DEVELOPING CHEMICAL INFORMATION SYSTEMS

AN OBJECT-ORIENTED APPROACH USING ENTERPRISE JAVA

Fan Li
Merck & Company, Inc.
Rahway, New Jersey

BICENTENNIAL 1807 WILEY 2007 BICENTENNIAL

WILEY-INTERSCIENCE
A John Wiley & Sons, Inc., Publication

DEVELOPING CHEMICAL INFORMATION SYSTEMS

DEVELOPING CHEMICAL INFORMATION SYSTEMS

AN OBJECT-ORIENTED APPROACH USING ENTERPRISE JAVA

Fan Li
Merck & Company, Inc.
Rahway, New Jersey

BICENTENNIAL 1807 WILEY 2007 BICENTENNIAL

WILEY-INTERSCIENCE
A John Wiley & Sons, Inc., Publication

Published by John Wiley & Sons, Inc., Hoboken, New Jersey
Published simultaneously in Canada

Library of Congress Cataloging-in-Publication Data is available.

ISBN-13: 978-0-471-75157-1
ISBN-10: 0-471-75157-X

Printed in the United States of America
10 9 8 7 6 5 4 3 2 1

For Yingduo, Melodee, and Michael

PREFACE

Although I have published several scientific articles throughout my academic career spanning 15 years, this is my first book. I consider it both an opportunity to share my experience of developing chemical information systems for the pharmaceutical industry and an opportunity for me to learn. Therefore, I do not expect this book to be perfect. I welcome feedback from the readers so that I can improve on the material for my next book.

Hundreds of books are in the marketplace about object-oriented analysis, design, and programming. A handful of books are about cheminformatics. But no book exists about how to apply object technology to the cheminformatics domain. This book is an attempt to fill that gap.

For a long time, chemical information systems have been considered special and have been dominated by a few vendor proprietary solutions. The costs for development and support of these systems are extremely high. I strongly believe that era is over. More and more cheminformatics software vendors provide open APIs for their proprietary implementations or develop their software using open technologies altogether, which offers tremendous opportunity for organizations to acquire or develop their cheminformatics solutions at a much reduced cost and with increased productivity. There is no need to rely on a single vendor to provide end-to-end solutions. This book shows how to apply the software industry's best practices, principles, and patterns while effectively integrating vendor tools to solve chemical informatics problems.

Chemical information systems are complex. This book does not cover every aspect of them. However, it uses a chemical registration system as an example of how to use an object-oriented approach to develop systems in the cheminformatics domain.

This book assumes the reader has basic knowledge of object-oriented analysis, design and programming, UML, Java, and concepts of chemical registration and searching.

Fan Li

Edison, New Jersey
fan_li_1129@yahoo.com

ORGANIZATION OF THE BOOK

Chapter 1 gives an introduction to the book: some historical background, the purpose of the book, and some basic information on chemical information systems.

Chapters 2–8 provide some general information and guidance for developing enterprise chemical information systems using object technology and the agile iterative process. I firmly believe that both object-oriented analysis and design principles and the agile iterative process are important to the success of any software development projects. The combination of the two helps a team to do the right things and to do the things right.

Chapters 9–15 use the chemical registration system as a case study to illustrate how to develop chemical information systems using the object-oriented approach and the Java technology. Chapter 9 presents an example of capturing functional requirements using a use case specification document. Other chapters talk about the implementations of each layer of the chemical registration system. Many analysis and design techniques are presented in great detail, and there are many code examples and UML diagrams in these chapters.

Chapter 16 summarizes the key points of the whole book.

ACKNOWLEDGMENTS

During my career at Merck, I received support from many people. I would first like to thank the management team of Merck Research Information Services for Basic Research: Dr. Ingrid Akerblom, Dr. Allan Ferguson, Dr. Gary Mallow, and Dr. Sanjoy Ray who supported my idea of writing this book. Without their encouragement, this book would have not been possible.

