

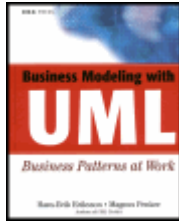
OMG PRESS

Business Modeling with UML

Business Patterns at Work

Hans-Erik Eriksson • Magnus Penker

Authors of UML Toolkit



Business Modeling with UML: Business Patterns at Work

by Hans-Erik Eriksson and Magnus Penker

ISBN: 0471295515

[John Wiley & Sons](#) © 2000 (459 pages)

An introduction to the Unified Model Language, and lessons and examples of practical business applications for software developers.

Table of Contents

[Business Modeling with UML: Business Patterns at Work](#)

[Introduction](#)

[Chapter 1](#) - Business Modeling

[Chapter 2](#) - UML Primer

[Chapter 3](#) - Modeling the Business Architecture

[Chapter 4](#) - Business Views

[Chapter 5](#) - Business Rules

[Chapter 6](#) - Business Patterns

[Chapter 7](#) - Resource and Rule Patterns

[Chapter 8](#) - Goal Patterns

[Chapter 9](#) - Process Patterns

[Chapter 10](#) - From Business Architecture to Software Architecture

[Chapter 11](#) - A Business Model Example

[Appendix A](#) - Eriksson-Penker Business Extensions

[Appendix B](#) - Business Patterns Summary

[Glossary](#)

[References](#)

[Index](#)

[List of Figures](#)

[List of Tables](#)

Business Modeling with UML: Business Patterns at Work

Hans-Erik Eriksson

Magnus Penker

Publisher: Robert Ipsen

Editor: Theresa Hudson

Associate Developmental Editor: Kathryn A. Malm

Managing Editor: Micheline Frederick

Text Design & Composition: Thomark Design

Designations used by companies to distinguish their products are often claimed as trademarks. In all instances where John Wiley & Sons, Inc., is aware of a claim, the product names appear in initial capital or ALL CAPITAL LETTERS. Readers, however, should contact the appropriate companies for more complete information regarding trademarks and registration.

Copyright © 2000 by Hans-Erik Eriksson and Magnus Penker.

All rights reserved.

Published by John Wiley & Sons, Inc.

Published simultaneously in Canada.

No part of this publication may be reproduced, stored in a retrieval system or transmitted in any form or by any means, electronic, mechanical, photocopying, recording, scanning or otherwise, except as permitted under Sections 107 or 108 of the 1976 United States Copyright Act, without either the prior written permission of the Publisher, or authorization through payment of the appropriate per-copy fee to the Copyright Clearance Center, 222 Rosewood Drive, Danvers, MA 01923, (978) 750-8400, fax (978) 750-4744. Requests to the Publisher for permission should be addressed to the Permissions Department, John Wiley & Sons, Inc., 605 Third Avenue, New York, NY 10158-0012, (212) 850-6011, fax (212) 850-6008, E-Mail: PERMREQ@WILEY.COM.

This publication is designed to provide accurate and authoritative information in regard to the subject matter covered. It is sold with the understanding that the publisher is not engaged in professional services. If professional advice or other expert assistance is required, the services of a competent professional person should be sought.

Library of Congress cataloging-in-Publication Data:

Eriksson, Hans-Erik, 1961-

Business modeling with UML: business patterns at work / Hans-Erik Eriksson, Magnus Penker.

p. cm

Includes bibliographical references and index.

ISBN 0-471-2955-5 (cloth : alk. paper)

1. Application software--Development. 2. UML (Computer science). 3. Business--Data processing. I. Penker, Magnus, 1968- II. Title.

QA76.76.A65 E794 2000

650'.0285'5117--dc21 99-058911

Printed in the United States of America.

10 9 8 7 6 5 4 3 2

Advance Praise for Business Modeling with UML

"Attempts to model businesses have been around for a long time. Unfortunately, the confusion surrounding object methodologies and notations that existed before 1997 slowed the progress in this field while arguments ensued over notation and structure of business representation. With the worldwide acceptance of OMG's UML standard, business modellers can now focus on the business mechanics and semantics of interactions rather than the graphical form. Eriksson and Penker leverage this newfound focus with an excellent hands-on book for practitioners eager to document the internal structure and everyday workings of business processes. This clear and practical book belongs on the shelf of everyone dedicated to mapping, maintaining, and streamlining business processes."

Richard Mark Soley

Chairman and CEO, OMG

"Eriksson and Penker have not just written another patterns book; this is a significant contribution to the key field of business-IT alignment. While capturing profound academic insights, what makes the book so refreshing from a practitioner's viewpoint is the richness of accessible, down-to-earth examples and its pragmatic, unpretentious style."

Paul Allen

Principal of CBD Strategies and Architectures, Sterling Software

"UML may have been designed by and for software engineers, but Eriksson and Penker have defined a practical extension to UML for describing business processes. They put this extended UML immediately to use with a gallery of common business patterns that should jump start any BPR effort."

Philippe Kruchten

Director of Process Development, Rational Software

"This book is a marriage between proven business modeling concepts and the techniques of the UML. It provides real-world strategies for developing large-scale,

mission-critical business systems in a manner accessible to both software and business professionals.”

Scott W. Ambler

Author of *Process Patterns*

About the Authors

Hans-Erik Eriksson is a consultant, specializing in object-oriented technology. He has been doing system development for more than 15 years. With a background in program and system design, in the last years, Mr. Eriksson has focused on software architectures and the integration of business processes with technology.

Mr. Eriksson has authored numerous articles and five books on object technology, among them the best-selling *UML Toolkit* (with Magnus Penker), which was one of the first UML books available. He has worked as a trainer in object-oriented technologies for more than 10 years and has taught over 100 courses within this field.

Mr. Eriksson is the Chairman and Chief Technical Officer of Open Training, a company that specializes in advanced online learning solutions and the integration of e-Business and e-Training.

He can be contacted at hanserik.eriksson@opentraining.com.

Magnus Penker has more than 10 years of experience in the field of object-orientation and business process engineering. His previous positions include project manager, senior management consultant, and VP of training (at the European e-Business company Adera+). Mr. Penker has published several books, including the best-selling *UML Toolkit* (with Hans-Erik Eriksson). He is a frequent speaker at conferences and also the author of a number of articles and papers on business and software engineering.

Mr. Penker is the Chief Executive Officer of Open Training, a company specializing in advanced online learning solutions and the integration of e-Business and e-Training.

He can be contacted at magnus.penker@opentraining.com.

OMG Press Advisory Board

Karen D. Boucher
Executive Vice President
The Standish Group

Carol C. Burt
President and Chief Executive Officer
2AB, Inc.

Ian Foster
Business Director
PeerLogic, Inc.

Michael N. Gurevich
Chief Technology Officer and Vice President of Development
Concorde Solutions, Inc.

V. “Juggy” Jagannathan, Ph.D.
Senior Vice President of Research and Development and Chief Technology Officer
CareFlow|Net, Inc.

Cris Kobryn
Chief Architect
EDS

Nilo Mitra, Ph.D.
Principal System Engineer
Ericsson

Richard Mark Soley, Ph.D.
Chairman and Chief Executive Officer
Object Management Group, Inc.

Sheldon C. Sutton
Principal Information Systems Engineer
The MITRE Corporation

Andreas Vogel, Ph.D.
Chief Scientist
Inprise Corporation

OMG Press Books in Print

(For complete information about current and upcoming titles, go to:

www.wiley.com/compbooks/omg/)

[*Building Business Objects*](#) by Peter Eeles and Oliver Sims, ISBN: 0471-191760.

Business Component Factory: A Comprehensive Overview of Component-Based Development for the Enterprise by Peter Herzum and Oliver Sims, ISBN: 0471-327603.

[*Business Modeling with UML: Business Patterns at Work*](#) by Hans-Erik Eriksson and Magnus Penker, ISBN: 0471-295515.

CORBA 3 Fundamentals and Programming, 2nd Edition by Jon Siegel, ISBN: 0471-295183.

CORBA Design Patterns by Thomas J. Mowbray and Raphael C. Malveau, ISBN: 0471-158828.

Developing C++ Applications with UML by Michael Sandberg, ISBN: 0471-38304X.

Enterprise Application Integration with CORBA: Component and Web-Based Solutions by Ron Zahavi, ISBN: 0471-327204.

The Essential CORBA: Systems Integration Using Distributed Objects by Thomas J. Mowbray and Ron Zahavi, ISBN: 0471-106119.

Instant CORBA by Robert Orfali, Dan Harkey and Jeri Edwards, ISBN: 0471-183334.

[*Integrating CORBA and COM Applications*](#) by Michael Rosen and David Curtis, ISBN: 0471-198277.

Java Programming with CORBA, 2nd Edition by Andreas Vogel and Keith Duddy, ISBN: 0471-247650.

Mastering XML: Java Programming with the XML Toolkit, XML and UML by Stephen Brodsky and Tim Grose, ISBN: 0471-384291.

The Object Technology Casebook: Lessons from Award-Winning Business Applications by Paul Harmon and William Morrissey, ISBN: 0471-147176.

The Object Technology Revolution by Michael Guttman and Jason Matthews, ISBN: 0471-606790.

Programming with Enterprise JavaBeans, JTS and OTS: Building Distributed Transactions with Java and C++ by Andreas Vogel and Madhavan Rangarao, ISBN: 0471-319724.

Programming with Java IDL by Geoffrey Lewis, Steven Barber, and Ellen Siegel, ISBN: 0471-247979.

UML Toolkit by Hans-Erik Eriksson and Magnus Penker, ISBN: 0471-191612.

About the OMG

The Object Management Group (OMG) was chartered to create and foster a component-based software marketplace through the standardization and promotion of object-oriented software. To achieve this goal, the OMG specifies open standards for every aspect of distributed object computing from analysis and design, through infrastructure, to application objects and components.

The well-established CORBA (Common Object Request Broker Architecture) standardizes a platform- and programming-language-independent distributed object computing environment. It is based on OMG/ISO Interface Definition Language (OMG IDL) and the Internet Inter-ORB Protocol (IIOP). Now recognized as a mature technology, CORBA is represented on the marketplace by well over 70 ORBs (Object Request Brokers) plus hundreds of other products. Although most of these ORBs are tuned for general use, others are specialized for real-time or embedded applications, or built into transaction processing systems where they provide scalability, high throughput, and reliability. Of the thousands of live, mission-critical CORBA applications in use today around the world, over 300 are documented on the OMG's success-story Web pages at www.corba.org.

CORBA 3, the OMG's latest release, adds a Component Model, quality-of-service control, a messaging invocation model, and tightened integration with the Internet, Enterprise JavaBeans, and the Java programming language. Widely anticipated by the industry, CORBA 3 keeps this established architecture in the forefront of distributed computing, as will a new OMG specification integrating CORBA with XML. Well-known for its ability to integrate legacy systems into your network, along with the wide variety of heterogeneous hardware and software on the market today, CORBA enters the new millennium prepared to integrate the technologies on the horizon.

Augmenting this core infrastructure are the CORBA services which standardize naming and directory services, event handling, transaction processing, security, and other functions. Building on this firm foundation, OMG Domain Facilities standardize common objects throughout the supply and service chains in industries such as Telecommunications, Healthcare, Manufacturing, Transportation, Finance/Insurance, Electronic Commerce, Life Science, and Utilities.

The OMG standards extend beyond programming. OMG Specifications for analysis and design include the Unified Modeling Language (UML), the repository standard Meta-Object Facility (MOF), and XML-based Metadata Interchange (XMI). The UML is a result of fusing the concepts of the world's most prominent methodologists. Adopted as an OMG specification in 1997, it represents a collection of best engineering practices that have proven successful in the modeling of large and complex systems and is a well-defined, widely-accepted response to these business needs. The MOF is OMG's standard for metamodeling and metadata repositories. Fully integrated with UML, it uses the UML notation to describe repository metamodels. Extending this work, the XMI standard enables the exchange of objects defined using UML and the MOF. XMI can generate XML Data Type Definitions for any service specification that includes a normative, MOF-based metamodel.

In summary, the OMG provides the computing industry with an open, vendor-neutral, proven process for establishing and promoting standards. OMG makes all of its specifications available without charge from its Web site, www.omg.org. With over a decade of standard-making and consensus-building experience, OMG now counts about 800 companies as members. Delegates from these companies convene at week-long meetings held five times each year at varying sites around the world, to advance OMG technologies. The OMG welcomes guests to their meetings; for an invitation, send your email request to info@omg.org.

Membership in the OMG is open to end users, government organizations, academia, and technology vendors. For more information on the OMG, contact OMG headquarters by phone at +1-508-820 4300, by fax at +1-508-820 4303, by email at info@omg.org, or on the Web at www.omg.org.

Acknowledgments

We would like to acknowledge the work of the people who helped us to write this book, and whose efforts have been vital to its completion.

Some very insightful and important comments were made by our team of reviewers, most notably:

- Philippe Kruchten at Rational Software
- Paul Allen at Sterling Software
- Jos Warmer at IBM

Several reviewers were anonymous to us and therefore can't be named, but a sincere thanks goes to them for their constructive and helpful comments.

A tremendous job was done from the Wiley team to get this book ready: Kathryn Malm, Terri Hudson, Gerrie Cho, Micheline Frederick, and others. Producing this kind of book is always an endeavour, and when the authors and reviewers are spread across the world it becomes quite an achievement.

Special thanks goes to the consultants at Astrakan, who have vast experience in business modeling and have provided many of the theories and ideas presented in this book. Many have been individually credited in the pattern chapters, but a thanks also goes to the group as a whole.

Any remaining mistakes or errors in the text are the responsibility of the authors. Readers who would like to contact us to pose a question or to discuss the contents of the book may feel free to do so at the addresses listed below.

The book's Web page is located at www.opentraining.com/bm. Readers are invited to visit for further information, updates, and links to topics related to the book.

Stockholm, December 1999

Hans-Erik Eriksson

hanserik.eriksson@opentraining.com

Magnus Penker

magnus.penker@opentraining.com

Introduction

Since its standardization by the OMG (Object Management Group) in November 1997, the Unified Modeling Language (UML) has had a tremendous impact on how software systems are developed. The role of modeling in specifying and documenting complex software systems is being accepted, and an industrial approach to software engineering is on its way to becoming reality. With the acceptance of UML, a new generation of tools and processes that use UML have also emerged, and important concepts and techniques such as the role of architecture, requirements engineering, and tool integration also have been emphasized.

However, a common problem is that software systems do not properly support the businesses of which they are an integrated part. There are several reasons for this: a correct requirement specification is not available, the software team does not have a proper understanding of the business, or the business changes so frequently that the software can't keep up. With the emergence of e-commerce, the solutions of tomorrow need to be a combination of technology and business—to provide real excellence you need an understanding of both. Business modeling helps you understand by modeling the actual business and its goals, processes (activities), resources (such as people, machines, and material) and rules.

To identify the proper requirements on the software systems is not the only reason to do business modeling. Business modeling creates an abstraction of a complex business and establishes a common understanding that can be communicated to the business's stakeholders (e.g., owners, management, employees, and customers). A better understanding of how the business functions facilitates improvements to the business, and helps to identify new business opportunities (i.e., business improvement or innovation).

Although UML in its first years has been used mainly for modeling software systems, it is also a very suitable language for business modeling. It has the ability to describe both the structural aspects of a business (such as the organization, goal hierarchies, or the structures of the resources), the behavioral aspects of a business (such as the processes), and the business rules that affect both structure and behavior. Many developers are already familiar with UML since they have used it to model software. Using the same modeling language for both business and software modeling ensures that the documentation is consistent and facilitates communication between business modelers and software modelers. In addition, a large amount of tools become available for use in business modeling when you use UML.

This book shows how to use UML for business modeling, and how to use the business models to identify the correct requirements for the software that supports the business. We use standard extension mechanisms, such as stereotypes, to define new model elements suited for business modeling. This book also presents guidelines for how to produce a business model and what it should contain. It introduces you to how to define business rules using the Object Constraint Language (OCL).

To help you get started with business modeling using UML, 26 business patterns are provided. These patterns are working, reusable, and illustrative examples of how different aspects of a business can be modeled. In addition, you'll learn how the information and knowledge in a business model can be used to identify the proper requirements on software systems that support the business, as well as how the information can be reused in the software models.

Finally, you'll find an example business model. The model applies the concepts, steps, and patterns described through out the book to an example mail order firm that has to migrate into the new world of e-business and network economy. This example is based on our experience modeling these types of projects. The company is fictitious, but the business structure is based on existing businesses.

What the Book Is and Is Not

This book provides valuable information that will help you to effectively use UML to model your business. You'll learn:

- A combination of techniques that melts knowledge and experience in the business modeling field together with object-oriented modeling.
- An approach that shows how to use the well-established UML language for business modeling. UML has so far been used mostly for modeling software systems.
- A set of business extensions, the Eriksson-Penker Business Extensions to UML, that extend UML with adapted model elements for business modeling. The extensions are defined using the standard extension mechanisms in UML.
- New techniques, such as the Assembly-Line diagram, that integrate the business processes in a business model with the use-cases to define functional requirements on a software system.
- Common business modeling experience and knowledge packaged in the form of reusable patterns (Resource and Rule Patterns, Goal Patterns, and Process Patterns).
- An approach to designing support systems that are in harmony with the business they are supposed to support.
- Many powerful examples of business models and illustrations of non-trivial features of the UML language, such as powertypes and stereotypes.
- A practical and pragmatic approach to doing business modeling with UML, an approach that can be used together with other techniques or methods.

This book is **not**:

- A new "method" for either business modeling or software modeling. Most of the ideas presented are acknowledged techniques for modeling, but have not been used with UML before.
- A book describing the UML language in detail. There are other books for this, such as our previous book *UML Toolkit* [Eriksson 98]^[1]

- A programming book. The models in this book do not model programs and should not be translated into code. The models, however, can be the basis for other UML models that model the software in an information system.
- A book about object-oriented technology. Knowledge of the basic concepts of object-oriented technology are required for reading this book.
- A substitute for doing bad analysis. A very important factor for succeeding with business modeling is staffing the project with knowledgeable, experienced, and dedicated people. There is no “silver bullet” in business modeling either.
- A book showing statistical proof of the success rate of these techniques. The techniques presented are based on well-established theories and have been used in practice for many years (though not necessarily using UML as the modeling language). There is a vast world of literature related to business modeling that provides further “proof” of the success of these techniques.

^[1][Eriksson 98] Eriksson, Hans-Erik and Magnus Penker. *UML Toolkit*. John Wiley & Sons, Inc., 1996.

Who Should Read This Book

This book is for you if:

- You want to understand the concept of business modeling for creating abstractions of your business that in turn can be used to communicate, improve, or innovate the business.
- You want insight into how UML can be used to describe the complexities of a business instead of just software systems.
- You are a software manager or developer who is looking for an answer to questions such as “But what should I do before I start producing the software system?” and “How do I know if I have identified the proper requirements (e.g., use cases)?”
- You are an analyst or a modeler who is looking for working and reusable patterns that demonstrate how the processes, resources, rules, and goals of a business can be modeled.
- You are an experienced business modeler who wants to understand how UML can be used in a business context, and how business modeling can be integrated with software development.
- You are a CASE Tool vendor who would like to integrate business modeling features with software engineering features in your tool.

You should have a basic knowledge of object-oriented concepts and of UML.

How this Book Is Organized

[Chapter 1](#), “Business Modeling,” presents the concept of business modeling and the purposes for modeling a business. It also provides arguments for why UML is suitable for business modeling and what elements are required in UML to do business modeling. [Chapter 2](#), “UML Primer,” is an overview of the UML language. If you are proficient in UML, feel free to skip this chapter. There are, however, some important sections on activity diagrams, power types, and UML extensions that highlight some areas of UML that are relevant to business modeling and the rest of the book.

[Chapter 3](#), “Modeling the Business Architecture,” defines the major concepts used in business modeling: processes, goals, resources, and rules. It also introduces the Eriksson-Penker Business Extensions that are defined using the standard extension mechanisms in UML to facilitate business modeling.

[Chapter 4](#), “Business Views,” describes the different views of a business model. It defines the techniques and diagrams used to capture a specific aspect of a business, and provides examples to illustrate their use.

[Chapter 5](#), “Business Rules,” describes how to define business rules using the Object Constraint Language (OCL), part of standard UML. Examples show you how to use different categories of rules in all of the views in order to regulate how to run and structure the business. We also introduce Fuzzy Business Rules, a technique for defining rules without ordinary binary logic.

[Chapter 6](#), “Business Patterns,” introduces the 26 business patterns, common solutions to business modeling problems, that are covered in chapters 7 through 9. The chapter defines a pattern, its structure and characteristics, and how patterns are represented in the UML language.

[Chapter 7](#), “Resource and Rule Patterns,” describe patterns that can be used to resolve typical problem situations that can arise when modeling the structures and relationships (including rules) between resources.

[Chapter 8](#), “Goal Patterns,” describes patterns for defining the goals of a business. Goals are what the business and their corresponding models strive for, and form the basis for designing the processes, finding the right resources, and tuning the business rules.

[Chapter 9](#), “Process Patterns,” defines high quality, well-proven, and easy-to-use patterns that are used to model business processes. The chapter defines the different categories of process patterns, and covers important areas such as layering, decomposition, interaction, process type and instance, and workflow.

[Chapter 10](#), “From a Business Architecture to a Software Architecture,” discusses how the business model is used to produce the business’s supporting information system. The business model identifies the requirements on the information systems (the use cases of the system), and can also be used to define the architecture of the information system. Using the same business model for several information systems also helps to create systems that are more easily integrated with the business they support.

[Chapter 11](#), “A Business Model Example,” demonstrates how the techniques and notation defined throughout the book can be used to model a practical example of an e-business. The business views, business extensions, business rules, and business patterns are demonstrated in this case study.

We’ve also included additional resources, a visual glossary of the Eriksson-Penker Business Extensions, a table summary of the Business Patterns, and a glossary. Also visit the Web site at www.opentraining.com/bm/ for further articles, information, and links on subjects related to the book, as well as our recommendations for tools that support business modeling. You can also contact us through our email addresses hanserik.eriksson@opentraining.com and magnus.penker@opentraining.com.

Chapter 1: Business Modeling

Overview

Running a business today is more competitive than ever. The globalization of world markets, brought about by technology in general and the Internet in particular requires businesspeople to acquire and adapt to new business logic. Anyone who doesn’t continuously strive to improve the operation, products, and services of their business will find it difficult to succeed in this challenging environment.

To keep up and remain competitive, companies and enterprises must assess the quality of their products and the efficiency of their services. In doing so, they must consider the world around them: their competitors, their subcontractors, their suppliers, the ever-changing laws and regulations, and, above all, their customers. They must also

objectively examine their products or services, asking such questions as: Is my internal operation working smoothly? Can I improve my product or service in any way? Is production running as efficiently as possible? Can I expand my product or service portfolios to reach new markets and customers?

Today's businesspeople must also evaluate their information systems: Do they effectively support their way of working? Do the systems adapt easily to change? Is information used as an important strategic resource in the business? Is the information adequate and correct? All businesses will benefit by gaining a deeper understanding of how their business interacts with its environment, which comes from honestly answering these questions.

In today's marketplaces, information systems no longer merely support businesses. Increasingly, they are an integral part of them. All businesses make some use of information technology, and it is important that their systems be designed to support them. The business is, after all, what defines the requirements of the information systems.

To answer these questions, it is essential to make a *model* of the business, a simplified view of a complex reality ([Figure 1.1](#)). It is a means of creating abstraction; it enables you to eliminate irrelevant details and focus on one or more important aspects at a time. Effective models also facilitate discussions among different stakeholders in the business, helping them to reach agreement on the key fundamentals and to work toward common goals. Finally, a business model can be the basis for other models, for different information systems that support the business, for example. Modeling is an accepted and established means of analyzing and designing software. To create appropriate software, the businesses in which the software systems operate must also be modeled, understood, and improved as required.

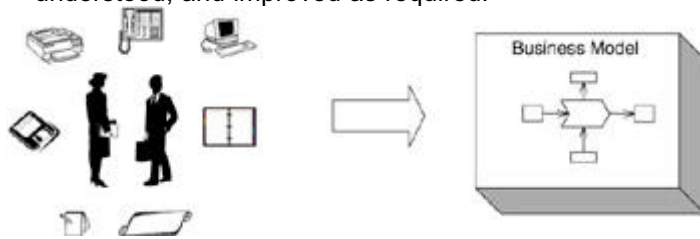


Figure 1.1: A business model is a simplified view of a business.

A *business model* is an abstraction of how a business functions. Its details differ according to the perspective of the person creating the model, each of whom will naturally have a slightly different viewpoint of the goals and visions of the business, including its efficiency and the various elements that are acting in concert within the business. This is normal, and the business model will not completely resolve these differences. What the business model will do is provide a simplified view of the business structure that will act as the basis for communication, improvements, or innovations, and define the information systems requirements that are necessary to support the business. It isn't necessary for a business model to capture an absolute picture of the business or to describe every business detail.

The business model is the focal point around which business is conducted or around which business operations are improved. The evolving models also help the developers structure and focus their thinking. Working with the models increases their understanding of the business and, hopefully, their awareness of new opportunities for improving business.

The word business in this context is used as a broad term. The businesses that can be modeled with the techniques presented in this book do not have to be profit making (e.g., a relief organization for the homeless or for war victims is also a business that needs to be organized as effectively as possible). Any type of ongoing operation that has or uses resources and has one or more goals can be referred to as a business. The *owner* of the business sets the goals and allocates resources to make the business run. The *business modeler* then creates the structure, designs the processes, and allocates the resources in order to achieve the goals. The *system developer* then adapts, designs, or develops appropriate information systems that support the running of the business.

This chapter explores the role of models in general, and how businesses can benefit from using them, and, more specifically, where the Unified Modeling Language (UML) fits in.

The Role of Models

Anyone who has played chess knows that you need a strategy or plan, and that even a bad plan is better than no plans at all. During the game, not everything happens as planned, but the plan nevertheless points in a direction, making the decision-making easier and faster. A more skilled player will change the plan, in part or in whole, during the game and think ahead about alternative moves he or she will take as a result of possible moves by the opponent.

The business model functions as the plan for conducting a business. It acts as the basis for decision-making and affects decisions about prioritizing goals, obtaining the right resources, or negotiating with subcontractors. It also serves as an up-to-date description of how the business is performed, and allows for changes and improvements in the process, such as cutting costs, improving quality, or shortening time-to-market. The model can anticipate and forecast changes that are necessary to maintain an edge on the competition. The model can't provide all the answers, but as for the chess player, it is a basic strategy or plan to follow. An advantage of modeling in a language such as UML is that it visually depicts functions and relationships that are usually difficult to visualize clearly.

Ideally, a business model would consist of a single diagram that included all the important aspects of a business. That is, however, never possible, since a business is so complex and has so many aspects that a single diagram can't capture all that information. Instead, a business model is composed of the following:

Views. A business model is illustrated with a number of different views, each of which captures information about one or more specific aspects of the business. A view is an abstraction from a specific viewpoint, omitting details that are irrelevant to that viewpoint. Multiple views are necessary to separate purposes and perspectives in a controlled way, without losing important information about the business.

Diagrams. Each view consists of a number of diagrams, each of which shows a specific part of the business structure or a specific business situation. Several diagrams are necessary to visualize a single view of the business model, since each type of diagram has a different purpose and expresses one important aspect or mechanism within the business model view. A diagram can show a structure (e.g., the organization of the business) or some dynamic collaboration (a number of objects and their interaction to demonstrate a specific process). The diagrams contain and express the objects, processes, rules, goals, and visions defined in the business situation.

Objects and Processes. Concepts are related in the diagrams through the use of different objects and processes. The objects are the "things" in the business; they may be physical, such as people, machines, products, and material, or more abstract, such as debts, instructions, and services. Objects can also represent other objects by containing information about other things in the business. Processes are the functions in the business that consume, refine, or use objects to affect or produce other objects.

A model is motivated by objectives. For example, the purpose of modeling a building might be to construct it or, later, to sell apartments in it. The purpose of modeling a business might be to understand or to improve its functionality. It is important to distinguish models used as an exploration tool from models used as a specification tool (modeling can be used for both). To build a supporting information system, such as a sales system, the surrounding business is modeled in order to understand it, and from that understanding the appropriate information system can be designed. Models facilitate understanding and communicating about systems, but only when the objective of the model is kept in mind. If the objective is to understand a business well enough to specify a supporting system, it is not necessary to model the entire business in detail; producing such a detailed model is simply too time-consuming and too expensive for that purpose.

However, if the goal of a serious business process innovation project is to redefine how the entire business is run and to find new and improved ways of conducting business, a more elaborate effort might be necessary. In any case, it is important not to overwork the models; if you keep the objective of the model in mind at all times, you will profit from business modeling.

UML

Since its introduction in November 1997, the Unified Modeling Language has quickly become the standard modeling language for software development. Many users of other methods (Booch, OMT, Fusion, etc.) have adopted UML; a number of books have been written about UML; and most modeling tools have implemented support for the language. All predictions for the future of UML indicate that it will become even more dominant and important, and that the tool support for producing and exchanging UML models will become more advanced.

The UML consists of nine different diagram types, and each diagram shows a specific static or dynamic aspect of a system. The basics of UML are easy to learn, and very powerful constructs, such as stereotypes or power types, are available for the more advanced modeler. [Chapter 2](#), “UML Primer,” includes an overview of UML and some of the advanced concepts used later in this book.

The UML standardizes notation for describing a process, but it doesn’t standardize a process for producing those descriptions (a well-defined order of activities, a set of artifacts produced, and ways to monitor or control the work). UML, in fact, can be used by many different developmental processes, more or less formally specified. This is not an oversight by the developers of UML; a development process for a project involving three persons is very different from a project involving a thousand people who work in different countries. The process is also affected by other factors, such as the type of system or the experience of the developers. Attempts at defining de facto standards for development processes, such as the Rational Unified Process^[1] or Select Perspective^[2], are often configurable, meaning that they can be adapted or modified to suit the project at hand. It is important to understand that UML doesn’t prescribe a specific way of how it should be used in a project; there is no *one* specific or correct way of using it.

Several development processes that use UML advocate that the system development should start with *use-case modeling* to define the functional requirements on the system. A *use case* describes a specific usage of the system by one or more *actors*. An actor is a role that a user or another system plays. The objective of use-case modeling is to identify and describe all the use cases that the actors require from the system. The use-case descriptions then are used to analyze and design a robust system architecture that realizes the use cases (this is what is referred to as “use-case driven” development).

But how do you know that all of the use cases, or even the correct use cases that best support the business in which the system operates, are identified? To answer such questions, you need to model and understand the system’s surroundings.

