

目次

序文	3
謝辞	5
本書の構成	7
はじめに	14

第一部 ソフトウェア工学入門

第1章 ソフトウェア工学 入門編	19
1.1 ソフトウェアエンジニアリングとは何か?	19
1.2 ソフトウェアプロセスとオブジェクト指向設計概説	21
1.3 オブジェクト指向ソフトウェアプロセス	23
1.4 オブジェクト指向とオブジェクト指向言語	24
1.5 オブジェクト指向言語 Java	25
1.6 コンポーネント指向、モデリング、UML	27
第2章 ソフトウェア開発の現状	31
2.1 新しい流れ オープンソース	31
2.2 現状分析 CASE ツール、プロトタイピングツールの問題点	32

第二部 IIOSS の概要

第3章 IIOSS プロジェクト	37
3.1 IIOSS プロジェクトの概要	37
3.2 開発立ち上げに至る経緯	38
3.3 設定された目標	38

3.4	IPA への応募	39
3.5	オープンソース方式での提案	41
3.6	開発メンバーの結集	41
3.7	9社1大学によるコンソーシアムの結成	42
3.8	広域分散環境での開発作業	42
3.9	インターネットを介してのゆるやかな連帯	42
3.10	開発に際してのシステム工学的な考察	43

第4章 IIOSS システム 47

4.1	ファシリティの概要	47
4.2	IIOSS で扱う UML ダイアグラム	54
4.3	インストールと起動	66

第三部 IIOSS ユーザーマニュアル

第5章 MEF ユーザーマニュアル 77

5.1	はじめに	77
5.2	MEF とは	78
5.3	MEF の起動と終了	79
5.4	MEF 編集操作の基礎	83
5.5	プロジェクトとは?	103
5.6	ダイアグラムの編集	110
5.7	より UML らしく——情報の付加	192
5.8	高度な使い方	205
5.9	ダイアグラムの診断	241
5.10	ヘルプメニュー	246

第6章 MDF ユーザーマニュアル 249

6.1	はじめに	249
6.2	MDF とは	250
6.3	MDF の起動と終了	251

6.4	MDF の基本操作	252
6.5	MDF の基本画面	263
6.6	MDF の高度な使い方	268
6.7	おわりに	272
第7章	IBF ユーザーマニュアル	273
7.1	はじめに	273
7.2	IBF とは	274
7.3	IBF の起動と終了	276
7.4	モデル情報の読み込み	278
7.5	エディタ呼び出し機能	279
7.6	GUI の編集 / 生成	282
7.7	データベースインターフェイスの編集 / 生成	296
7.8	ネットワークインターフェイスの編集 / 生成	306
7.9	コンパイル / 実行管理機能	310
7.10	おわりに	315
第8章	FCF ユーザーマニュアル	317
8.1	はじめに	317
8.2	FCF とは	318
8.3	XMI から Java への変換 / xmi2java	319
8.4	Java から XMI への変換 / java2xmi	322
8.5	おわりに	328
第9章	DBF ユーザーマニュアル	329
9.1	はじめに	329
9.2	DBF とは	331
9.3	サーバのセットアップ	331
9.4	画面を使ってデータベースを管理する	338
9.5	DBF API について	354
9.6	OQL を使ってデータベースを検索する	357
9.7	おわりに	361

第 10 章 IDE ユーザーマニュアル 363

10.1	この章について	363
10.2	IDE とは	364
10.3	IDE をご利用になる前に	365
10.4	IDE の起動と終了	366
10.5	ファイル管理	368
10.6	プロジェクト管理	370
10.7	ヘルプの作成	372
10.8	プロキシ設定	377
10.9	ファシリティメニュー	377
10.10	Java 実行環境設定	380

第 11 章 MEF を使ったモデリング 383

11.1	はじめに	383
11.2	クラス図の作成	387
11.3	コラボレーション図の作成	390
11.4	シーケンス図の作成	394
11.5	ステートチャート図の作成	397
11.6	アクティビティ図の作成	399
11.7	MDF でのシミュレーション	402
11.8	Java ソースプロトタイプの生成	404

Appendix

Appendix A	機能比較表	409
-------------------	--------------	------------

Appendix B	メニュー一覧	411
-------------------	---------------	------------

Appendix C	用語集	417
-------------------	------------	------------

Appendix D	参考文献	431
-------------------	-------------	------------

開発者略歴	437
著者略歴	444
IIOSS マスコット「タンチョーくん」作者略歴	446
索引	447

はじめに

「IIOSS」は、「Integrated Inter-exchangeable Object-modeling and Simulation System for Open Source Software Environment」の略称で、「イイオス」と発音します。IIOSS はオープンソースのライセンスで公開されている、オブジェクト指向の設計 / 開発支援ツール群で、Linux 上で 100% Java で開発されています。

