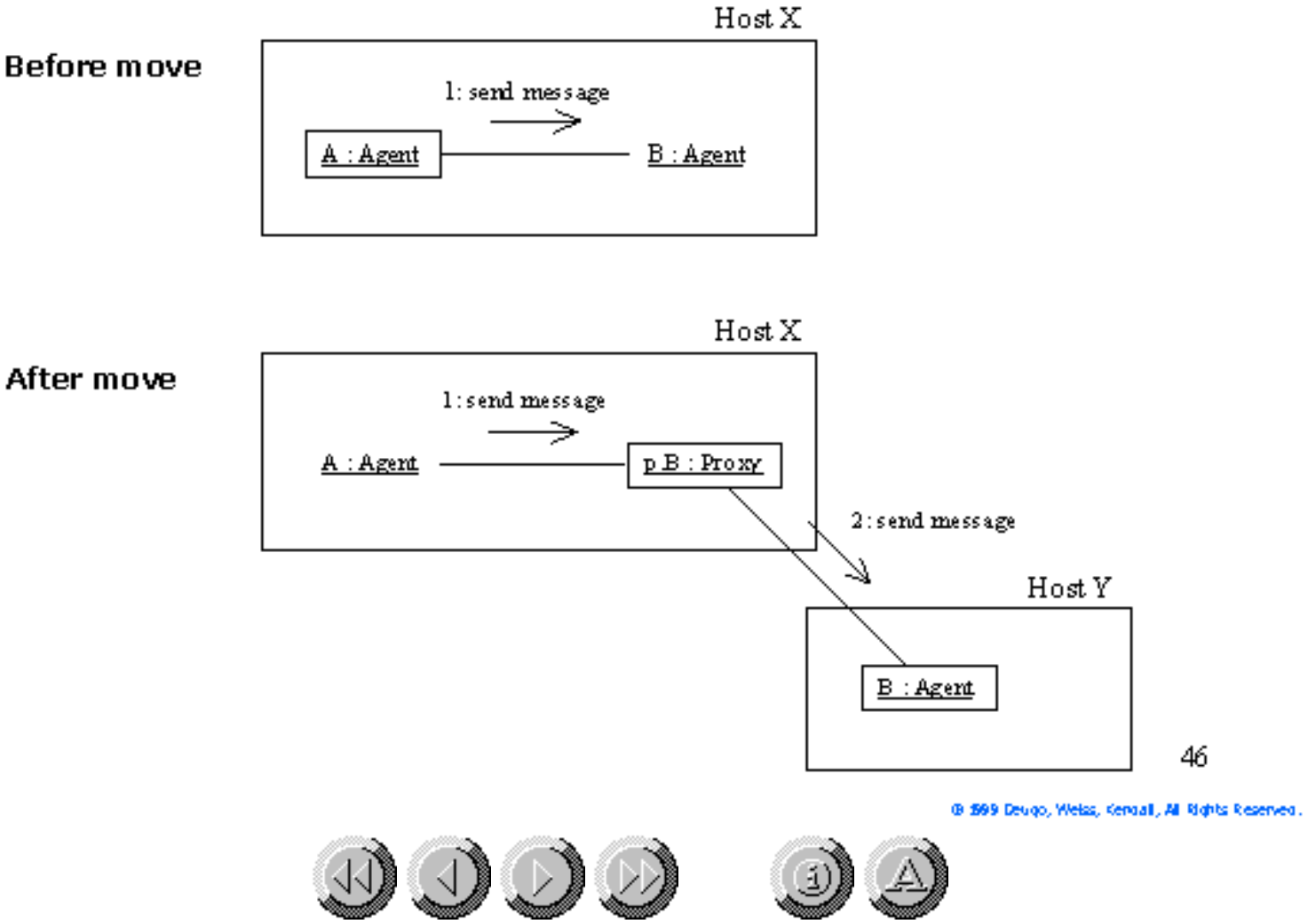
Slide 44 of 126

Proxy

- Problem: How do agents communicate if the sender does not expect receivers to move?
- Context: Agents must always be reachable at the same address. You therefore want to keep changes to an agent's location transparent.
- Solution:
  - When an agent moves, it creates a Proxy in the old place to hide the change of place.
  - A proxy either forwards messages to the agent (active proxy) or stores them for later pick-up by the agent (passive proxy).

45

Proxy

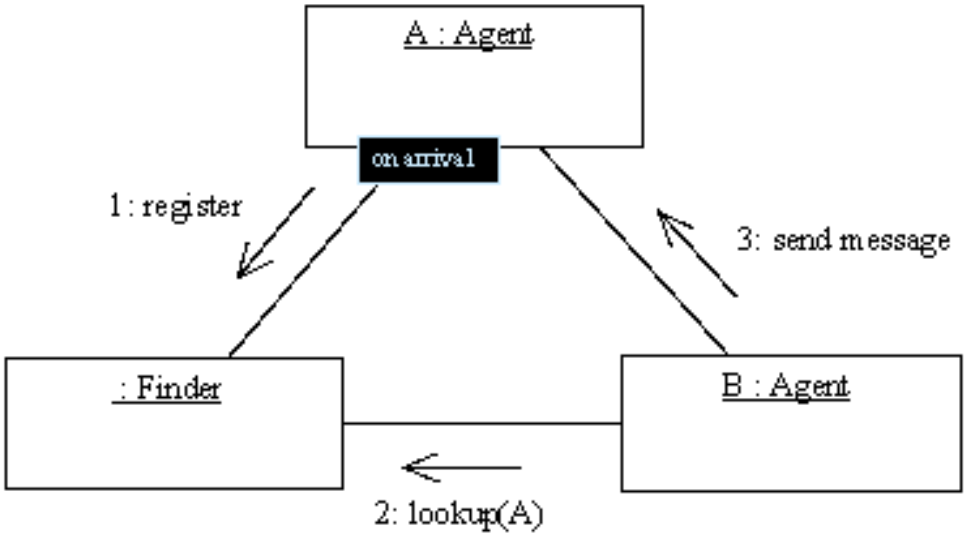


Finder

- Problem: How do mobile agents locate other agents if the know their name?
- Context: Agents move so frequently that the Proxy pattern creates too much overhead.
- Solution:
  - Each agent is assigned a unique identity.
  - A stationary finder agent maintains a name and location database of agents.
  - Agents register their current location upon arrival at a new place.
  - Other agents send lookup requests to the finder.