Modeling business surroundings involves answering questions such as:

- How do the different actors interact?
- What activities are part of their work?
- What are the ultimate goals of their work?
- What other people, systems, or resources are involved that do not show up as actors to this specific system?
- What rules govern their activities and structures?
- Are there ways actors could perform more efficiently?

The answers to these questions come from tackling the entire business and looking beyond the functions of the information system currently being built (and using

techniques other than use-case modeling). The ultimate objective of all software systems is to give correct and extensive support to the business of which it is a part. However, when modeling the surroundings of the information system, you are no longer modeling software. Enter the world of *business process modeling*.

^[1][Kruchten 98] Kruchten, Philippe. *The Rational Unified Process*. Reading, MA: Addison-Wesley, 1998.

^[2][Allen 98] Allen, Paul and Stuart Frost. *Component-Based Development for Enterprise Systems: Applying the Select Perspective*. New York: Cambridge University Press, 1998.

Business Process Modeling

A business is a complex system, consisting of a hierarchical organization of departments and their functions. Some of these functions, however, are not restricted to one department; they cross horizontally across several departments. The traditional method for documenting a business is to draw an organization chart, which divides the business into a number of departments or sections (e.g., research and development, marketing, sales, manufacturing, etc.), represented vertically. This documentation method is limited to how the business is built and organized. It does not document the business processes that flow through horizontally and affect all the vertical departments (e.g., the development of a new product will affect all divisions).

Other structures in the business, such as business processes, resources that participate in or are used in the process, rules that govern the execution of the business, the goals, and problems that hinder the achievement of those goals, cannot be captured in the traditional organizational view. A good business model contains all of this information. Capturing and documenting this information in itself can be the basis for making better decisions that result in a business that runs more smoothly, and better documentation for specifying the requirements of the information system.

In the process modeling field, many different theories strive to explain and improve how to structure and run a business. Very few standards, or even methods, exist in this area, and most of the literature concentrates on how to describe a business rather than on the well-defined techniques for actually running a business. The central concept used for process modeling is the business process, which describes activities within the business and how they relate to and interact with the resources in the business to achieve a goal for the process.

A business model can never be totally accurate or complete, simply because no two observers of a business will have an identical perception of the business or agree on an accurate model. As noted earlier, the business model cannot and should not contain all the details of the business. A model that attempts to do so risks becoming just as complex and as hard to comprehend as the business. Not every detail can be captured due to restrictions in the modeling language or the concepts used to build the model. Thus, the model should focus on the core business tasks and its key mechanisms. Pinpointing the core business tasks and determining what to depict in the model is the responsibility of the modeler.

Likewise, a business model of a future view of the business cannot be expected to ever be completely realized as planned. Changes in the real world can affect the basis on which the model was created, rendering the model as no longer completely valid. In addition, during implementation, the model may meet with active or passive resistance by either the business management or the employees.

Even with these limitations, however, the following arguments for producing business models are still very strong:

To better understand the key mechanisms of an existing business. By providing a clear picture of their roles and tasks in the overall organization, the models can be used

to train people. (They may be used in both a hierarchical or a process-oriented organization.)

To act as the basis for creating suitable information systems that support the business. Descriptions of the business are used to identify necessary information systems support. The models are also used as a basis for specifying the key requirements of those systems. Ideally, large sections of the business model can be mapped directly onto software objects. As more infrastructure software systems are added, potentially the systems being developed may become more business-driven, and the developers can concentrate more on functionality that supports the business rather than on solving technical incompatibilities or problems.

To act as the basis for improving the current business structure and operation. The models identify changes in the current business that are necessary to implement the improved business model.

To show the structure of an innovated business. The model becomes the basis for the action plan. Innovation suggests that radical change, rather than incremental changes, have been made to the business processes.

To experiment with a new business concept or to copy or study a concept used by a competitive company (e.g., benchmarking on the model level). The developed model becomes a sketch of a possible development for the business. The model can be a new idea, inspired by modeling other businesses, or can take advantage of new technologies, such as the Internet.

To identify outsourcing opportunities. Elements of the business not considered the part of the “core” are delegated to outside suppliers. The models are used as the specification for the suppliers.

Let's take a detailed look at each of these motives for business modeling and see how they differ.

Understanding the Business

One of the primary motives for developing any model is to increase the understanding of the business and facilitate communication about the business. A visual model is easier to comprehend and discuss than a textual description or no description at all (which is often the case). The model is a current snapshot of how modelers currently view the business. The model will change and evolve either as modelers better understand the business or as the business changes. Once the models are fairly stable, because they give a clear picture of the roles and tasks in the overall organization, they can be used to train people.

Information System Support

Today, most businesses use some type of information system. In fact, it may be said that information technology is an integral part of the daily operation of many, if not most, companies. This trend continues to gain momentum with the recognition that the effective use of computer systems will enhance almost any business. In some domains, computer systems are obligatory to handle the massive amounts of information and to respond to the need for fast and reliable communication with other companies and customers. With the Internet as a technical infrastructure for communication and financial transactions, a wealth of new business opportunities is emerging. Existing business models need to be adapted with the new possibilities that the Internet provides.

As widespread as this trend is, however, many companies are dissatisfied with the quality of their information systems, citing they offer insufficient or ineffective business support, they are awkward to use, are not reliable, and are not integrated with other systems. In many cases, this is due to the fact that the systems haven't been developed with a correct understanding of the business it supports. Developing the actual software for computer systems is still a job for qualified computer experts who are familiar with all the intricate details of programming languages, operating systems, and databases. Because of this, software systems development is often technology-driven, rather than business-driven.

The common approach to solving this problem is to have representatives of the business write a requirement specification for the systems that are to be developed. Frequently, though, this requirement specification is often inadequate and incomplete because it is usually written specifically with the information system in mind, but concentrates on describing the appearance of the user interface rather than on the relationships and interactions between the objects used in the business.

Often, the authors of the specifications are also technicians who make design decisions. Without a full understanding of the business and its needs, they prematurely make important decisions about how the information should be constructed. Sometimes these choices are in direct conflict with the functions and characteristics the business really requires from the system.

The solution is to create a model of the overall business that can be used to decide which information systems are required, how those information systems should be developed, and what functionality the systems should contain (see [Figure 1.2](#)). If the requirement specification is based on a good business model, there is a much greater chance that the information system will support the business adequately. There are several advantages to basing all the information systems on the same basic business model:

- The information systems become an integrated part of the overall business, supporting the business and enhancing the work and the results.
- The systems integrate easily with each other and can share or exchange information.
- The systems are easier to update and modify as dictated by changes in the business model, which result from changes in the surrounding environment, goals of the business, or improvements or innovations to the business model. This in turn reduces the cost of maintaining the information systems and of continuously updating the business processes.
- Business logic can be reused in several systems.

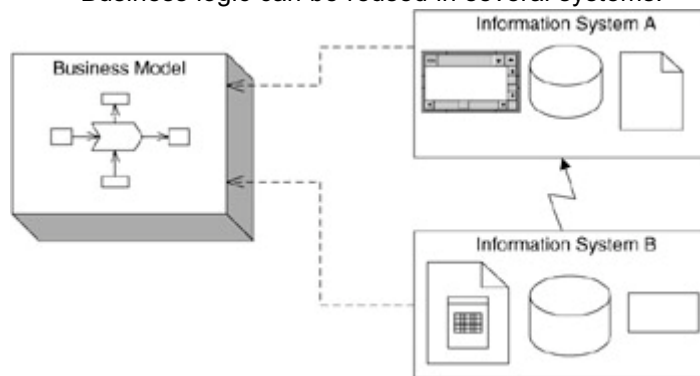


Figure 1.2: A business model can be used as the basis for defining the requirements on information systems.

Ideally, the objects presented in the business models translate or map to objects in the information system. Normally, this is not a one-to-one mapping. One object or process in the business model cannot always be translated into one object in the information system, since there are objects in the information system that are not present at all in the business model, and vice versa. This makes a direct mapping between the “real world” as described by the business model and the software system impossible. Nevertheless, even if a one-to-one mapping between the business model and the analysis model of the information system is not possible, the way the business operates and the functionality and design of the information system used to support it are more tightly integrated. Why? Because an information system that is built to support the requirements of a process will offer the appropriate services. Using object-oriented techniques, the modeling concepts and the structures used in the business model can be the same as those used in the

analysis models for the information systems. Furthermore, the information system can be implemented using the very same concepts. This book shows you how.

Using business models as a basis for the information systems also presents the opportunity for software reuse. If there are several information systems that support the same business, they often will have an overlapping set of objects. For example, several information systems that act within the same process can use the same objects from the business model. These objects have to be implemented only once and can be reused in other information systems. Often, the objects in a business model participate in several processes, and the information systems supporting different processes then can reuse reoccurring objects. It is difficult to succeed with this kind of reuse if the system has been developed with only one software system in mind. The objects become too “attached” technically to a particular system; they have not been generically modeled after the requirements of the business.

The advantage of reuse also applies to models. If the same business model can act as the basis for several information systems, it can be reused as the basic input for defining the requirements of each system. Without a common business model, each system development team creates its own analysis model to understand the real world. Not only is this redundant work, but this heightens the risk that the different teams will interpret reality differently and thus develop incompatible systems. The trend for reuse within system development is leaning more toward high-level reuse through architectural frameworks or patterns, instead of simple reuse of code (which hasn’t lived up to its promise). Reusing business models is another step in that direction.

In addition to new information systems, most businesses already have a number of information systems known as *legacy systems* that are expensive and hard to replace. These systems are functional parts of the business; they actually become a part of the current business model. In many cases, the legacy systems are the parts of the business that are most likely to be the target of an improvement or innovation plan (e.g., to substitute or remove some of these old systems).

There are two ways to depict a legacy system in the business model: it may be modeled as a single entity, such as a person acting in the business, or it may be *reverse-engineered*. Reverse engineering means that a model is created by analyzing the current information system; that is, modeling an already completed system (which is opposite from what normally is advocated). Reverse engineering breaks down the current system in order to discover the set of information objects present in the system. These information objects then become part of the business model. These two techniques are discussed in more detail in [Chapter 10](#), “From a Business Architecture to a Software Architecture.”

Improvement

A business model can be used to improve the current business. This technique, sometimes called *business process improvement* (BPI), is used to identify possible ways to make the business more efficient. The current business is modeled and then analyzed for opportunities for enhancement or improvement. The improvement part of BPI suggests that the business is changed incrementally (i.e., step-by-step improvements; see [Figure 1.3](#)) rather than through immediate and radical means. When an opportunity for improvement is identified, a new business model is produced to demonstrate how the business should look after those changes are implemented.

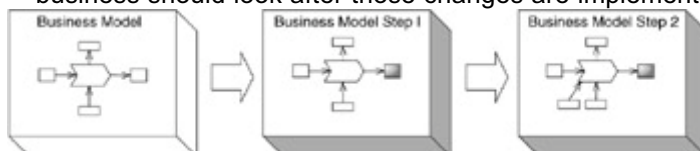


Figure 1.3: Business improvement means that changes are done incrementally.

A number of activities must be completed in order to change the business and implement the new business model:

- Describe new routines and create administrative support for these routines.

- Train the people affected by the changes; teach them the new processes and motivate them to become a part of the changes.
- Change the information systems that participate in the business to better support and enhance the operation of the business.
- Negotiate with subcontractors and other partners who will need to adapt to the changes.

Depending on the extent of the changes, improving the business can be simple or complex work. Changes occur in small steps, but they can be implemented continuously, acting upon changes in the business environment, such as changed customer needs (i.e., whenever a customer need is changed, an incremental step is performed to meet and handle that change in the business).

Innovation

Business innovation involves analyzing the current business and searching the model for new ways of doing things. The business model and its processes are changed significantly to create different and improved processes (see [Figure 1.4](#)). Often, routines in a business exist because of historical reasons (it always has been done that way) or because the infrastructure demands them to be done a certain way (the documents or the information systems don't allow them to be made in any other way).

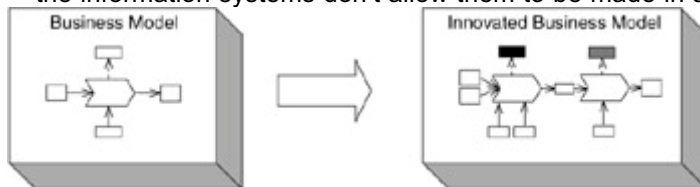


Figure 1.4: Innovation implies making radical changes to the processes.

Innovation places even higher demands on correct implementation than business improvement. Motivating and instructing the people involved and ensuring that the new processes are integrated in the existing business are essential for success. To be sure, innovation is a much bigger risk than improvement, but if innovation succeeds, the resulting business can achieve much larger gains in efficiency. Innovation is therefore typically used in companies that require radical change prompted by poor performance, missed budgets, and inefficient productivity.

An extreme form of business innovation is *business process reengineering* (BPR). BPR advocates radical changes to the business processes; this means that everything about the current way of running the business is questioned and, subsequently, often substantially changed. BPR strives to achieve dramatic improvements in efficiency in the order of several hundred percentage points, in comparison with process improvement or traditional rationalization where changes of 5 to 10 percent are deemed satisfactory.

Succeeding with such innovation is much more difficult and bears a higher risk of failure. Consequently, there has been strong resistance to BPR. Many BPR projects have failed because those involved in the business or the environment objected to the radical changes and therefore did not actively or wholeheartedly participate in implementing the new business models. Failures are also caused by incorrect design of the business process; they simply don't work when implemented.

Information systems are the key enablers to achieving process innovation or BPR. The introduction of new information systems is not, however, a guarantee for innovation. More commonly, new information systems are designed according to a current, inefficient business model, which effectively cements the current way of doing things, and makes innovation impossible to achieve. True innovation takes place first in the minds of the modelers, before it can be described and achieved in a business model. Only then can new information systems be built, either as a central and required part of that new business model or simply as a support for it.

Debate continues in the management world over whether improvement or innovation should be used when making changes to the business processes. We will not choose sides in this debate; instead, we point out that the business modeling techniques described in this book can be used for either purpose.

Design New Processes

Business modeling can be used to create new models, not previously part of the business, in order to experiment with how new business concepts fit into the current model. Models are used to determine if the current organization, resources, and information systems can be easily used in or adapted to the new process. Business models also are used to benchmark a business, that is, to copy or study business processes used by competitors in order to measure one's own business against the competition.

Often, new processes are designed based on a vision of new opportunities. Modeling this vision or idea creates a "first try" that tests the feasibility of the process. Obviously, other activities have to be performed before implementing the process, including profit calculations, cost analysis, and market studies. These activities can use the new process model as a specification of the projected goals for the new process and of the necessary resources, and to determine how the new process is implemented within the current business.

New opportunities stem from finding new combinations of or additions to objects that are already present in the organization. For example, a business can identify new ways to use customer information that it already possesses, or new services to add to the products that the business sells. By visualizing the processes in models, it is easier to see the strengths and weaknesses of the business and how to use the strengths and eliminate the weaknesses in order to turn opportunities into successes.

Outsourcing

A common practice among business leaders today is to focus on the core business, fundamental processes at which the company excels and that give them the competitive edge. Processes that are not part of the core business are good candidates for *outsourcing*. The companies hire subcontractors to handle support functions or even entire departments, rather than managing them themselves. Information systems are not the only candidates for outsourcing; other parts of the business that aren't considered vital, such as marketing or maintenance, may be sent out as well.

Business modeling can be used not only to identify and define the core processes, but also to define the processes that are candidates for outsourcing. The model then acts as a resource for the supplier and becomes a valid specification for how these processes should be performed and integrated with the core processes. The business model acts as a map that integrates the processes with each other, indicating where the processes are run by different companies and subcontractors.

Business Modeling with UML

Why use object-oriented modeling techniques to describe a business? Isn't object-oriented modeling and programming restricted to analyzing and designing computer programs? The answer is there are several advantages to using object-oriented concepts and techniques to model a business:

Similar concepts. A business can be described in terms of processes that achieve goals by collaborating with different types of resource objects. Rules define conditions and constraints as to how the processes and resources may relate to each other and how they may behave. All of this can be mapped onto objects, relationships between objects, and interaction between objects; for example, through creating static and dynamic object-oriented models.

Well-proven established techniques. Object-oriented modeling and programming has been used for several years now and has proven that it can handle large and complex systems. New techniques, such as patterns, have been introduced in the field of object-oriented modeling, and a number of patterns are available for modeling businesses.

Standard notation. Business modeling methods and techniques are in need of a standard notation: every method uses its own notation and its own tool, if notation is used at all. Object-oriented modeling finally has a standard notation: UML. That means the tools are already there, and that the same tools that are used to model the information systems can be adapted and used to describe the business models. It also opens up the possibility of enabling continuous traceability of business requirements all the way to the implementation code. The absence of this capability is a major weakness in most tools today.

Short learning curve. It is a major advantage when the same basic concepts (objects, classes, etc.) used to describe the information systems that support the business can also be used to describe the business as a whole. Just as object-oriented models have decreased the semantic gap between those who analyze and design systems and those who program them; using object-oriented techniques and notation will decrease the gap between the business modelers and information systems modelers and architects (assuming both have prior knowledge of object orientation).

New and easier ways to view an organization or a business. The traditional way of describing and viewing an organization doesn't show much of how the business is performed. The functional division of a business into organizational charts can't be used to describe modern business processes that are horizontal to the organization and affect many functions within the business. Object-oriented techniques can easily show these processes, as well as the traditional organizational structure.

There are, of course, a number of other ways to create business models, including using other notations such as IDEF0; it is also possible to simply use textual descriptions of the processes. But UML is a well-defined standard, supported by many tools. It is the dominant modeling language used to model object-oriented information systems.

Summary

The purpose of business modeling is to generate descriptions (abstractions) of a complex reality that capture the core functions of the business. The models show agreed-upon fundamentals, and can serve as a point of discussion. There are many motives for business modeling: to better understand the key mechanisms of a business, to act as the basis for supporting information systems, to improve the business, to radically change the business, to identify new business opportunities, and to identify outsourcing opportunities.

UML has quickly been adopted as the standard modeling language for modeling software systems. This book illustrates how it also can be used for business modeling, based on object-oriented concepts and thus demonstrates that the same modeling language can be used for business and for software models.

This book shows a business model from a number of views. Each view is expressed in one or more diagrams. The diagrams can be of different types, dependent upon the specific structure or situation in the business that it is depicting. The diagrams capture the processes, rules, goals, and objects in the business, and their relationships and interactions with each other.

Naturally, business models alone can't guarantee the success of a business. Good leadership and management are still needed, both in terms of having a long-term strategy and in running and coaching the daily operations. Nor are business models a substitution for finding the right people to work in the business, or for motivating and rewarding these people. All of these issues are very important, but beyond the scope of this book.

[Chapter 2](#) offers a short introduction to the basics of UML and describes some important concepts such as powertypes and stereotypes. Beginning in [Chapter 3](#), "Modeling the Business Architecture," we'll use UML to do business modeling.

Chapter 2: UML Primer

Overview

The Unified Modeling Language (UML) was created by Grady Booch, James Rumbaugh, and Ivar Jacobson, and later standardized by the Object Management Group (OMG) in 1997. Since then many people and companies have contributed to making UML the language for modeling software systems that it is today.

UML has become one of the most important topics of discussion in modern software engineering circles; and it is becoming of interest to management consulting firms, business analysts, system analysts, software developers and programmers, and people working with requirement specifications. In short, UML has had and continues to have a tremendous impact. Many companies have decided that all their software should be modeled with UML, and thousands of people have taken training courses in the language. Many books have been written about UML, and many software tools support it. All this progress and UML's importance as a generic modeling language is still only in its infancy; its use is expected to grow substantially in the years to come.

UML Basics

A modeling language has a *notation*—the symbols used in the models—and a set of rules that govern the language. The rules are *syntactic*, *semantic*, and *pragmatic*. The syntactic rules dictate how the symbols should look and how they may be combined. The syntax can be compared to words in a natural language; as in a natural language, it is important to know how to spell them and how to combine them to form sentences. The semantic rules tell us what each symbol means and how it should be interpreted by itself or in the context of other symbols. These rules can be compared to the definitions of words in natural languages (what each word represents). Pragmatic rules explain how to use the language. They may be compared to writing guidelines in natural language. They define the intentions of the symbols, through which the purpose of the model is achieved and becomes understandable to others.

In UML the symbols are geometrical, such as rectangles, circles, and lines. It has a well-defined set of syntactic and semantic rules that define what the symbols mean and how they can be combined. But UML does not have pragmatic rules, that is, specific guidelines for how to use it. Therein lies one of the purposes of this book: to show pragmatic ways of using UML in business modeling.

This chapter gives an overview of the basic UML symbols, syntax, and semantics. The pragmatics of UML in the context of modeling businesses are explored in the remainder of this book. This chapter focuses on introducing UML fundamentals; it presumes a basic knowledge in object-oriented methodology. Those readers with a good knowledge of UML and object orientation can regard this chapter as a refresher and feel free to skim it. The chapter also addresses some special areas, such as powertypes and UML extensions, which will be of interest to readers already proficient in basic UML.

Unified Modeling Language

UML has nine predefined diagrams:

Class diagram. Describes the structure of a system. The structures are built from classes and relationships. The classes can represent and structure information, products, documents, or organizations.

Object diagram. Expresses possible object combinations of a specific class diagram. It is typically used to exemplify a class diagram.

Statechart diagram. Expresses possible states of a class (or a system).

Activity diagram. Describes activities and actions taking place in a system.

Sequence diagram. Shows one or several sequences of messages sent among a set of objects.

Collaboration diagram. Describes a complete collaboration among a set of objects.

Use-case diagram. Illustrates the relationships between use cases. Each use case, typically defined in plain text, describes a part of the total system functionality.

Component diagram. A special case of class diagram used to describe components within a software system.

Deployment diagram. A special case of class diagram used to describe hardware within a software system.

These diagrams capture the three important aspects of systems: structure, behavior, and functionality. Because of UML's unique capability to adapt and extend, it is possible to add new diagrams and elements to UML, making it a very flexible language that can be used in many situations. Extending UML is discussed in more detail later in this chapter.

Class Diagram

Systems are built from *objects*, which can be physical, such as computers, people, and raw materials, or abstract, such as information or knowledge. Objects are described by their internal properties and their relationships to other objects. For example, Bill's bank account (an object) has attributes common to all bank accounts, such as balance, account number, and credit on current account. In addition, Bill's bank account has relationships to other objects, in this case to a specific bank (a bank object) and to Bill himself (a person object). Attributes are described with a name and a type (integer, Boolean, string, date, etc.). For example, the account number attribute is an integer (a type); and in UML it is shown as 'Account Number : Integer'. Objects also have behavior; for example, one can ask a bank account for its balance. Behavior is described with operations attached to the objects. A bank account could, for example, have "cash withdrawal" and "get balance" operations.

A *class* is a set of objects with the same characteristics. Typical classes are Person, Invoice, Company, Supplier, Order, Product, and Goal. Classifying and grouping objects into classes reduces the complexity and number of elements when modeling and facilitates describing more complicated systems.

Classes are modeled and related to each other in a *class diagram*. The classes are described with names, attributes, and operations. The relationships between the classes are described with a name, roles, and multiplicity. For example, many companies (employers) can employ many persons (employees). This relationship can be described in terms of classes: the class Company has an employ relationship with the class Person. The company plays the role of employer and the person the role of employee. This relationship also has multiplicity, because companies employ many persons, and a person can be employed by many companies.

Class diagrams are used to describe the objects and relationships of a system. Class diagrams capture and describe information within an information system (physical items in mechanical systems, parties in organizations) or they can be used to describe objects in biological systems, such as the ecosystem. [Figure 2.1](#) is a class diagram for a small system for insurance companies. The model specifies that:

- A person can be a policyholder, and a policyholder has one or many insurance contracts.
- One insurance contract has one or many policyholders, which are persons.
- One insurance company plays the role of an insurer who has zero, one, or many insurance contracts with policyholders. The insurance contracts

represent one insurance, which can be a car insurance, life insurance, or homeowner's all-risk insurance. The insurance is regulated by the insurance contract, which specifies policyholder, term of insurance, and policy value. The contract includes all of this information (as attributes) in an insurance policy document.

- An insurance contract is expressed in one (or zero, when it has not been printed yet) insurance policy.



Figure 2.1: A class diagram describing a small system for insurance companies.

Classes and Objects

Another way to introduce the basic concepts of object technology is to quote from *The Universal Dictionary of the English Language* (Wordsworth Editions Ltd, 1989). All the basic concepts are defined there:

Class. Order, group, category of organisms, etc., which have some essential character or feature in common.

Object. (1) That which is presented to, or observed by, the senses; a material thing, anything visible or tangible. (2) That which is presented to, and grasped by, the mind; that which can be apprehended or known.

Attribute. A quality ascribed to, and considered as inherent in and essential to, any person or thing.

Operation. Act performed, transaction, in the way of business: operation on the stock exchange.

Association. (1) State of being associated, companionship, intimacy. (2) Connexion, bond.

Generalization. (1) Mental process of generalizing. (2) A notion, rule, law, etc. resulting from such a process; derived, evolved, formulated by observation of specific instances.

Multiplicity. Quality, state, of being very numerous or various.

Role. Part played by actor.

In UML, classes are defined as a set of objects with a name, attributes, and operations. The classes can be associated to each other via associations; and the associations have names, roles, and multiplicity. The only concept quoted from *The Universal Dictionary of the English Language* that was not explicitly discussed in the introduction is generalization. The insurance class in the model shown in [Figure 2.1](#) is a generalization of car insurance, life insurance, and homeowner's all-risk insurance. The class Insurance then includes the car insurance, life insurance, and householder's all-risk insurance. In a mathematical sense, a class is a set of objects, and a generalization is just a set of sets. The Insurance class is a superset containing the subsets of car insurance, life insurance, and householder's all-risk insurance.

A class is drawn in UML with a rectangle that is divided horizontally into three compartments. The top compartment contains the name of the class; the middle compartment contains the attributes of the class; and the bottom compartment contains the operations of the class. A compartment can be suppressed in a diagram. Attributes with a plus sign (+) in front of them are public, meaning that other classes can access

them. Attributes preceded by the minus sign (-) are private, indicating that only the class itself and its objects can access the attribute. Protected attributes, marked with the number sign (#), can be used by the class and any descendant (specialization) of the class. Attributes can also be marked as *class scope global* by underlining the attribute. This designates that the attribute has only one value in common with all objects. An example of a class scope global attribute is a counter, such as number of invoices, that is common to all instances of the class. An attribute's possible values can be enumerated, in which case they are shown in the form {a,b,c...}.

[Figure 2.2](#) shows class Invoice with the name and attribute compartments (the operations compartment is suppressed). Invoice has the attributes amount, date, customer, specification, administrator, number of invoices, and status. The amount attribute is of type Real (real number), date is of type Date with the initial value Current date, which means that when a new object of the class Invoice is created the attribute date of that object will be assigned the current date.

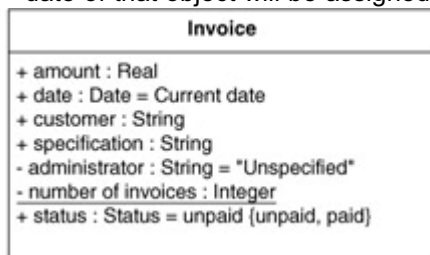


Figure 2.2: A class with attributes.

Classes also have *operations*, services that the class provides. Operations are similar to functions in programming languages; but the operations in a class have unique access to all the attributes in that class. An operation has a name, visibility, a list of parameters, and a return type. An operation performs some service that the class provides; it sometimes requires parameter values as input; and it returns a result of the return type.

[Figure 2.3](#) shows a Bank Account class with the operations cash deposit, cash withdrawal, and bank statement.

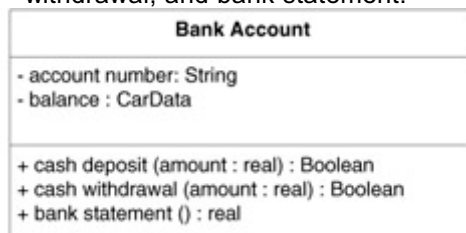


Figure 2.3: Examples of operations.

Operations can also be class scope global. Two examples of operations that always are global (for a class) are create and destroy (sometimes referred to as the constructor and the destructor of the class). A create operation is used to create new objects. It is global to the class because it is not possible to ask an object to create itself (because it needs to exist to ask it that). The destroy operation is used to model the destruction of objects. Associations are given names (usually verbs) and are represented in UML with lines. The association name appears near the association line, and multiplicity and role names appear on each end, as shown in [Figure 2.4](#). Multiplicity is presented as a range, such as 1..5 or 0..* or as a specific number, such as 4. The asterisk (*) indicates many (an open interval). The multiplicity 0..* indicates zero or more. For example, [Figure 2.4](#) shows that the class Insurance Company is associated to zero or more insurance contracts (specified at the end of the association line, near the Insurance contract class). Multiplicity is indicated at the ends of the association, also called the roles of the association. Specifying a name for each role is optional.

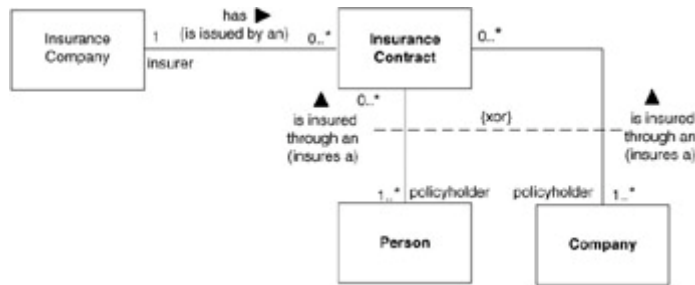


Figure 2.4: A class diagram with classes, associations, association names, roles, multiplicity, and a constraint (xor). The reversed name of the association is indicated in parentheses.

Associations can also be given a name direction, specifying how the name of the association should be read. (For example, the Insurance Company has Insurance contracts, not vice versa.) The name direction is drawn near the association name and marked with a small, solid black triangle. If needed, the reversed name (the name in opposite direction) of an association can be indicated in parentheses. The constraint {xor}, drawn between the “is insured through” associations, indicates that only one of these associations can be valid at a time.

The class diagram in [Figure 2.4](#) indicates that the Person class (a noun and a subject) is associated to the Insurance contract class (a noun and a object) with the association of “has” (a verb) and the multiplicity of 0..* at the end of Insurance contract. The Person is insured through one or more Insurance contracts.

More about Relationships

The *aggregate relationship* is a specialization of association used in situations where a whole is connected with its parts, as shown in [Figure 2.5](#). The hollow diamond on the association in this figure denotes that a delivery aggregates many products (the parts). A delivery consists of a number of products. The constraint {ordered} indicates that a specific order exists between the different products.