現在のオブジェクト指向の設計 / 開発の重要な技術の 1 つに、I. Jacobson、G. Booch、J. Rumbaugh 等によって考案された UML (Unified Modeling Language) があります。UML を使うことで、システムの仕様や構造の記述をオブジェクト指向的に行うことができます。UML は、OMG (Object Management Group) によって標準化が推進されており、現在では業界標準になっています。関連書籍も数多く出版されていますが、本当の意味での本格的な普及はこれからです。IIOSS は、標準的なオブジェクト指向技術 (UML、Java、XML/XMI など) に焦点を当てて開発されています。もちろん、UML でのシステムのモデリングや記述を可能にする UML エディタ (MEF: Model Editing Facility) も含まれています。MEF を使って描いた UML モデルをシュミレーションし、その可動性を検証し、インタラクティブにデバッグすることもできます。UML のクラス図から、Java ソースコード (スケルトン) を自動的に生成することもできます。しかも、IIOSS はオープンソースソフトウェアです。誰もが自由に無料で入手し、使用することができます。このような素晴らしいソフトウェアが無料で公開されているのには、何か裏があるに違いない、と思われる方がいらっしゃるなら、IIOSS が IPA (経済産業省の外郭団体である「情報処理振興事業協会」) の予算で開発された、つまり、われわれの税金の一部を使って開発された、ということを感じておいてください (この辺のいきさつについては、「3.4 IPA への応募」をお読みください)。

「IIOSS」は、オープンなソフトウェアプロセスを支援するコンソーシアムによって開発されました (詳しくは、「3.8 9 社 1 大学によるコンソーシアムの結成」をお読みください)。すでに触れたとおり、IIOSS の開発予算を提供したのは、IPA です。

読者の中には、オブジェクト指向の設計 / 開発支援ツール群がなぜオープンソースなのかと思われる方も多いと思います。

現在、設計支援ツールは数種類が商品化されているのは事実です。しかし、考えてみてください。それらは、いずれも非常に高価なだけでなく、特定の OS 等への依存性が高いものばかりです。これらの高価な市販製品となんら遜色なく、しかも機能によってはより優れていて、(1) ユーザーが無料で入手して自由に使い、(2) カスタマイズや改良を自由に施すことができ、かつ、(3) プラットフォーム依存性のない設計支援ツールはほとんど存在しない、というのが現状です。オープンソースのライセンスで公開されたソフトウェアは、そのバイナリやソース

コードを誰もが自由に入手できます。そしてそのソフトウェアを自分好みに自由に改変(改良)することができます。変更したソフトウェアを公開するには、オープンソースのライセンスを使用するのが原則です。これは、公開されたソフトウェアが、未来永劫、自由に入手可能な状態にありつづけることを意味します。ソフトウェアが改良され続け、良い方向へ発展していく可能性を示唆しています。われわれが、IIOSS システムをプラットフォームへの依存性が少ない Java で開発し、オープンソースのライセンスで公開し、誰でも自由に無料で入手して使えるようにした理由もまさにここに 있습니다。IIOSS を多くの人に利用してもらい、改善 / 改良に参加してもらうことでオブジェクト指向設計 / 開発支援ツールとして発展させていきたい。これがわれわれの希望なのです。もちろん、読者のあなたにも IIOSS の開発に参加していただきたいと願っています。

では、なぜ今さらオブジェクト指向の設計 / 開発支援ツールなのでしょう？ これも、当然すぎるほど当然な疑問です。現在、オブジェクト指向の設計 / 開発支援ツールが今まで以上に重要になっているのは、情報システムの構築やソフトウェア開発の世界が、設計 / 製造上での規格化や共通化の面において、一般に考えられているよりもずっと「ローテク」だからです。ソフトウェアの世界では、機械や建築などの世界で広く実践されているような、機械部品や建築材料と同じように、ソフトウェアのインターフェイスを規格化し、部品化を通じて共通化を図るアプローチがまだまだほとんど採用されていません。産業として 100 年以上の歴史を持つ機械や建築の世界では、設計図面は万国共通といってもよい標準の形式で表現されます。また、部品の共通化も行われており、機械や建築物を効率良く作製するための工夫がなされています。しかし、ソフトウェアの世界では、設計図面(設計書)の標準化や、プログラム / ソフトウェアを部品化するための取り決めがいまだに統一されていません。オブジェクト指向の開発手法は、そのような状況を改善し、高品質で、ソースコードが理解しやすく、保守も容易なソフトウェアを作成するための技術やノウハウを裏付ける開発手法として登場しました。一般に、これらの技術やノウハウを理論的に研究する分野はソフトウェア工学(ソフトウェアエンジニアリング)と呼ばれています。本書では、IIOSS のインストール方法や使用方法について説明する前に、ソフトウェア工学の現状と、IIOSS がオープンソースソフトウェアとして開発されるに至った経緯、および IIOSS の全体像についてできるだけ簡潔に述べます。