47

Finder



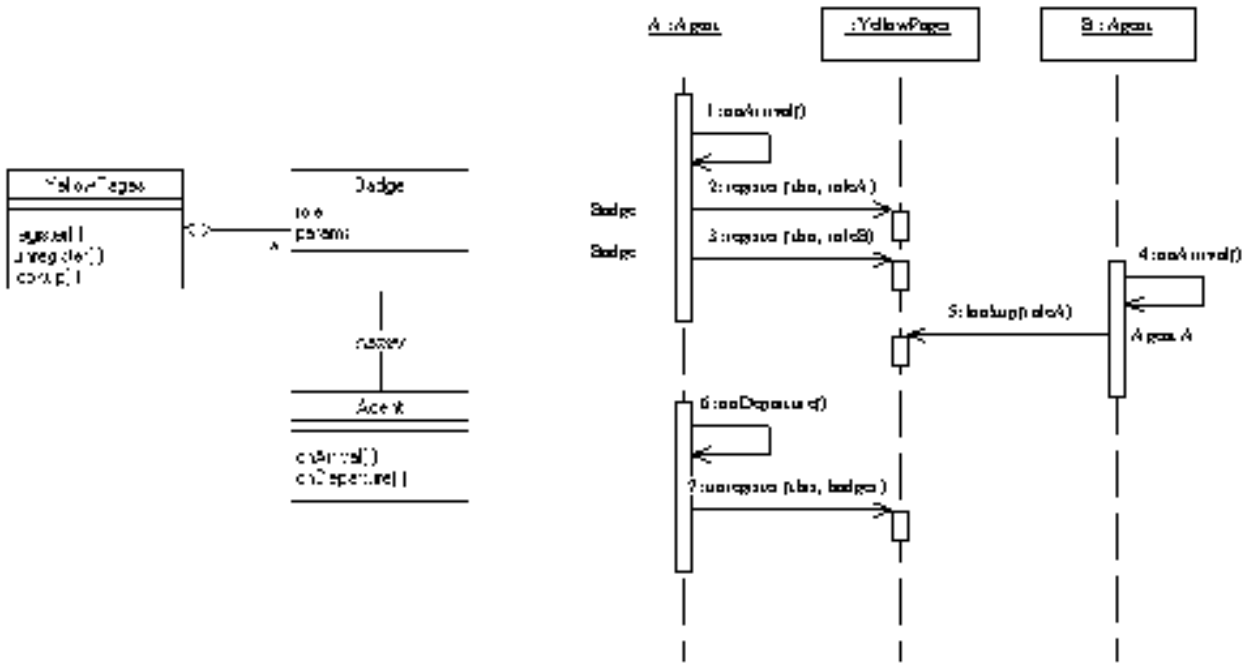
48

Slide 48 of 126

Badges

- Problem: How do agents find a suitable peer without supplying a specific agent?
- Context: A group of agents of a given type need to collaborate in the same place.
- Solution:
  - Attach badges to agents. Every badge has a unique id and agents can carry several badges.
  - A place provides a service to find a local agent carrying a certain badge.
  - Agents belonging to the same group can carry the same badge.

Badges



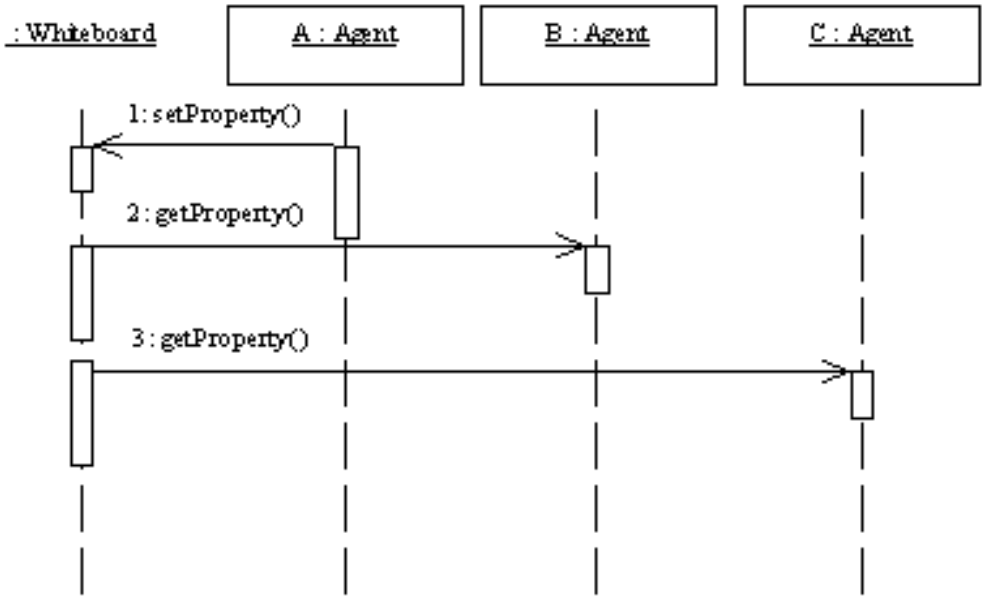
50

Slide 50 of 126

Whiteboard

- Problem: How do agents exchange location-specific data with other mobile agents?
- Context: A mobile agent wants to store data at a place for other agents to read.
- Solution:
  - Provide an area at every place where agents can leave messages for each other.
  - For example, an agent needs to know that a colleague just visited the place.
  - Allows agents to post and read messages.

Whiteboard



52

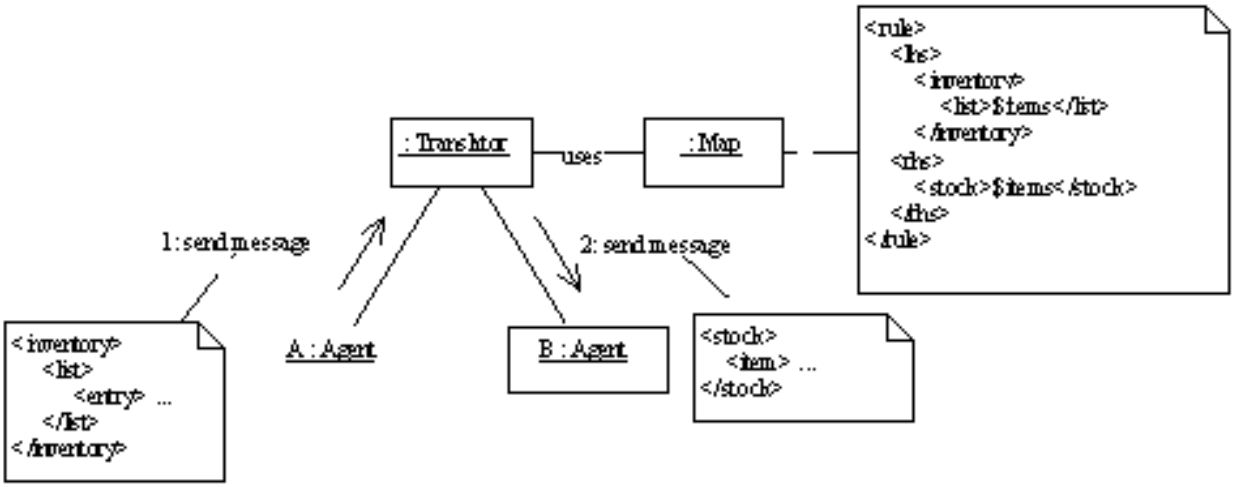
Slide 52 of 126

Translator

- Problem: How can agents with different ontologies communicate?
- Context: You are writing an agent application where the agents use different ontologies.
- Solution:
  - The Translator agent translates an agent's message into another language or semantics.
  - In simple cases it can consult a mapping table with corresponding messages.
  - In other cases it can convert messages via a rule-based transformation language.