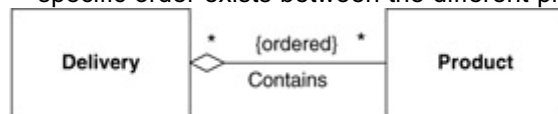


Figure 2.5: A delivery aggregates a number of products.

Object composition, shown with a filled diamond, is a stronger form of aggregation. This relationship indicates that the parts can only be in one entity at time. [Figure 2.6](#) shows object composition.

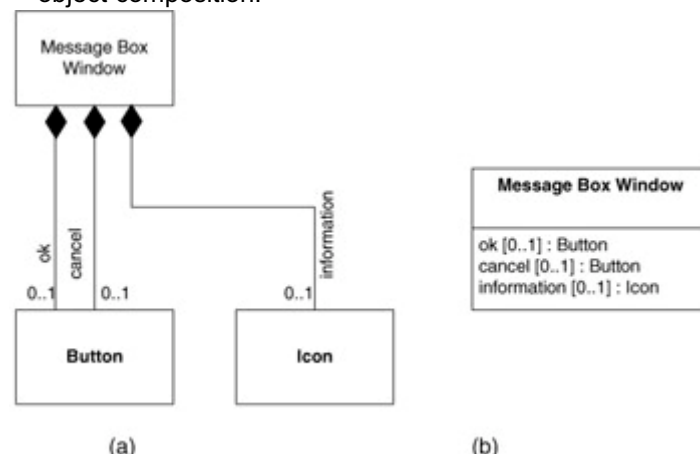


Figure 2.6: Object composition. Buttons and an icon compose a message box. The right side (b) is another way to show object composition.

Where three classes are connected, a *ternary association* is used, as shown in [Figure 2.7](#). Associations are normally bidirectional but there may be unidirectional associations in which the association is only valid in one direction. The association in [Figure 2.7](#) is unidirectional. In this figure, a Plan and a Project are connected to several Resources. Ternaries can be used with both unidirectional and bidirectional associations.

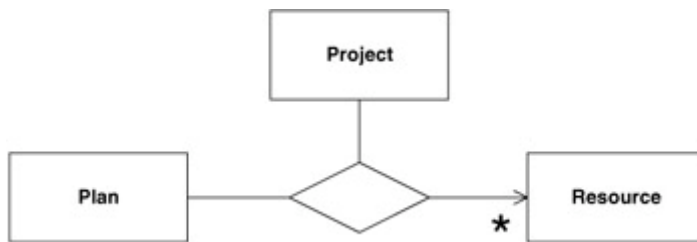


Figure 2.7: A ternary association is shown with a large open diamond.

Dependency is another type of relationship expressed between two classes in which one class relies on the other. Dependency also can be used to express dependencies among all the model elements (objects, packages, activities, etc.) included in UML. A dependency is indicated by a dashed line with an arrow from one element pointing to the element on which it depends. *Realization*, shown with a dashed line ending in a hollow triangle, shows that one element is used to realize another element. One class can be a specification or an idea, and another class can implement or detail that specification or idea. [Figure 2.8](#) shows a class *Company* that is dependent on an *Information System* that is realized as a *Computer-based Information System*.

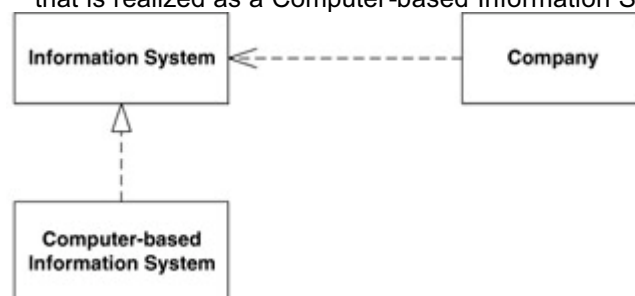


Figure 2.8: Dependency and realization.

UML also provides the concepts of *generalization* and *specialization*. The generalization mechanism is used to organize objects in hierarchies. Recall that a class is just a set of objects (e.g., the set of all red cars). A set can be divided into subsets or generalized. For example, a set of all red cars can be divided into subsets of red cars with two doors and red cars with more than two doors. The subset is then a specialization. Specializations are always made upon one or more properties (attributes or operations), such as the number of doors, in this example, or different behavior of red cars. A class can be generalized into a superclass (or in fact, several superclasses if multiple generalization is used) and specialized into subclasses. [Figure 2.9](#) shows an example in which resources are specialized into *Products*, *Documents*, *Information*, *Staff*, and *Raw Material*. *Products* are then further specialized to *Items* and *Services*. The line with the hollow triangle shows the generalization; the triangle points at the more general class.

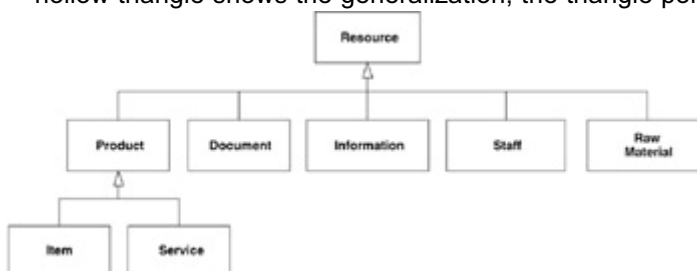


Figure 2.9: A generalization hierarchy.

[Figure 2.10](#) shows another generalization hierarchy, this time concerning figures. All figures have a drawing operation, which is redefined for each specialization. The figures also have the position attribute, which is protected (meaning that only those from the class itself and its specializations can use it). The note symbol shown as a rectangle with a “bent corner” in the upper right corner is used in the figure to comment on the subclasses’ *draw()* operation.

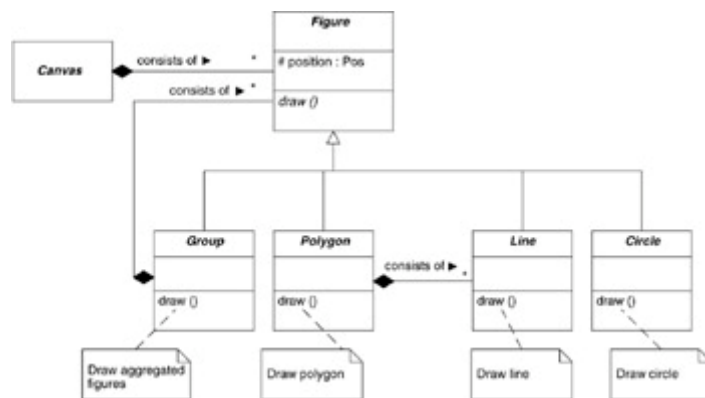


Figure 2.10: A class diagram with generalization, protected attribute, redefined operations and notes.

The class name, *Figure*, in [Figure 2.10](#) is italicized to indicate that it is an abstract class. Abstract classes cannot be directly instantiated into objects themselves, but their subclasses can be instantiated (unless the subclass itself is defined as being abstract as well). In *The UML Reference Manual* (Addison-Wesley, 1999), James Rumbaugh writes that “An abstract class may not have direct instances. It may have indirect instances through its concrete descendants.” Abstract classes are used to define generic superclasses whose only purpose is to be subclassed and specialized into more concrete classes.

The `draw()` operation is also italicized to indicate that it is an abstract operation that must be defined in the subclasses. An abstract operation has no implementation in the class in which it is defined, but is a way for a superclass to force its subclasses to implement that specific operation (or again define it as abstract). A class with at least one abstract operation is also automatically an abstract class.

Note that subclasses are not always disjointed. If two subclasses are disjointed, none of the objects belongs to more than one class. In many cases, especially in business modeling, classes overlap (hold overlapping sets of objects). For example, the class **Book** represents all books in a certain context. These books can be subclassed into management books and software engineering books. However, there is an overlap between the set of all management books and the set of all software engineering books, namely business modeling books. [Figure 2.11](#) illustrates this discussion. The fact that an overlap exists is expressed with the constraint `{overlapping}`.

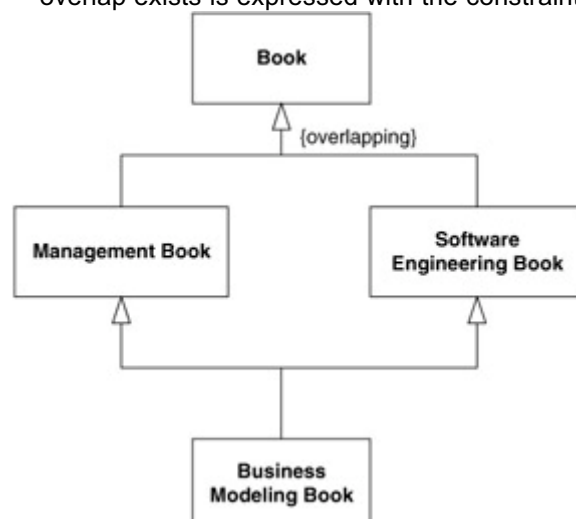


Figure 2.11: Overlapping specialization.

A special model element in UML is used to express specifications. This model element is the *interface* and it is denoted by a line with a circle at the end (the “lollipop” symbol). The interface is a specification of a collection of operations that can be implemented by one or more classes. [Figure 2.12](#) shows how the interface concept is used to specify that Microsoft Word and Microsoft PowerPoint both implement the spell-check feature. An interface can’t have attributes; it only has abstract operations. Classes that have this

interface attached to them (that choose to “implement the interface”) must implement all the operations defined in the interface. An interface typically describes a generic feature or functionality that different classes or components can choose to support.

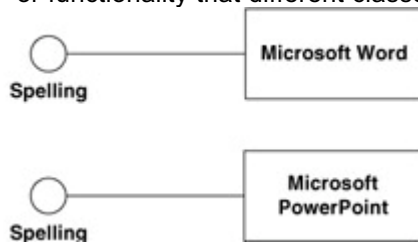


Figure 2.12: An example with interface.

Packages

Packages are used to organize and handle complexity in large models. A package groups model elements, such as classes, states and activities, and names that group so that the package can then be referred to as a whole. Dependencies are used between packages to show how the packages are dependent on each other, as shown in the class diagram depicted in [Figure 2.13](#). This type of diagram sometimes is referred to as a package diagram, but there is no such diagram type in UML; it is a class diagram that shows packages. It is also possible to draw the packages directly in a model, a class diagram, or a sequence diagram. The package symbol is shown in [Figure 2.13](#).

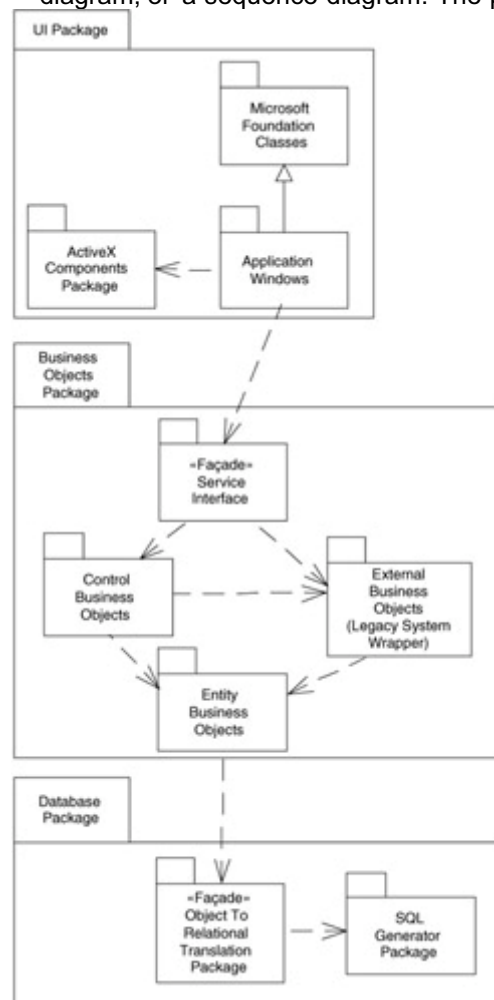


Figure 2.13: Three packages with subpackages. The relationships between the packages are dependency and generalization.

Powertypes

Often it is desirable to model a system generically, for example, to handle articles, production equipment, or a sales system. A common problem that arises when developing generic models is how to handle new types of objects that are added over time. In a system built, say, for organizing and registering vehicles, it would include vehicle types such as cars, boats, airplanes, helicopters, and bikes, to name a few. But what if new types of vehicles were to appear in the future? It is a daunting task to discover and include all object types in a system, not to mention impractical (if not impossible) to model it all beforehand without knowing exactly which types will be required.

Instead, objects and types of objects can be separated. [Figure 2.14](#) shows that the class Car is a set of many cars, such as Bob's car or Bill's car. These cars can be grouped into new subsets such as Volvo, Ford, or Toyota. The subsets (subclasses) are nothing more than objects to the set and class Kind of Car. The class Kind of Car is called a *powertype* to the class Car. The objects to the class Kind of Car (Ford, Volvo, Toyota, etc.) are subclasses to the class Car. Powertypes are used to model types of objects; the objects themselves have no knowledge of all objects of either class.

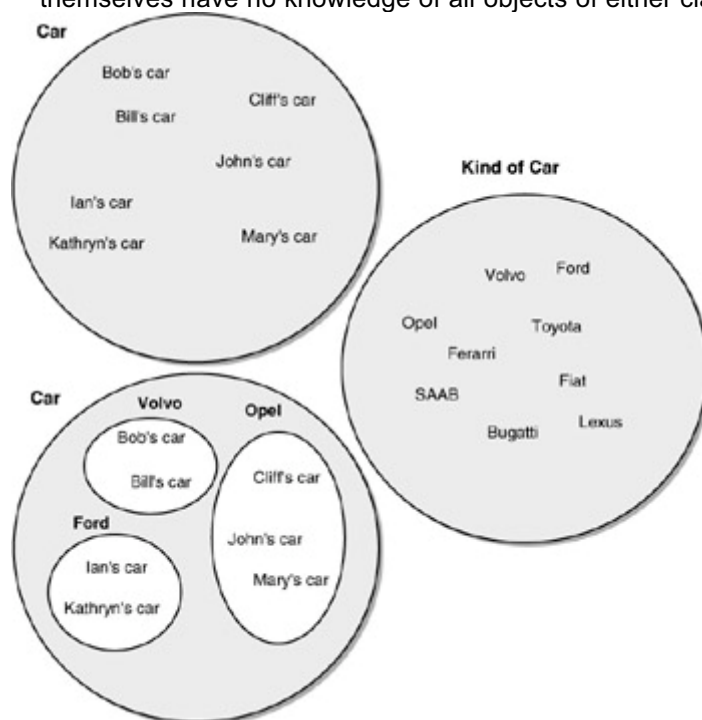


Figure 2.14: Class, subclass, and powertype.

Powertypes are widely used in more complex business models and in many of the commercial database applications (in databases this entity is often called metadata). There is a mathematical definition of powertypes as well. In terms of set theory, a class is a set and its subclasses are subsets. The number of objects (elements), let's say X , of the set A can be divided into maximal 2^X (2 power X) subsets. The set of all subsets is the *powerset*. The set of all possible subsets of set A is the powerset of A . For example, if the set Color includes the objects red, green, and blue, there are eight possible combinations of these color objects. The possible combinations (subsets) are: RGB, RG, R, GB, G, B, RB, and $\{\emptyset\}$, where $\{\emptyset\}$ is the empty set. The set that contains these combinations is called the powerset of the set Color.

[Figure 2.15](#) shows how UML powertypes can be visually modeled. In it, the powertype Kind of Car is related to the class Car. The powertype Kind of Car is a class in itself (a stereotyped class; stereotypes will be further explained in the section on extending the UML). [Figure 2.16](#) shows another, more pragmatic, way of indicating powertypes in UML. It is easy to implement in computer-based systems. The model says that a Volvo, Ford, and Opel are examples (instances) of some kind of cars. Volvo, Ford, and Opel are

also cars; they are specializations to the class Car. However, there are more kinds of cars, and that is shown by the constraint {incomplete}.

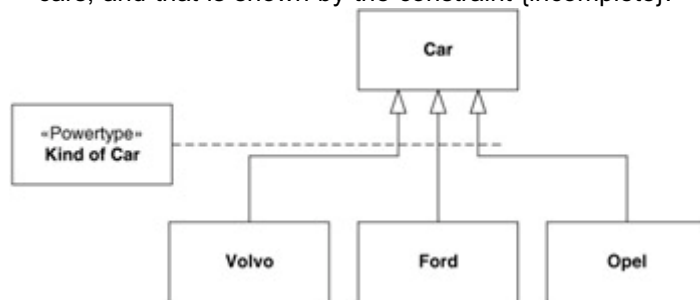


Figure 2.15: The formal way of showing how powertypes relate to classes [Rumbaugh 1999].

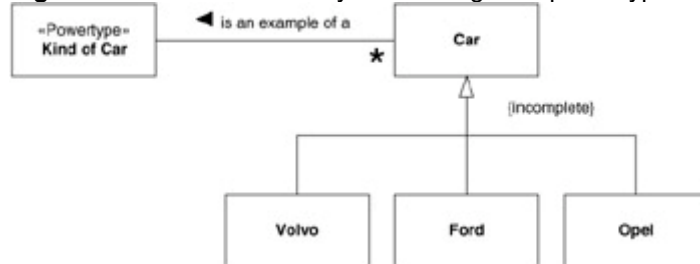


Figure 2.16: A pragmatic way of showing powertypes.

Figure 2.17 shows a business example in which a powertype is used, specifically, a generic class diagram for an organizational chart. Individual organizational units can be connected to each other to show the relationships among the units (which unit is a subunit to another). For example, the Organizational unit Software Inc. is connected to the units Production, Sales department, and Development, which all are subunits to the Software unit. Each Organizational unit is also linked to an Organization type (which is a powertype). For example, the Organization type Department is connected to the Organizational units Production, Sales Department, and Development, showing that all of these units are departments. Corporation, subsidiary, division, team, and group are other examples of organization types. By being linked to a specific powertype object, the type of the organizational unit can easily be denoted and new types of organizational units can be added later without having to change the class diagram. Powertypes will be illustrated and used in further examples in [Chapter 7](#), “Resource and Rule Patterns.”

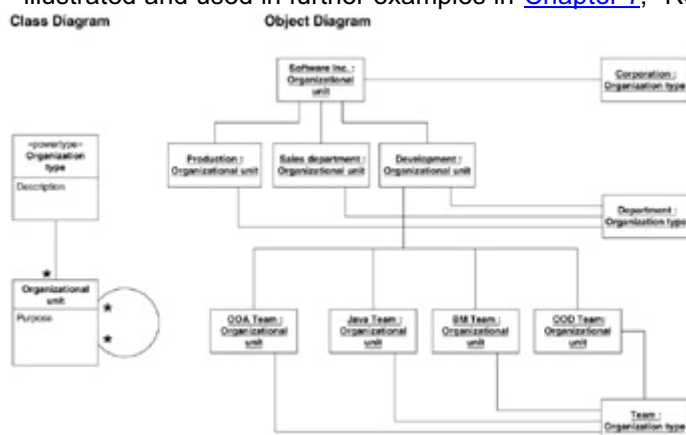


Figure 2.17: A business example with a powertype.

Note that [Figure 2.17](#) also models a specific organization, Software Inc., as an *object diagram*. An object diagram can be used to show an example of a class diagram. In [Figure 2.17](#), Software Inc. is organized into three departments: Production, Sales, and Development. The Development department is in turn organized into four teams: OOA Team, Java Team, BM Team, and OOD Team. This object diagram shows how powertypes can be used in a simple class diagram. Powertypes also are used to express more complex object diagrams. Note that new types of organizational units can be added as new objects, without having to modify the class diagram. The [next section](#) explores object diagrams in more detail.

Object Diagram

Object diagrams explain and illustrate complex class diagrams; they are pictures of objects and their relationships at a specific moment in time. Object diagrams are drawn with objects and links. Links are instances of associations and aggregations. Because an object diagram is a snapshot, multiplicity is not shown (the actual links at that moment in time are shown instead). [Figure 2.18](#) is an object diagram that exemplifies the class diagram in [Figure 2.10](#). The object names, which are underlined, are composed of their names, followed by a colon and the class name; for example, G1 : Group. Three examples of naming objects are:

- Ben : Person (An object named Ben of a class named Person)
- Bob (An object named Bob of an unnamed class)
- : Person (An unnamed object of a class named Person)

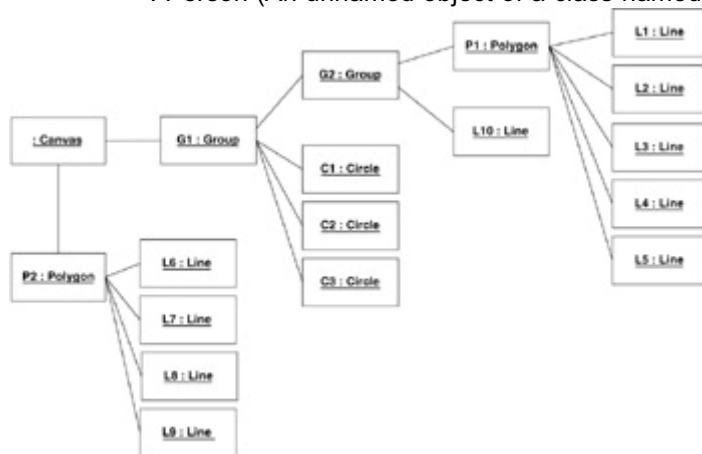


Figure 2.18: An object diagram.

Named objects (e.g., Ben : Person, Bob) are used when the name of an object is of primary interest. The anonymous name form (: Person) is used when only the class is of interest and no specific name is needed for the object.

In addition to object names, the state and attribute values of an object can be shown. The state is shown in brackets after the object name; for example:

Bill's Invoice #34 : Invoice [paid]

Though the attribute compartment is normally suppressed when showing objects, it can be shown, as was the case with class diagrams. If shown, the attribute's value is indicated as follows:

description : string = "this is a sample description"

This technique is often used in combination with activity diagrams and is discussed further in the section about activity diagrams later in this chapter.

Statechart Diagram

A *state* is the result of the attribute's value and links to other objects. It specifies how the object reacts to events occurring around it. Statechart diagrams capture the life cycles of objects, subsystems, and systems. They indicate what states an object can have and how different events affect those states over time. Statechart diagrams should be attached to classes that have clearly identifiable states and complex behavior, but they also can be defined for a subsystem or an entire system. The diagram specifies the behavior of an object and how that behavior differs from state to state. It also shows which events change the state of the objects of the class.

A state is typically represented by the attributes' values and links to other objects. Examples of object states are:

- The invoice (object) is paid (state).
- The car (object) is standing still (state).
- The engine (object) is running (state).
- Jim (object) is playing the role of a salesman (state).
- Kate (object) is married (state).

An object changes state when an event occurs, that is, when something happens. There are two dimensions of dynamics, the *interaction* and the *internal state change*.

Interactions describe the object's external behavior and how it interacts with other objects (by sending messages or linking and unlinking to other objects). Internal state changes describe how the object changes state, for example, the values of its internal attributes in different states. Statechart diagrams show how objects react to events and how they change their internal state as a result of the interaction. Interaction between objects is described in more detail later.

Statechart diagrams can have a starting point and several end points. A starting point, or *initial state*, is represented with a solid circle; an end point or *final state* is represented with a small solid circle surrounded by a larger empty circle (to form a bull's-eye). A state is represented as a rectangle with rounded corners. Changes in states, or *state transitions*, are indicated with a line ending in an arrow pointing from one state to another. The state transition is labeled with its cause (the event that generates the state transition). When the event occurs, the transition from one state to another is performed (sometimes spoken of as "the transition fires" or "the transition is triggered"). [Figure 2.19](#) is an example of a statechart diagram.

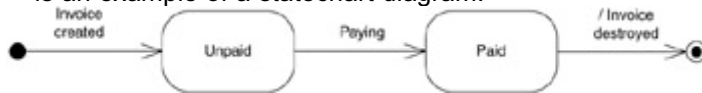


Figure 2.19: A statechart diagram for invoices.

[Figure 2.20](#) shows a statechart diagram for the Stock Order resource, which represents a stock order placed by a customer. It will be created when an order is received from a customer. At some point, a communication with a marketplace will put this order onto the market. The marketplace will then try to match this order with other orders on the market; that is, match a buy order with a sell order. If a match is reported from the marketplace, the order will be marked as concluded; thereafter, an action will generate a security holding that represents the shares bought. An order can also be canceled and withdrawn from the market; or the trading day can end without making a match. According to the state diagram, there must be an explicit decision to put the order back onto the market for the next day. If it's not put back on the market, it will be marked as a canceled order.



Figure 2.20: A statechart diagram for a stock order.

Event Types

There are four types of events that trigger state transitions: *call events*, *time events*, *signal events*, and *change events*. A call event is a called operation that affects the attributes and links of an object. An example of a call event is:

withdrawal(amount)

Time events cause a change in time. They are expressed with the keyword **after** and occur at a specific moment in time. An example of a time event is:

after (15sec)

Change events cause a change in a link or attribute value. Change events are expressed with the keyword **when** and occur because of a change in one of the attribute values or links. An example of a change event is:

when (t = 2 days)

A fourth event, called *signal event*, occurs when a signal is received. The signal event is handled and described in the [“Activity Diagram”](#) section later in this chapter.

Nested and Parallel States

States can also be *nested* and in *parallel*. A state that contains itself and is divided into other states is nested. A parallel state is when an object has several states at the same time. A car, for example, can be new or old and be moving forward, backward, or standing still.

[Figure 2.21](#) shows a statechart diagram for a production plan that has both nested states and parallel states. Initially, the production plan is proposed and not suspended, which means that the production plan has two states at the same time, namely the state proposed and the state not suspended. Between the time the production plan is started and accomplished, it can be suspended and even abandoned. If the production plan is suspended, it can be restarted; but if it is abandoned, it cannot be restarted (i.e., there is no state transition, triggered by a restart, from the state abandoned to the state suspended or the state not suspended). The production plan has two parallel states at all times, and it is composed of substates (nested states). The state Started has four internal states (nested states), namely Preparation, Negotiation, Accomplishment, and Acceptance.

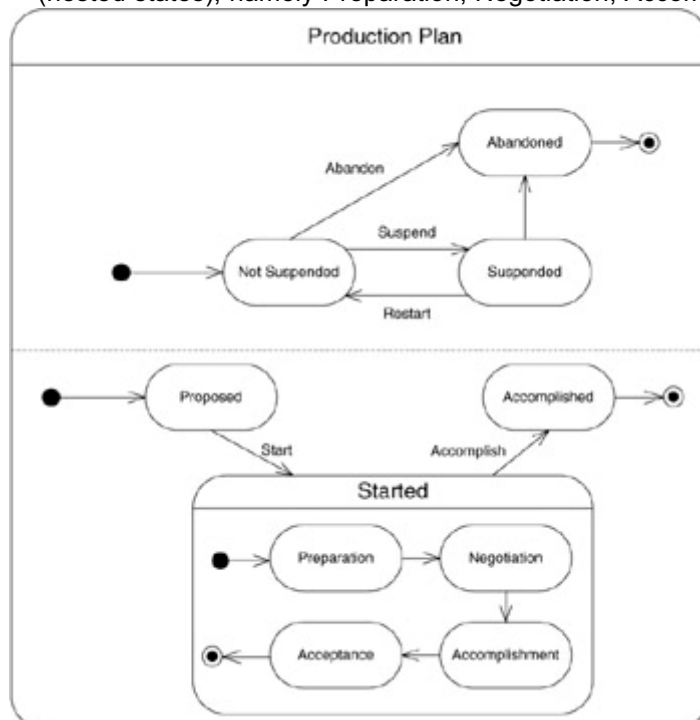


Figure 2.21: A statechart diagram with nested and parallel states.

State Transitions

A state transition usually has an event attached to it, but it is not necessary to attach one. If an event is attached to a state transition, the state transition will be performed when the event occurs. A *do-expression* within a state can be an ongoing process (e.g., waiting, producing, following some plan, etc.) performed while the object is in the given state. A *do-expression* can be interrupted by outside events, meaning that an event in a state transition can interrupt an ongoing internal do-action. [Figure 2.22](#) shows a state with a do-expression.

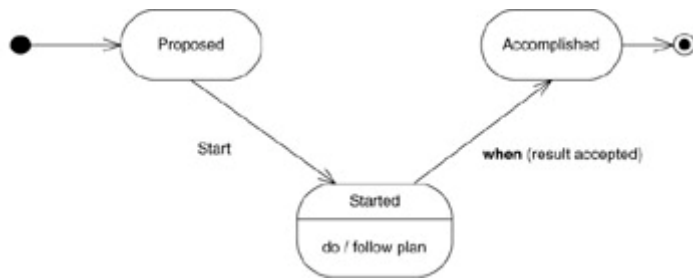


Figure 2.22: A statechart diagram for a plan.

If a state transition does not have a specified event, the state will change when the internal actions in the source state are executed (i.e., the do-expressions). When all the actions in a state have been performed, a transition is triggered automatically, without an event.

An *action-expression* can be connected to a state transition. An action-expression is a procedural expression that is executed if the transition fires. It can be written in terms of operations and attributes within the owning object (the object that owns all of the states) or with parameters within the event-signature. It is possible to have more than one action-expression on a state transition, but they must be delimited with a forward slash (/). Action-expressions are executed one by one in the order specified (from left to right). Action-expressions cannot be nested or recursive. It is, however, possible to have a state transition that contains only an action-expression.

An example of a state transition with an action-expression is:

`add (n) / sum = sum + n`

A special case of the action-expression is the *send-clause*, an explicit syntax for sending a message during a state transition. The syntax consists of a *destination-expression* and a *message*. The destination-expression points out an object or a set of objects. The message should correspond to a state transition in the destination object that is pointed out by the destination-expression. The caret (^) indicates that a send-clause follows.

An example of a state transition with a send-clause is:

`out_of_paper() ^indicator.light()`

[Figure 2.23](#) shows a class and its statechart diagram. It demonstrates how operations within the class are related to the call-events specified by the statechart diagram. The watch has three states: its normal display state showing the time and two states for setting the clock (hours and minutes, respectively).

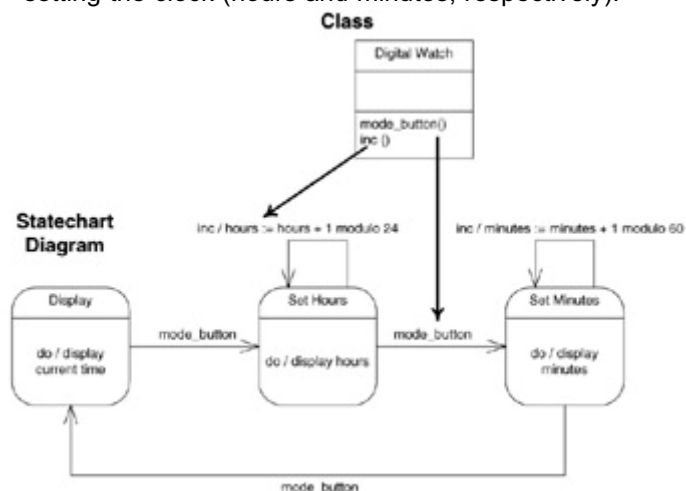


Figure 2.23: The class Digital Watch with its corresponding statechart diagram.