Special thanks to the Merck Chemical Informatics Application Engineering Team: Rachel Zhao, Arik Itkis, Xiping Long, LiMiao Chang, Sean Morley, Vaniambadi Venkatesan, Jarek Pluta, Irene Fishman, Jeanette Cabardo, and Dr. Hank Owens, without whom much of my research at Merck would not have been possible. A lot of information in the book is inspired by their work. I also thank Dr. Christopher Culberson of Molecular Systems of Merck Research Laboratories, who helped tremendously during the development of the Merck compound registration system.

Also, I thank my other colleagues at Merck: John Simon, Dr. Yao Wang, Dr. Annie Samuel, Dr. Jay Mehta, James Goggin, Andrew Ferguson, and Marianne Malloy. They were all part of the Merck Chemical Registration System Project Team, and many of them shared invaluable knowledge about compound data management with me.

POSTSCRIPT

I made a career change after I finished this book. I am now working at Goldman Sachs as a Technical Lead. This book was in the production phase when I joined Goldman Sachs. I am grateful to Allen Hom and Johnathan Lewis, managing directors at Goldman Sachs, for their support. Thanks to Sue Su, who helped me to establish contact with Wiley. Also, I thank the editorial and production team at Wiley: Dr. Darla Henderson, Senior Editor, Rebekah Amos, Editorial Assistant, and Kris Parrish, Production Editor.

CONTENTS

CHAPTER 1

Introduction

1.1 BACKGROUND

In 1999, I was asked by my manager to lead an application development team to lay out a strategic plan for the next generation of chemical information systems for Merck Research Laboratories. Back then, Java technology was entering its fifth anniversary, and the J2EE 1.0 specification was just launched by Sun Microsystems. However, almost all chemical information systems used by chemical, pharmaceutical, agricultural, and biotech companies were developed using vendor proprietary technologies such as MDL ISIS, which is the de facto industry standard. Although many people recognized that the cost of licensing, developing, and maintaining these legacy systems was high, an alternative to those systems was unclear. I have to admit that there was probably no viable alternative at all back then.

Since its inception 30 years ago, object-oriented technology has been successfully applied in software development in many industries for many years. However, it is a new beast even now in the chemical informatics domain. Many chemistry software vendors have been slow in reacting to technology evolution. As a user or developer, not many technological choices are available. As an employer, it is difficult and costly to find and recruit developers who have experience in those vender proprietary development platforms. There is also a fear factor in many organizations; moving away from existing technologies to new ones, no matter how promising they may be, is risky. This risk is true even though many of the limitations of the existing technologies justify the changes: performance and flexibility are low, whereas development, maintenance, and licensing costs are high.

From the middle to late 1990s, the situation changed when major chemistry software vendors started migrating their chemical information databases from proprietary formats to Oracle-based relational databases. Another positive move was that these vendors also started releasing chemical structure data cartridges using the Oracle® Extensibility Framework. These products included Accelrys® Accord for Oracle, CambridgeSoft® Oracle Cartridge, Daylight® DayCart, Tripos® Auspyx for Oracle, and MDL® MDLDirect. These changes were caused at least in part by the competition among these vendors. These cartridges enable people to use direct SQL to query and update chemical databases, something that could only be done using vendor proprietary programming interfaces in the past. Software developers in the chemical informatics field now have the opportunity to use open, industry standards and more interesting technologies to do their work (like it or not, having fun is one of the biggest factors of software development productivity).

Having programmed in Java since its inception, I was a firm believer that Enterprise Java could be one alternative to vendor proprietary technologies. I proved to my managers that I was right when we finally released the first compound registration system using J2EE at Merck in 2003.

Chemical information systems are complex because they process chemical structures–a very special and complex sort of data. Indexing and querying chemical structure data require special techniques, and a handful of software vendors that have the domain expertise have come up with data storage and query solutions. The complexity also deterred many organizations from developing customized chemical information systems in-house. Instead, they hire outside consultants to implement these systems on their behalf. Many software developers in these consulting firms are not professional software devolopers by training but ended up becoming programmers for one reason or another. I remember during the technology boom in the 1990's, many "seasonal" programmers wanted to find IT jobs. Many of them did so simply because they were tired of what they were doing and believed IT jobs were easy and less stressful. People were under the impression that one could become a good programmer by just attending a two-week programming training course and learning how to write a "Hello World" program—a gross misperception. Software development projects are challenging and costly. They require special skills and disciplined practices, or they may fail badly.