53

Translator



54

Slide 54 of 126

Travelling Patterns

- Itinerary
- Ticket
- Router
- Relocator

Forces

- Destinations might be unknown or not respond.
- Access to destinations is restricted.
- Agents must choose between alternative routes.
- Agents need to move together.

Correspondence to Patterns

	Unknown locations	Access restricted	Alternative routes	Moving together
Itinerary	X			
Ticket		X		
Router			X	
Relocator				X

57

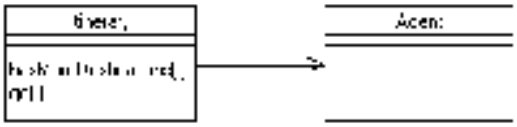
Slide 57 of 126

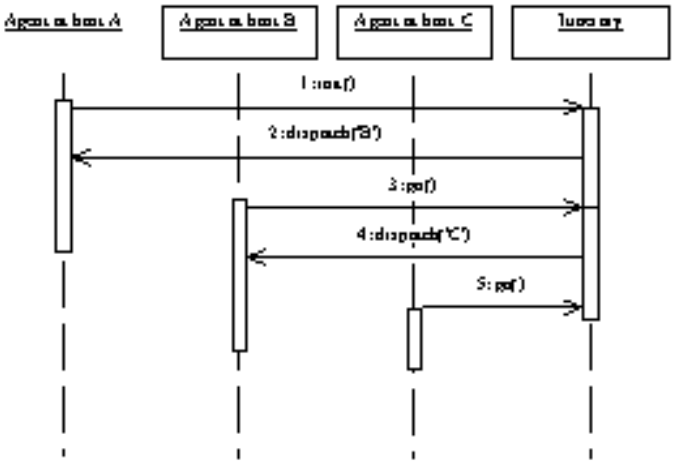
Itinerary

- Problem: How do you encapsulate an itinerary?
- Context: You are concerned with routing agents among multiple destinations.
- Solution:
  - An Itinerary maintains a list of destinations and defines how to navigate them.
  - The agent dispatches itself by asking the Itinerary to go to the next destination.
  - An Itinerary handles exceptions, e.g. if a host is unknown or does not respond.

58

Itinerary





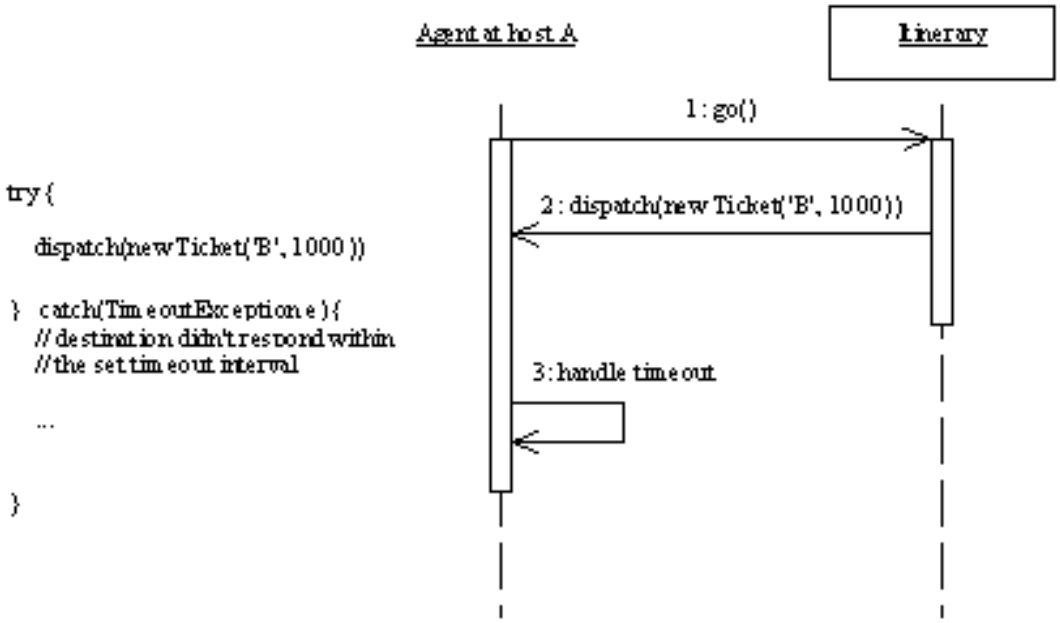
59

Slide 59 of 126

Ticket

- Problem: How can an agent make reasonable decisions while traveling?
- Context: An agent dispatches itself to another destination, but the destination doesn't reply.
- Solution:
  - A ticket is an enriched form of an URL specifying the agent's destination and other terms of the trip.
  - It embodies e.g. the means by which it must be made and the time by which it must be completed.

Ticket

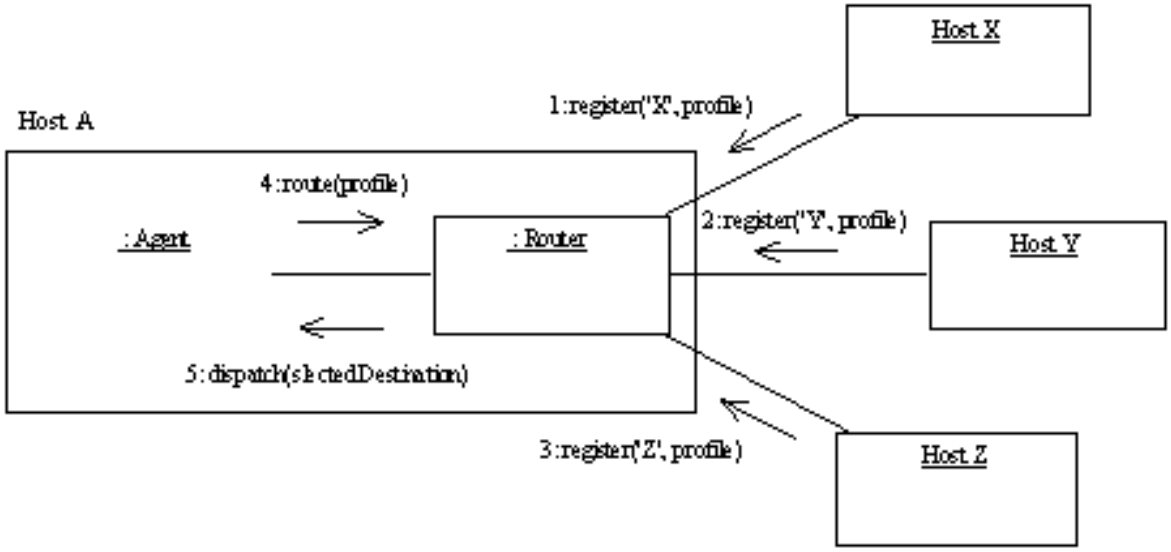


Router

- Problem: How can agents select a destination in which a task can be performed best?
- Context: You are writing an e-commerce application. Your agent acts as a middleman guiding shopping agents to suitable sellers.
- Solution:
  - Attach a set of attributes (the agent's profile) to each agent such as the language it uses to send and receive messages, or quality of service requirements.
  - Destinations register desired profiles with a router that dispatches arriving agents to the destination that "best" matches the agent's attributes.