Activity Diagram

Activity diagrams are used to explore and describe a workflow, the actions performed in an operation in a class, similar to traditional program flowcharts. In addition, activity

diagrams are used to describe business processes, workflows in the context of organizations. They are described in great detail in this section because of their importance to business process modeling. (The term “business process” will be explained and illustrated in greater detail in Chapters 3 and 4, but for the purpose of this discussion, a business process describes a simple flow or sequence of activities.) A workflow can be a simple operation, such as how to place an order in an order system, or it can be more complex, such as how to control production and product development. [Figure 2.24](#) shows an activity diagram for an order process. The order process begins with the Prepare Inquiry activity and then sends the inquiry. After the inquiry is sent, the Order Process waits for a notice of delivery. When the delivery is received by the Order Process, it is checked, and an acceptance is sent to the supplier. An invoice is received by the Order Process and the payment is prepared and sent. Note that there are no events specified on the transitions; in an activity diagram there is an automatic transition to the next activity once the preceding activity has been performed.

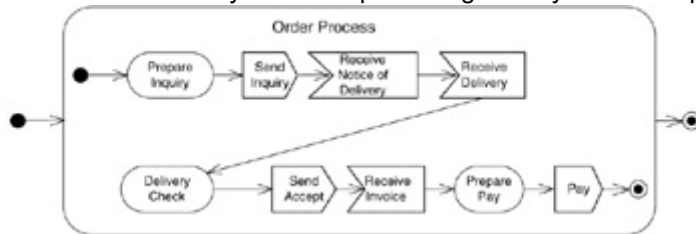


Figure 2.24: An order process.

Activity States

The activity diagram has some similarities to the statechart diagram, but the “states” in an activity diagram are actually activities to be performed, and the transitions are fired by ending activities (i.e., when the activity specified in a “state” has been performed, a transition to the next activity happens automatically; no event is necessary, as is the case in the statechart diagram). The states in an activity diagram are called *activity states*, or more simply, *activities* (states with a do-expression). Activity states are states in which some activity is performed. Activities can be divided into *subactivities*. Subactivities that cannot be further divided are called *actions* (atomic activities); they are indicated with the same symbol, the activity symbol (rectangles with rounded corners). Activities and actions are connected via transitions; transitions in an activity diagram make up the *control flow*.

Control Flow

All syntax used in statechart diagrams (with the exception of events) can be applied to the control flow. A control flow can have guards, actions, and send-clauses. The send-clauses are marked by a special symbol, which is shown in the [“Send and Receive”](#) section upcoming. [Figure 2.25](#) illustrates an activity diagram with start, stop, and a set of activities connected with a control flow. The start and stop symbols are the same as in statechart diagrams.

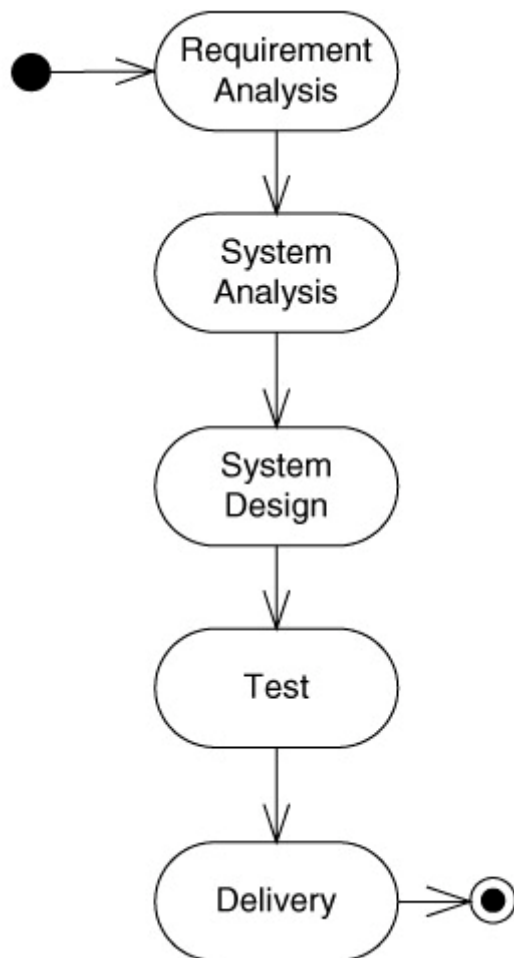


Figure 2.25: A simple software development process without iterations.

Guards can be applied directly to the control flow or in combination with a decision symbol. The *decision symbol* is a hollowed diamond. [Figure 2.26](#) shows an example in which the decision symbol is used to model iterations.

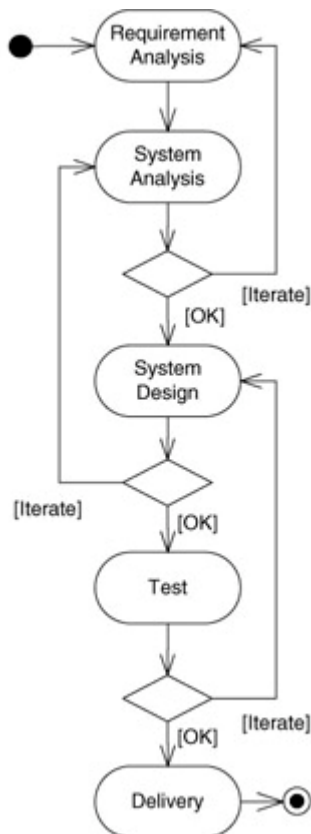


Figure 2.26: An iterative software development process.

Forking and Joining

A control flow can be *forked* or *joined*. Forking a control flow means that the flow of control is split into several flows, each with its own activities. This technique is used when activities are performed in parallel with some form of *synchronization*.

Synchronization means that the control flow is joined, as in [Figure 2.27](#) (which shows a very simplified view of a business process). The synchronization bar, a bold line, is used to show forking (split into several control flows) and joining of several control flows.

When flows are joined, if one flow reaches the synchronization bar first, it will wait for the other flows before continuing.



Figure 2.27: Sales and marketing are done in parallel with the activity series product development, production, and delivery. In the automotive industry this is called concurrent engineering; marketing is done in parallel with sales, product development, and so on.

Object Flow

Object flow is a technique used to capture how objects participate in activities and how they are affected by the activities. [Figure 2.28](#) shows an example in which a production activity (a business process) receives an order object and feedback from manufactured products. The production delivers products that are then evaluated in the product evaluation activity. The product evaluation activity delivers checked products. Object flows are shown with a dashed line ending with an arrow. If the arrow points at an activity, it is an input object; otherwise, it is an output object.

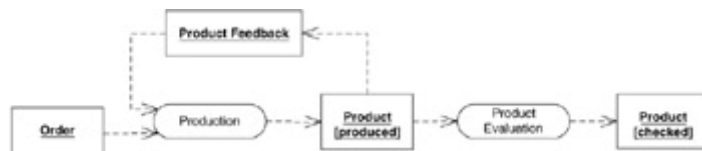


Figure 2.28: An activity diagram with object flow.

Send and Receive

Signals are objects that are explicitly sent or received in activity diagrams. The send symbol corresponds to the send-clause attached to a transition in a statechart diagram. The send symbol is a convex pentagon that looks like a rectangle with a triangular point on one side. The receive symbol is a concave pentagon that looks like a rectangle with a notch in its side. As with the send-clause, both the send and receive symbols are attached to a transition (control flow). The transition is divided graphically into two transitions with a send or a receive symbol in between.

The send and receive symbols are attached to the receiver or sender objects with a dashed line with an arrow from the send or receive symbol to the object. If it is a send symbol, the arrow points to the object. If it is a receive symbol, the arrow points away from the object, to the receive symbol. Showing the objects receiving or sending the signals is optional.

[Figure 2.29](#) shows an example in which signals (i.e., objects) are sent and received. The model indicates that a delivery process begins with the receipt of an inquiry. Once an inquiry is received, the process begins to negotiate with the sender of the inquiry, the customer. After a while, an order is placed and the order is accomplished. While the order is being filled, the estimated time for delivery is sent to the inquiry sender. When the inquiry is accomplished, the delivery is made and the delivery process waits for an acceptance. Finally, it sends an invoice and receives payment. This model shows only the sending and receiving of signals; it is also possible to indicate which objects are the receiver or sender of the signals.



Figure 2.29: A delivery process.

This process can be further detailed; for example, it is possible to add the handling of nonaccepted deliveries. It also might be of interest to see how the delivery process interacts with the order process or to examine how the supplier and the customer interact. As you probably have guessed, the delivery process does not take place only in the supplier's organization; it also takes place in the customer's organization. This is thoroughly examined in [Chapter 3](#), "Modeling the Business Architecture," [Chapter 4](#), "Business Views," and [Chapter 5](#), "Business Rules." The interaction between the customer and the supplier is discussed in the "[Sequence Diagram](#)" section next in this chapter.

Swimlanes

A *swimlane* groups activities with respect to their responsibility. Swimlanes can be used for several different purposes, including to explicitly show where actions are performed (in which object), or to show in which part of an organization activities are performed. Swimlanes are drawn as vertical rectangles in the activity diagram, and the activities that belong to a swimlane are placed within its rectangle. The swimlane is given a name that is placed at the top of the rectangle. In [Figure 2.30](#), swimlanes are used to model organizational units (Product Development, Marketing and Sales, and Manufacturing). The activities are placed in the organizational units in which they are carried out.

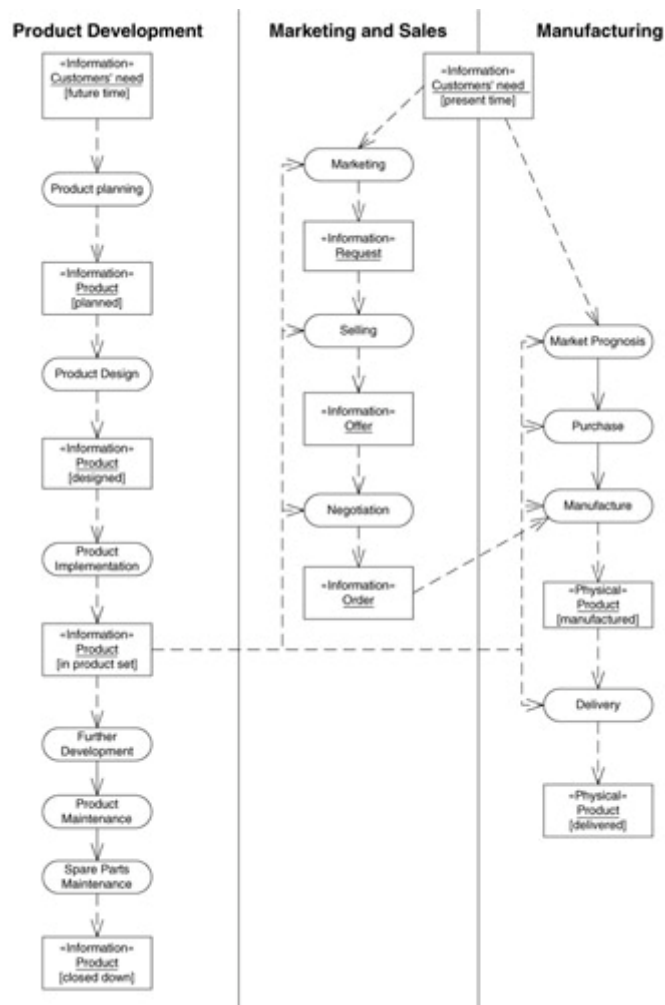


Figure 2.30: A simple business process view with organizational units.

Another technique used in [Figure 2.30](#) is *stereotyping*. Stereotypes enable you to create new building blocks in UML based on existing blocks but specific to the problem at hand. All model elements in UML (classes, objects, states, etc.) can have stereotypes. For example, if it is important to denote that an object is an information object holding only information, we can stereotype the object as “information object”; if it is a physical item, then we can stereotype it as “physical item.” There are no predefined building blocks for information objects and physical objects in UML, but by defining stereotypes for them based on the standard class concept we can adapt and extend UML to support these concepts. In [Figure 2.30](#), the information and physical stereotypes are used. A stereotype name is enclosed in double angle brackets (the guillemet symbol), in the form «stereotype-name». Stereotypes are discussed in more detail later in this chapter and are used in this book to adapt UML for business modeling.

Sequence Diagram

Sequence diagrams are used to explore and visualize the sequence in which objects interact with each other. The objects can be organizational units, companies, computers, people, processes, or mechanical things. Typically, sequence diagrams depict the sequence of messages between a set of objects, where the order and timing of the messages are clearly depicted. [Figure 2.31](#) is one example of such a sequence, the interaction between a customer and a supplier. This figure is based on a repeatable four-phase model for interaction. The first phase, Preparation, consists of two activities: Prepare inquiry and Send inquiry. The Negotiation phase consists of these activities: Prepare offer, Send offer, Prepare counterbid, Send counterbid, Send offer to the customer, Prepare order, Send order, and Obligation. The next phase, Accomplishment, consists of three activities: Confirm, Accomplishment, Notice of delivery, and Delivery. The last phase, Acceptance, comprises these activities: Delivery check, Accept, Invoice,

Send invoice, Prepare payment, and, finally, Pay. The figure shows the exact interaction between the customer and the supplier.

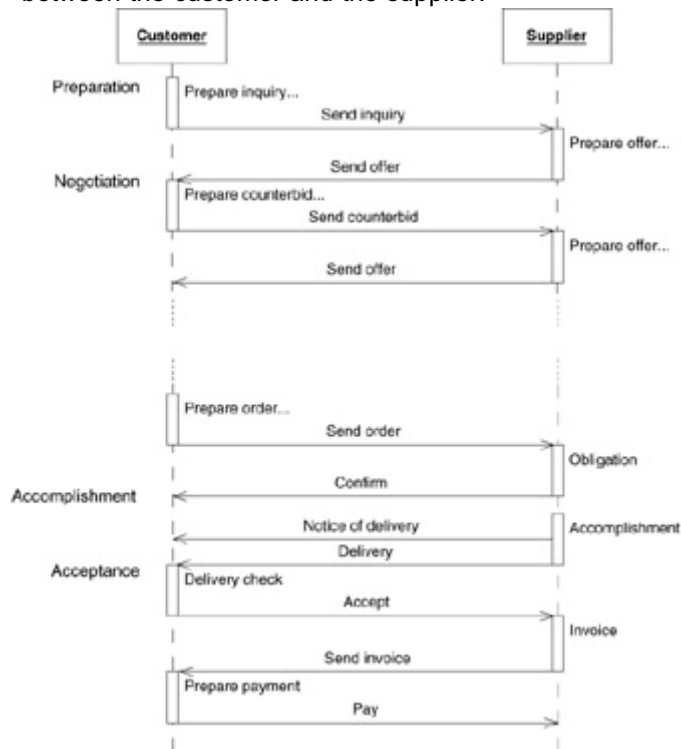


Figure 2.31: The interaction between a customer and a supplier modeled with a sequence diagram.

Notice in [Figure 2.31](#) that participating objects are placed at the top of the model and *lifelines* are drawn vertically from the objects. The lifelines show when objects are created and destroyed. In this figure, all objects exist from the beginning; [Figure 2.32](#) shows how objects are created and destroyed during the execution of the sequence. The creation and destruction is shown by the lifeline in sequence diagrams.

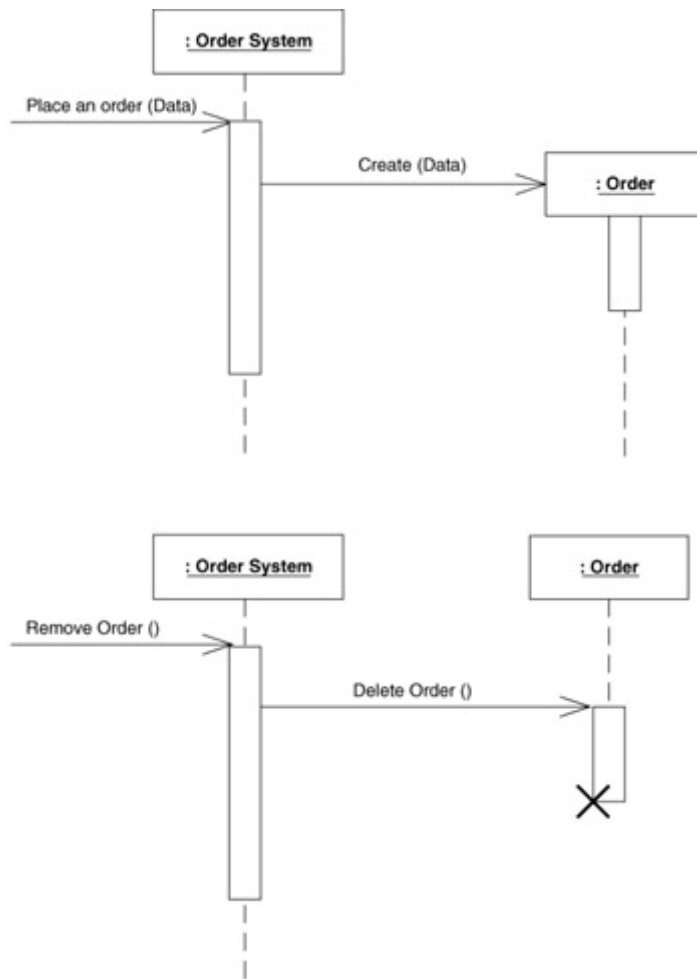


Figure 2.32: Create and destroy object in a sequence diagram.

Activities, such as Prepare Order and Deliver Check, can be shown on the lifelines. The activities are the connection-points to both statechart diagrams and activity diagrams; the objects are the *connection-points* to the object diagrams and the class diagrams. Scripts can be placed on the sides of a sequence diagram, as shown in [Figure 2.31](#) (Preparation, Negotiation, etc.), to clarify the sequences. Messages in a sequence diagram can have parameters, as shown in [Figure 2.32](#), and guards—just like statecharts and activity diagrams.

Collaboration Diagram

Collaboration diagrams focus on how objects collaborate, and express situations similar to those modeled by the sequence diagram. Sequence diagrams handle sequences and simple selections (with the guards) while collaboration diagrams handle iterations. Collaboration diagrams make it easier to express more complex interactions and to show the relationships between the collaborating objects. Sequence diagrams are straightforward and easy to read, but because collaboration diagrams are more powerful, they are less readable. Both are useful but in different situations.

[Figure 2.33](#) is a collaboration diagram that shows how to calculate the value of a portfolio. The messages are numbered, indicating the order in which they are sent, beginning with 1, followed by 1.1, 1.1.1, 1.2, 1.2.1, 1.3, 1.3.1 and so on.



Figure 2.33: Calculating the value of a portfolio.

Figure 2.33 also shows *synchronous messages*, those messages that must be fulfilled before anything else can be accomplished. For example, message 1.1.1 must be sent and accomplished before message 1.2 can be sent.

Iterations are indicated with an asterisk, followed by an iteration clause and the operation that should be called (i.e., the messages that are sent). For example:

* [when there are orders left] get amount ()

Figure 2.34 shows a complex collaboration diagram for a computer-based information system with a graphical user interface. A sales statistics window is shown (message 1), and the window then creates a statistics summary object (1.1), which will collect the statistics to show in the window. When the statistics summary object is created, it will iterate over all salespersons and get the total order sum (1.1.1) and the budget (1.1.2) for each salesperson. Each salesperson object gets its order sum by iterating over all its orders (getting the amount for each (1.1.1.1) and adding them together), and its budget by getting the amount from a budget sales object (1.1.2.1). When the statistics summary object has iterated over all the salespersons, it has been created (the return of message 1.1). The sales statistics window will then get result lines from the statistics summary object (the result line is a formatted string describing the result of one salesperson) and show each line in its window. When all result lines have been read, the show operation on the sales statistics window returns, and the collaboration is finished. At first, the message-sequence numbering scheme can take some time to get used to, but after a while you will find it easy to read.



Figure 2.34: A collaboration diagram showing a collaboration that summarizes sales results.

Use-Case Diagram

Use cases capture the functional requirements on a system. They are defined through plain text. The use-case diagram summarizes which use cases are available and their relationships to each other. Use cases and use-case diagrams are formulated in terms of an *actor* and a *system*. An actor is a role that a user or another system has in relationship to the system. An actor always has some interest in using the functionality the system provides. A use case is one use of the system (information system, mechanical system) by an actor.

[Figure 2.35](#) shows a use-case diagram for a computerized insurance system. The actors using the system are the insurance customer, the insurance salesperson, and the claims adjuster.



Figure 2.35: A use-case model for an insurance information system.

The customer can sign insurance policies, pay insurance fees, notify the insurance company of damage, and retrieve compensation. Notification of damage also involves the claims adjuster actor. The insurance salesperson can view different types of statistics about customer, insurance policies, and damages.

Use cases are described with textual descriptions, such as:

Signing Insurance Item

- The use case is initiated by the insurance customer.
- Information about the customer (name, address, social security number) is entered.
- Type of insurance is entered.
- Information about the insured object is entered. Depending on type of insurance, either Life Insurance Information or Car Insurance Information is included.
- System calculates the monthly insurance fee.
- A preliminary insurance number, fee, and insurance date are returned to the customer.

Use cases can be linked with three types of relationships:

Include. An include relationship between use cases means that one use case can include and use other use cases in an explicitly defined place in its description. An include relationship is shown as a dependency stereotyped as «include» (see [Figure 2.35](#)).

Extend. An extend relationship between use cases means that the base use case (the use case that is extended) is extended with additional behavior by the extending use case. The extensions can be seen as optional functionality added to the base use case, and the base use case does not need to know of the extending use case; it can also be used as-is. An extend relationship is shown as a dependency stereotyped to «extend».

Generalization. Generalization among use cases is the same as generalization among classes. One use case can be specialized to one or several use cases that inherit and add features to it. Generalization is shown with the same symbol as between classes (a line with a triangle).

Component Diagram

Component diagrams are used to structure components in software systems. They examine and manage dependencies between components or between interfaces of components. Components in UML are physical, such as source code files, libraries, dynamic components, or executable programs. [Figure 2.36](#) is an example of a component diagram in which six software components depend upon each other. Business modeling does not utilize this diagram directly; it is a part of UML specially designed for software engineering (which in turn could implement the support for a business model).

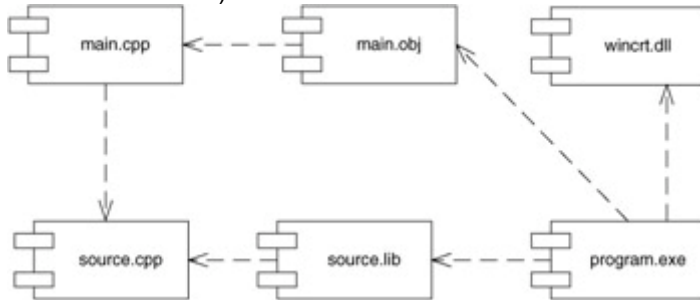


Figure 2.36: A component diagram.

Deployment Diagram

Deployment diagrams are used to explore different processor configurations for implementing a solution. It is a view of the physical hardware in a system. [Figure 2.37](#) is an example of a deployment diagram with three PCs, a server, a printer, an Internet hookup, and a database server. All of these are viewed as *nodes*, visualized by boxes. The diagram can also be detailed to show the allocation of components or processes to nodes (e.g., show which programs or components execute in different nodes). Business modeling does not utilize this diagram; it is a part of UML specially designed to model the physical architecture of computer systems.

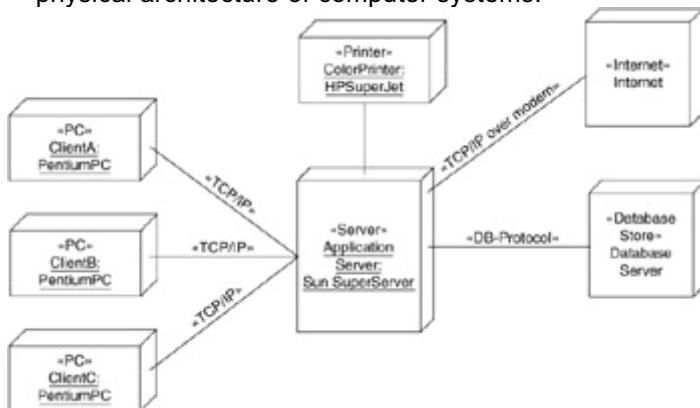


Figure 2.37: A deployment diagram.

Extending UML

Classes, objects, states, and activities are some of the building blocks that form the foundation of UML. However, there may be situations in which you will want to customize these building blocks to suit your needs. UML provides three mechanisms for extending UML to apply to your particular modeling requirements: stereotypes, tagged values, and constraints.

Stereotypes

Recall that stereotyping is a technique used to define new kinds of building blocks in UML based on existing blocks. It allows you to customize an existing UML modeling element to suit your particular modeling needs. For example, activities can be used for many things: to specify operations, to specify business processes, and to specify data

flow. To constrain activities to just business processes, you can define or stereotype this UML building block as a new type of UML modeling element.

Any UML modeling element can be customized; essentially, you can almost define your own modeling language based upon the UML foundation. Although it is not possible to add new kinds of elements, all existing UML elements can be customized, extended, or adapted by defining and naming the stereotypes. The definitions are textual descriptions; furthermore, a stereotype can have a specific graphical icon attached to it, so that the stereotyped element has a symbol. This new symbol can then be used to identify the new modeling element in the diagrams.

There are three methods for showing and modeling stereotypes. The first, already mentioned, is to show the original model element (a class, an object, a state, etc.) with the stereotype name enclosed within guillemets (double angle brackets): «stereotype-name» (see the left-hand example in [Figure 2.38](#)). The second method is to use an optional icon, a symbol that represents the stereotype, inside the original symbol (middle example in [Figure 2.38](#)). The last approach is to show the selected icon only, and treat the stereotype as any predefined building block in UML (right-hand example in [Figure 2.38](#)).



Figure 2.38: Three ways of representing the stereotype «Physical».

In [Figure 2.38](#), the stereotype «Physical» denotes a special kind of a class that is built upon a normal class; its objects are physical things in the real world.

[Figure 2.39](#) shows another example of a stereotype. Here, an activity is stereotyped to a business process. The icon also includes the stereotype string (which is a part of the icon) in order to make interpreting the icon easier (the string tells us what it is about). The name of the activity, in this case the business process, is shown inside the icon. Handling the process stereotype as shown in [Figure 2.39](#) is practical when modeling large-scale models, especially large-scale business models.

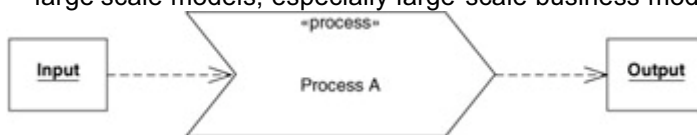


Figure 2.39: A business process, which is a stereotyped activity with object flow (input object and output object).

Tagged Values

All model elements can be extended with a *tagged value* that consists of a *tag* and a *value* (it can be a version tag and a version number (value) for a class, etc.). Another example of a tagged value is the modeler (author) of a process and a value that is a text string containing the name of the modeler. Like stereotyping, this is a technique used to extend UML. All model elements in UML can be extended with tagged values, which are depicted within curly braces—{ }. [Figure 2.40](#) shows a class with a tag for version and a tag for date on the class.

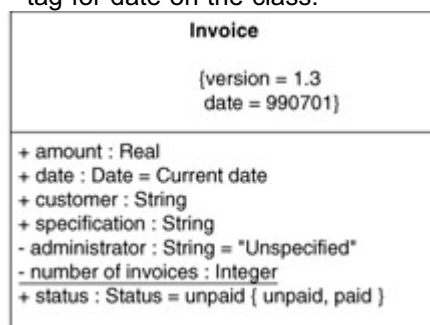


Figure 2.40: A class with two tags: version and date.

Constraints

A *constraint*, a term we've used already, is the third way to extend UML. Constraints are rules applied to UML models; they can be applied to just one or to several model elements. A constraint that is applied on several model elements uses a dashed line that crosses the involved model elements; the constraint itself is written within curly brackets. A constraint that is applied to just one model element is defined close to the element, again within curly brackets. [Figure 2.41](#) shows the two different ways of defining constraints.

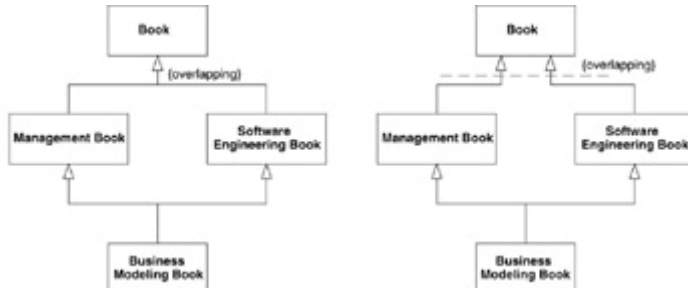


Figure 2.41: Two ways of modeling the same book model.

In addition to using the predefined constraints, such as xor and ordered (discussed earlier in this chapter), you can define your own constraints. [Figure 2.42](#) has an example with the predefined ordered constraint and a customized constraint that specifies that the shipping cost of a delivery clip must not be larger than 100. Constraints are often defined using the Object Constraint Language (OCL), discussed in detail in [Chapter 5](#), "Business Rules." In the context of business modeling, constraints and OCL are used to define business rules for the business.



Figure 2.42: A model with both a predefined and a user-defined constraint.

The three extension mechanisms, stereotypes, tagged values, and constraints, can be used to customize and add new functionality to UML. The Eriksson-Penker Business Extensions presented in the following chapters use these techniques to customize UML for real-life business modeling. After a while, you will learn to benefit from customizing UML on your own, but the best way to begin is to use someone else's extensions in some of your projects. You can then try to adjust them and develop your own extensions to suit your line of business and your modeling requirements. Remember, UML should not dominate or constrain you; rather, you should master UML.

Finally, a note on CASE tools that support UML. Some CASE tools on the market have not yet implemented the use of stereotype icons, but most support stereotype strings, tagged values, and constraints. If the tool you use does not support icons, simply use the string variant of stereotypes instead.

Summary

UML is a modeling language that can be used for both business modeling and software engineering. It is a language that can be extended and customized. The following chapters introduce the Eriksson-Penker Business Extension for UML. These extensions make it possible to use UML even in complex and difficult business modeling, all the way to designed and implemented computer-based systems.