The advantage for chemists in developing chemical information systems is obvious: they know the domain subject e.g., chemistry and what the systems are supposed to do very well. The disadvantage is that they do not necessarily know what it takes to develop enterprise strength software systems. There are certain people who know both very well, but it is not always the case. The consequence is that the systems developed can be hard to maintain and debug and are not as good in performance and scalability as you may expect. In many cases, only the person who wrote the code can understand and maintain it. I do not mean to offend anybody because this is purely due to a lack of training and experience and has nothing to do with talent. Neither am I suggesting that being trained in software engineering automatically makes a person a good software developer. In fact, many chemists working in the pharmaceutical and chemical industries have advanced degrees and have trained themselves to be good software developers. I was a physicist by training initially myself and acquired a computer science degree later in my career. I learned low coupling and high cohesion principles in graduate school. They turned out to be the two most important principles in software development that have guided me since then. Software development is both an art and an engineering discipline, which in my mind requires formal training, years of practice, and continuous learning and exploration of new and better techniques.

Chemical informatics may mean different things to different people. I am not here to provide an authoritative definition. However, as it is the topic of this book, I will give a definition from the IT aspect. Chemical informatics is about capturing, storing, querying, analyzing, and visualizing chemical data electronically. Modern chemical information systems are challenged to facilitate industry's productivity growth by effectively handling a huge amount of data. Making sure these systems are robust and high-speed is crucial to the competitive advantage of any discovery research organization. Chemical information systems usually require the following tools.

1.2 CHEMICAL STRUCTURE ENCODING SCHEMA

One of the most widely used chemical structure-encoding schemas in the pharmaceutical industry is the MDL® Connection Table (CT) File Format. Both Molfile and SD File are based on MDL® CT File Format to represent chemical structures. A Molfile represents a single chemical structure. An SD File contains one to many records, each of which has a chemical structure and other data that are associated with the structure. MDL Connection Table File Format also supports RG File to describe a single Rgroup query, rxnfile, which contains structural information of a single reaction, RD File, which has one to many records, each of which has a reaction and data associated with the reaction, and lastly, MDL's newly developed XML representation of the above—XD File. The CT File Format definition can be downloaded from the MDL website: http://www.mdl.com/downloads/public/ctfile/ctfile.jsp.

Other structure-encoding schemas are developed by software vendors and academia such as Daylight® Smiles, CambridgeSoft® ChemDraw Exchange (CDX), and Chemical Markup Language (CML), and they all have advantages and disadvantages. The MDL CT File Format is the only one that is supported by almost all chemical informatics software vendors.

Figure 1.1 is the structure of aspirin.

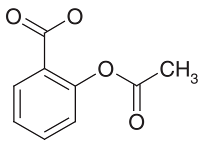

Figure 1.1 Structure of the aspirin molecule.

