Architecture-Based Design of Multi-Agent Systems

Danny Weyns

Architecture-Based Design of Multi-Agent Systems

Foreword by Len Bass

Springer

Danny Weyns
Katholieke Universiteit Leuven
Dept. Computer Science
Celesijnenlaan 200a
3001 Leuven
Belgium
danny.weyns@cs.kuleuven.be

ISBN 978-3-642-01063-7 e-ISBN 978-3-642-01064-4
DOI 10.1007/978-3-642-01064-4
Springer Heidelberg Dordrecht London New York

Library of Congress Control Number: 2010920463

ACM Computing Classification (1998): I.2.11, D.2.11, C.3, J.7

Cover design: KuenkelLopka GmbH, Heidelberg

Printed on acid-free paper

Springer is part of Springer Science+Business Media (www.springer.com)

To Tessa, Eva, and Marina

Foreword

One of the most important things an architect can do is reflection. That is, examine systems, organizations, people and ask "What alternatives were considered and why was that particular decision made?" Thinking about the response gives an architect insight into the motivations and decision processes that others have used and this, in turn, should help the architect make better decisions in the future. A pre-requisite for doing this type of reflection is that the decisions and alternatives are made explicit. One venue that gives an architect an opportunity to do this type of reflection is during an architectural evaluation. Another venue is from a book such as this. This book lays out the design process used in building a collection of multi-agent systems.

In addition to providing a case study of a design process and the rationale for the design decisions, the topic of the book also is of great interest. Systems of the future will increasingly have the characteristics of the autonomous systems described here: they are simultaneously becoming more interconnected and more autonomous. Think of your smart phone that is mostly connected but can operate many functions while it is disconnected. These systems of the future will also increasingly operate without central control. Again, the telephone system and how cellular communication is managed provides a good example of this phenomenon.

Problems of connectivity raise issues of a node discovering that it is disconnected, other nodes discovering that a particular node is disconnected, how the node operates while it is disconnected, and reconnecting the node. The case study provides solutions to this problem in the context of autonomous vehicles within a limited geographic area. The essence of the solution provided—define the concept of a neighborhood for a node and treat neighboring nodes in a different fashion from other nodes—seems like it is more general than the particular application in the case study but that is still to be determined.

Communication and protocols seem to be basic to providing solutions to relatively autonomous nodes, and the structure of the middleware for such systems is of interest independently from the particular application area. The middleware needs to provide not only communication structure and neighborhood definition but also security and authentication services. The case study describes a middleware structure, and a portion of the reflection process of the architect is to ask not only about the rationale for the specific decisions made but also how well these decisions will generalize to other situations that the architect can envision. Designing

systems is one aspect of the work of an architect but as stated in the introduction "Developing multi-agent systems software is 95% software engineering and 5% multi-agent systems theory." All of the portions of the software engineering life cycle must operate efficiently in order for systems to be effectively constructed. This means that the important requirements must be identified, a design generated, the design documented and evaluated, and the system constructed from the design. Each of these topics is treated in the book.

- The important requirements are typically the quality attribute requirements. Eliciting these requirements requires a different mindset from the normal requirements process whether through a formal process, through user stories, or through some other technique. Quality attribute requirements tend to be the requirements that are taken for granted by the user until they have not been met.
- Documentation is important for helping the designer think through difficult design issues and for communicating the design to others. From the perspective of a reflecting architect, the documentation provided in this book provides some understanding of the division of functionality and design rationale.
- Evaluation of a design is an important step for verifying nothing has been missed by the architect. An evaluation is an application of the multiple eyes principle—get an outside, knowledgeable perspective to look at the design. As was pointed out, this process takes time. Partially this is because it takes time to educate the outside eyes and partially because evaluation requirements look at the design with a variety of different concerns. In the ATAM process, these concerns are expressed as scenarios but, in general, looking at a complicated design in sufficient detail to determine potential problems will take time. This time could be done as one activity when the outside eyes have to travel, as in the ATAM, or the time could be spread over a collection of shorter activities when the outside eyes are generally available.

In summary, this book is interesting both for its expressed topic—the design of multi-agent systems—and as a case study where a reader can read, and reflect on, the rationale for the approach taken in building such a system.