UML has nine predefined diagrams; together, they cover structure, behavior, and functionality.

- *Class diagrams are used to describe structures.* The structures are built up from classes and relationships such as associations, aggregate, and generalization. The classes can represent and structure things like information, products, documents, and organizations.

- *Object diagrams are used to exemplify a class diagram.* These diagrams can show objects, the name of the objects, the object states, and links (instances of relationships) among the objects.
- *Statechart diagrams are used to express which states a class object can have.* They also show what triggers the change from one state to another.
- *Activity diagrams are used to describe activities and actions taking place in a system.* These diagrams are used in this book to describe business processes (i.e., to describe the activities taking place in an organizational system).
- *Sequence diagrams show one or several sequences of messages among a set of objects.*
- *Collaboration diagrams describe a complete collaboration among a set of objects.*
- *Use-case diagrams and their attached use-case text specifications are used to describe system functionality.*
- *Component diagrams are a special case of class diagram used to describe components within a software system.* Component diagrams are not used in business modeling.
- *Deployment diagrams are a special case of class diagram used to describe hardware within a software system.* Deployment diagrams are not used in business modeling.

This chapter did not touch upon patterns in UML. Patterns are described in detail in [Chapter 6](#), “Business Patterns.” The object constraint language, OCL, is discussed further in [Chapter 5](#), “Business Rules.” This chapter also did not address those areas of UML not relevant to business modeling. A more comprehensive description of UML can be found in our previous book, *UML Toolkit* (New York: John Wiley & Sons, Inc., 1998).

The next two chapters provide more detail on how UML can be used and adapted for business modeling. Though the material in these chapters is based on standard UML, the UML extension mechanisms are used to adapt the language to better suit business modeling.

Chapter 3: Modeling the Business Architecture

Overview

The role of architecture in building any type of structure is well defined. A well-designed architecture makes it possible to thoroughly understand the structure being built, to plan the actual construction, and to estimate costs; it serves as the basis for the blueprints of the structure. Once construction has been completed, a good architecture remains as the documentation of the process and the result, making it possible to understand, maintain and, if so desired, to extend the structure.

Likewise, the importance of architecture in constructing information systems has been acknowledged for some time and is currently the focus of much research. An architecture captures the vital parts of a structure in an organized manner and is a practical tool for managing a complex system, such as a software system or a business. Even though the architecture form of a business is different from that of construction projects, it is an equally important concept. Architecture defines the business structure, so modeling this architecture is key to understanding the business and how it functions.

This chapter discusses the characteristics of a business architecture, the concepts that are used in defining such an architecture, and the set of extensions to UML that will be used in this book to model a business architecture effectively. These extensions, called Eriksson-Penker Business Extensions, have been defined using the built-in extension mechanisms available in UML.

Architecting a Business

Business architecture is the basis for describing and understanding an enterprise: It lists the required parts of a business, how the parts are structured and interact, and how the architecture should evolve. Although it is difficult to clearly define the term, a working definition of business architecture is:

...an organized set of elements with clear relationships to one another, which together form a whole defined by its functionalityThe elements represent the organizational and behavioral structure of a business system, and show abstractions of the key processes and structures in the business. [Vernadat 96]^[1]

All businesses have some sort of architecture. But because an organizational chart is usually the only description available of the business, many of the situations and structures in it have never been documented or visualized. It is thought-provoking to realize that companies have many drawings for their buildings and/or their products, but none for how their business is conducted.

By defining and documenting how business is conducted, one can capitalize on the business knowledge that is already available. The architecture acts as a knowledge base; it is a strategic asset for the business. Documenting the business system makes it easier to make improvements or innovations to the business, and to identify new business opportunities. Today, with such heavy reliance on information system technology, a business model can also provide the correct requirements of the information system, so that the system best supports the operation of the business. This book treats a business as a type of system. It is important not to confuse this general term with the more specific term *information system*. There are many types of systems, such as systems in nature and different constructed systems such as a business or a machine. An information system is only one type of system.

Good Architecture

A good architecture allows the modeler to abstract the business into different aspects or views and to concentrate on only one aspect at a time. Achieving abstraction, suppressing the details and irrelevant information, is essential to understanding complex systems and relationships.

A good architecture has the following characteristics:

- *Captures the real business as truthfully and correctly as possible.* It defines an architecture that is realistic and feasible to implement and that achieves the goals of the business.
- *Focuses on the key processes and structures of the business at an appropriate level of abstraction.* The appropriate level is different from case to case and depends on the purpose of the architecture.
- *Represents a consensus view among the people operating in the business.* For example, both managers and workers agree that the architecture correctly describes how the business operates.
- *Adapts easily to change and extensions.*
- *Is easy to understand and fosters communication among the different stakeholders of the business.* An architecture is useful only if it can be understood by its users.

Achieving these characteristics is not simple. It requires several things:

- Modelers that have a high knowledge of the business, or at least access to people with such knowledge who can be interviewed and can participate in the construction of the architecture.
- A modeling language that can capture all the important concepts in the business, along with the relationships between these captures. Both static

and dynamic structures must be captured. The language must be simple enough to be understood by many different people, without losing accuracy or power. The language must also be scalable so that things can be described at different levels.

- A capability to organize visual diagrams into different views of the business, where each view illustrates a specific aspect of the business. The complete description of the business can't be defined in a single view.
- A design based on experience, on what works and what does not. If possible, well-defined modeling patterns that have proved to work should be used.
- A development process that ensures the quality and accuracy of the models produced.

Many of these requirements and how to achieve them will be discussed in this book.

Obviously, we need a good technique or language to define the architecture of a business. Although there are several reference architectures (ISO Reference Model, CIMOSA, PERA, etc.) that use different techniques to describe a business, a common factor runs through all of them: the use of models.

If the models are to be effective, they must be expressed in a common language. This book uses process and object-oriented modeling with the Unified Modeling Language (UML) to build business architectures. As stated, UML has already been established as the standard modeling language for modeling information systems. This book shows that it also has the capabilities to express business models. With the techniques presented in the book, a business architecture is based on four views of the business, each of which consists of a number of diagrams that contain the common elements and concepts (e.g., objects) of the business. The views and diagram types show the structure of and interactions between the different aspects of the business. The contents of the views and their internal relationships are defined in [Chapter 4](#), "Business Views."

^[1][Vernadat 96] Vernadat, Francois. *Enterprise Modeling and Integration*. London, England: Chapman & Hall 1996.

Business Concepts

A business, an *enterprise*, is a complex system that has a specific purpose or goal. All the functions of the business interact to achieve this goal. The business system also can be interlinked with and affected by the decisions and events that take place in other systems, and so can't be analyzed in isolation. Because of this, defining the boundaries of the business can be difficult. The resources within the organization can also have separate goals that do not always reflect those of the business.

Many of the important elements in a business, such as customers, suppliers, laws, and regulations, are external to the business and are not defined within the business itself. Thus, the business system is an *open system* whose objects and parts are often also parts of other business systems. As such, it cannot be viewed as a *black box* system, which is analyzed by looking only at the input to and output from the system, but as a system whose parts are visible, as shown in [Figure 3.1](#).

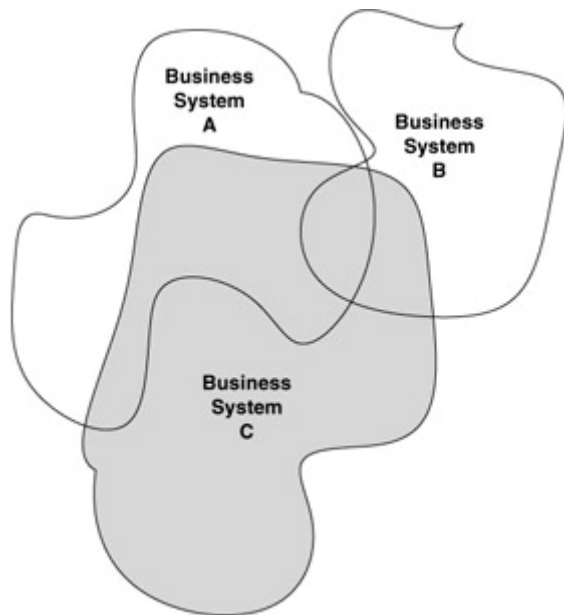


Figure 3.1: A business system is interlinked with other business systems.

Individual businesses have different goals and internal structures, but they use similar concepts to describe their structure and operation: the objects that are part of the system, their relationships and structure, and their dynamic interaction with each other in various situations. A model of the business system describes these concepts. The concepts used to define the business system are:

Resources. The objects within the business, such as people, material, information, and products, that are used or produced in the business. The resources are arranged in structures and have relationships with each other. Resources are manipulated (used, consumed, refined, or produced) through processes. Resources can be categorized as physical, abstract, and informational.

Processes. The activities performed within the business during which the state of business resources changes. Processes describe how the work is done within the business; they are governed by rules.

Goals. The purpose of the business, or the outcome the business as a whole is trying to achieve. Goals can be broken down into subgoals and allocated to individual parts of the business, such as processes or objects. Goals express the desired states of resources and are achieved by processes. Goals can be expressed as one or more rules.

Rules. Statements that define or constrain some aspect of the business, and represent business knowledge. Rules govern how the business should be run (i.e., how the processes should execute) or how resources may be structured and related to each other. Rules can be enforced on the business from the outside by regulations or laws, or they can be defined within the business to achieve the goals of the business. Rules can be categorized as functional, behavioral, and structural.

All of these concepts are related to each other: A rule can affect the way some resources are structured; a resource is allocated to a specific process; a goal is associated with the execution of a specific process. The goal of business modeling is to define these concepts and show the relationships and interactions among them.

A *meta-model* is a model of the basic business concepts and their relationships. The concepts depicted in this model are used to create other models. [Figure 3.2](#) is a meta-model that summarizes the concepts used in business modeling and their relationships to each other. It will be used for the business models developed in the rest of the chapters in this book. This meta-model is a UML class diagram in which each concept is depicted as a class, and the relationships between the concepts are either an association or a specialization. The meta-model also indicates which factors would hinder or prevent the business from achieving its goals.

stereotypes used for each of these business concepts are explored in the discussions that follow.

For the reader who regularly uses UML to describe the code structure of a program, remember that this book uses UML to create high-level business models; the concepts described here are *not* code specifications and should *not* be translated, even mentally, to lines of programming code. Information systems will be designed to support these business models at a later point in the development process, but that step is a long way from the business modeling stage. Attempting to capture reality and business thinking and directly translating it into the syntax of a programming language is very dangerous and subverts the focus of business modeling. Translating the completed business models to information systems is discussed in [Chapter 10](#), “From a Business Architecture to a Software Architecture.”

Eriksson-Penker Business Extensions

Keep in mind that UML was defined to model the architecture of software systems, and though there are similarities between software and business systems, there are also some differences. Business systems have many concepts that are never intended or suitable to execute in a program, such as the people working in the business, manufacturing production equipment, and rules and goals that drive the business processes. Because UML was initially designed to describe aspects of a software system, it had to be extended to more clearly identify and visualize the important concepts of processes, goals, resources, and rules of a business system. To address this issue, we have created a set of extensions based on the existing model elements of UML. We introduced these extensions briefly earlier; they are called, simply, the Eriksson-Penker Business Extensions and provide symbols for modeling the processes, resources, rules, and goals of a business system.

The standard extension mechanisms in UML allow you to adapt UML to accommodate new concepts. The extension mechanisms are:

Stereotype. An extension of the vocabulary of the UML, which allows you to create new building blocks specific to your problem from existing ones. [Booch 98]^[2] Stereotypes may have their own icons.

Tagged value (property). An extension of the properties of a UML element, which allows you to create new information in that element's specification. [Booch 98]^[2]

Constraint. An extension of the semantics of a UML element that enables you to add new rules or to modify existing ones. [Booch 98]^[2]

The UML specifications contain a five-page document called the “UML Extension for Business Modeling” that describes a set of extensions for business modeling; however, it offers no detailed explanation of how to apply them. It only briefly specifies the extensions with short text descriptions and samples of their icons. These extensions are based heavily on applying use-case modeling to describe business processes and are used as part of the Rational Unified Process, a process product sold by Rational Software Corporation.

Use cases are a common technique for capturing the functional requirements of an information system. They describe the functions provided by a black box system, which is further used to drive the design of the software inside the system. However, viewing a business system as a black box doesn't offer sufficient insight into how the existing or future processes operate. An information system can be characterized by its interactions with a user or a customer, but implementing business process improvement or innovation requires looking at the existing processes and their operation, not just the interactions with a user. Defining the user and the system boundaries is also typically much more difficult since processes from many businesses interact; and if the boundaries are difficult to define, the interactions between the user and the system will also be difficult to capture. Again, a business system is, and therefore must be viewed as, an open system. The fact that a business interacts with its environment is important; describing that alone isn't enough to do business modeling.

The Eriksson-Penker Business Extensions form a basic framework for business extensions to UML to which a business architect can add stereotypes or properties suitable to his or her line of business. Thus, the extensions should not be viewed as a definitive set of business extensions; they are intended to serve as a base upon which further development or adaptations can be made for specific modeling situations or as the use of UML for business modeling advances. The extensions merge UML with process modeling, so that it is easier to use UML for business modeling. Anyone familiar with UML should be comfortable with these extensions, as will those who are knowledgeable in business process modeling. To that end, the extensions have been defined using the standard mechanisms for extensions in UML, and are fully compliant with standard UML. We repeat, they are not in any way an attempt to modify or specify a new modeling language. This means that these extensions can be quickly adopted by those already proficient in UML and can be incorporated into an UML tool that supports the full language (including support for the extension mechanisms) without difficulty. Currently, the Eriksson-Penker Business Extensions are supported in the Qualiware CASE tool; more tool support is expected to become available.

The Eriksson-Penker Business Extensions, described with examples in more detail in the [next chapter](#) and summarized in a reference manual in [Appendix B](#), have been designed with simplicity as the highest priority. New icons, although possible in UML, have been avoided in favor of textual stereotypes, extant UML concepts, and symbols. Some adaptations, such as defining a process icon for a UML activity and a special icon for highlighting the information resource objects, have been made to ensure that an experienced process modeler is comfortable using UML for process modeling. Of particular note is the unique use of the activity diagram in the extensions, renamed as *assembly line diagram* when used in this manner. An assembly line diagram is a process model shown in an activity diagram where links are made to objects used during the process. The objects could be information objects in an information system or other resource objects of interest to the process.

Business Processes

The business processes are the active part of the business. They describe the functions of the business, and involve resources that are used, transformed, or produced. A business process is an abstraction that shows the cooperation between resources and the transformation of resources in the business. It emphasizes *how* work is performed, rather than describing the products or services that *result* from the process. A more formal definition of a business process is the following:

A business process is a collection of activities that takes one or more kinds of input and creates an output that is of value to the customer. A business process has a goal and is affected by events occurring in the external world or in other processes. [Hammer 94]^[3]

Another, more elaborate, definition is this:

A process is simply a structured set of activities designed to produce a specified output for a particular customer or market. It implies a strong emphasis on how work is done within an organization, in contrast to a product's focus on what. A process is thus a specific ordering of work activities across time and place, with a beginning, an end, and clearly identified inputs and outputs: a structure for action. [Davenport 92]^[4]

Michael Porter states that the basis for every enterprise is the process of taking some source materials and manipulating them such that value is added to those materials [1985]. The process is a chain of smaller value-added steps (a value chain). The value of a product is decided by those who consume or use the product, demonstrated by what they are willing to exchange for that product (e.g., money). Thus Porter defines that every enterprise is made up of a number of business processes, each producing value to a customer.

To summarize the similarities between these different definitions, a business process:

- Has a goal.
- Has specific input.

- Has specific output.
- Uses resources.
- Has a number of activities that are performed in some order, depending on conditions and events that occur during the execution of the process. The activities within the process can be seen as subprocesses.
- Affects more than one organizational unit. It is horizontal rather than vertical in regard to the traditional organization of the business.
- Creates value to some kind of customer. The customer can be either internal or external to the business.

A business process has an explicit goal, a set of *input objects* and a set of *output objects*. The input objects are resources that are transformed or consumed as part of the process, such as raw material in a manufacturing process. The input objects also can be refined by the process, in which case the process adds value to them, so that the value of the output of the process is larger than the input. The output objects represent the accomplishment of the goals and are the primary result of the process, such as a finished product in a manufacturing process. The output object is also a resource. An output object can be a completely new object created during the processes or it can be a transformed input object. The transformations made by the process can be physical, logical, transactional, or informational.

During its execution, the process interacts with other *resource objects*, objects other than the input and output objects, that are just as vital. These objects carry information required by the process or they are resources responsible for executing the activities in the process, such as people or machines. In a manufacturing process, it is the people who operate the machines that transform the raw material into a finished product.

Manipulating resources in a process in these ways is not limited to manufacturing processes; it can also be used to describe any type of process, such as a service process or document or transaction processing.

The Eriksson-Penker Business Extensions represent a process in a UML class diagram with the process symbol shown in [Figure 3.3](#). This symbol is used in many existing process modeling techniques (e.g., it is used in all process modeling at the telecommunications company Ericsson), and it doesn't overlap with any existing symbol in UML. In formal UML, the symbol is a *stereotyped activity* (an action state) from an activity diagram; it is also the UML activity diagram that is used to illustrate business processes. A process—that is, an activity stereotyped to process—takes input resources from its left-hand side and indicates its output resources on its right-hand side. The goal of the process is either illustrated as a goal object above the process symbol or, more informally, specified in the tagged value Goal of the process (the tagged value is not visible in the diagram but is available in a tool). Resource objects used as part of the process are shown below the process symbol. A special variant of an activity diagram, the aforementioned assembly line diagram, can be introduced to show how a process interacts with a specific resource (e.g., an information system). The assembly line diagram illustrates resource objects that are affected by the process in one or more assembly lines below the process. The objects in the assembly line are typically information objects from which information is read or written during the execution of the process. Assembly line diagrams are shown and discussed in more detail in [Chapter 4](#), “Business Views.”

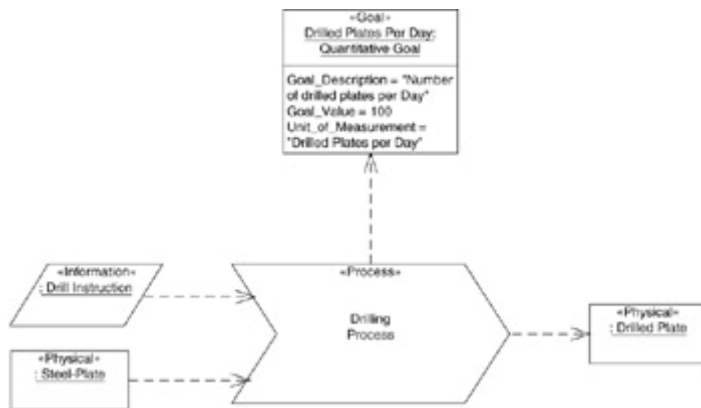


Figure 3.3: Business process symbol, illustrating a goal and the input and output objects.

A process can have tagged values attached to it that contain additional information about the process. The tagged values are not visually represented in the diagram but are a function accessible by double clicking on the symbol in a modeling tool. Although defining all the values of these tags is optional, the goal and the purpose of a process must always be defined. The UML tagged values attached to a process in the Eriksson-Penker Business Extensions are:

Goal. A textual value that describes the goal of the process if a goal object is not explicitly attached to it.

Purpose. A textual value that informally describes the purpose of the process; for example, what the process does and, in the case of a new process, its anticipated effect.

Documentation. A textual value that informally describes the work of the process; for example, the activities completed and the resources involved.

Process owner. A textual value that defines the process owner, the person in the organization who has the overall responsibility for this process and who manages the changes and plans for changes.

Process actors. A textual value that defines actors needed to run the process. Typically, their skill levels are described.

Priority. A textual value that describes the priority of this process; for example, whether it's a core process, a support process, an administrative process, and so on.

Risks. A textual value that describes the risk of the process; for example, what can go wrong either when executing this process or when implementing this process into the business.

Possibilities. Textual values that describe the potential of this process; for example, the opportunities for improving or using this process in the future.

Time. A numerical value that approximates the execution time of the process.

Cost. A numerical value that approximates the cost of executing the process.

The process concept is the center around which business modeling is performed. It is the execution of the process that transforms and refines resources, and in doing so creates value; it is this creation of value that fulfills the goals of the business. The process is also responsible for coordinating and controlling the activities that are performed by the resources that act within the business (e.g., people or machines). A concept called *the Business System Diamond* proposed by Michael Beedle states that the definition of business processes leads to the definition of jobs and structures, which in turn requires management and measurement systems, which in turn reinforce a set of values and beliefs. [Beedle 97]^[5]

Process Steps

The process contains a number of steps or activities that are performed as part of the process. Each of these activities can be considered a process of its own and as a subprocess to the containing process. The activities are categorized as follows:

Direct. An activity directly involved in creating the product or the service; that is, the value created by the process.

Indirect. An activity that supports the direct activities; includes maintenance, administration, planning activities.

Quality assurance. An activity that ensures the quality of the other activities; for example, inspections, controls, or reviews.

These categories are not explicitly used during the modeling process.

The processes can also be categorized in other ways, such as development, improvement, or management processes. Processes of different categories typically interact or control each other, as will be illustrated in [Chapter 9](#), “Process Patterns.”

The process steps are illustrated as nested activities of the process. Recall that an activity in a UML activity diagram can refer to another nested activity diagram that contains other, more detailed, activity descriptions. The process steps can either be viewed by expanding a process symbol in a tool to view the nested activity diagram or by directly drawing and viewing the nested activities inside the process symbol.

A subprocess is drawn using the same symbol as for the enclosing process, shown in [Figure 3.4](#). However, if a process does not contain any subprocesses, the standard activity symbol in UML is used. The activity symbol then represents an atomic process, a process that can't further be broken down into more detailed steps. [Figure 3.4](#) shows a process that contains three subprocesses, two of which are atomic processes (activities) and one that itself contains two atomic subprocesses.

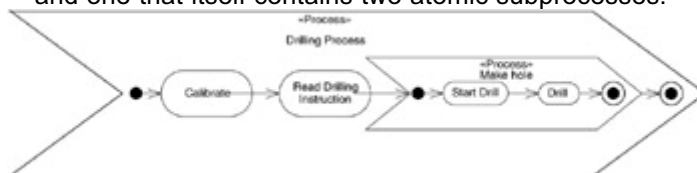


Figure 3.4: A process that contains three subprocesses, consisting of two atomic processes (activities) and one subprocess with yet another two atomic processes (activities).

Processes can span several organizational borders (e.g., departments or divisions in a business), as shown in [Figure 3.5](#). Therefore, a process is broken down into subprocesses independent of the organization and its structure, and shouldn't be divided just because it crosses an organizational border.



Figure 3.5: Processes span organizational borders.

The number of levels between an initial process and an atomic process is arbitrary.

[Figure 3.6](#) shows a process with two layers. In some business process methods, the number of levels has been predefined in the method, and each level has a specific name (process, activity, action, task, etc.). This is an artificial and inadequate construction, since different processes have a different number of levels, and a process on a specific level is a process in its own right. The only exception made in these extensions is the atomic process, activity that visually shows that this step has not been broken down into subprocesses. Note that in practice, processes are intertwined and are rarely sequential as in the example shown in [Figure 3.6](#).

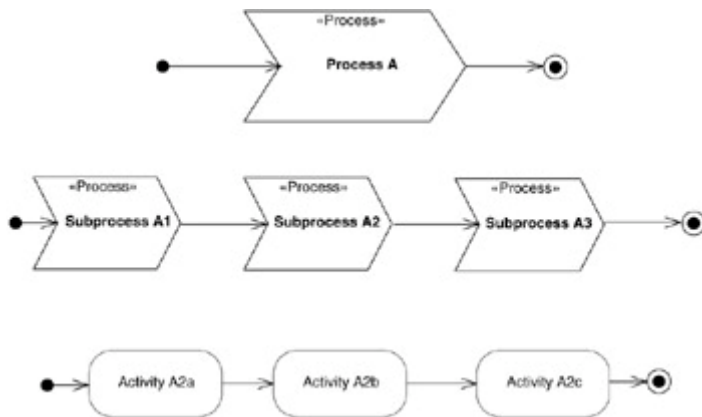


Figure 3.6: A process can be described by subprocesses and finally activities (i.e., atomic processes).

A process may be required to hold information about itself, such as how far along in execution it has gone or what accumulated results have been gathered as part of the process. This holding of information is best illustrated by using the stereotype «Information» on one or more resource objects connected to the process. A resource object stereotyped with «Information» is an information object that represents part of the state of the process. Further, a process support object or a business process object is used to implement the handling of a process and its state in an information system. Remember, though, that many parts of the process (people, machines, human judgment, etc.) can't be computerized and put into the information system; the objects in the information systems are only representations (i.e., abstractions) of the real process. It is rare that the entire process can be implemented into the information system; typically, some supporting parts of the process are implemented.

Business Events

A process is affected by events occurring in the surrounding environment or generated by other processes that cause a process to be activated. Business events are triggers that initiate activities or that control which activities are performed. A business event can be defined as follows:

- A business event is an external real-world happening that requires certain action. [Gale 96]^[6]
- A business event represents a record of a change in the business at a particular instant in time. [OMG 98]^[7]

Several events can occur during the operation of a process to which the process must react, such as a cancelation of an order by a customer, a delivery of material, or the misplacement of a specific resource. A process can also generate events to other processes within the business or in other businesses that will cause these other processes to react in a specialized manner.

An event can:

- Initiate the execution of a process.
- Affect the behavior and execution of the process.
- Conclude a process by generating an event.

Whether the event is generated from the external world or by another process is usually not important, since in a process-oriented view the external world is also defined in terms of processes that generate the events. Some of the most interesting events will come from the external world of the business (i.e., from the processes in that world), such as events generated by customers (orders, payments, complaints), subcontractors (deliveries, invoices), or competitors (product announcements, price cuts).

In the Eriksson-Penker business notation, a business event is represented as a class (the event type) and objects (instances of the event type). The event classes are stereotyped as a business event. Their relationships (i.e., generalization hierarchies) can be illustrated in a class diagram, as shown in [Figure 3.7](#).

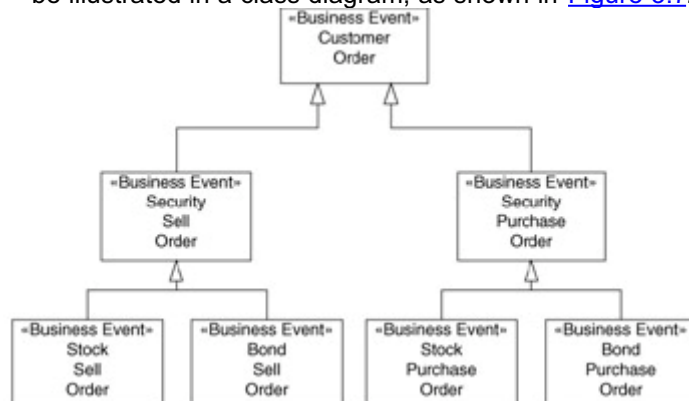


Figure 3.7: Business events are modeled as classes in a class hierarchy.

The receive symbol (concave pentagon) and the send symbol (convex pentagon) are used in a process diagram to illustrate receiving and sending events, respectively, as shown in [Figure 3.8](#). The receive symbol can be read so that the process waits for this event to occur, or, in case of a decision point with several alternatives, the events that will trigger each of the specified alternatives. The send symbol indicates that the process generates and sends the event between two activities. Either of the event symbols (send or receive) can be attached to an object with a dependency arrow, which shows from which object the event is sent or received. If an event symbol is drawn to a process symbol, an expansion of the process symbol shows exactly how and where in the process the event is received or sent. An event symbol can also be attached to its opposite symbol in another process, (i.e., a send symbol to a receive symbol, or vice versa) to indicate how the processes communicate.



Figure 3.8: Receiving and sending business events during a process.

Modularizing the Business System

A large system will have many processes. To handle all the processes, they must be grouped, or *modularized* so that a modeler can concentrate on only one set of processes at a time. Processes are modularized using the UML *package mechanism*. Processes are allocated to a specific package, sometimes referred to as a subsystem in information systems (see [Figure 3.9](#)), each of which contains a set of processes and their subprocesses. A package can in turn also contain other packages.

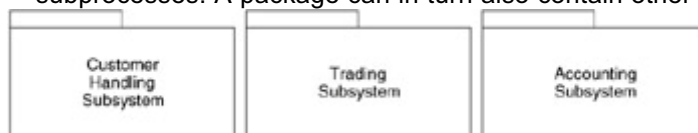


Figure 3.9: Packaging processes together.

A package can contain processes that are not directly linked to each other (e.g., by being subprocesses to each other), but are related in some way in the modeler's judgment. This grouping can be based on similar or related functions, shared structure or organization, or related locale. The exact basis for the grouping is up to the modeler, and so cannot be standardized.

Resources

Resources are the objects that act or are used in the business. They are the concepts consumed, produced, transformed, or used by the business processes. Examples include material, energy, products, people, information, and services. Most of the elements normally modeled as a class in object-oriented modeling are resources in business modeling terms. Vernadat proposes a definition of a resource:

A resource is an entity which can play a role in the realization of a certain class of tasks [1996].^[6]

Another definition is:

A resource is a concept used in the business, and represents anything that we choose to evaluate as a whole. [Darnton 97]^[8]

Resource types are represented as classes. Resource instances are represented as objects. The Eriksson-Penker Business Extensions define some stereotypes to indicate different categories of resource types. A meta-model of these resource types categories is shown in [Figure 3.10](#), in which each class illustrates a specific category of resource types.

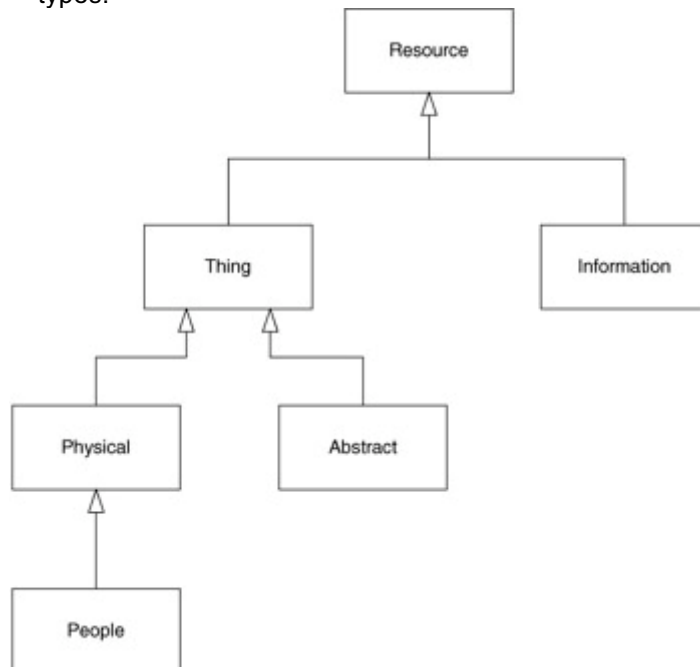


Figure 3.10: A meta-model showing a hierarchy of the different resource types.