The Molfile representation of the above structure is as follows.

```
-ISIS−  07240513032D

 13  13  0  0  0  0  0  0  0  0999  V2000
−1.1556   −0.1291   0.0000 C   0  0  0  0  0  0  0  0  0  0  0  0
−1.1568   −0.9565   0.0000 C   0  0  0  0  0  0  0  0  0  0  0  0
−0.4419   −1.3694   0.0000 C   0  0  0  0  0  0  0  0  0  0  0  0
  0.2745   −0.9560   0.0000 C   0  0  0  0  0  0  0  0  0  0  0  0
  0.2716   −0.1255   0.0000 C   0  0  0  0  0  0  0  0  0  0  0  0
−0.4437     0.2836   0.0000 C   0  0  0  0  0  0  0  0  0  0  0  0
−0.4462     1.1086   0.0000 C   0  0  0  0  0  0  0  0  0  0  0  0
−1.1667     1.5250   0.0000 O   0  0  0  0  0  0  0  0  0  0  0  0
  0.9846     0.2897   0.0000 O   0  0  0  0  0  0  0  0  0  0  0  0
  1.7006   −0.1201   0.0000 C   0  0  0  0  0  0  0  0  0  0  0  0
  2.4135     0.2951   0.0000 C   0  0  0  0  0  0  0  0  0  0  0  0
  1.7037   −0.9451   0.0000 O   0  0  0  0  0  0  0  0  0  0  0  0
  0.2677     1.5221   0.0000 O   0  0  0  0  0  0  0  0  0  0  0  0
 1  2  2  0  0  0  0
 6  7  1  0  0  0  0
 3  4  2  0  0  0  0
 7  8  2  0  0  0  0
 5  9  1  0  0  0  0
 4  5  1  0  0  0  0
 9  10  1  0  0  0  0
 2  3  1  0  0  0  0
 10  11  1  0  0  0  0
 5  6  2  0  0  0  0
 10  12  2  0  0  0  0
 6  1  1  0  0  0  0
 7  13  1  0  0  0  0
M  END
```

The Smiles representation of the same structure is far simpler: C(=O)(O)c1ccccc1OC(=O)C.

1.3 CHEMICAL STRUCTURE RENDERING AND EDITING TOOLS

MDL® ISISDraw and CambridgeSoft® ChemDraw are probably the most widely used structure editing tools. Both companies have a Web browser plug-in version of these structure editing tools—MDL® ChimePro Plug-in and CambridgeSoft® ChemDraw Plug-in. MDL ChimePro also includes a JavaBean component, which can be used either as applets or in Java Swing based client applications.

Other products on the market include Daylight® Depict Toolkit, Accelrys® Discovery Studio ViewerPro, and Chem Axon® Marvin Bean.

1.4 CHEMICAL INFORMATION DATABASES

Data storage and querying are the most fundamental requirements of all informatics systems. Thanks to the Oracle® Extensibility Framework (a.k.a. Oracle Data Cartridge Technology), chemical structure data can be stored and queried using direct SQL and special query operators, such as substructure search, flexmatch search, similarity search, and formula search. Also, some indexing techniques make these otherwise slow searches fast. Detailed discussions about these databases and cartridges are beyond the scope of this book. Please refer to the vendor's website and product documentation for more information.

1.5 CHEMISTRY INTELLIGENCE SYSTEMS

These tools perform structure validations, making sure molecule structures follow certain conventions that are defined by an organization, property calculations such as molecular weight, molecular formula, pK_a, and so on, and salt handling. Many chemistry software vendors provide chemistry intelligence software. Some vendors may encapsulate chemical intelligence components in their data cartridge products. Some may bundle it with their structure editing tools. Some may offer it as independent products. MDL, for example, used to have it as part of its ISIS product suite. Now it has a product called Cheshire that is independent of ISIS and can be integrated with both Microsoft and Java platforms.

Since each organization has unique business rules, it is highly desirable that the chemistry intelligence software is flexible to allow customized implementations of chemistry rules handling. MDL Cheshire does a pretty good job from that perspective.

The above tools provide fundamental building blocks of chemical information systems. With these tools in place, you can pretty much develop customized solutions that meet your specific technical and business needs.

CHAPTER 2

Software Development Principles: High–Low Open–Closed Principles

One of the biggest challenges of all software projects is managing changes. This is true for several reasons. First, most programmers prefer developing new systems over maintaining existing systems because they feel the former is more challenging and creative and has a better sense of achievement than the latter. Developers do not want to spend most of their time supporting existing systems. Second, many software systems are poorly documented and hard to understand. Changes in one place may have unpredictable side effects in other places. Many software systems are poorly designed such that it is impossible to make changes without breaking the system.

However, no matter how much you hate it, changes in software systems are inevitable. Usually software systems that cannot be changed are short-lived and cannot survive when the business evolves, which happens all the time in drug discovery research. Isn't it nice that you could always add new behaviors to or alter the existing behavior of your software by adding new code without even touching the existing code? Wouldn't it be even nicer if there were proven solutions that could help you achieve this? This is exactly what software design principles and design patterns are about.