第3章 IIOSSプロジェクト

3.1 IIOSSプロジェクトの概要

IIOSS プロジェクトは、情報処理振興事業協会(IPA)の平成 10 年度「ビジネスオブジェクト関連システム開発事業」への応募案件としてスタートしました。応募テーマは「オブジェクト指向設計およびプロトタイピング統合開発環境の開発」でした。ここから推測されるとおり、IIOSS システムはオブジェクト指向技術に基づくソフトウェア設計 / 開発支援ツール群で、UML のモデルエディタ、モデルデバッガ、インターフェイスビルダ、モデルとソースとのコンバータ、オブジェクトデータベースなどを含んでいます。本書の執筆時点(2001 年 4 月)において IIOSS システムには、Java で実装され、相互に連携して動作する、以下の 6 つのサブシステム(ファシリティ)があります。これらはすべてオープンソースで公開されています。

- **モデル編集ファシリティ** (Model Editing Facility : MEF)
Rational Rose のような、UML (Unified Modeling Language) モデルエディタ
- **モデル検証ファシリティ** (Model Debugging Facility : MDF)
UML モデルのデバッガとシミュレータ
- **インターフェイス構築ファシリティ** (Interface Building Facility : IBF)
GUI、ネットワーク、データベースの各種インターフェイスの定義 / 編集、インターフェイスの生成ツール
- **フォーマット変換ファシリティ** (Format Conversion Facility : FCF)
XML による UML モデルと Java ソースプログラムのフォーマットとの変換ツール
- **データベースファシリティ** (DataBase Facility : DBF)
ネットワーク環境で利用可能な Java によって実装されたオブジェクト指向データベースシステム

・統合開発環境(Integrated Development Environment : IDE)

IIOSS 全体の統合管理ツール

IIOSS プロジェクトは、1998年3月末より2000年9月までの、およそ18か月間にわたって推進されました。その間、開発と実証実験が何度か行われました。2000年の7月には、ソフトウェア開発環境展(東京ビックサイト)に出展し、開発の途中のアルファ版をソースコード付きで開示しました(これは、IPAの公募開発としては、大変珍しいことです)。その後、ソースコードCD-ROMの配付、IPA成果発表展示会への参加、専門雑誌への寄稿、解説書の出版等、本格的な普及活動のための準備を進めています(IIOSSプロジェクトの最新情報については、IIOSSプロジェクトのWebサイト(<http://www.iiooss.org/>)をご覧ください)。

3.2 開発立ち上げに至る経緯

筆者らは、UNIXやオブジェクト指向技術を活用して、10数年にわたってさまざまなソフトウェア開発や情報システムの構築を行ってきました。その開発および構築の経験の中で、常々、以下のような問題に悩ませされてきました。

- ・ソフトウェアエンジニアリング研究の成果が設計/開発の現場にあまり活かされていない。
- ・オブジェクト指向技術が、設計/開発の現場で、正しく理解、活用されていない。
- ・設計情報を開発者間で流通する枠組みがない。
- ・上流工程の設計支援ツールと、下流工程の開発支援ツールがほとんど連携していない。
- ・本来エンドユーザコンピューティング(EUC)や小規模なプロトタイピングのためのツールが、比較的大規模な企業システムの構築に誤用されることによる弊害が多い。
- ・ソフトウェアモジュールの「部品」としての標準化/共通化/再利用があまり行われていない。
- ・特定の海外企業の製品である、設計ツール、開発ツールへの依存度が高い。

これらの課題に対して、筆者らは、技術教育と設計/開発の現場で共通のツールとして自由に利用できる、ソフトウェア設計支援ツールや、開発支援ツールの必要性について議論してきました。このような問題意識と議論が、IIOSSプロジェクトのコンセプトの原形となっています。