Thornton Gale and James Eldred defines four concrete resource types [1996]:

Physical. An entity with material reality that occupies a volume of space. It is something that can be seen and touched upon. Commodities, raw materials, parts, or products used in a process are examples of physical resources. A physical object is often constructed from other physical objects.

Abstract. An idea or concept, often a composite of other objects (e.g., a purchase order is a concept relating to a collection of things). Involves things and concepts that aren't physical and cannot be touched upon but are of importance to the business. Contracts, roles, accounts, and energy are examples of abstract resources.

Information object. A representation of a concept, thing, or another information object. It holds information about other resources and works as a surrogate for the resource, for example, in an information system (though not all information objects must be placed in an information system). It is important to separate the information object from the concept or thing it represents; the information object holds facts or knowledge about other objects in the business. An information object can hold information about a bank account, a product, or a contract (i.e., it represents the account or the contract in the

information system, but it still isn't the same object as the actual account or the contractual agreement).

People. A human being acting in the process. It's a specialization to the physical resource to emphasize and identify the people in the process (people are, of course, a very important part of the processes). People don't like viewing themselves as physical objects, so they have been given their own class in the meta-model. People are also sometimes more unpredictable than machines (which can be both positive and negative), so they should be highlighted in the process. People could be viewed as a special case of a physical resource and as such are also a subtype to Physical in [Figure 3.10](#).

These four resource types are also the stereotypes used to categorize the resource classes. There may well be room for more stereotypes, such as Machine for a machine or Document to represent a document as part of the business (both subclasses to Physical in the meta-model). To keep the size of the business extensions down, such stereotypes have not been defined. However, it may be appropriate to add stereotypes to a specific domain or business.

[Figure 3.11](#) shows some examples of the different types of resources, where the classes are stereotyped to their respective categories. Note that the information stereotype has a special kind of icon. The icon is a slanted class rectangle. Typically, there are well-defined structures and relationships among resources. A product that has been produced in the business is an aggregate of some parts (i.e., a physical resource is built from other physical resources); the people in the business are organized in a functional organization, such as departments (i.e., a people resource is part of an abstract resource such as an organizational unit), and there are relationships handled by the business between customers and abstract objects, such as orders and contracts.



Figure 3.11: Different types of resources stereotyped to their respective categories.

Goals

A goal describes the desired state of one or more resources. Goals are attached to the entire business and to individual business processes. Goals motivate activities leading to state changes in a desired direction. A goal must be measurable in order to track progress. The measure may be defined in quantifiable terms such as profit, volume, time, or quality, or it may be defined in qualitative terms, such as "our company will be regarded as one of the most highly respected firms in this town," or "our company will attract more competent labor." Putting a measure on the goal will make it easier to determine whether it has been achieved. Goals are stated in terms that indicate optimization, such as increase the result, increase the quality, or decrease the time-to-market. The sum of all the efforts of the processes, resources, and rules should achieve the goals.

The business can have an overall vision or goal, such as "we will be the market leader in five years," as well as very specific goals, such as "we will have sales for more than 10 million in Southeast Asia within 12 months." The deeper into the goal hierarchy one gets, the more specific and quantified the goals become. Thus, the vision is typically more loosely specified. The more specific and quantified a goal is, the easier it will be to determine later if the goal has been achieved.

A goal can be broken down into (or composed of) subgoals. Achieving the superior goal is dependent upon achieving the subgoals. Subgoals can also substitute or compensate for other subgoals that have failed or have not been achieved. For example, if the superior goal is "Increase profits by 15 percent," this goal will be achieved if two out of three subgoals make such an increase and the third subgoal makes no profit at all. The two subgoals that were successful compensate for the failed goal.

Dividing goals into subgoals is called *goal modeling*. The basis upon which the goals are divided is up to the modeler. For example, the goal for Southeast Asia can be divided

into subgoals for different countries within the region or into subgoals for different products within the region (or into both of these hierarchies).

Goals are closely related to problems, because a problem is an obstacle to a goal. It is a situation or object that stands in the way of achieving the goal. Therefore, it is common to model the attached problems when modeling the goals. By using both the concepts of goals and problems, it becomes easier to accurately describe them. For example, when presented with a goal, one can ask: “What are the problems or obstacles that stand in the way of achieving this?” On the other hand, when presented with a problem, one can ask: “If this is eliminated, what goal has been achieved?” By defining the problems, subgoals can be identified that will eliminate them. The subgoals can then be allocated to business processes in order to eliminate the obstacle.

Representing goals and problems in UML is not simple. The solution used in the Eriksson-Penker Business Extensions is to represent goals as objects, and to use an object diagram to show the dependencies between goals and subgoals. An object diagram is a UML diagram that illustrates object instances. A goal object is indicated as a goal object of a goal class stereotyped to Goal. In the Eriksson-Penker Business Extensions, there are two predefined goal classes: Qualitative Goal and Quantitative Goal, both of the stereotype Goal. A quantitative goal can be described with a target value in a specific unit of a measurement (a quantity), while a qualitative goal is described more loosely, without a specific target value (a quality). There must be a process to determine whether a goal has been fulfilled. However, when using qualitative goals, that process could rely on human judgment rather than a specific value. Objects shown in the goal object diagram are instances of either of these classes, and dependencies between the objects show how they depend on each other.

Problems are shown as notes stereotyped to Problem, and described in plain text. The note is attached to the goal to which the problem relates. In [Figure 3.12](#), the problem “Understaffed Sales Force” indicates a further subgoal, “Increase Sales Force.” When the dependencies between a goal and its subgoals are constrained to be complete, the fulfillment of all of the subgoals will guarantee fulfillment of the goal. If they are constrained to be incomplete, the fulfillment of all subgoals will not guarantee fulfillment of the superior goal.

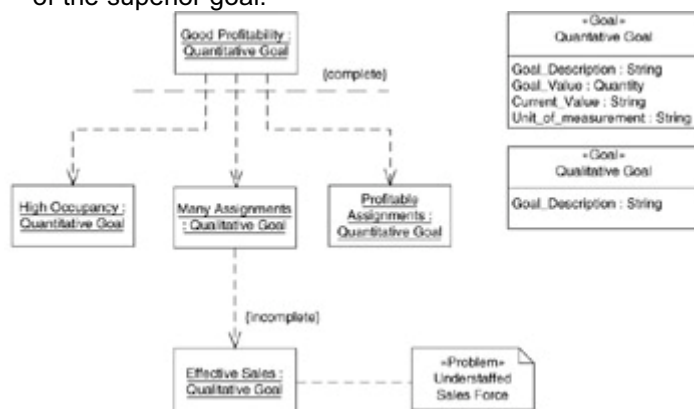


Figure 3.12: Goals, subgoals, and problems in an object diagram.

Because it is a vital part of the business vision view, goal/problem modeling and the syntax for it will be discussed in further detail in [Chapter 4](#), “Business Views.”

Business Rules

A business model contains business rules that define constraints, conditions, and policies for how the business processes are to be performed. Business rules can affect all the other concepts; they can constrain the execution of a business process, the behavior of a resource, and the means to achieve a specific goal. Often a rule concerns the relationships among concepts, defining the way the concepts can be related to each other and which conditions must hold in order for a relationship to be valid at a specific point in time. A practical definition of a business rule is:

... a statement that can control or affect both the execution of a business process as well as the structure of the resources in the business. The statement specifies a condition that must be upheld, or a condition that controls which activity should follow next. It can express a business goal, specify the way a process should execute, detail the conditions of a relationship, or constrain the behavior of a resource.

Rules control the business. They are defined either to satisfy external requirements on the system (regulations and laws, or restrictions imposed by other businesses) or internally to safely achieve the goals of the business.

There are three types of business rules:

Derivations. Rules that define how knowledge in one form may be transformed into other knowledge (i.e., how some information is derived from other information). A derivation can be either a computation rule, such as a formula for calculating a value, or an inference rule, meaning that if a certain fact is true, then another inference fact must also be true.

Constraints. Rules that constrain either the possible structure or behavior of objects or processes; for example, the way objects are related to each other or the way object or process state changes occur. The constraints uphold the integrity of objects as they are created and relationships are changed. An operation of an object can be constrained through the use of operation preconditions and postconditions. A precondition must be fulfilled before the operation is executed; a postcondition must be fulfilled after the operation has been executed.

Existence. Rules that define when something may exist and when it should come into existence, that is, when an object is created or destroyed.

Rules can be specified formally in a particular language (that could be executable) or informally in plain English. Sometimes a rule needs to be specified in both ways, using the formal specification for the information systems and the informal specification for the people in the business. Rules complement the diagrams; together they contain all the information about how the business should be run. Rules are used in all of the views, and can relate to other rules; therefore, care should be taken that the rules in each view do not conflict with each other.

Many of the UML diagrams have built-in support for defining rules, using the generic construct of defining a constraint. A class diagram has structural constraints in its relationships (e.g., the multiplicity of an association). A state diagram has behavior constraints in its state and state transitions (e.g., actions to perform when a transition occurs). Derivation rules can be defined as a computational constraint in a UML diagram (e.g., that the value of an attribute is calculated from the value of another attribute). An activity diagram has behavioral constraints in its activity flow (e.g., which activity must take place before another or which condition must be true before an activity is carried out). As noted in [Chapter 2](#), a constraint in UML is expressed within curly braces placed close to the model element that is being constrained (e.g., close to an association that is being constrained). UML has a recommended formal language for specifying constraints, the Object Constraint Language (OCL), which is presented in more detail in [Chapter 5](#), "Business Rules." Business rules specified in OCL can easily be transferred to an information system that supports the business, and in that way become executable.

The Eriksson-Penker Business Extensions use a stereotyped note to define rules. Such a note explicitly connects a rule to a specific element or part of an element in the diagram, and defines more informal *soft rules* that are expressed only in plain text. A soft rule could be an axiom, a principle, or a rule that involves the use of human judgment. Again, rules can be present in all views and in all diagrams. [Figure 3.13](#) shows a class diagram with some built-in business rules (the multiplicity of the associations), a constraint (specifying that a Person's salary must be greater than \$50,000 in order to get a Lease contract), and a business rule in a note (specifying how the rent is calculated).

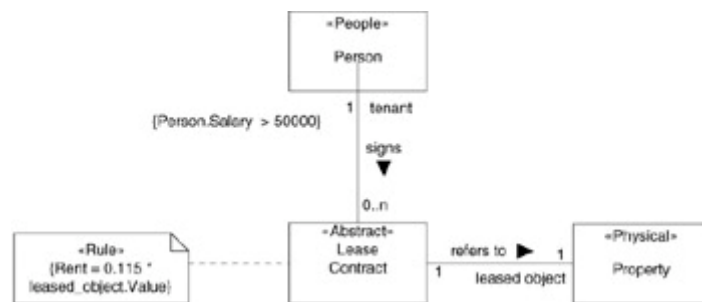


Figure 3.13: Business rules in a class diagram as constraints, rule note, or multiplicity on relationships.

Relationships

In addition to relationships and structures within each resource category, there are also relationships between different concept categories. A business process is associated with one or more goals, resources are allocated to a business process, resources can have their own goals, and there can be rules that constrain the possible states of a process or a resource. Since the diagrams used in the different business views are based on standard UML diagrams, the relationships between the concepts are the common object-oriented relationships used in those diagrams.

The relationships most commonly used in UML are:

State/activity transition. A relationship between activities or states, showing which activity or state will follow another. This is called a *temporal relationship*, one that shows the order of activities. See [Figure 3.14](#), where the lines connecting the activities in the process indicate the order.

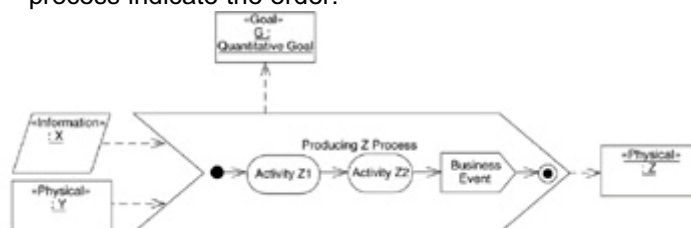


Figure 3.14: An activity diagram showing relationships between processes, resource objects, and a goal object.

Generalization. A specialization/generalization relationship, in which objects of the specialized element are substitutable for objects of the generalized element (subtype conformance). Generalization can be used, for example, to show a hierarchy of different resources. See [Figure 3.15](#), where the line with the open arrow shows how an element is a specialization of another element; the open arrow points to the generic element.

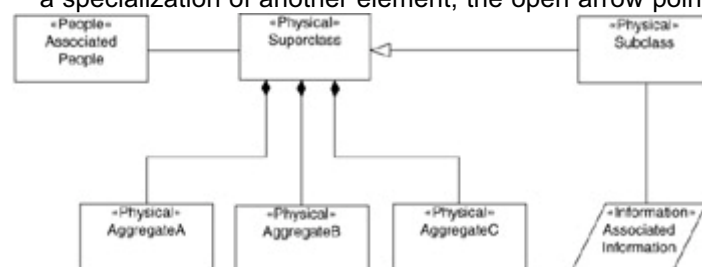


Figure 3.15: A class diagram showing static relationships, such as association, generalization, and aggregation.

Association/aggregation. A relationship that describes a set of links, in which a link is a connection among objects. For example, a customer object has an association to one or more order objects. Aggregation is a special case of association where one of the objects contains or controls the other object; for example, a product is an assembly of some components. See [Figure 3.15](#), where association is the straight line between two classes and aggregation is shown with lines ending with a filled diamond pointing to the controlling class.

Dependency. A relationship between two elements that in some way are dependent on each other. Typically, one object uses the other in some way, and indicates that a change in the object being depended upon will affect the dependent object. A dependency can also be used between an activity and an object to show that an object is consumed or produced by the activity, or between packages to show that packages are dependent on each other. See [Figure 3.14](#), where the dashed line between the process and different resources shows how the process is dependent on the resources.

Refinement. A relationship between two elements where one element is a refinement of the other. For example, refinement describes the same concept but in a more detailed or explicit manner. A refinement is shown using a dashed line with an open arrow. The arrow points to the unrefined element; the refined element is at the other end of the line. In the Eriksson-Penker Business Extensions, the activity transition has a stereotype «non-causal» that should be read as “the activity at the end of the transition arrow *might* happen.” In the world of information systems, no such relationships exist because computers running well-designed programs are reliable and the result is predictable. But in the real world, reliability is less assured. That’s the reason for the addition of the non-causal stereotype describing this relationship. An example is when you are driving too fast on the highway. A policeman *might* stop you. Or, if your employees work 16 hours a day for months, they *might* become sick or they *might* resign from the company. To illustrate those risks and chances, the non-casual relationship between activities can be used in a process model.

General Mechanisms

There is only one new general mechanism in the Eriksson-Penker Business Extensions: the *reference note*. A reference note is a stereotyped note that contains a reference to another diagram or another document (see [Figure 3.16](#)). This feature, which is not included in the current UML specification, allows the modeler to refer to another diagram in a standardized manner (sometimes a tool can allow such references, but not in a standard manner). The reference note can also be used to refer to a document; for example, double-clicking or some other user action performed on a note referring to a Word document would start Word so that the document could be read or printed.



Figure 3.16: A reference note allows the modeler to visually reference another diagram or document in a standard manner.

^[2][Booch 98] Booch, Grady, Ivar Jacobson, and James Rumbaugh. *The Unified Modeling Language Users Guide*. Reading, MA: Addison-Wesley, 1998.

^[3][Hammer 94] Hammer, Mike and James Champy. *Reengineering the Corporation: A Manifesto for Business Revolution* New York: Harper Business Books, 1994.

^[4][Davenport 92] Davenport, Thomas. *Process Innovation: Reengineering Work through Information Technology* Cambridge, MA: Harvard Business School Books, 1992.

^[5][Beedle 97] Beedle, Michael A. *cOOherentBPR: – A Pattern language to Build Agile Organizations*. PLoP-97 conference, 1997.

^[6][Gale 96] Gale, Thornton and James Eldred. *Getting Results with the Object-Oriented Enterprise Model*. New York: SIGS Books, 1996.

^[7][OMG 98] OMG, *Business Object Architecture Proposal*. Framingham, MA: OMG, 1998.

^[8][Darnton 97] Darnton, Geoffrey and Moksha Darnton. *Business Process Analysis*. Cambridge, U.K.: Thomson Business Press, 1997.

Summary

A business architecture is a practical tool for managing the complexity of a system such as a business. An architecture captures an organized set of elements with relationships to one another. The elements and their structure represent the organization and behavior of a business system and show abstractions of the key processes in the system. By clearly defining and documenting the architecture of the business, it is possible to discuss, communicate, adapt, and improve the business. But finding the right level of detail in an architecture can be difficult and is dependent on the purpose of the architecture. The architecture is defined in four different views, each containing a number of diagrams, that will be described further in the [next chapter](#).

The concepts that are used in defining the architecture are:

Process. An abstraction showing a number of activities that form a set of input objects and create a set of output objects that are of value to a customer. The input and output objects, along with objects used during the process, are all resources in the business. A process has a goal and is affected by events.

Events. A change of state that notifies that something has happened in the business. It is generated by one process and received by one or more other processes. Often it will be generated by a process outside the business.

Resources. Concepts or “things” used in the business, such as physical things (e.g., a machine), abstract things (e.g., an agreement), people, or information resources (e.g., information about other resources such as data about an employee stored in a information system).

Goals. The desired state of one or more resources. Goals are attached to the entire business and to processes within the business.

Business rules. A statement that defines or constrains some aspect of the business, and represents business knowledge. It governs how the business should be run (i.e., how the processes should execute).

General mechanism. Mechanisms that can be used in all diagrams. A reference note can contain the name of an external document or another diagram to show where further model information is available.

This chapter explained the UML notation used and the Eriksson-Penker Business Extensions defined for these model elements, along with examples of their basic implementation. The [next chapter](#) uses the extensions in diagrams, defines how the diagrams make up the different business views, and explores the relationships among the different concepts and views.

Chapter 4: Business Views

Overview

Modeling a complex business requires the use of multiple views. Each view focuses on a particular aspect of the business and is described through a number of diagrams, sometimes complemented with a textual document. Links to the information in the views make it possible to navigate from one view to another, thereby providing a more complete picture of the business.

Designing a model is very much like assembling a jigsaw puzzle: At first there are numerous isolated pieces. You begin by putting pieces together, typically by examining the pieces and their structure, form, and appearance. Slowly, a picture evolves until all the pieces come together in a complete image. The same process is done in business modeling: The views are not modeled in isolation one by one, but incrementally as information about the business is collected and understood. The information is modeled in different diagrams, and each diagram is allocated to the view that best captures that particular aspect of the business. This is where the metaphor ends, however, because unlike a puzzle, a business model rarely shows the complete picture of the business, and

seldom do all pieces of information fit together seamlessly. Rather, the business model is a picture of a particular aspect of the business.

As a basis for defining a business architecture with these views, it is preferable to have:

- A knowledge of the business, collected from the experience and information of those working in this type of business.
- Previous models of the business.
- Reference models for this type of domain; that is, generic architectural styles or patterns for this type of business.

Frequently it is the case, however, that previous models do not exist, that there are very few, if any, textual descriptions of processes, and that the people in the business openly admit that the current processes are not efficient. Consequently, modeling must begin with group meetings or interviews with the business employees and managers. Rough diagrams are sketched based on the information collected at this early stage. As the modeling process progresses, details are added and the sketches evolve. There is usually one person, the business architect, who is responsible for overseeing the modeling process and envisioning the complete architecture.

A common question asked when defining a model is what level of detail should go into the definition? It is not an easy question to answer. If the model presents a very high-level overview of the business, the description will be too general and only express the obvious. On the other hand, if the model tries to capture everything in great detail, it will be a very complex model that is difficult to use, evolve, or navigate.

The ideal solution is somewhere in between these two extremes. The level of detail included in the model is dependent upon the purpose of the model. If the model will be used for constructing information systems, the definition of the information used and the format of that information should be emphasized. If the model will be used to improve or innovate the business, the interactions in the processes should be emphasized, and possible changes that could improve the overall performance should be identified. Knowing the intended use of the business model is essential to be able to decide the appropriate level of detail necessary for each view or diagram.

Using a tool such as a computer-aided software engineering (CASE) or computer-aided business engineering (CABE) tool to draw these diagrams makes it easier to navigate from a diagram in one view to a diagram in another. The tool stores all information in its internal database and provides links to a specific concept that appears in several diagrams. This makes it easy to find all instances where that concept is defined or used. These tools also make administering larger models easier than would be possible if they appeared only on paper (smaller models, however, can be handled on just paper). Using a tool allows the modeler or the reader of the model to focus on one aspect of the business at a time, to gradually gain an understanding the business and its architecture. This is not to say that CASE tools should be treated as the miracle tools that they are sometimes marketed to be; what they can do is provide administrative help and simplify the development of complicated models.

This chapter defines four business views using standard UML diagrams and the Eriksson-Penker Business Extensions introduced in [Chapter 3](#), "Modeling the Business Architecture." Recall that these extensions are created using UML grammar and UML's standard extensibility mechanisms that include stereotypes, tagged values, and constraints. The views and the diagrams are illustrated with an ongoing example of a business model for the fictional Sample Business, Inc., a company that markets and sells financial services through the Internet.

Four Common Business Views

The Eriksson-Penker Business Extensions use four different views of a business. We chose the four views as the basis for the descriptions in this book because they are aspects common to businesses in general and blend well with the capabilities of UML. Their incorporation as given here is not mandatory; other views can be defined and used in different modeling processes, such as an economic effects and results view, or a human aspect view that highlights the human role and human goals in the business.

The four views used in this book are:

Business Vision. The overall vision of the business. This view describes a goal structure for the company and illustrates problems that must be solved in order to reach those goals.

Business Process. The view that represents the activities and value created in the business and illustrates the interaction between the processes and resources in order to achieve the goal of each process. The view also demonstrates the interaction between different processes.

Business Structure. The structures among the resources in the business, such as the organization of the business or the structure of the products created.

Business Behavior. The individual behavior of each important resource and process in the business model.

The views are not separate models; they are different perspectives on one or more specific aspect of the business. Combined, the views create a complete model of the business ([Figure 4.1](#)).

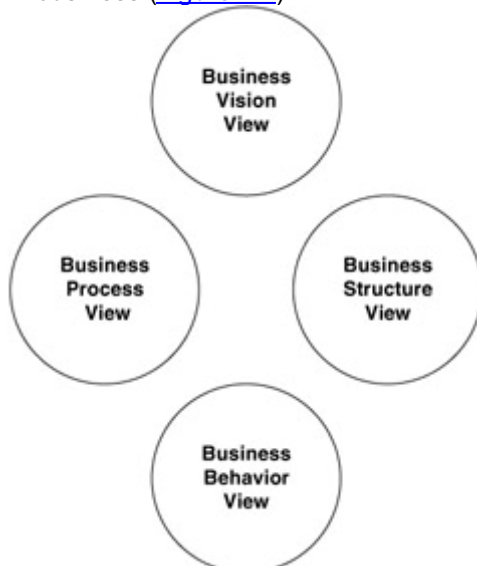


Figure 4.1: A business architecture is described with four views: business vision, business process, business structure, and business behavior.

The following sections discuss each of these views in turn, its overall purpose, the modeling effort required to define the view, the different diagrams within the view, and the techniques for capturing and defining those diagrams using UML. Recall from [Chapter 2](#) that most UML diagrams are applicable for business modeling, but there are three that are not used at all: use case, component, and deployment. These three are used to analyze and design information systems, and have a less significant role in business modeling. The use-case diagram identifies the requirements on the information system and links the business models to the information system models. The component and deployment diagrams are used to model software architectures. (Note that there are other opinions on use cases and business modeling, some of which advocate including use cases for business modeling. Our experience is, however, that the most important role of use cases is to specify the requirements on systems, while business modeling requires a more process-oriented approach.)

The four views are presented here in the order in which they are most commonly defined; again this is not a sequential process and order will vary. [Chapter 5](#), “Business Rules,” discusses how business rules can be applied in all of the views. Chapters 6

through 9 present business patterns that use these views, and provide additional examples of how they can be used and arranged in modeling different business situations.

Business Vision View

The Business Vision view depicts the company's goals. It is an image of where the company is headed. This view sets up the overall strategy for the business, defines the goals of the business, and acts as a guide for modeling the other views. Many of the goals in this view will not be visible in the other views or in the information systems, despite the fact that they indirectly affect the functionality of the information systems. This is because the goals have been broken down into more detailed goals, or because the information system doesn't actually store information about the goals (the system is part of a larger business process that strives toward the goal). The view guides the modeling process, acting as a point of reference for design decisions when constructing the information systems. The Business Vision view also is used as a motivational tool. It should be shared with the employees of the business, and provide the information and resources, as well as assign the responsibility, to achieve those visions.

Note that the Business Vision view does not contain a detailed description of the goals to be achieved. Rather, it is a basic strategy for moving forward. The vision is often focused on keeping and strengthening the strong aspects of the business and eliminating or improving on the weak ones.

There are some important factors to consider when creating the Business Vision view, according to Geoffrey Darnton [Darnton 97]^[1]:

Company Mission. The overall goal for the company.

Objectives. More specific goals, measurable over a period of time.

Strengths. The specific aspects at which the business excels.

Weaknesses. The specific aspects of the business that require improvement.

Opportunities. Areas of potential growth for the future of the business.

Threats. External conditions that might negatively affect the business (e.g., competitors, changes in technology, etc.).

Critical factors. Elements that are required for the business to succeed or grow (e.g., a quick time-to-market, good adaption to new technology, etc).

Strategies. Action plans that, if applied, will achieve the objectives.

Core competencies. Areas of the business that are of most importance.

Roles. The specific functions of the people working in the business.

Organization units. Groups into which the business is divided.

Key processes. The key steps to achieving the objectives.

It is also important to look carefully at the customers, competitor companies, and what's going on in the world to identify future changes and trends in the industry among competitors, in the market segment, in technology, or in regulations or policies. These changes can either impose threats or create opportunities for the business, and must be met with changes in business performance. Economic measures such as profitability, solidity, and cash flow must also be studied and benchmarked against similar companies within the same business domain.

The ultimate result of the Business Vision view is a definition of the desired future state of the company, and how that state can be reached. The primary result is expressed in a vision statement, and one or more goal/problem models. A vision statement is a short text document that outlines the vision of the company some years into the future. The goal/problem model is a formal diagram that breaks down the major goals of the business into subgoals, and indicates the problems that must be solved in order to achieve those goals. Thus, the vision statement is the readable form of the vision and the goal/problem model shows the definition of more concrete goals and subgoals that can be attached to business processes.

A member of upper management, such as a CEO, president, or member of the board of directors, is responsible for creating the vision statement and for defining the top-level

goals. This person must be able to communicate the vision to all members of the company and act as a leader and mentor for the project. Breaking down the goals and modeling the processes is the responsibility of a business architect or process modeler, but the upper management member acts as a supporter of the modeling efforts, continuously emphasizing its importance to the employees.

There are three techniques used in the Business Vision view:

Strategy definition. Position of the company with regard to the current and future world, and the strategic goals or necessary changes in the business.

Conceptual modeling. Definitions of the important concepts used in the business, along with their relationships to each other.

Goal/problem modeling. Definition of the goals of the company, including the breakdown of goals into subgoals, and the definition of the problems that hinder the achievement of goals.

The results of these techniques are used in other views as well, since they define the goals of the business, the concepts used when describing the business, and strategies that describe necessary changes in the business.

Strategy Definition

Defining an overall strategy for the future means that the business must be viewed in context of its surrounding world in order to identify threats and opportunities that will require the business to change. This also involves decisions about the future direction of the company, raising this question: Does the management or owner(s) want to expand or maintain their position in the market? An expansion normally involves a higher risk, and ultimately it is the owners who decide if they are willing to take that risk.

The strategy is normally based on conclusions that result from evaluating the core processes of the business or by creating new processes that extend the business into new areas or new ways of working. Support processes or administrative processes normally are not considered at this stage, nor are details of how the core processes should be designed or refocused. The strategy shows the direction in which the business is headed; the business processes, as well as the organization, should then be adapted accordingly. The strategy indicates required changes, and presents strategic ideas for the transition. The strategy also looks at previous failures (such as lost deals or product failures) and defines plans that would avoid such failures in the future.

The following are typical considerations and questions for strategy planning:

Customer. Who is the customer and what characteristics does he or she have? How is the customer's interaction with the business changing?

Competitors. Who are the competitors and what are they doing? In what way are they changing their business model?

Size and position in industry. How is the business positioned in the industry? Does it need to expand to increase the market share? How is the industry changing?

Profitability and growth. How do the profitability and growth of the business compare to other businesses in the same domain (benchmarking)?

The surrounding world. What changes (specifically political or technological) are taking place?

Public perception. How does the public perceive the company? In what ways is it desirable to change that perception? Does the company need a new public image?

Service level. What is the customer service level? Could service be improved or extended?

There are two techniques used to define strategy: a TOWS matrix and a vision statement.

TOWS Matrix

One technique used to summarize the business situation is to evaluate threats, opportunities, weaknesses, and strengths. Threats and opportunities are external attributes of the business, and weaknesses and strengths are internal attributes of the business. These attributes are listed in a Threats, Opportunities, Weaknesses, and Strengths (TOWS) matrix. [Weirich 82]^[2] The internal attributes are shown along the vertical axis and the external attributes along the horizontal axis. The matrix then is filled with strategies for handling each intersection between an internal attribute and an external attribute (e.g., one part of the matrix shows how to take advantage of internal strengths and external opportunities, another how to handle internal weaknesses that intersect with the external opportunities). The strategies suggested in the matrix are the basis for a more formal strategy. Strategies that are repeated in several parts of the matrix become especially significant because repetition suggests that those strategies would make a significant impact on the business. The TOWS matrix is often used as a basis for writing the vision statement, the textual document that outlines the future of the business. The TOWS matrix does not use UML, demonstrating our pragmatic view that techniques that don't use UML can also be used when there is no suitable UML diagram available.