62

Router



63

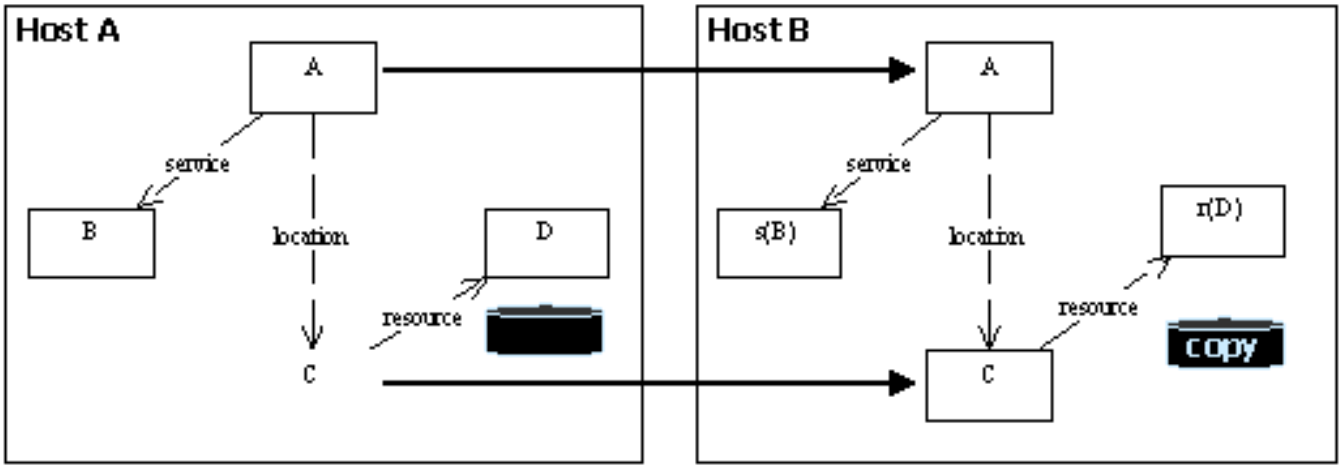
Slide 63 of 126

Relocator

- Problem: How do you preserve references between mobile agents?
- Context: Agents expect other agents to move with them to a new place.
- Solution:
  - Same location: members move together.
  - Same service: service relationship needs to be reestablished in the new location.
  - Same context: resources are replicated in the new place if they cannot be moved.

64

Relocator



65

Slide 65 of 126

Coordination Patterns

- Master-Slave
- Marketplace
- Specialist
- Negotiating Agents
- Subsumption

Forces

- Partitioning of work can be used to increase reliability, performance or accuracy.
- Partitioning should be transparent to clients.
- Agents need to collaborate to perform complex tasks.
- Agent behaviors may conflict.
- Agents and agent applications should be evolvable.

Correspondence to Patterns

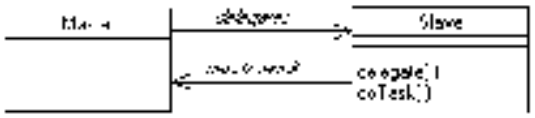
	Partitioning	Partitioning transparency	Need to collaborate	Conflicting behaviors	Evolvability
Master-Slave	X	X			
Marketplace			X		X
Specialist		X	X		
Negotiating Agents				X	
Subsumption				X	X

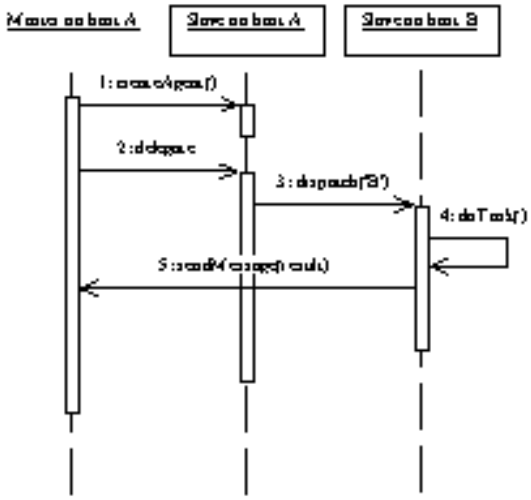
Master-Slave

- Problem: How do you delegate subtasks and coordinate their execution?
- Context: You decide to partition a task to increase reliability, performance or accuracy.
- Solution:
  - The master agent divides the task into subtasks, delegates the subtasks to slave agents and computes a final result from the partial results returned.
  - The master agent creates a slave agent for each subtask and dispatches it. The master can continue its work in parallel with its slaves.

69

Master-Slave





70

Slide 70 of 126

Marketplace

- Problem: How do you match up producers and consumers of resources with one another?
- Context: Your agent system evolves constantly. Instead of setting up links between agents up-front you let the agents create them on-the-fly.
- Solution:
  - Buyers request resources with certain attributes. Sellers with products that match or approximate those attributes issue bids.
  - Buyers evaluate bids and choose the product that "best" meets the attributes they care most about (least expensive, highest quality).

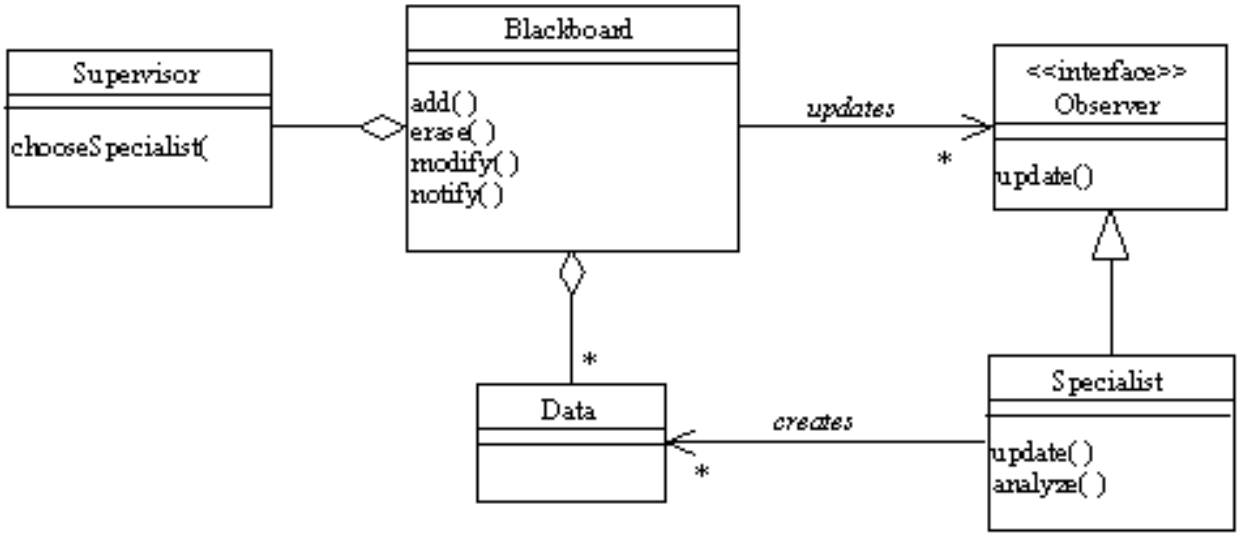
71

Specialist

- Problem: How do you coordinate the joint action of a group of specialized agents?
- Context: An agent system consists of agents that are each specialized to perform a particular task. You need to coordinate their actions.
- Solution:
  - A group of specialists continually observe a blackboard for items of interest and signal when they want to add, erase, update data.
  - A supervisor chooses a specialist to make a contribution and then decides if the state of the blackboard represents a solution or not.