There are four fundamental and yet important software design principles—*low coupling, high cohesion, open for extension,* and *closed for changes*. We can simply call them *high–low open–closed principles*.

2.1 LOW COUPLING

The low coupling principle tells us that a software module should be loosely coupled with other modules in the system. Coupling is a measure of how strongly one module is connected to, has knowledge of, or depends on other modules (Larman, 2005). High coupling makes the system hard to understand, change, or extend.

Low Coupling Principle: Complexity can be reduced by designing the system with the weakest possible coupling between modules.

There are two aspects to coupling: one is the number of modules to which one module is coupled; the other is how rigid these couplings are. The low coupling principle says both of them should be low. Low coupling reduces the impact of changes in one module on the rest of the system. A good analogy to this is a business organization that requires collaborations between employees. A well-organized and efficient business requires only a few collaborators for an employee to do his or her work; whereas in a poorly organized business, each employee needs many collaborators to do his or her work. In such an organization, there is a greater chance that things will break.

In object-oriented software systems, there are two types of couplings. One is inheritance (also referred to as Is-A relationship). The other is composition (also referred to as Has-A relationship). Inheritance is a more rigid coupling than composition and should be avoided if possible. In an inheritance hierarchy, changes in the interface or in the base class impose the same changes in all the subclasses. This is not necessarily a bad thing as long as all classes in the same class hierarchy share the same behaviors. (I mean behavior at the interface level, not at the implementation level, because each class in the hierarchy can have its own implementation of the behaviors.) In fact, inheritance gives you the benefits of code reuse. However, if classes in a class hierarchy do not always have the same behavior, then inheritance is not a good choice; in which case, you should consider using composition.

Figure 2.1 shows coupling by inheritance and how changes in Base propagate to all its concrete subclasses.

In a composition relationship, one object can shield changes in another object that it "owns." In Figure 2.2, Class1 owns Class2. Changes in Class2 are hidden to the clients of Class1 because Class1 wraps Class2. Figure 2.2 shows coupling by composition.

Composition is a very powerful technique and is used in many Gang of Four (GoF) design patterns (Gamma et al., 1995) such as *Strategy, State,* and *Command.* You can further reduce coupling by having Class1 referencing an interface or an abstract class instead of a concrete class as in Figure 2.3. This design enables the system to dynamically swap implementation Class2 and Class3 at runtime. Figure 2.3 shows coupling by composition through interface.

This kind of reduced coupling has direct benefits to the goals of open-closed principles as you can see later in this chapter.

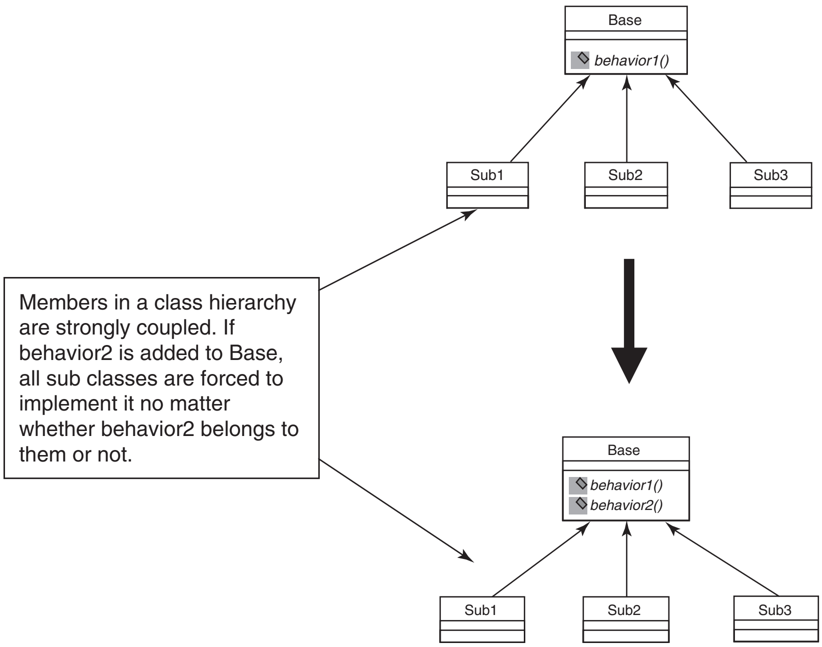
Base
behavior1()
Sub1
Sub2
Sub3
Members in a class hierarchy are strongly coupled. If behavior2 is added to Base, all sub classes are forced to implement it no matter whether behavior2 belongs to them or not.
Base
behavior1()
behavior2()
Sub1
Sub2
Sub3