Figure 4.2 is a TOWS matrix for Sample Business, Inc. Remember, this is a company that sells financial services via the Internet. Sample Business, Inc. uses the Internet as its primary means of communicating with its customers. Customers use a bulletin board available on the company's Web site to freely discuss different stocks and to read articles about the stock market. The site also shows current stock market prices, offers company profiles, and provides other services of interest to a private financial investor. The company will also act as an online stockbroker, enabling customers to maintain stock portfolios and buy and sell stocks on the markets via the Internet. The brokerage services, along with more advanced services, are available only to subscribing customers who pay a monthly fee. Other services, such as the bulletin board, are available without cost to visitors. Sample Business, Inc., expecting to become a popular site, also gains additional profit by selling advertisement space on the Internet page. Advertisers pay a fee based on the number of visitors to its Internet pages.

Overall Business Strategy To become a leading provider of financial services on the Internet, by having a complete range of services offered at competitive prices.	Internal Strengths 1. Good knowledge of Web design. 2. Good knowledge and experience of Internet programming solutions. 3. Knowledge of financial markets and services. 4. Trading knowledge and good contacts with stockbrokers.	Internal Weaknesses 1. Insufficient sales staff. 2. Insufficient financial resources. 3. Company not known—no "branding."
External Opportunities 1. Big interest in financial services on Internet. 2. No complete supplier on market. 3. Inexpensive marketing channel. 4. International market unexplored. 5. Investment interest in Internet companies.	Strategy: 1. Develop easy-to-use and complete financial Web site. 2. Target international customers interested in investing in U.S. market.	Strategy: 1. Sell advertisements through external agents. 2. Find subcontractors that deliver financial information on a royalty basis (based on actual usage). 3. Attract institutional shareholders. 4. Attract international customers.
External Threats 1. Competition already operating in U.S. marketplace. 2. Economic recession could happen. 3. Only free services are used by customers. 4. Hard to get known on the Internet.	Strategy: 1. Devise slogan and PR campaign about the complete financial site. 2. Advertise site on other Internet sites. 3. Define price strategy that reduces initial cost for customers.	Strategy: 1. Employ good sales manager. 2. Attract institutional shareholders. 3. Launch PR campaign to make site name known.

Figure 4.2: A TOWS matrix example.

Important strategies are identified by analyzing the TOWS matrix for Sample Business, Inc. in Figure 4.2. Again, those that are repeated or are similar are key. Three important parts of a business strategy that can be seen in this figure are:

- Emphasize that it's a complete online financial service provider (the strategy to stress that it is a complete service is mentioned in several places in the matrix).

- Target international customers who want to invest in the American market (the international market is mentioned as an external opportunity).
- Obtain institutional shareholders to secure initial financing (the weakness of not having enough financing must be addressed).

Even though these three strategies are most dominant, many other strategies mentioned in the matrix can and should be used in the vision statement. These strategies are stated as strategic goals for Sample Business, Inc. The means for achieving these goals must be determined. The TOWS matrix is a supporting technique often used prior to the writing of the vision statement.

Vision Statement

The strategy is summarized in a vision statement, a textual document that outlines the future of the business. It contains descriptions of the current business context (what is changing, what is important), the requirements on the business, the problems that can be visualized, and possible scenarios of what might happen. The vision statement is a description of what the company is to become, how it is going to operate, and what kinds of results are expected. The vision statement should contain clearly stated high-level goals that will later be broken down into operative goals. These operative goals can be linked to goals for individual processes. The vision statement also presents systems or ways to measure whether the high-level goals are achieved.

A vision statement also often contains a plan of the process effort to follow, that is, a description of how to organize the business models that support the vision. The vision statement is not only management's vision of the business, but is also intended to motivate the entire company to be involved in and contribute to the modeling work and, later, the implementation of the business processes. In order to evoke change, the organization must be convinced of the need for change. All too often, the employees are not convinced that the proposed plan will work, or do not support it. Such situations naturally decrease the chances for change. The vision statement should therefore be supported by managers who communicate the ideas and motivate the employees.

Conceptual Modeling

A conceptual model defines the important concepts used in the business. This model establishes a common vocabulary for all the concepts, and demonstrates the relationship among different concepts. Clear definitions of the basic concepts are central for understanding the business or modeling the business in more detail. Without a common vocabulary, individuals may have different understandings and interpretations of the concepts.

It is important to realize that there is a difference between the terms used to refer to an object, the concept that describes the object, and the object in the real world (this theory is called Ogden's triangle). For example, the term "car" is used to refer to an automobile. The concept of a car could be defined as a vehicle with an engine, chassis, and four wheels; and an object example of a car in the real world might be a Toyota Celica 94. Mixing these different views in discussions or models can result in misunderstandings or ambiguity. If several terms are used to refer to the same concept, clear references must be present in a conceptual model to validate all of these terms. If possible, the conceptual model should define one term that will be used consistently throughout all models.

The conceptual model is a high-level model of the concepts that has nothing to do with a database design or with class design in an information system, even though a class diagram is used to create the conceptual model and the concepts are modeled as classes.

Class Diagram for Conceptual Modeling

A conceptual model is presented with a standard UML class diagram. The names of the classes and associations used in the model are important, since they are the concepts being defined. In UML, arrows on association names are used to specify how to read a relationship and to create a clear and understandable model. The classes can have significant attributes attached to them to further describe the concepts, as well as a textual explanation defined in the standard UML property, named Documentation (this text is not visible in the UML diagram but can be retrieved with the help of a tool). Such textual descriptions create a term catalog in which each concept is defined in a few sentences. The name chosen for each concept is then used in other models and documents that describe the business.

Note that this class diagram is not a final diagram describing all possible concepts and all of their relationships. It is a first attempt to define the key concepts used to describe this business. New concepts and relationships can be introduced in this diagram as modeling continues. Also at this stage, the attributes and operations of the classes are not as important as they would be in a class diagram depicting software classes. More important is to capture the concepts as such and their relationships. If adding attributes and operations helps in characterizing the concepts, they also can be defined.

[Figure 4.3](#) is a conceptual model of Sample Business, Inc. The important concepts are modeled as classes, and the relationships are expressed through the UML relationships known as association, aggregation, and generalization. Reading the diagram provides much information about the primary concepts in the business. The customers are divided into three types:

- Ordinary customer, who is anonymous and just visits the Internet site.
- Registered customer, who has registered with name and e-mail address.
- Subscribing customer, who also pays a monthly fee to use some of the more advanced services.



Figure 4.3: The conceptual model of the most important concepts in Sample Business, Inc.

The model shows that a subscribing customer owns one or more portfolios in which both securities and orders to buy or sell securities are stored. Owning a security is modeled as a security holding, where a security holding represents a purchase date, a purchase price, and the number of stocks owned. The Security class represents one individual security on the market and is sub-classed into Stock and Stock Option (naturally, other securities could be modeled as well). All securities are associated with price information from the market, and have a grading with a security recommendation (e.g., buy, sell, hold). A security is associated with the company that has issued it and includes a company profile with a detailed description about the company (according to regulations) and news that concerns the company. Finally, all customers can read and write messages on a bulletin board, which is divided into different discussion threads

suggested by the customers. The bulletin board also contains articles about stocks and displays paid advertisements.

Each of the classes in the model should also store a few descriptive sentences in the UML Documentation property for the class. The attributes have been suppressed in [Figure 4.3](#), but important attributes should also be defined and shown.

Goal/Problem Modeling

A goal model describes the goals of the business and the problems that stand in the way of achieving the goals. Goals control the behavior of the business and show the desired states of some resources in the business, such as units produced per month, revenue for a specific product, or profit margins. The goal model establishes why the business exists, what the business is trying to achieve, and what the business strategies for achieving these goals are. It does this through a repetitive definition of subgoals, in which each goal is broken down into its subgoals, each of which is in turn broken down to its subgoals. A goal model also indicates ways to improve the business or resolve conflicts between goals. Some of the goals in a goal model can be legacy parts of the business, while other goals might show areas for improvement or entirely new business opportunities.

The goals are achieved by the business processes and the resources that participate in the process. Typically, the more detailed subgoals in a goal model are allocated to individual processes in the business. If the goal is new, a completely new process may need to be designed and implemented in the business.

The goals and problems are modeled in a goal/problem diagram, which is based on a UML object diagram and some stereotypes in the Eriksson-Penker Business Extensions.

Goal/Problem Diagram

A goal/problem diagram is a UML object diagram of objects and their relationships. The complete goal model is actually a number of object diagrams, wherein each diagram shows a specific high-level goal broken down into subgoals. Goals are described as objects of classes that have the stereotype «goal». Such objects are used in this diagram to show goals, the dependencies between goals, and the relationships between goals and the problems they solve.

There are two predefined goal classes in the Eriksson-Penker Business Extensions, though nothing prevents the modeler from defining other, more specialized, goal classes. The goal classes predefined are Quantitative Goal and Qualitative Goal. A Quantitative Goal can be easily measured through some value that is to be achieved. A Qualitative Goal is difficult to describe in measurable terms, and therefore determining if it has been achieved is more complex for than quantitative goals, and involves human judgment. Both of these goal types have a goal description as an attribute. A quantitative goal also has a goal value, a current value, and a unit of measurement.

Modeling goals as classes is required in order to define a goal hierarchy as a set of objects, and doing so also shows the possible implementation of these classes in an information system. Often the goal of a process is not represented in the information system, although the information system provides an excellent opportunity to measure how the goal is being achieved. That is because the information system has all the information necessary to calculate the results of the business (which can be in terms of productivity, economics, quality, etc.) and compare against the goals. By implementing a goal as a class, it is possible to have operations that decide if and to what extent the goal is fulfilled, as well as perform process simulations in an information system.

A specific goal in the business is described as an object of a goal class. The relationships between goals are dependencies and associations. A dependency between two goals shows that one of the goals is a subgoal, or a dependent, of another goal.

Associations are used to show links between two goals, such as contradictions between goals.

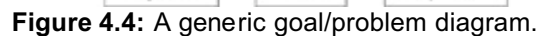
A dependency is represented by a dashed line from the super-goal to the subgoal, ending with an open arrow. The super-goal is then dependent on the subgoal, which should be interpreted so that the fulfillment of a subgoal contributes to the fulfillment of the super-goal. If a goal can be completely broken down into subgoals, a dashed line is drawn across the dependencies and a constraint is written next to the line, in this form: {complete}. If the goal can't be completely broken down into subgoals, {incomplete} is written (this is also the default if nothing at all is written). A goal that has been completely broken down into subgoals (i.e., the constraint {complete} is specified), indicates that the goal will automatically be filled if all of the subgoals are met. This is what could be seen as a logical AND condition between the subgoals. There is currently no corresponding logical OR condition in the extensions, but the contradictory stereotype, explained later, is used to denote mutually exclusive goals. If a goal is not completely broken down, other events or results might be necessary to fulfill the goal, even if all subgoals are achieved.

The other key concept shown in a goal/problem diagram is a problem. A problem is an obstacle that hinders the achievement of a goal. Identifying the problems is just as important as finding the goals! By finding the problem, new goals or subgoals are discovered that attempt to eliminate the problem. A problem is therefore always linked to a goal. Similar to goals, problems can also be broken down into subproblems. Because problems are linked to goals, this structure is normally only shown indirectly in a goal hierarchy, with the problems attached to their respective goal. In the Eriksson-Penker Business Extensions, a problem is more informally specified in a note with the stereotype «problem» attached to its goal object. A problem can be a temporary problem that can be solved once and for all, or it can be a continuous problem that requires continuous action in order to prevent the problem from reoccurring. Problems are eliminated—solved—by actions.

An action plan can be formulated from the goal model, where temporary problems are resolved as soon as possible, and the goals linked to the continuous problems are allocated to processes in the business (i.e., they achieve the goal through ongoing activities). The action plan contains a list of the problems, the cause of each problem, the appropriate action for each problem, the prerequisites for each action, and, finally, the resource or process responsible for solving it. This can also be shown visually in the goal/problem diagram through the use of stereotyped notes. The stereotypes «problem», «cause», «action», and «prerequisite» specify the purpose of the note. All of these optional concepts are defined through the use of informal notes, because they are normally described in simple text and can't be formally defined. The processes, designed later, that implement the actions and provide the prerequisites are formally defined. The problem note is attached to the goal, the cause-note to the problem note, and so on.

A technique for identifying goals is to ask these questions: Why should we achieve this goal? and “How should we achieve this goal?” The answers to the why question will identify higher goals (goals to which the current goal is a subgoal), and the answers to the how question will identify subgoals. Answering these two questions makes it possible to identify new goals from existing goals, and to reveal additional goals that might not have been discovered by the people in the business.

[Figure 4.4](#) is a generic goal/problem diagram. It shows the Quantitative Goal and Qualitative Goal classes, both of which are stereotyped to «goal», and a number of goal objects of these classes with their dependencies. The goal X is completely broken down into three subgoals (of which two are quantitative and one is qualitative). There is a problem linked to subgoal X1; and the subgoals to X1 (one or both) typically contribute to eliminating that problem. The subgoals to X1 are noted as being incomplete, meaning that other events or results might have to occur to fulfill goal X1. The problem linked to a goal can be used to find new subgoals of that goal.



[Figure 4.5](#) is a more concrete goal/problem diagram from Sample Business, Inc. The overall goal of this diagram is to have many customers. The goal value has been set to 500,000 customers that are known to the business; that is, where the names and addresses of the customers have been registered (an Internet business can also have anonymous customers). This goal has been broken down into three subgoals that also describe the categories of the customers:

- [illegible]

Figure 4.5: An example of a goal/problem diagram.

All of these categories are considered customers, and the business concept is to provide quality services that will attract as many new subscribing customers as possible. Problems in [Figure 4.5](#) are linked to each of the three subgoals. For example, the problem attached to attracting more Internet visitors is that Internet users are not aware of the site. This problem could be handled by generating links to the site from other sites within the financial community; by ensuring that the site is visible through the most common search engines; or by mentioning the name of the site in other media. All of

these actions have been modeled as subgoals. Again, these subgoals also have problems linked to them that in turn are used to find further subgoals.

The subgoal “Links from Other Sites” also is defined with the optional sequence of a problem, cause, action, and prerequisite definition through the use of notes. Again, these are optional; they don’t have to be defined for all problems, or sometimes not all of these notes have to be defined (e.g., often a problem and an action note are enough). They are, however, helpful in visualizing what needs to be done in order to achieve the goals, and what the prerequisites are to an action. This information then is used when designing or changing the business processes.

The goal/problem diagram in [Figure 4.5](#) shows only one of the primary goals of this business. Other goals have their own diagrams. All the primary goals of the business can be summarized in a diagram of their own, where any conflicts between the goals (contradictory goals) are shown. Each of the primary goals then can be described in a diagram of its own, with its corresponding subgoals. For example, the goal of attracting many customers may be contradictory to obtaining high revenue from advertisement since Internet visitors will avoid sites with too much advertising content. This contradiction could be illustrated in the goal model using an association between these goals stereotyped to «contradictory» (an example of using this stereotype was shown in [Figure 4.4](#)).

It is important to realize that the goal/problem diagram shouldn’t be over-formalized or described in too computational terms. The purpose of the goal/problem diagram is to identify and structure the different goals of the business and to break down the goal descriptions to a level at which these goals can be allocated to individual processes.

Business Process View

The Business Process view is at the center of business modeling. The processes show the activities that must be undertaken to achieve an explicit goal, along with their relationships with the resources participating in the process. Resources include people, material, energy, information, and technology; these resources can be consumed, refined, created, or used (i.e., act as a catalyst) during the process. There are relationships between a process and its resources and between different processes that interact, and there is a coupling of processes to goals. The processes have a purpose and a specific goal, and all the processes collectively attempt to achieve the overall goals of the business. The process definitions are used to understand the business, to see threats or opportunities in the business, to improve or innovate, and to act as a basis for other models (such as information system models).

The goals of the company as stated in the Business Vision view are the basis for modeling processes. Existing process descriptions serve as the basis for creating the models, but it is important to note that these must be reviewed with a critical eye. The business processes are modeled by utilizing interviews, discussions with the people in the business, the results of brainstorming sessions composed of carefully selected groups of people, and practical studies of how the business operates. In performing these activities, the modeler attempts to understand and capture how the business operates and how the resources in the company are handled.

The result is the creation of a number of process diagrams that describe at least the *core processes* of the company. A core process is that which has interactions with the external world or is critical for the delivery of goods and services offered by the company. Core processes are normally customer-oriented and horizontal to the traditional organization of the company.

A process description should be a generic description, while an actual execution of a process executes a specific path in the process. This means that the description of a process should contain all the execution alternatives (i.e., including exceptions and error conditions that can occur). A process instance is an example of an execution, a specific way through the general description.

The business architect creates the business process models, possibly with the support of a team of process modelers.

The Business Process view is described with an UML activity diagram. To use the activity diagram as a process diagram, the Eriksson-Penker Business Extensions established a set of stereotypes that define a process and the various resources. In addition, a variant of a process diagram, called an *assembly-line diagram*, is used to more clearly depict how the process interacts with resources during its execution. In the assembly-line diagram, the resources are often information resources (i.e., information objects) in an information system.

It is essential to define the following when modeling businesses in order to identify and specify the business processes:

- *Which activities are required?* These are specified as processes or activities in a process diagram.
- *When are the activities performed, and in what order?* This is specified through the control flow in a process diagram.
- *Why are the activities performed; what is the goal of the process?* This is specified in a process diagram through the attached goal object and a goal diagram that puts the goal into a context with other goals.
- *How are the activities performed?* This is specified in a process diagram, often by breaking down the processes into subprocesses that define the activities in more detail.
- *Who or what is involved in performing the activities?* This refers to the resources that participate in the process.
- *What is being consumed or produced?* This refers to the resources consumed or produced in the process.
- *How must the activities be performed?* This is defined through the control flow in a process diagram or through business rules.
- *Who controls the process?* This refers to the owner of process who runs the process or is responsible for its success.
- *How is the process related to the organization of the business?* This can be shown through the use of swimlanes in a process diagram.
- *How does the process relate to other processes?* This is shown through interaction modeling, which is discussed in the Behavior View section later in the chapter.

As your understanding of the business increases, the answers to all of these questions become apparent, enabling you to accurately model the processes in the business.

Process Modeling

There are several techniques used to model processes, most of which are customer-oriented and require evaluating the products or services produced in the business. By discovering the interfaces to the customer, it is possible to identify the business events that are generated and then analyze them to indicate the appropriate action the business must take for each event. By working backward from defining the product or service, you discover the activities required to produce the product or service. Once the process has been identified, it can be broken down into subprocesses. The flow of both control and objects (e.g., resources) are thus captured and described in a process model, in which each step in the process adds value to the resources created or refined in the process.

Processes are modeled first by concentrating on and completely modeling the core processes of the business and then by moving on to the support processes. A business usually does not have more than five to ten core processes, but may have many more support processes. Ideally, all the processes are modeled and integrated into each other, but if that isn't practical, the goal should be to concentrate on the core processes. The core processes have interfaces to the customers and are the processes the customers use to evaluate the business. It is also important to determine whether the indirect support processes in any way constrain the core, direct value processes. That should, of course, be avoided.

Process modeling creates an accurate documentation of the way in which work is performed, perhaps in order to develop better information systems. It is also used to improve or innovate processes to make the business function more efficiently. Process modeling also identifies new opportunities in the business and creates and designs new processes that take advantage of the resources and knowledge present in the organization. Clarifying the purpose before beginning to model more finely aims the focus on what is important in the model. You can better determine if you are on the right track as you model because a clear purpose provides the modeler with a goal to work toward. Knowing the future use of the result makes it easier to decide what is important and what is not.

Process management is an additional area of concern related to process modeling. Because processes aren't strictly allocated to just one organizational unit, but span over several units, a specific process owner within the organization is designated as responsible for that process. The process owner is then given authority over this process across the organization. This is important to ensure that the result of a process is distributed fairly, for example, preventing one organizational unit from profiting if a process goes well while another organizational unit notes a loss by the same process. Ideally, the vertical way of building result units is eliminated and replaced with a complete process organization.

Many books have been written about process modeling, so there's definitely more to say about this subject. To that end the reference list at the end of the book suggests titles to read on the most important work in this area.

Process Diagram

A process diagram is a UML activity diagram with a set of stereotypes that describe the activities performed within the processes and how they interact; the input and output objects; the supplying and controlling resources that participate in the process; and the goal of the process. [Figure 4.6](#) shows a generic process diagram.

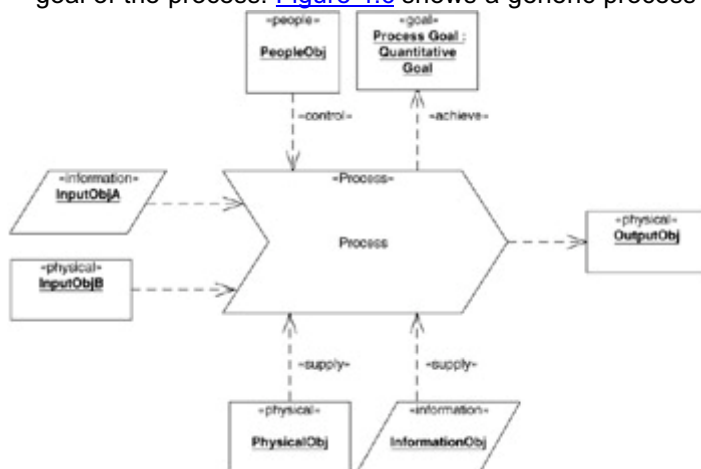


Figure 4.6: A generic process diagram.

A process is an activity stereotyped to «process». This stereotype has the traditional process icon shown in [Figure 4.6](#). A process can contain other processes, or subprocesses, that describe the internal steps taken within the overall process. An activity symbol (a rectangle with round corners) is used to show that a process cannot be broken down further or that doing so wouldn't be meaningful; that is, an activity is atomic. An atomic process should be intuitively understood as specified by the text within the activity symbol, so that the resources that are responsible for performing it can't misinterpret it.

Resource objects and goal objects that are involved in the process are placed around the process. These objects are:

Goal objects. A goal object from a goal/problem diagram that has been allocated to a process. A goal object is drawn above the process diagram and attached with a

dependency that is stereotyped to «achieve» from the process to the goal object (showing that the process attempts to achieve the goal).

Input objects. Objects that are either consumed or refined in the process. The input objects are resources, and as such can be stereotyped to «physical», «abstract», «people», or «information». They are connected with dashed lines from the objects to the process (i.e., the activity diagram notation for an input object). Input objects are normally placed to the left of the process.

Output objects. Objects that are produced by the process or that are the results of the refinement of one or more input objects. The output objects are also resources and are connected with a dashed line from the process to the output object (i.e., the activity diagram notation for an output object). Output objects are placed to the right of the process.

Supplying objects. Resources that are participating in the process but are not refined or consumed. These objects are drawn below the process with a dependency (a dashed line) from the object to the process. The dependency is stereotyped to «supply».

Controlling objects. Resources that control or run the process. Such objects are normally drawn above the process, with a dashed line from the object to the process. The stereotype of the dependency is «control».

Refinements by a process to the input objects can change the location of the object, the appearance of the object, or the content of or information in the object. It is difficult to separate an input object from a supplying object because a supplying object can also change its state during the process. An input object represents a key object that is refined or consumed in order to produce the output object. A supply

object is one that the process requires (it participates in the process) in order to be able to perform that refinement or consumption. For example, in a manufacturing process, an input object could be raw material, and a supply object could be a machine used in the process. In many cases, the output object is of the same class as an input object, but with additional value resulting from the process.

Note that there is no multiplicity shown in the process diagram. Multiplicity is not shown on dependencies where only one input object of a specific class or one output object is required. Instead, the process is a continuous operation that proceeds to consume input objects and produce output objects. If the number of output objects is required, it should be specified as a goal for the process (e.g., the number of products per day or per month), not through multiplicity as used to specify relationships in UML.

[Figure 4.7](#) is an example of a process diagram that shows the advertisement sales process for selling ad space on Web pages. It has a specific quantitative goal that includes a sales sum, a cost sum, and a yearly budget (note that this goal class is a special class designed for this case). To the left are the input objects: information resources Suspect and Prospect. A suspect is information about a company that may be willing to advertise. A prospect is information about a company that has expressed an interest in advertising. The result of the sales process is Orders. Order is an abstract resource because it is an agreement between a customer and Sample Business, Inc. Participating in the process are the salesperson and sales material resources. The process is controlled by a sales manager and by the corporate sales directives, usually an instruction book on how to conduct sales within this company.

which object is performing the activity (shown by placing the activity in a specific swimlane).

In [Figure 4.8](#), the Order object from the sales process is an input object to the Web Design process, where a profile of the company that wants to advertise and a representative for that company are also required as input. The Web Design process, involving a Webmaster and a Copywriter, creates the actual ad and an advertisement plan that describes how often and on which pages the ad should be shown. This then becomes input to the Web site deployment process, the key process in the delivery section of the company. The delivery section, with the Web site deployment process, administers and maintains the Web site, updating the material and making sure the site is available to the customers. The deployment process uses the ad and the advertisement plan to update the current Web site accordingly.

Another technique that can be used in process diagrams is to show the business events that are received or generated by the process. [Figure 4.9](#) shows the customer login process. After login, the process expects a Service Select Event from the customer. These business events are also objects, and are defined in a class diagram, as shown in [Figure 4.10](#). Depending on which business events the customer generates, the process continues to execute one of a set of different alternative processes. This is an example of object-oriented polymorphism, a technique in which the type of an object decides what will happen next. Each process handles a type of request; and by looking at each of these processes, the handling for each request type is detailed. The figure also illustrates the UML symbol for generating a event. The Trading Service generates a Stock Order Event, which is sent to another process (not shown here). This is all standard UML notation for sending and receiving event objects.

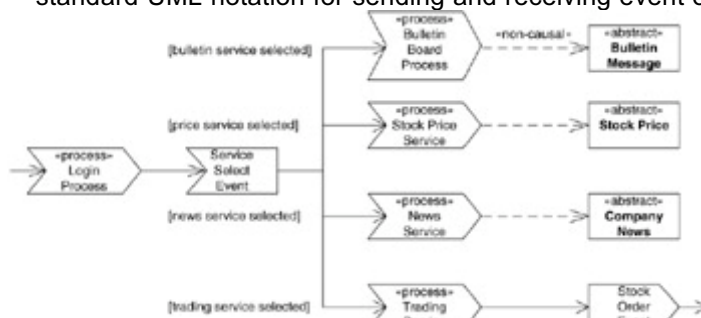


Figure 4.9: A process diagram showing the business events received and generated.



Figure 4.10: A class hierarchy of the business events used in Figure 4.9.

The bulletin board example in [Figure 4.9](#) demonstrates yet another stereotype, «noncausal», which when interpreted might cause the process to produce the object. The bulletin board process has a noncausal dependency to the bulletin message object, meaning that such an object might be the result of the process (but not always). Relationships and dependencies are usually causal, meaning that the object linked is always used or created as part of the process. In a noncausal dependency, which defines either an object flow or a control flow, the connection might not exist, and there are no well-defined conditions for when it does exist. This is used when there is a noncausal object flow from a process to an object, and the object might become a result of the process (it is impossible to define a clear condition for whether or not the process will produce this object). The noncausal stereotype can also be applied to the control flow lines between processes to indicate that a process might lead to the execution of another process. If this is a result of a well-defined rule condition, a business rule should be defined and specified within brackets to indicate that when the condition is fulfilled, the process will always be executed and no stereotype will be applied. However, often a strict rule cannot be defined and so the noncausal stereotype is used. Remember that

here we're trying to describe a complex business that is often very unpredictable, not computer software that, when written correctly, is very predictable.

If a model contains many processes, the processes can be allocated to packages to organize the model, as shown in [Figure 4.11](#). Recall that in UML, a package is a generic grouping of elements. It is up to the modeler to determine which packages are necessary and what they represent. The packages could be mapped to represent the organization of the company, the different types of processes, or a number of abstractions above the process level.

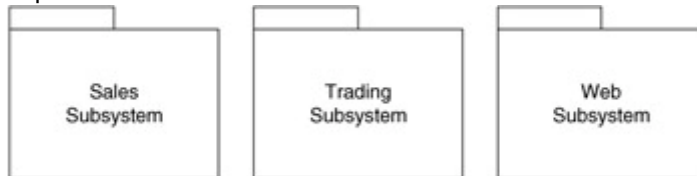


Figure 4.11: Packages of processes.

Assembly Line Diagram

The assembly line diagram is a unique diagram in the Eriksson-Penker Business Extensions. As with the process diagram, it is based to a large extent on the UML activity diagram. The assembly line diagram (introduced in the Astrakan method) has been used successfully for process modeling, particularly when the purpose of modeling is the production of information systems that support the processes. It is named assembly line because of the way it looks—processes that write and read objects placed in an assembly line.

At the top of the assembly line diagram is a process diagram. Below the process diagram are a number of horizontal packages that are called assembly-line packages, each representing a group of objects (the objects in the package can be of one specific class or of different classes; that decision is up to the modeler). An assembly line package is a UML package that is stereotyped to «assembly line» and drawn as a long horizontal rectangle, as shown in [Figure 4.12](#). The purpose of this diagram is to demonstrate how the processes in the upper part of the diagram write and read objects in the assembly line. A reference from a process to an assembly line package is indicated with a dashed line (object flow) between the process and an object within the assembly line package. The type of operation performed on the object is written along the line (as the name of the object flow). The diagram is read from left to right in the sequence of references to the assembly line packages.

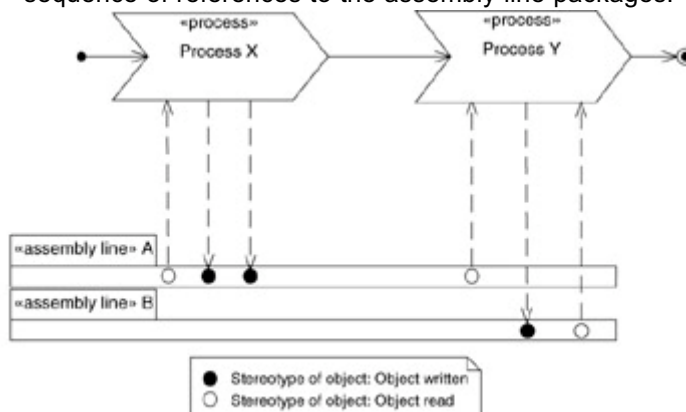


Figure 4.12: Generic assembly line diagram.

Because the objects in the assembly line package are often information objects in an information system, the assembly line diagram shows what information is accessed through the system and how that information is used by the processes. The objects can be other resources as well, and it is up to the modeler to determine what the assembly line packages include. The description of what each assembly line package represents appears in the tagged value “Documentation” of each package, since the packages are all stereotyped to «assembly line». A package could, for example, represent an entire

information system, a subsystem in an information system, a special category of classes in an information system, or a specific type or group of resources.