73

Specialist



74

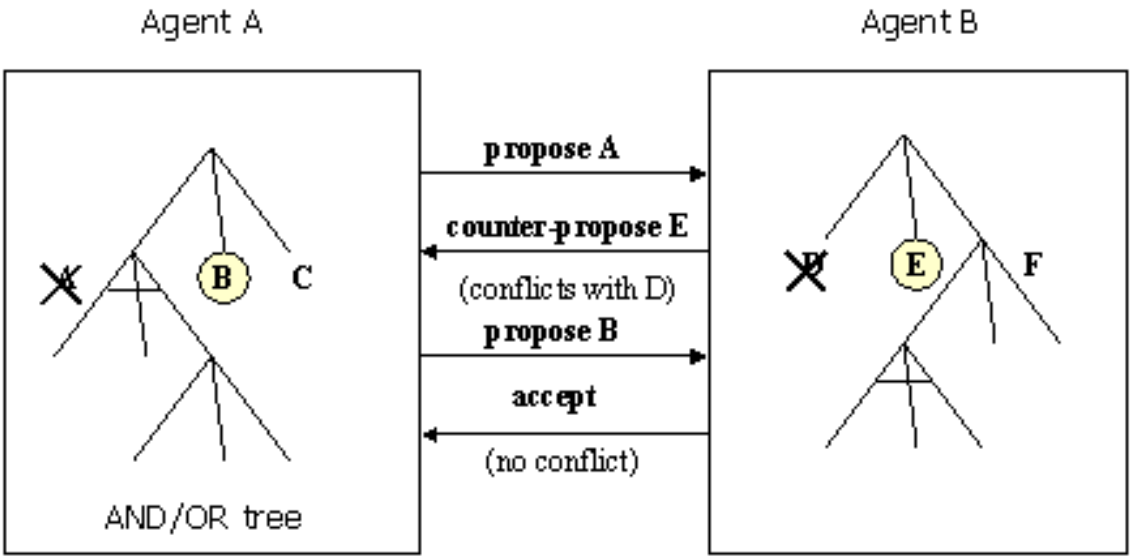
Slide 74 of 126

Negotiating Agents

- Context: How do you detect and resolve between agents with conflicting intentions?
- Problem: Agent in your system may give conflicting instructions to the environment. They need to align their actions with each other.
- Solutions:
  - Agents make their intentions explicit by exchanging constraints on what they can do.
  - Agents replan their actions to meet these constraints and may make counter-proposals.
  - Policies guide agents in their choice among alternative courses of actions.

75

Negotiating Agents



76

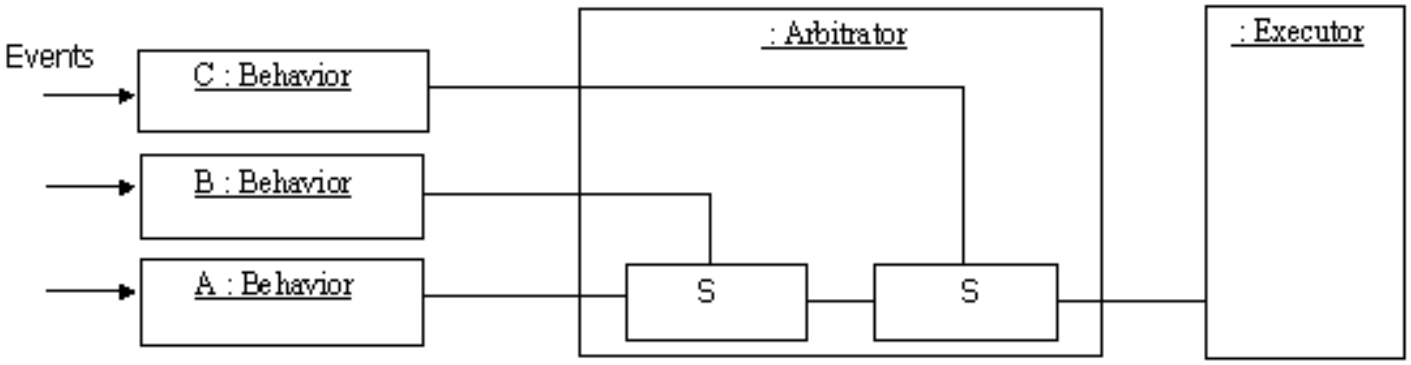
Slide 76 of 126

Subsumption

- Problem: How do you compose an agent from independently written behaviors?
- Context: You are composing the behavior of an agent from simpler behaviors. These may define conflicting actions in response to an event.
- Solution:
  - Make behaviors adhere to a protocol of declaring their intention to perform an action (e.g. by setting a flag) and waiting to be selected by an arbitrator.
  - An arbitrator selects the behavior to execute and suppresses the others (e.g. using a priority scheme)

77

Subsumption



S: suppress

78

Slide 78 of 126

Agent Pattern Sources

- Amund Aarsten and Davide Brugali, Giuseppe Menga, Patterns for Cooperation, PLoP, 1996
- Yarif Aridor and Danny Lange, Agent Design Patterns: Elements of Application Design, Conference on Autonomous Agents, 108-115, ACM, 1998
- D. Deugo, F. Oppacher, A. Ashfield, M. Weiss, Communication as a Means to Differentiate Objects, Components and Agents, Proceedings of Technology of Object-Oriented Languages & Systems (TOOLS-30 USA'99), IEEE Computer Society Press, 376-386, 1999
- D. Deugo, F. Oppacher, J. Kuester, I. Von Otte, Patterns as a Means for Intelligent Software Engineering, Proceedings of The International Conference on Artificial Intelligence (IC-AI'99), Vol II, CSREA Press, 605-611,1999
- Eric Freeman, Susanne Hupfer and Ken Arnold, JavaSpaces Principles Patterns and Practice, Addison-Wesley, 1999
- Elizabeth A. Kendall, P.V. Murali Krishna et al, Patterns of Intelligent and Mobile Agents, Conference on Autonomous Agents, 92-99, ACM, 1998
- Danny Lange and Mitsuru Oshima, Programming and Deploying Java Mobile Agents with Aglets, Addison-Wesley, 1998
- Alberto Silva and Jose Delgado, The Agent Pattern: A Design Pattern for Dynamic and Distributed Applications, EuroPLoP, 1998
- Michael Weiss, A Hierarchical Blackboard Architecture for Distributed AI Systems, Conference on Software Engineering and Knowledge Engineering, 349-355, IEEE, 1992
- Michael Weiss, Tom Gray and Aurora Diaz, Experiences with a Service Environment for Distributed Multimedia Applications, Feature Interaction in Telecommunications and Distributed Systems IV, 242-253, IOS Press, 1997

Writing Patterns

Approach

Formats

Patterns for Writing Patterns

80

Approach

- You have to write to a VERY HIGH STANDARD to have a work incorporated into the patterns literature.
- Your pattern will never be perfect
  - but you have to iterate, and incorporate feedback
- It's all about making your documentation easy to read and understand
  - BUT, you don't get to determine your own style/ way of documenting your pattern
- Patterns follow a certain format

81

Why Patterns?