Figure 2.1 Coupling by inheritance.

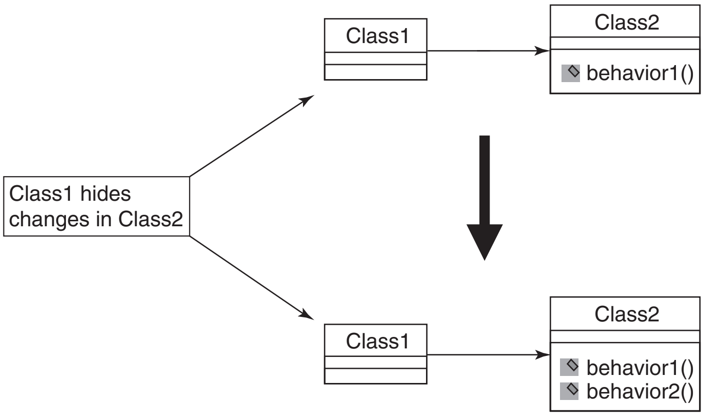

Figure 2.2 Coupling by composition.

2.2 HIGH COHESION

Cohesion is a measure of how strongly related or focused are the responsibilities of a module. A module is highly cohesive if its responsibilities are highly focused, which can be translated to the notion that a module's responsibilities should all be related. Or to be more extreme, a module should have only one

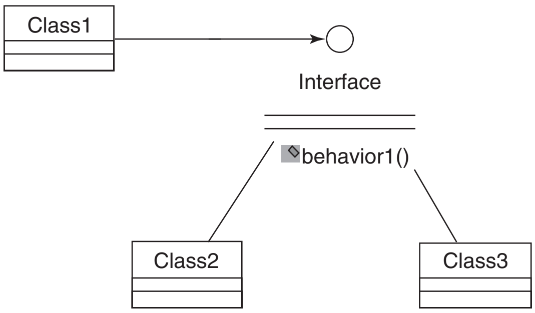

Figure 2.3 Coupling by composition through interface.

responsibility or one reason to change. Robert Martin's (2003) book has very good explanations about the high cohesion principle.

High Cohesion Principle: Responsibilities of a module should be highly related and focused so that the module has only one reason to change.

Some techniques can help you to achieve high cohesion—one of which is to use descriptive names for your classes and methods. Descriptive names can help you to keep the classes and methods focused. When you add responsibilities to your classes or methods, think about whether these responsibilities have any relevancy to the names of the class and method. If not, most likely it does not belong there. Never use ambiguous names for your classes and methods because they make the code hard to understand and most likely lead to low cohesive design. The same rule applies to member and local variables. Here are some bad names: MyClass and myMethod. These names should never be used in your code (although I use these names in this chapter to describe some concepts, they are not recommended in the real world). Here are some good names: Molstructure, ChemistryConventionChecker, and CompoundRegistrationService. Another technique is to keep the module short. If the size of a class or a method is large, usually it is a bad sign indicating the class or method is not focused enough and you should consider moving some of the responsibilities out of the class or method.

High cohesion makes the system easy to understand, reuse, and extend.

2.3 OPEN FOR EXTENSION AND CLOSED FOR CHANGES

These two principles are closely related.

Open (for Extension) – Closed (for Changes) Principles: Modules should be open for extension and adaptation and closed for modification in ways that affect its clients.