3.3 設定された目標

われわれは、IIOSSのプロジェクトの企画立案に際し、以下の目標をかかげました。

- ・実用的な、オブジェクト指向設計支援ツール、プロトタイピング開発環境、およびコンポーネント構築支援ツールを構築し、広く普及させる。

- ・上流工程の設計支援ツールやモデリングツールの使いにくさ、および上流工程のツールと下流工程の開発ツールとの連携の欠如に対応するだけでなく、プロトタイピングツールを用いた開発における上流設計工程の不足や欠如に対応できるシステムとする。
- ・オープンソースソフトウェア(少なくとも POSIX 準拠^{*1})のプラットフォーム上で開発する(この目標は、Linux 上での開発と実証実験に現れています)。
- ・ポータブルで拡張性の高いシステムを構築する(この目標は、100% Java による実装に現れています)。
- ・構築に際して、既存のオープンソースソフトウェアのソースコードを活用し、開発の効率化を図る。
- ・オープンソースソフトウェア環境下での、新しいシステム開発、保守の枠組みを構築する。
- ・開発システム自体のオープンソース化と、参考書の出版などによって、ソフトウェアエンジニアリングを身につけた技術者の育成の促進、業界の活性化に寄与する。

3.4 IPAへの応募

正式名称を「情報処理振興事業協会」という IPA(<http://www.ipa.go.jp/>)は、「情報処理の促進に関する法律」に基づき、日本の情報処理の振興を図ることを目的に、1970年10月に設立されました。ドメインが、"go"であることから明らかなように、経済産業省の管轄の特殊認可法人(所謂外郭団体)です。IPAは、以下のような事業を行っています。

1. 情報処理事業等に対する支援事業
2. 基盤技術関連事業
3. 応用実用化技術関連事業
4. 教育、人材育成関連事業
5. 情報システムに関するセキュリティ対策関連事業
6. 補正予算による情報処理、情報化促進などの事業

もともと、IPAのソフトウェア開発、情報技術開発を支援するプロジェクトは、情報処理産業の振興に寄与するような、企業の研究開発や製品技術開発(商品開発)を助成する案件が多かったようです。しかし近年では、成果をオープンソースにするプロジェクトもいくつか現れています。われわれが、IIOSSプロジェクトをスタートした1999年には、20件以上の公募がありました。

たとえば、1999年2月4日には、以下のような要件を掲げて「ビジネスオブジェクト関連シ

*1 Portable Operating System Interface, IEEE1003.

STEM開発事業」の公募が開始されました（IPAの公募資料より抜粋）。ちなみに、この公募は、「平成10年度第3次補正予算事業」の1つである「技術的環境整備システム開発推進事業」の一貫として実施されたものです。

- 事業の目的

情報通信の活用が経済社会活動に浸透する中、21世紀初頭においては、オープンなネットワーク環境下での情報化への適応がユーザー企業の事業活動になくってはならない要素となりつつある。それに伴い、今後のユーザー企業の情報化戦略は、単なる業務への情報機器の導入による効率化に留まらず、経営戦略そのものとなりうるものである。

これらのニーズに対応し、産業の情報化を実現するためには、オープンなネットワーク環境下において、企業の情報化をきめ細かくサポートする環境や企業体制を整備することが極めて重要な課題である。

したがって、本事業は、従来から情報サービスを提供してきた情報処理産業が、上記課題を解決するために必要となる情報技術やシステム開発等を実施することにより、我が国の情報インフラとしての産業基盤の強化を図るものである。

- 事業公募の対象

今後の情報処理産業にとっては、オープンなネットワーク環境の下、ソフトベンダ企業ごとの特徴を活かしたソフトウェアやサービス等の新たなビジネス展開が必要であると考えられる。このことから、本事業では、企業の枠を越えた複数企業の連携に伴い、個々の企業の持つ技術やノウハウの有効活用することによりはじめて実現が可能となるソフトウェア製品や情報技術の開発及び実地検証、新しいビジネスシステムサポートサービスを提供するために必要となるソフトウェア技術やシステム構築技術の開発及び実地検証、等を対象とする（実地検証は、必要に応じ対象に含めることとする）。