- Capture domain expertise
- Document design decisions and rationale
- Reuse wisdom and experience of master practitioners
- Convey expert insight to novices
- Give a shared vocabulary for discussion
- Provide reusable solutions
- Restore the human element to software development, as Alexander strives to restore the human element to architecture Are about people

82

What isn't a Pattern?

- A solution to a problem in a context
  Problem: How do you allocate objects in a memory
  - **Context**: A large OO system in a virtual memory computer
  - **Solution**: Run some typical problems and figure out which object communicate frequently in a time locale and put them on the same page

What isn't a Pattern Cont'd?

- An untested idea/theory
- A simple rule or recipe
- An algorithm or a data structure
- An old thing written in a pattern form
- A solution that has worked only once
- A "silver bullet" or panacea

What is a Good Pattern?

- It is a mature (passed the test time), proven (successful), and recurring solution (rule of 3)
- It builds upon the insight of the expert problem solver
- It resolves a dense set of forces
- It tells the problem solver when to use this pattern, what to do, and how to solve the problem (context specific)
- It has the quality without a name and contributes to human comfort
- It is an element in a pattern language and a configuration in a system
- It is generative and cannot be automated

85

Writing Patterns and Workshops

- Patterns are a literary form for communication and sharing
- Patterns are a formalism for capturing knowledge and wisdom
- You can have a better understanding of writing patterns by writing your own patterns
- Writing patterns is hard. Many workshops are designed to improve pattern writing skill
  - PLoP(US), EuroPLoP, ChiliPLoP, OOPSLA
  - TOOLS Pacific '98

86

Pattern Formats

There are two main pattern formats:

- Alexander's original format, which is reflected in the AG Template
- Gang of Four (Gamma, Johnson, Vlissides, and Helm)
- You have to select a format for your pattern

87

Alexander's Pattern Language

The Timeless Way of Building (TTWoB)

- **The Quality**
  - To seek the timeless way we must first know the quality without a name. (a.k.a. QWAN)
- **The Gate**
  - To reach the quality without a name we must then build a living pattern language as a gate.
- **The Way**
  - Once we have built the gate, we can pass through it to the practice of the timeless way.

88

Alexander's Pattern Language Cont'd

- Consider a small pattern language for building a porch:
  - PRIVATE TERRACE ON THE STREET (140)
  - SUNNY PLACE (161)
  - OUTDOOR ROOM (163)
  - SIX-FOOT BALCONY (167)
  - PATHS AND GOALS (120)
  - CELLING HEIGHT VARIETY (190)
  - COLUMNS AT THE CORNERS (212)
  - FRONT DOOR BENCH (242)
  - RAISED FLOWERS (245)
  - DIFFERENT CHAIRS (251)

89

Slide 89 of 126

Alexander's Pattern Language Cont'd

- A picture shows an archetypal example
- An introductory paragraph sets the context for the pattern, explaining how it helps to complete certain larger patterns
- Three diamonds mark the beginning of the problem
- A headline, in bold type, gives the essence of the problem
- The body of the problem describes a set of forces which occur in the problem

Alexander's Pattern Language Cont'd

- The solution, in bold type, describes the field of physical and social relationships which are required to solve the stated problem, in the stated context
- A diagram shows the solution's structure
- Three diamonds shows the finish of the main body of the pattern
- A paragraph ties the pattern to all those smaller patterns in the language, which are needed to complete this pattern

Alexander's Pattern Language Cont'd

180 WINDOW PLACE

... this pattern helps complete the arrangement of the windows given by ENTRANCE ROOM (130), ZEN VIEW (134), LIGHT ON TWO SIDES OF EVERY ROOM (159), STREET WINDOWS (164)... ...

❖❖❖

Everybody loves window seats, bay windows, and big windows with low sills and comfortable chairs drawn up them.

... These kinds of windows which create "places" next to them are not simply luxuries; they are *necessary*. A room which does not have a place like this seldom allows you to feel fully comfortable or perfectly at ease. Indeed, a room without a window place may keep you in a state of perpetual unresolved conflict and tension -- slight, perhaps, but definite.

92

Alexander's Pattern Language Cont'd

180 WINDOW PLACE

This conflict takes the following form. If the room contains no window which is a "place", a person in the room will be torn between two forces:

1. He wants to sit down and be comfortable
2. He is drawn towards the light

Obviously, if the comfortable places ... are away from the windows, there is no way of overcoming this conflict... A room where you feel truly comfortable will always contain some kind of window place ...

Therefore:

In every room where you spend any length of time during the day, make at least one window into a "window place".

❖❖❖

93

Slide 93 of 126

Alexander’s Pattern Language Cont’d

180 WINDOW PLACE

Make it low and self-contained if there is room for that -- ALCOVES (179); keep the sill low -- LOW SILL (222); put in the exact positions of frames, and mullions, and seats after the window place is framed, according to the view outside -- BUILT-IN SEATS (202), NATURAL DOORS AND WINDOWS (221). And set the window deep into the wall to soften light around the edges -- DEEP REVEALS (223). Under a sloping roof, use DORMER WINDOWS (231) to make this pattern ...

94

Slide 94 of 126

Pattern Form and Essential Elements

- **Name**: a meaningful "conceptual handle" for communication
- **Context**: what must be true in order to apply this pattern
- **Problem**: statement of the problem, usually start with a question, with one or more examples
- **Forces**: conflicts, concerns, difficulties

95

Pattern Form and Essential Elements

- **Solution**: an instruction on how to solve the problem and a diagram showing the solution structure
- **Resulting Context**: benefits, consequences, and how forces are resolved; what other patterns are needed to complete this one?
- **Known Uses**: at least 3 applications of the pattern (Rule of 3)

96

Slide 96 of 126

The AG Format

- Pattern Name:
  - Name of pattern goes here
- Aliases: Aliases (or none)
- Problem
  - Give a statement of the problem that this pattern resolves. The problem may be stated as a question.
- Context
  - Describe the context of the problem.
- Forces
  - Describe the forces influencing the problem and solution. This can be represented as a list for clarity.
- Solution
  - Give a statement of the solution to the problem.