Pittsburgh, Pennsylvania, USA Len Bass
October 23, 2009

Acknowledgements

This book is based on 8 years of applied research conducted by a team of researchers at DistriNet Labs in collaboration with multiple industrial partners. First and foremost, I must acknowledge the contributions of Tom Holvoet and Kurt Schelfthout. Their insights and stimulating discussions have significantly contributed to the development of architecture-based design of multi-agent systems. Kurt invented the ObjectPlaces middleware described in Chapter 5. The following people must also be acknowledged for their invaluable contributions to the approach: Alexander Hellboogh, Nelis Boucké, and Elke Steegmans. The advanced coordination mechanisms described in Chapter 6 are the result of an enjoyable collaborative effort with Nelis over multiple years. I thank my former colleagues Koen Mertens and Tom De Wolf for their contributions. I also thank the Master students that have supported the development of various aspects of the approach described in this book, in particular Robin Custers, Olivier Glorieux, Bart Demarsin, Wannes Schols, Els Helsen, and Koen Deschacht. I am grateful to Jan Wielemans, Tom Lefever, Walter De Feyter, Rudy Vanhoutte, Wim Van Betsbrugge, Jan Peirsman, Raf Sempels, and Jan Vercammen for the collaboration in the EMC2 project. I wish to acknowledge the international colleagues I worked with during the past years for the stimulating collaborations. I am particularly grateful to Van Parunak and Fabien Michel for the enjoyable joint efforts. I would like to thank the researchers and architects of the Software Engineering Institute for the inspiring discussions on the design and evaluation of decentralized architectures. Several anonymous reviewers commented on earlier versions of this manuscript and suggested many improvements. Ralf Gerstner provided useful advice toward getting this book published. Thank you.

Danny Weyns

Contents

1 Introduction 1
1.1 Software Architecture and Middleware 1
1.1.1 Software Architecture 2
1.1.2 Middleware 3
1.2 Agent-Oriented Methodologies 4
1.3 Case Study 5
1.4 Overview of the Book 6

2 Overview of Architecture-Based Design of Multi-Agent Systems 9
2.1 General Overview of the Approach 9
2.1.1 Architectural Design in the Development Life Cycle 9
2.1.2 Steps of Architecture-Based Design of Multi-Agent Systems . 11
2.2 Functional and Quality Attribute Requirements 12
2.3 Architectural Design 14
2.3.1 Architectural Patterns 14
2.3.2 ADD Process 16
2.4 Middleware Support for Multi-Agent Systems 17
2.5 Documenting Software Architecture 17
2.5.1 Architectural Views 18
2.5.2 Architectural Description Languages 19
2.6 Evaluating Software Architecture 20
2.7 From Software Architecture to Downstream Design and Implementation 23
2.8 Summary 24

3 Capturing Expertise in Multi-Agent System Engineering with Architectural Patterns 27
3.1 Situated Multi-Agent Systems 28
3.1.1 Single-Agent Systems 28
3.1.2 Multi-Agent Systems 30
3.2 Target Domain of the Pattern Language for Situated Multi-Agent Systems 32

3.3 Overview of the Pattern Language 33
3.4 Pattern Template 34
3.5 Virtual Environment 35
3.5.1 Primary Presentation 35
3.5.2 Architectural Elements 35
3.5.3 Interface Descriptions 37
3.5.4 Design Rationale 38
3.6 Situated Agent 39
3.6.1 Primary Presentation 39
3.6.2 Architectural Elements 39
3.6.3 Interface Descriptions 41
3.6.4 Design Rationale 41
3.7 Selective Perception 43
3.7.1 Primary Presentation 43
3.7.2 Architectural Elements 43
3.7.3 Interface Descriptions 44
3.7.4 Design Rationale 44
3.8 Roles and Situated Commitments 45
3.8.1 Primary Presentation 45
3.8.2 Architectural Elements 45
3.8.3 Design Rationale 47
3.8.4 Free-Flow Trees Extended with Roles and Situated Commitments 47
3.9 Protocol-Based Communication 50
3.9.1 Primary Presentation 50
3.9.2 Architectural Elements 50
3.9.3 Interface Descriptions 52
3.9.4 Design Rationale 52
3.10 Summary 53

4 Architectural Design of Multi-Agent Systems 55
4.1 Designing and Documenting Multi-Agent System Architectures 55
4.1.1 Designing and Documenting Architecture in the Development Life Cycle 56
4.1.2 Inputs and Outputs of ADD 57
4.1.3 Overview of the ADD Activities 57
4.2 Case Study 58
4.2.1 The Domain of Automated Transportation Systems 58
4.2.2 Business Case 60
4.2.3 System Requirements 61
4.3 General Overview of the Design 63
4.3.1 Challenges at the Outset 64
4.3.2 The System and Its Environment 65
4.3.3 Design Process 67

4.3.4 Design Rationale . . . 68
4.3.5 High-Level Design . . . 69
4.4 Architecture Documentation . . . 75
4.4.1 Introduction to the Architecture Documentation . . . 75
4.4.2 Deployment View . . . 76
4.4.3 Module Uses View . . . 79
4.4.4 Collaborating Components View . . . 83
4.5 Summary . . . 92

5 Middleware for Distributed Multi-Agent Systems . . . 93
5.1 Middleware Support for Distributed, Decentralized Coordination . . . 93
5.1.1 Middleware in Distributed Software Systems . . . 94
5.1.2 Middleware in Multi-Agent Systems . . . 95
5.2 Case Study . . . 96
5.2.1 Scope of the Middleware and Requirements . . . 96
5.2.2 Objectplaces . . . 97
5.2.3 Views . . . 99
5.2.4 Coordination Roles . . . 103
5.3 Middleware Architecture . . . 106
5.3.1 High-Level Module Decomposition . . . 106
5.3.2 Group Formation . . . 109
5.3.3 View Management . . . 111
5.3.4 Role Activation . . . 113
5.4 Collision Avoidance in the AGV Transportation System . . . 114
5.4.1 Collision Avoidance . . . 114
5.4.2 Collision Avoidance Protocol . . . 115
5.4.3 Software Architecture: Communicating Processes for Collision Avoidance . . . 119
5.5 Summary . . . 122