この公募要件に対して、われわれは、システム構築の現場で一緒に仕事をしている同年輩の仲間たちと、常日頃感じていたソフトウェア開発の、分析、設計、実装、テスト等における問題点を再検討しました。また、設計/開発支援ツールの問題も再検討し、いくつかの解決策を模索しました。そして、その結果に基づき、設計/開発の一貫ツールの開発を企画し、IPAに応募しました。われわれの企画は、自分たちが使う道具として便利だと思ふ機能を集め、上流の設計/モデリング工程と、下流のプロトタイピング、開発、ドキュメンテーション工程を統合化するツールで、しかもネットワーク/分散環境、オープンソース環境に対応するツールを、オブジェクト指向の最新技術および既存のPC用速成プロトタイピングツールの特徴を取り入れて開発しよう、というものでした。これが、IIOSSプロジェクトのはじまりです。

3.5 オープンソース方式での提案

近年、Linux、Apache、PostgreSQL、Tomcatなどに代表される、「オープンソースソフトウェア」(OSS: Open Source Software)と呼ばれる体系がソフトウェア業界の新しいトレンドになってきています。

オープンソースソフトウェアは、ソフトウェア開発業界における新しいパラダイムであり、かつ、いくつかの基盤的なソフトウェアの開発形態として少なからず成功しています。また、関係各方面から注目されています。たとえば、フランスなどのヨーロッパの一部の政府機関、米国の一部の連邦政府関連の組織や一部の州政府関連組織では、税金で購入する情報システムやソフトウェアに関して、ソースコードがオープンになっていることを「義務付け」しているところもあります。元来、IPAは、日本政府が国内の情報産業の活性化、技術促進を支援するために設立された組織であり、そのIPAの予算の原資は政府の予算、つまり「税金」ですから、そのIPAの予算を使った活動の成果を「社会的知的共有財産」としてオープンソースとするのは、まったく「道理に適っている」といえます。

IIOSSプロジェクトでは、公募への提案の最初の段階から、開発成果をオープンソース化することを全面的にうたいました。また、現在主流となっている設計支援ツールが特定の海外企業製品への依存性が高いことなども考慮して、特定の企業の製品に依存せず、誰でも自由に使える国内共通ツールの提供という意味でも、プロジェクト成果物のオープンソースでの開示を提案しました。さらに、開発環境のOSとしてオープンソースのOSの代表であるLinuxを採用することも提案しました。

3.6 開発メンバーの結集

IIOSSの最初の構想は、オープンテクノロジーズ社(<http://www.opentech.co.jp/>)からのアイデアをまとめたものを出発点としましたが、IPAの提案書作成の準備段階で、より多くのメンバーの意見を取り入れ、提案を洗練したものにするために、設計チームを拡大することになりました。これがIIOSSコンソーシアムのはじまりです。

IIOSSコンソーシアムのメンバー集めの主力となったのは、オープンテクノロジーズのSRA社出身の社員です。彼らは、自分のSRA社時代の同僚や後輩にまず当たりました。それらの人びとの中で、UNIX上での開発やオブジェクト指向技術を活用した開発などの経験が豊富で、自分で独立してソフトウェア開発を行っている人びとに声をかけました。オープンテクノロジーズ社と提携関係にあるアステックグループなどにも声をかけ、提案書の作成とコンソーシアムへの参加を要請しました。そして開発メンバーに加わった人の1人が、自分の友人である河野助教授(琉球大学工学部情報工学科)に協力を依頼しました。琉球大学からは非公式な大学院生のアルバイトグループがまず参加し、やがて多くの学生が正式なインターンシップのプログラ

ムの一貫として参加し、開発に協力してくれました。

3.7 9社1大学によるコンソーシアムの結成

IIOSS プロジェクトの IPA との開発委託契約は、株式会社オープンテクノロジーズを代表とするコンソーシアムとして締結されました。コンソーシアムのメンバーは、本書執筆時点(2001年4月)の段階で以下のとおりです。

株式会社 アステック
株式会社 アステック・イーコマス
有限会社 インフォ・ウエイブ
有限会社 エイビス システム ソリューション
小野商店
株式会社 オープンテクノロジーズ
有限会社 マルチパラダイムシステムズ
有限会社 メタボリックス
メランジ 株式会社
琉球大学 工学部情報工学科