97

The AG Format Cont'd

- Resulting Context
  - Describe the context of the solution.
- Rationale
  - Explain the rationale behind the solution.
- Known Uses
  - List or describe places where the pattern is used.
- Related Patterns
  - List or describe any related patterns.
- Example
  - Give an example implementation of the pattern. This can be code, pseudo code, etc.

GoF Format

- Pattern Name (Scope, Purpose)
  - The pattern's name conveys the essence of the pattern succinctly. A good name is vital, because it will become part of your design vocabulary.
- Intent
  - A short statement that answers the following questions: What does the design pattern do? What is its rationale and intent? What particular design issue or problem does it address?
- Also Known As
  - Other well-known names for the pattern, if any.
- Motivation
  - A scenario that illustrates a problem and how the structures in the pattern solve the problem. The scenario will help you understand the more abstract description of the pattern that follows.

99

Slide 99 of 126

GoF Format Cont'd

- **Applicability**
  - What are the situations in which the pattern can be applied? How can you recognize these situations?
- **Structure**
- **Participants**
  - The classes and/or objects participating in the design pattern and their responsibilities.
- **Collaborations**
  - How the participants collaborate to carry out their responsibilities.
- **Consequences**
  - How does the pattern support its objectives? What are the trade-offs and results of using the pattern? What aspect of system structure does it let you vary independently?
- **Implementation**
  - What pitfalls, hints, or techniques should you be aware of when implementing the pattern? Are there language-specific issues?

100

Slide 100 of 126

GoF Format Cont'd

- Sample Code and Usage
  - Code fragments that illustrate how you might implement the pattern in C++ or Smalltalk.
- Known Uses
  - Examples of the pattern found in real systems. We include at least two examples from different domains.
- Related Patterns
  - What patterns are closely related to this one? What are the important differences? With which other patterns should this one be used?

Which Format?

- The formats are very similar
  - Intent - Problem
  - Structure - Solution
  - Applicability - Context
  - Consequences - Resulting Context, Rationale
- The main difference is that the Alexander form explicitly captures the forces acting on the problem. These forces must be resolved by the solution
- The forces should push and pull the reader in different directions, but eventually they are pulled toward the solution
- A well written Motivation section should also do this!

Patterns for Writing Patterns

A Pattern Checklist by Doug Lea...

- A Pattern
  - Describes a single kind of problem.
  - Describes the context in which the problem occurs.
  - Describes the solution as a constructable software entity.
  - Describes design steps or rules for constructing the solution.
  - Describes the forces leading to the solution.
  - Describes evidence that the solution optimally resolves forces.
  - Describes details that are allowed to vary, and those that are not
  - Describes at least one actual instance of use.
  - Describes evidence of generality across different instances.
  - Describes or refers to variants and subpatterns.

103

Patterns for Writing Patterns Cont'd

- Describes or refers to other patterns that it relies upon.
- Describes or refers to other patterns that rely upon this pattern.
- Relates to other patterns with similar contexts, problems, or solutions.

104

Slide 104 of 126

Qualities of a Pattern

Doug Lea's Qualities of a Pattern

- Encapsulation and Abstraction
- Openness and Variability
- Generativity and Composability
- Equilibrium

105

Slide 105 of 126

Seven Habits of Successful Pattern Writers

1. **Take Time to Reflect**

   Accumulate your own written experience. Work/ learn from others.

2. **Adhere to a Structure**

   A pattern is a structured exposition of a solution to a problem in a context.

3. **Be Concrete Early**

   GoF's Motivation section has a concrete example. Other pattern forms frequently employ a running example. Examples are important. Tell the whole truth.

4. **Keep Patterns Distinct and Complementary**

   Your patterns should be orthogonal and work together synergistically. How are your patterns different ? The Intent section should tell you. Compare and contrast your patterns.

5. **Present Effectively**

   Writing style and typesetting. Use drawings. Use a conversational style. You should be able to hear yourself talk.

106

Slide 106 of 126

Seven Habits of Successful Pattern Writers

6. Iterate TIRELESSLY

   You'll never get the pattern to be perfect. HOWEVER, you have to try!

7. Collect and Incorporate Feedback

   Shepherds. Writer's Workshops.

References on writing style:

- Strunk, William, Jr. and E.B. White. *The Elements of Style*, Macmillan, New York, 1979. Third edition.
- Trimble, John R. *Writing with Style: Conversations on the Art of Writing*, Prentice-Hall, Englewood Cliffs, NJ, 1975.
- Williams, Joseph M. *Style: Ten Lessons in Clarity and Grace*, Scott, Foresman and Co., Glenview, IL, 1985. Second edition.

A Pattern Language for Pattern Writing

G. Meszaros, Jim Doble

A Pattern Language for Pattern Writing

D. Patterns For Making Patterns Understandable
D.1 Clear Target Audience
D.2 Terminology Tailored to Audience
D.3 Understood Notations
D.4 Code Samples
D.4.1 Code Samples as Bonus
E. Pattern Language Structuring Patterns
E.1 Pattern Language Summary
E.1.1 Problem/Solution Summary
E.2 Common Problems Highlighted
E.3 Running Example
E.4 (Distinctive) Headings Convey Structure

A.1 Pattern: Pattern

- **Context:**
  - You are an experienced practitioner in your field. You have noticed that you keep using a certain **solution** to a commonly occurring **problem**. You would like to share your experience with others.
- **Problem:**
  - How do you share a recurring **solution** to a **problem** with others so that it may be reused?
- **Forces:**
  - Keeping the **solution** to yourself doesn't require any effort
  - Sharing the **solution** verbally helps a few others but won't make a big impact in your field.
  - People are unlikely to use a **solution** if you don't explain the reasons for using it.

110

Slide 110 of 126

A.1 Pattern: Pattern Cont'd

- Writing down your understanding of the **solution** is hard work and requires much reflection on how you solve the **problem**.
- Transforming your specific **solution** into a more widely applicable **one** is difficult.
- Writing down the **solution** may compromise your competitive advantage

Solution:

- Write down the **solution** using the pattern form. Capture both the **problem** and the **solution**, as well as the reasons why the **solution** is applicable. Apply *Mandatory Elements Present* to ensure that the necessary information is communicated clearly. Include *Optional Elements When Helpful* to capture any additional useful information. Distribute the resulting pattern to the largest audience that does not compromise your competitive advantage.

111

Slide 111 of 126

A.2 Pattern: Pattern Language

- **Context:**
  - You are trying to use the "pattern form" to describe a procedure with many steps or a complex **solution** to a complex **problem**. Some of the steps may only apply in particular circumstances. There may be alternate **solutions** to parts of the **problem** depending on the circumstances. A single pattern is insufficient to deal with the complexity at hand.
- **Problem:**
  - How do you describe the **solution** such that it is easy to digest and easy to use parts of the **solution** in different circumstances?

112

Slide 112 of 126

A.2 Pattern: Pattern Language Cont'd

Forces:

- A single large **solution** may be too specific to the circumstance and impossible to reuse in other circumstances.
- A complex **solution** may be hard to describe in a single pattern. A "divide and conquer" approach may be necessary to make the **solution** tractable.
- Factoring the **solution** into a set of reusable steps can be very difficult. Once factored, the resulting pieces may depend on one another to make any sense.
- Other pattern languages may want to refer to parts of the **solution**; they require some sort of "handle" for each of the parts to be referenced.