6 Task Assignment . . . 123
6.1 Schedule-Based Task Assignment . . . 124
6.2 FiTA: Field-Based Task Assignment . . . 124
6.2.1 Coordination Fields . . . 125
6.2.2 Adaptive Task Assignment . . . 127
6.2.3 Software Architecture . . . 127
6.2.4 Dealing with Local Minima . . . 130
6.3 DynCNET Protocol . . . 131
6.3.1 Adaptive Task Assignment . . . 132
6.3.2 Monitoring the Area of Interest . . . 135
6.3.3 Convergence . . . 137
6.3.4 Synchronization Issues . . . 137
6.4 Evaluation . . . 137
6.4.1 Test Setting . . . 137

6.4.2 Test Results . . . 139
6.4.3 Tradeoff Analysis . . . 144
6.5 Summary . . . 147

7 Evaluation of Multi-Agent System Architectures . . . 149
7.1 Evaluating Multi-Agent System Architectures with ATAM . . . 149
7.1.1 Architecture Evaluation in the Development Life Cycle . . . 150
7.1.2 Objectives of a Multi-Agent System Architecture Evaluation . 151
7.1.3 Overview of the ATAM Activities . . . 151
7.2 Case Study . . . 152
7.2.1 AGV Transportation System for a Tea Processing Warehouse 153
7.2.2 Evaluation Process . . . 153
7.2.3 Quality Attribute Workshop . . . 155
7.2.4 Analysis of Architectural Approaches . . . 156
7.3 Reflection on ATAM for Evaluating a Multi-Agent System Architecture . . . 161
7.4 ATAM Follow-Up and Demonstrator . . . 163
7.5 Summary . . . 163

8 Related Approaches . . . 165
8.1 Architectural Approaches and Multi-Agent Systems . . . 165
8.1.1 Architectural Styles . . . 165
8.1.2 Reference Models and Architectures for Multi-Agent Systems 168
8.2 Middleware for Mobile Systems . . . 172
8.2.1 Work Related to Views . . . 172
8.2.2 Work Related to Coordination Roles . . . 174
8.3 Scheduling and Routing of AGV Transportation Systems . . . 177
8.3.1 AI and Robotics Approaches . . . 177
8.3.2 Multi-Agent System Approaches . . . 178

9 Conclusions . . . 181
9.1 Reflection on Architecture-Based Design of Multi-Agent Systems . . . 181
9.1.1 It Works! . . . 181
9.1.2 Reflection on the Project with Egemin . . . 183
9.2 Lessons Learned and Challenges . . . 185
9.2.1 Dealing with Quality Attributes . . . 185
9.2.2 Designing a Multi-Agent System Architecture . . . 185
9.2.3 Integrating a Multi-Agent System with Its Software Environment . . . 186
9.2.4 Impact of Adopting a Multi-Agent System . . . 187

A π-ADL Specification of the Architectural Patterns . . . 189
A.1 Language Constructs . . . 189
A.2 Virtual Environment Pattern . . . 190
A.3 Situated Agent Pattern . . . 194

B Synchronization in the DynCNET Protocol 199
B.1 Synchronization of Abort and Bound Messages 199
B.2 Synchronization of Scope Dynamics 201

C Collision Avoidance Protocol 203
C.1 Overview 203
C.2 Invariant 204
C.3 Maintaining the Invariant 205

Glossary 209

References 213

Index 223

Acronyms

ADD	Attribute-Driven Design
ADL	Architecture Description Language
AGV	Automatic Guided Vehicle
APL	Agent Programming Language
ATAM®	Architecture Tradeoff Analysis Method®
AUML	Agent Unified Modeling Language
BDI	Belief, Desire, Intention
C&C	Component and Connector
cfp	call for proposals
CNET	Contract NET
CORBA	Common Object Request Broker Architecture
CPU	Central Processing Unit
DynCNET	Dynamic Contract NET
E'nsor®	Egemin navigation system on robot
E'pia®	Egemin platform for integrated automation
ERP	Enterprise Resource Planning
FiTA	Field-based Transport Assignment
IEC	International Electrotechnical Commission
IEEE	Institute of Electrical and Electronics Engineers
IP	Internet Protocol
ISO	International Standard Organization
JEE	Java Enterprise Edition
LAN	Local Area Network
LIME	Linda In a Mobile Environment
Mbps	Megabits per second
QAW	Quality Attribute Workshop
RMI	Remote Method Invocation
SOAP	Simple Object Access Protocol
TB	Transport Base
UML	Unified Modeling Language
XML	Extensible Markup Language