The assembly line diagram can be viewed so that the primary activities are shown in the process diagram. The references to the assembly line package are really support activities completed to make the execution of the primary activities possible. Because the UML package mechanism is used, an assembly line package is just a grouping mechanism. The assembly line diagram is an excellent way to show how resources are read or modified as part of the process, and how an object modified (or created) by one process at a later stage is read by another process. The interaction between processes through common resource objects is shown in the assembly line diagram.

[Figure 4.13](#) is an example of an assembly line diagram that shows the processes Stock Order Handling and Trade Handling and their interaction with a number of resources in the assembly line. The first three assembly line packages in this case correspond to classes for Portfolio, Order, and Security Holding, respectively; the assembly line packages Market and Administration are used more as general grouping mechanisms for a number of classes that represent a market and an administrative unit within the organization. For example, the Trade Handling process creates a Contract Note in the Administration package as its last reference to an assembly line, where a Contract Note (a document that is sent to the buyer or seller as verification of a trade) is an administrative object, that is, an object in the Administration package.

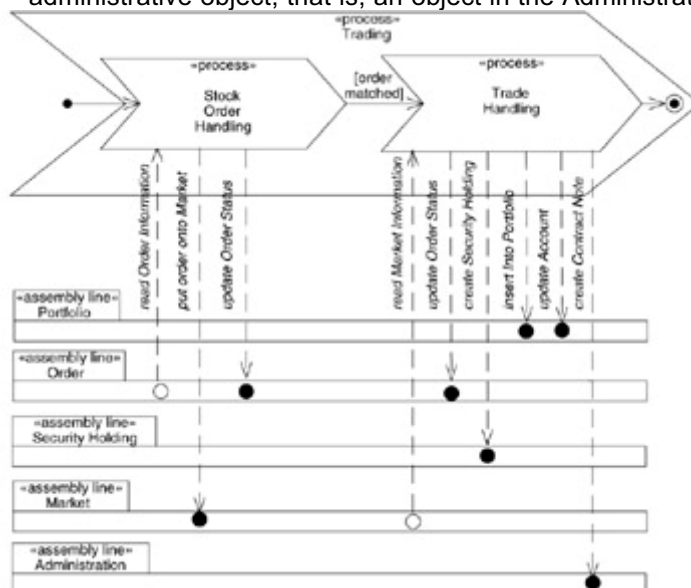


Figure 4.13: Specific example of assembly line diagram.

Assembly Line Diagram and Use Cases

[Figure 4.14](#) shows another assembly line diagram in which all the assembly line packages are objects in an information system. The diagram shows how the process interacts with the information system. The references to the assembly line packages comprise information flow to and from the information system and show the interface between the business process and the information system. This interface is described through use cases in object-oriented modeling; and a set of references in an assembly line diagram typically become a use case that the information system has to provide. This is very important, because it maps the business process to use cases that describe the functional requirements of an information system; it also identifies the proper actors of the use cases (the roles played by the process that uses the assembly lines). Assembly line diagrams provide the connection between business modeling and software system requirements modeling with use cases. A common question when modeling use cases in a software system is How do I know I have defined the right use cases in terms of the business? The assembly line diagram is a good technique to answer this.

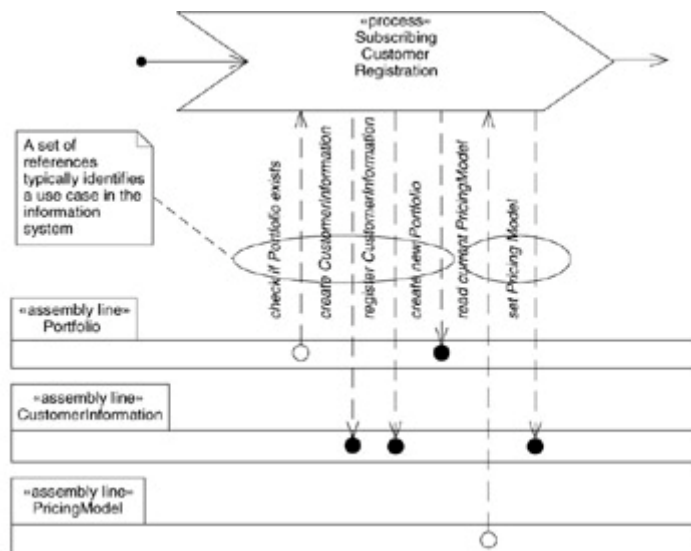


Figure 4.14: The references to the assembly line packages in an assembly line diagram can be mapped to use cases that define the requirements of an information system.

This analysis should start with the assembly line packages at a very high level, such as the system or subsystem level. Once the initial references from the process to the information system are identified, classes in the system can be identified, and the same analysis can be repeated with the packages now defined at a another, more detailed, level. At this more detailed level, the references from the process also become more detailed. A single reference in the initial diagram might be divided into several references. It is also important that each “set of references” that becomes a use case is not chosen in an ad hoc manner, but is selected according to the rules of use case identification (e.g., a use case is a service that brings a specific value to an external actor). The use case should have a clear initiation, a sequence of communication between actor and a system, and a well-defined end that brings value to the actor. If the references from the process to the assembly line packages are randomly put together into use cases, the result will be ill-formed, partial use cases.

Nonfunctional requirements of information systems (such as performance, availability, usability, etc.) can be determined by studying the process descriptions. Factors such as process lead-times will affect support systems, such as information systems, because they define the requirements of the information system. The desired characteristics of the process directly affect both the functional requirements (what should the system do?) and the nonfunctional requirements (e.g., how fast and how reliable must the system be?). There are also factors not directly described in the process that could affect the information systems, such as a change in government or a new law. By deriving all requirements of the support systems from the processes, with the assembly line and the process properties, it is also possible to verify them. Moreover, the processes themselves result from goal modeling, which means that the system requirements can be validated against the overall business goals! Validation is about doing the right things (that achieves the business goals) and verification is about doing things right (so that the goals actually are achieved). This means that by working with goals, processes, and the assembly line diagram, it is possible to verify and validate both functional and nonfunctional system requirements. It is important to point out that the assembly line diagram is not limited to use with information systems; it has also been successfully used to model logistic systems and human resource systems.

Business Structure View

The Business Structure view shows the structures of the resources, the products, or the services, and the information in the business, including the traditional organization of the company (divisions, departments, sections, business units, and so on). It does not show the structure of processes or the breakdown of processes into subprocesses; that is shown in the Business Process view. The Business Structure view is considered

supplemental to the Process view, depicting information that can't be shown in the process diagram but that is vital to the operation of the company. It too, is modeled by the business architect, again possibly supported by a team of process modelers.

The traditional organizational charts and descriptions, and descriptions of the products and services that the company provides, are the basis for the Business Structure view. The information from the Process view is also used since it shows which resources are used. Note that, typically, these two views are modeled in parallel, since they contribute to each other and must be consistent. It is difficult to model one of the views completely and then move on to the other, since their development interacts. During your discussions and interviews, you will find that the people in the business will interchangeably present information that belongs in either of the views.

The UML diagrams used to document this view are class and object diagrams. The class diagrams show the principal structure; the object diagrams show an actual configuration of the class diagram (e.g., how an organization looks at a specific point in time). Because the resources are modeled as classes, they have operations and attributes. The operations of a resource are used in a process or assembly line diagram; that is, the processes use the operations to trigger the resource to perform a specific service or action. Recall that a class diagram was also used in the Business Vision view to create a conceptual model of the basic concepts in the business. The class diagrams modeled in the Structure view depict the resources, information, and organization in more detail, in addition to more of the business rules that govern the structure. The conceptual model in the Vision view provides a high-level overview that defines a common terminology. The interactions among a number of resources can be indicated in a UML sequence or collaboration diagram. Sequence and collaboration diagrams show the interaction among a number of objects, in which both the order in time and the operations performed by the different objects can be shown. (This is further illustrated and discussed in the [“Business Behavior View”](#) section later in this chapter.) The more dynamic and flexible the principal structure of the resources is, the more the actual structure can be changed in the future.

Resource Modeling

Resource models show the structure of different resources. The generic model is depicted in a class diagram, while an actual configuration of a structure is shown in an object diagram. The inner structure of the resources, which are usually products or services offered by the company, can be presented in a resource model. The difference between a resource model and a conceptual model (as shown in the Business Vision view) is that the resource model concentrates on the more concrete structures of resources, such as products or services, while the conceptual model concentrates on defining the meaning and relationships of important concepts used when defining the business.

[Figure 4.15](#) shows a class diagram of the bulletin board resource structure for Sample Business, Inc. The bulletin board consists of different Web pages that contain messages, articles, or instructions. All messages are organized in discussion threads. A Web page can have up to four advertisements attached to it. This is modeled with traditional object-oriented techniques using class diagrams.

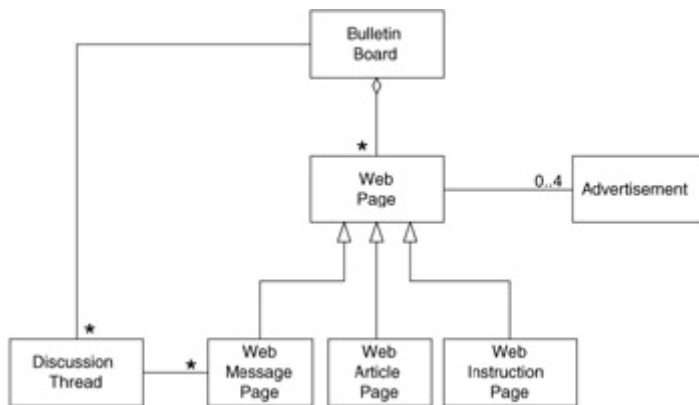


Figure 4.15: Example of resource class diagram.

Information Modeling

Information modeling creates models of strategically important information in the business. Even though information is also a resource in the business, it is worthwhile to model it separately using the techniques of class and object diagrams. The information is what goes into the information systems that support the business; it has a very strategic value to the business. Information modeling is an early step in defining the information stored in an information system, even though details such as database issues (e.g., keys, etc.) should not be part of the business model. Such details are defined during the modeling of the software information system.

The requirements of the information are governed by the business, but sometimes the information available also governs the business. For example, the information that the business has can create new opportunities for the business; the more information a business can capture about its customer, the more possibilities it has to adapt and configure its products and services. A very obvious example of this is an Internet business, where numerous commerce sites attempt to learn as much as possible about its visitors and customers in order to customize their Web pages. There are also many examples of companies that have information that is not fully used or explored for the purposes of improving the business or customer relationships.

[Figure 4.16](#) is a class diagram that contains classes for the most important information resources in Sample Business, Inc. Note that a business model can have classes for both Customer and Customer Information. The Customer class represents the actual customer, the physical resource, and how objects of that class behave and interact in the business processes. The Customer Information class represents the information about the customer, which the business stores in an information system (although a simple card file is also plausible). The Customer and Customer Information classes are two separate entities and must be modeled as such. A very common mistake in analyzing and designing information systems is that one class in the analysis model attempts to be both the actual Customer (typically an actor in a use case) as well as the Customer Information.

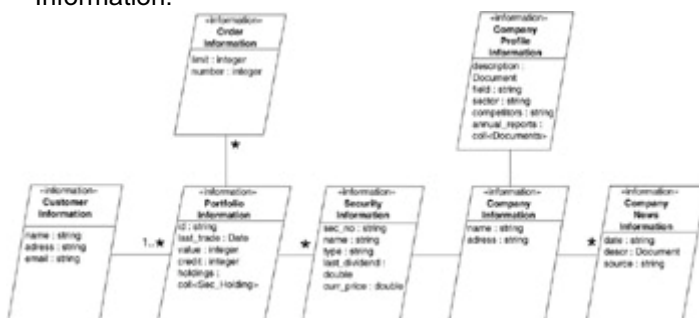


Figure 4.16: Example of an information class diagram.

Organization Modeling

Organization modeling is another special case of resource modeling, in which resources are allocated to organizational units that are related to each other according to specific rules. The allocated resources in an organization include employees, machines, and locations. Processes also can be allocated in an organization, or resources from an organization can be allocated to a process. Remember that a process often spans the borders of several organizational units. This doesn't mean that organization is unimportant. Often the responsibility of running a process is allocated to a process owner who is attached to an organizational unit. This owner manages the process even when it spans organizational units. The organization should strive to make optimum use of the resources within the business, and attempt to avoid internal suboptimization, which, unfortunately, is common in many organizations. The basic functions of an organization model are to show the allocation of resources, the reporting methods, task assignments, and the way the company is managed. The organization can include several dimensions, such as the organization units, geographical placement, and the allocation of processes.

The overall structure of an organization is expressed through class and object diagrams. A class diagram specifies the basic structure and rules for the organization, and an object diagram shows the actual organization currently in use. The more flexible and well designed the class diagram, the easier it will be to make organizational changes without affecting the business model or its supporting information systems. Resources are typically allocated through an object diagram in which resource objects are linked to organization objects. Processes are linked in an organization through swimlanes in a process diagram, as discussed previously in this chapter.

The trend in organizational modeling is to avoid the classical hierarchical structures in favor of more flexible and dynamic alignments. These organizations can be based on projects or missions in which resources are temporarily allocated to a specific process, and can either be based on a traditional organization from which resources are borrowed or have no organization at all (i.e., all work is done in project form and resources are moved into other projects as soon as a project is complete). The advantage of dynamic organization is that optimal workgroups are created for each task. The disadvantage is that the people in such resources might feel disoriented as a result of not belonging to a traditional organizational unit. Even in very strict hierarchical organizations, informal structures and methods of communication aren't planned for (though in practice they may take place).

Information technology in many situations can be an enabler of more flexible organizations, provided that the information systems aren't designed for a specific organization and can adapt to changes. At worst, if it is impossible or very expensive to adapt the information systems, its construction hinders changes to the organization or the business.

Some advanced organizational patterns and guidelines for creating flexible organizations are discussed in [Chapter 7](#), "Resource and Rule Patterns."

The class diagram describes the names of the organizational units and the business rules for arranging and linking them to each other. [Figure 4.17](#) shows a class diagram for a company with a management team that is organized into divisions. The divisions in turn are organized into sections. This figure depicts a rather static structure that doesn't allow much flexibility. A better solution is to use the concept of powertypes, introduced in [Chapter 2](#), "UML Primer," and explained further in [Chapter 7](#), "Resource and Rule Patterns."

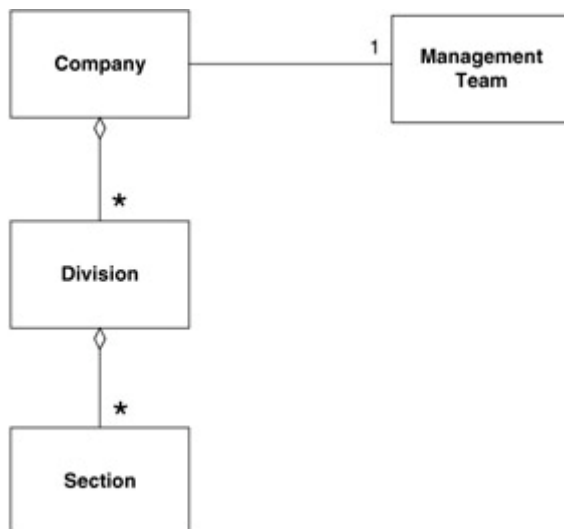


Figure 4.17: Organization model as a class diagram.

[Figure 4.18](#) shows an object diagram that adheres to the class diagram in [Figure 4.17](#). It shows actual instances of the classes and provides a view of the current organization. A model or a system based on the class diagram in this figure would allow new divisions or sections to be added or removed in the future; but would not allow any changes to the actual structure (e.g., for a section to have subsections within it or for the addition of new organizational concepts).



Figure 4.18: The actual organization shown as an object diagram.

Business Behavior View

The Business Behavior view illustrates both the individual behavior of resources and processes in the business as well as the interaction between several different resources and processes. The behavior of the resource objects is governed by the Business Process view, which shows the overall main control flow of the work performed. The Business Behavior view looks into each of the involved objects in more detail: their state, their behavior in each state, and possible state transitions. The Behavior view also shows the interaction among different processes, such as how they are synchronized. Used in this way, the Behavior view becomes an important tool for allocating the precise responsibility of the various activities, and for defining the exact behavior of each resource that takes part in each process.

The combined states of the processes and the objects define the current condition of the system. States are changed through the operation of the system, that is, through the processes. Remember that it is actually the resources that perform the work of the business; the process only drives or coordinates the work of many resources. When the state of a process is altered, events are generated to notify other processes about the state change. A state also decides what may happen, which actions will occur if a state changes, and how an object can be made to enter a specific state (i.e., which events must be generated to purposely alter the state of an object). State is thus an important part of the Behavior view, as are actions and events. The Behavior view is defined by the UML dynamic diagrams: statechart, sequence, collaboration, process, and assembly line diagrams.

What is the difference between the Business Behavior view and the Business Process view? The latter illustrates the activities of the system, the transformations and the functionality, while concentrating on the interactions among the resources, goals, and rules in the business. The Behavior view illustrates the dynamic behavior of each of the objects involved in these activities. Some activities are described at a more detailed level, and interactions and responsibilities that are not visible in the Process view are defined. Naturally, there must be consistency between these two views.

The Business Behavior view is fairly detailed and normally is created by the process modelers, with the support of a business architect who ensures that these models are consistent with the business process diagrams.

State Modeling

State modeling shows the behavior of an individual resource by identifying the possible states of a resource and the behavior of the resource object in each state. The behavior of a resource is depicted using UML statechart diagrams with the following key concepts:

States. The different states the object can have, including its start and end states.

Events. The cause of a state transition, in which the state of the object is changed to another state. The events that can be sent to a resource are shown as operations in the resource class.

Actions. The activities performed either in a specific state or when going from one state to another. The actions performed are modeled as the actions taken within a operation in the resource class.

Normally, the states of resources, not processes, are shown when modeling states. The different states of a process are the activities (i.e., subprocesses) in the process. A statechart diagram for a process is very similar to a process diagram and doesn't add any significant information.

Statechart Diagram

Figure 4.19 shows a statechart diagram for the Stock Order resource, which represents a stock order that is created when an order is received from the customer. At some point, a communication with a marketplace will put this order onto the market. The marketplace will then try to match this order with other orders on the market, that is, match a buy order with a sell order. When a match is reported from the marketplace, the order is marked as concluded, and an action creates a security holding that represents the shares bought. An order can also be canceled and withdrawn from the market; or the trading day can end without making a match. According to the statechart diagram, there must be an explicit decision to put the order back onto the market for the next day. If it's not put back on the market, it will be marked as a canceled order.

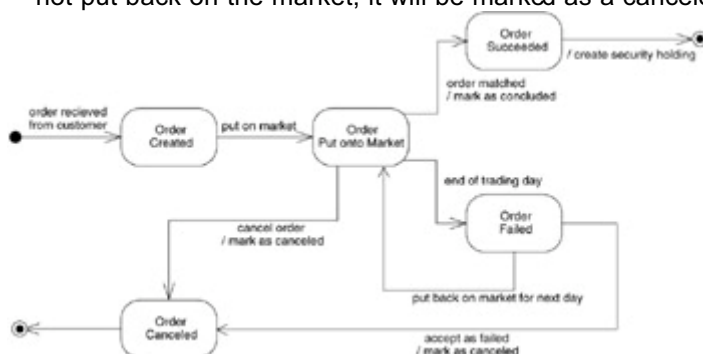


Figure 4.19: A statechart diagram for the Stock Order resource.

Interaction Modeling

The behavior of a business system is also composed of the interaction between processes and the interaction between resources. These interactions can be shown in the dynamic diagrams of UML, such as the sequence or collaboration diagram; this is

called interaction modeling. The statechart diagram models the individual behavior of a specific resource, while the sequence or collaboration diagrams show the behavior, the interaction, that occurs among several different resources.

To complicate matters, the interaction between resources is also a part of a process and can be illustrated in a process or an assembly line diagram. Sequence or collaboration diagrams should be used to show only the details of a process, for example, how a specific calculation is performed or how a detailed interaction between specific resources looks. The primary activities and interactions remain in the process diagram.

Sequence and Collaboration Diagrams

The traditional technique for depicting interactions between objects in UML is to draw sequence and collaboration diagrams. Both of these diagram types show how a set of objects interact through operation calls in a specific scenario. Sequence and collaboration diagrams can be used to show the detailed cooperation of a number of resource objects. This cooperation is part of an overall process as well, but is viewed as too detailed to include in a process or an assembly line diagram.

The interaction depicted in a sequence and collaboration diagram is triggered by a reference from a process to an object in an assembly line diagram. The sequence and collaboration diagrams show the detailed interaction between objects placed in different packages. A reference from a process to a assembly line package in an assembly line diagram triggers the interaction among a number of resources in the assembly line diagram. The interaction is modeled by allowing the resources to call operations on each other.

[Figure 4.20](#) is a sequence diagram that illustrates how a security price is updated. This interaction is triggered by the actual trading of securities, a process outside the business. This sequence diagram shows how the price is distributed to the resources within this business. The objects in it might be in an information system, but they might also be resources that interact as part of a process. The sequence diagram, which is read from top to bottom, highlights the order of a specific interaction.



Figure 4.20: A sequence diagram showing how a price update of a security is distributed to other resources.

[Figure 4.21](#) is a collaboration diagram that shows how a portfolio value is calculated. Again, this is a detailed description similar to the description of an algorithm; the objects are part of an information system. This interaction can be triggered from all processes that need to know the value of a customer's portfolio, such as a process that allocates credit to a customer or one that produces holding information to a customer.

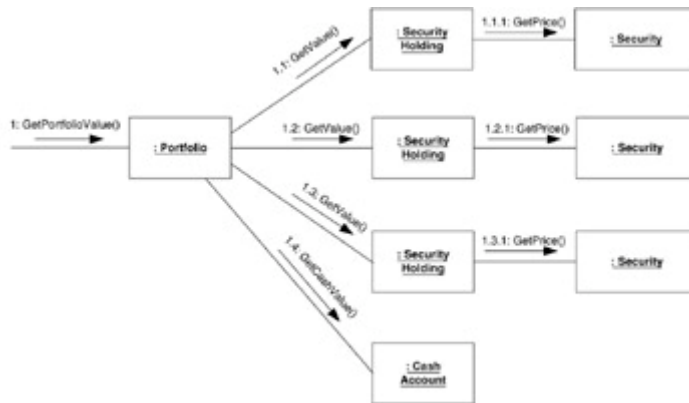


Figure 4.21: A collaboration diagram showing how the total value of a portfolio is calculated.

Sequence and collaboration diagrams both show an interaction, and the modeler can choose which one to use. In general, a sequence diagram emphasizes the sequence in time, whereas the collaboration diagram emphasizes the relationships between the objects (since it is an object diagram in which operation calls between objects have been added). It is also possible to use just one of these diagrams consistently throughout a project, to simplify learning UML syntax and to avoid using too many different diagram types.

Process Diagram

The interaction between processes can be shown in a process diagram, which you will recall is an UML activity diagram with stereotypes from the extensions. The output objects from one process are the input objects to another process. This is the case when the subprocesses of a process are shown, as well as when two independent processes are shown in the same diagram. Note that a process is not modeled as a single class like a resource; it is actually an abstraction of the interaction among and the activities performed by a number of resources. Complicating matters is that a process can have a Process Support class in an information system, but it rarely handles the entire process—the process is not performed entirely within the information system.

[Figure 4.22](#) illustrates how two different processes, A and B, are placed in the same process diagram. Swimlanes indicate the organization of the business. Note that objects created by subprocess A3b are input objects to subprocess B2. Also note the example of how processes are executed in parallel: subprocess A3a and A3b are run in parallel, and objects created by A3a are continuously used as input objects by A3b. The solid lines between the processes show the control flow of the processes and the dashed lines show the object flow between processes.

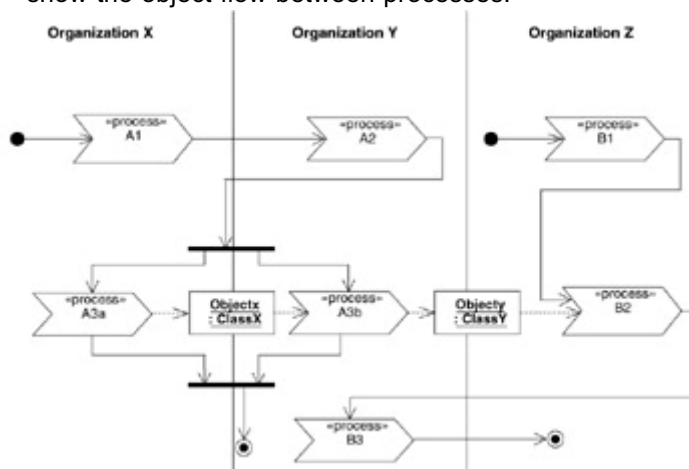


Figure 4.22.: A process diagram showing two processes and their interaction.

The assembly line diagram also illustrates the interplay between processes through their interaction with any common resources. One process could create an object that is placed in a particular assembly line package and is later read or used by another

process. This interplay becomes clearly visible through the use of the assembly line diagram. [Figure 4.23](#) shows an example in which the interaction between the processes Order Handling and Conclusion of Order is performed through objects in the assembly line packages. The reference from a process to an assembly line package in the assembly line diagram is also what causes an internal interaction between resources, which, for example, are present in an information system. That internal interaction can be depicted using a sequence or a collaboration diagram, as previously discussed in this chapter.

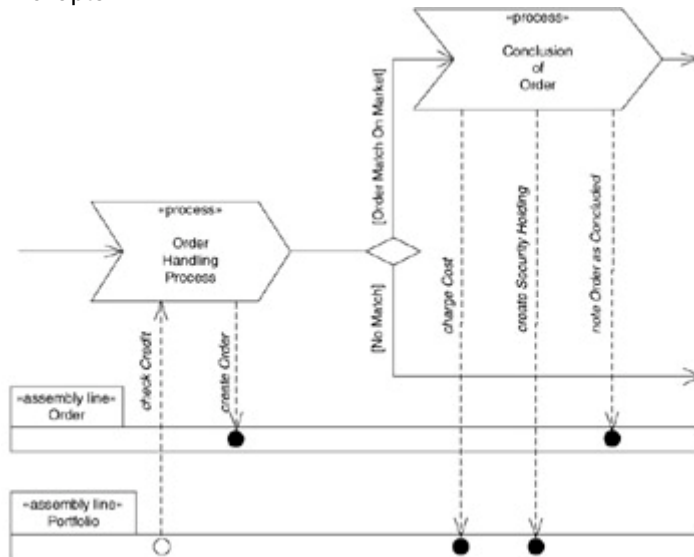


Figure 4.23: An assembly line diagram showing the interaction between processes through common resources in the assembly line packages.

[1][Darnton 97] Darnton, Geoffrey and Moksha Darnton. *Business Process Analysis*. Cambridge, U.K.: Thomson Business Press, 1997.

[2][Weirich 82] Weirich, H. *The TOWS Matrix: A Tool for Situational Analysis*. Long Range Planning, 1982.

Summary

The Business Vision view shows the future strategy for the business, and is created in collaboration with the upper management or business owners. The company is viewed in context to its customers, competitors, and expected technological and political changes in order to formulate the strategy, a vision of where the business is headed. The strategy is expressed in a textual document, called a vision statement, communicated to all employees in the business and used as input for modeling the views.

The vision statement is supplemented with a conceptual model and goal/problem diagrams. The conceptual model shows all the important concepts in the business and their relationships to each other. The goal/problem diagram set shows the high-level goals in the vision statement broken down into subgoals, analyzed in relationship to each other; it also points out the problems hindering achievement of the goals.

A goal/problem model is expressed in UML class and object diagrams. It leads to an action plan, in which temporary problems can be solved immediately but ongoing problems are allocated to continuous processes in the business.

The Business Process view is at the center of business modeling. It describes the processes in the business along with their goals, resources, and activities. Resources that are consumed, produced, used, or refined during a process include people, material, energy, information, and technologies. A process is described in a process diagram,

which is a UML activity diagram with suitable stereotypes to link resources and goals to processes.

The process diagram shows the process and the control flow of how subprocesses are linked to each other (a subprocess is called an activity when it can't be broken down further); it also shows stereotyped object flows to the resources, which are either input, output, or used in the process. A process can also have a dependency to a goal from the goal/problem model. Resource objects that take part in the process can either be supplying objects that feed the process or controlling objects that run the process. The organization of the business can be shown in context to the process through the use of swimlanes, whereby the processes are placed in a swimlane for a specific organization or span several organizational units. A process diagram also contains symbols for receiving or sending business events, where a business event is a means to communicate between processes. A process is triggered by receiving an event; it generates an event when it is finished with its task.

An assembly line diagram is an extended process diagram in which a number of assembly line packages have been added below the process diagram. These packages, represented as UML object packages, can be used to contain a type of information object in an information system. The assembly line diagram can then illustrate the references from the process to the information system—that is, how objects in the information system are used, written, or read during the process and the sequence of those references. This diagram can be used later to define the use cases that depict the functional requirements of the information system. The assembly line diagram is used to link the business models to the use cases in the software models, thus creating the opportunity to trace requirements of a software system all the way to the processes that the system supports. (It also is used to validate the requirements. This connection is discussed in [Chapter 10](#), “From a Business Architecture to a Software System Architecture.”)

The Business Structure view shows different structures of the resources in the business. It can be the structure of a product or a service (showing how the product or service is assembled of various resources), of the information in the business, or of the organization in the company. Class diagrams and object diagrams define the structure in each of these cases. A class diagram defines the relationships between concepts and the rules for how they can be assembled. The object diagram shows an actual configuration at a specific point in time (i.e., an object diagram can show the organization as it is right now).

The Business Behavior view describes the behavior of individual resources or the interaction among either resources or processes. A resource is modeled with a UML statechart diagram through its states, the events that affect it, and the actions it performs in a specific state or when it receives a specific event. Interactions are shown with either of the UML dynamic diagrams: sequence or collaboration. These diagrams show how a number of resources interact in a specific situation. These interactions can be viewed as “mini-processes,” those not modeled in a process diagram but triggered by a process in a process diagram. Often, the interaction takes place inside an information system and therefore is not shown in the process diagram. Interactions between processes can be modeled in a process diagram in which more than one process is depicted. It is then possible to show the exchange of resources or the synchronization required between the processes. The interaction between processes can also be illustrated in the assembly line diagram in which one process writes objects into an assembly line package and another process uses the same object later.

The [next chapter](#) addresses business rules. Business rules are used in all of the views and diagrams presented in this chapter as a way to detail and further specify the information in them. As mentioned, UML has a recommended language, Object Constraint Language (OCL), for defining business rules, and [Chapter 5](#) shows how OCL is used to define different categories of business rules.