A.2 Pattern: Pattern Language Cont'd

- **Solution:**
  - Factor the overall **problem** and its **solution** into a number of related **problems** with **solutions**. Capture each **problem/solution** pair as a pattern within a pattern language. Each pattern should solve a specific **problem** within the shared context of the language. Strive to ensure that each pattern could conceivably be used alone.
  - To give the pattern language an identity of it's own, give it an *Evocative Name* by which it can be known and referenced. Describe the overall **problem** and how the patterns work together to solve it in a *Pattern Language Summary*. Relate the patterns to each other *using Readable References to Patterns* within the pattern description, especially in the **Context** and **Related Patterns** elements.

114

Slide 114 of 126

B.1 Pattern: Mandatory Elements Present

- **Aliases: All Elements Present**
- **Problem:**
  - How do you make sure that all necessary information is covered in a pattern?
- **Context:**
  - You are writing a pattern, either standalone or as part of a pattern language.
- **Forces:**
  - All patterns do not require the same kinds of information. Capturing all elements regardless of need only clutters many patterns.
  - If the necessary elements are missing, it becomes much harder to determine whether the pattern solves the reader's problem in an acceptable way.

115

Slide 115 of 126

B.1 Pattern: Mandatory Elements Present

- **Pattern Name**: A name by which this problem/solution pairing can be referenced.
- **Context**: The circumstances in which the problem is being solved imposes constraints on the solution. The **context** is often described via a "situation" rather than stated explicitly. Sometimes, the **context** is described in terms of the patterns that have already been applied. The relative importance of the **forces** (those that need to be optimized at the expense of others) is determined by the **context**.
- **Problem**: The specific **problem** that needs to be solved. Use *Context-Free Problem* to ensure that the **problem** is kept separate from the constraints on the **solution**.

117

B.1 Pattern: Mandatory Elements Present

- **Forces:**
  - The often contradictory considerations that must be taken into account when choosing a **solution** to a **problem**. The relative importance of the **forces** (those that need to be optimized at the expense of others) is implied by the **context**.
- **Solution:**
  - The **solution** to the **problem**. Note that many **problems** may have more than one **solution**, and the "goodness" of a **solution** to a **problem** is affected by the **context** in which the **problem** occurs. Each **solution** takes certain **forces** into account. It resolves some **forces** at the expense of others. It may even totally ignore some **forces.** The most appropriate **solution** to a **problem** is the one that best resolves the highest priority **forces** as determined by the particular **context**. Use *Solution Clearly Related to Forces* to ensure the reader understands why this **solution** was chosen.

118

Slide 118 of 126

B.3 Pattern: Visible Forces

- **Problem:**
  - A pattern presents a solution to a problem within a context. How do you ensure that the reader understands the choice of solution?
- **Context:**
  - You are writing a pattern or pattern language that is intended to convey one of potentially several solutions to a problem. You have applied Mandatory Elements Present; you are now writing the Forces section.
- **Forces:**
  - There are many different styles of patterns, some more structured than others.
  - People like to have convenient handles for concepts such as the forces which affect the choice of solution.

119

Slide 119 of 126

B.3 Pattern: Visible Forces Cont'd

- Prose pattern descriptions can be very pleasing to read but may be hard to use as a reference because the forces are buried in the prose.
- Having a separate Forces heading makes the forces very easy to find but may make the pattern less pleasing to read.
- Too much structure can impinge on the literary quality of a pattern.

- **Solution:**
  - Regardless of the style chosen for the pattern description, ensure that the forces are highly visible. This can be done by defining a meaningful "name" for each force and visually setting if off from text by making them minor headings, or by highlighting them using fonts, underlining, or other typographic techniques.

B.4 Pattern: Single-Pass Readable

- **Problem:**
  - How do you help the reader understand your pattern in the least amount of time, in order to facilitate this search?
- **Context:**
  - You are writing a pattern with Mandatory Elements Present.
- **Forces:**
  - People sometimes only have a limited time to read a pattern.
  - People get frustrated and give up when the effort is too high.
  - A pattern that must be read several times before being understood is more likely to be misunderstood.
  - A simple message is more likely to be understood correctly.

121

Slide 121 of 126

B.4 Pattern: Single-Pass Readable Con'd

Solution:

- Single-Pass Readable is easier said than done, and probably merits a pattern language on its own. However, here are a variety of techniques which can be helpful to achieve single-pass readability:
- Use Evocative Pattern Names or Pattern Thumbnails in cases where some understanding of a forward referenced pattern is necessary for the reader to keep reading.
- Help the reader locate key information by using Findable Sections and Visible Forces to highlight the tradeoffs involved.
- Use Skippable Sections (such as Code Samples as Bonus) to highlight information which can be skipped on first reading.
- In a pattern language, provide a clear, concise summary of the structure of the pattern language, then remind the readers where they are within the structure as they go along, using Headings Convey Structure.
- If you need to introduce and/or define a number of concepts in the introductory sections of your pattern, try to pare down your list, and write your pattern using this reduced list.

122

Agent Patterns and Pattern Languages

- Differentiate between object, components and agents in your structures diagrams (colors, stereotypes, special symbols)
- Forces should push you to agent solution, or;
- Context should involve agents
- Don't use Gof format
- For conferences, write patterns and position your pattern in a language
- For journals, write pattern languages
- Isolate all general forces relating to agents

123

Slide 123 of 126

Concluding Remarks

- Patterns have a long history
- Used successfully by the OO community
- Effective way of transferring technology and knowledge
- Contexts, Problems, Forces and Solutions are a good way to promote the use of agent technology
- Our goal is to get patterns into agent conferences
- Starting an Agent Pattern web site (send email to deugo@scs.carleton.ca)
- Get Started!

124

References

- C. Alexander. The Timeless Way of Building. New York: Oxford University Press, 1977.
- J.O. Coplien. Software Patterns. SIGS Management Briefings Series, SIGS Books & Multimedia, 1996.
- E. Gamma, R. Helm, R. Johnson, J. Vlissides. Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley, Reading, MA, 1995.
- Doug Lea, "Christopher Alexander: An Introduction to Object Oriented Designers," http://gee.cs.oswego.edu/dl/ca/ca/ca.html.
- Doug Lea, "Pattern Checklist", http://hillside.net/patterns/Writing/Check.html
- Gerard Meszaros, Jim Doble, "A Pattern Language for Pattern Writing," http://hillside.net/patterns/Writing/pattern_index.html
- John Vlissides, "Seven Habits of Successful Pattern Writers," C++ Report, November/ December 1996.

Useful Links

- WWW Patterns Home Page (http://hillside.net/patterns)
- PLoP 99 (http://st-www.cs.uiuc.edu/~plop/plop99)
- PLoP 98 (http://st-www.cs.uiuc.edu/~plop/plop98)
- PLoP 97 (http://jerry.cs.uiuc.edu/~plop/plop97/Workshops.html)