開発チームの詳細やメンバーの略歴などは、付録の「開発者一覧」をご覧ください。

3.8 広域分散環境での開発作業

IIOSS プロジェクトの開発と実証実験には、18 か月間に、のべ 30 名以上のメンバーがかかり、全体の工数は 250 人月以上でした。これらの人びとは、埼玉(川口)、東京 23 区(練馬、新宿、文京-小石川、文京-千石)、神奈川(平塚、大磯)、愛媛、福岡、沖縄と、物理的に分散していました。IIOSS プロジェクトは、最初の段階から、開発メンバー全員が 1 か所に集まることを前提とせず、後のオープンソース化による開発継続を念頭において、完全な広域分散環境で開発を推進してきました。事実、18 か月の間、開発メンバー全員が一堂に会する会議は一度も開催されていません。こういう方式でソフトウェア開発が行われたのは、IPA の開発プロジェクトでは、IIOSS プロジェクトが初です。

3.9 インターネットを介してのゆるやかな連帯

IIOSS プロジェクトでは、物理的に分散しているメンバー同士の情報交換をほとんどすべてインターネット上で行いました。使用したツールは、インターネット上の最もプリミティブなツールである、E メールとメーリングリストです。また、設計資料やプログラム等の交換は、FTP

によって行いました。設計以降のフェーズでは、CVS サーバを使ってプログラムのバージョン管理を行いました。たとえば、オープンテクノロジー社とアステックグループは同じビルを共同使用しています。異なるフロアに事務所を構えるこの2つのメンバー間でさえも、IIOSS プロジェクトの推進に関しては、会議室でのミーティングをほとんど行っていません。情報交換は、主にメールとFTPによるやりとりを中心に行われました。つまり、IIOSS プロジェクトは、特別なツールに依存せず、オープンソースの開発で用いられているインターネット上のごく一般的なツールだけを用いて、広域分散環境での開発や情報交換を推進してきたと言えます。

3.10 開発に際してのシステム工学的な考察

IIOSS の設計は、「メソドロジスト」の諸先生によるものではなく、日常的に、システムインテグレーションやシステムコンサルテーションを行っている「現役の SE、コンサルタント」のチームによるものです。われわれは、IIOSS の設計 / 開発に際して、過去の経験とソフトウェア工学的な側面やシステム工学的な側面から種々の考察を行い、以下の項目を設計 / 開発の方針としてかけました。

- さまざまな既存のオープンソース実装系の活用と、緩やかな統合
- 物理的に分散した設計 / 開発チームへの対応
- ソースコード第一主義
- デザインパターンの利用
- Java の特性を活かす
- 業界標準への準拠

さまざまな既存のオープンソース実装系の活用と、緩やかな統合

IIOSS プロジェクトでは、当初の開発予算の制限と過大な要求仕様の問題を現実的に処理するために、ある程度の機能を網羅する実装系の開発を目標にしました。そして、すべての機能をスクラッチから開発するのではなく、既存のオープンソース実装系を活用する方針がとられました。

われわれは、基本設計と並行して数か月の調査を行い、われわれの開発の部品として活用できそうなオープンソースシステムを候補として選びだしては、IIOSS の要求仕様の一部の機能を持っているかどうかを検証しました。開発ベースとして使えそうなオープンソースシステムを見つけては、そのソースコードを注意深くレビューし、機能、クラスの構造、ソースコードの品質などを総合的に評価しました。そして最終的に、IIOSS の部品として採用するシステムを選びました。

IIOSS は、UML エディタ等の上流工程用ツール、インターフェイスビルダ等の下流工程用

ツール、OODB 等を包含しています。この点において IIOSS は、用途や機能が異なるツール群(スーツ)ですが、IIOSS のベースとして採用されたオープンソースシステムは、それぞれ相互に関係のないプロジェクトの成果物なので、それらのシステム間で無理に統一、統合を図っていません。IIOSS は、ソースコードレベルで、個々の大機能(ファシリティ)を独立したアプリケーションとして開発してあります。ファシリティ間の相互の連携は、

- Java のネットワーク機能
- XML フォーマットのデータ
- DBF(IIOSS の OODB)

などのやりとりで実現することとし、ベースソフトウェアの統合は緩やかなものになっています。ただし、異なるファシリティ間で共通リソースなどを制御する機能として、IDE(統合開発環境)的なファシリティが別途追加してあります。

物理的に分散した設計 / 開発チームへの対応

IIOSS プロジェクトは、ソースコードを公開する以前から、設計 / 開発チームが日本各地に分散してました。また、われわれは普段から、東京と愛媛、東京と福岡というような分散したチームで開発を進めているので、IIOSS の開発においても、分散した開発環境を支援する機能、分散したシステムの開発に有用なモジュールの開発を支援する機能などを重視しました。つまり、将来、IIOSS 自身の開発や、われわれの日常の開発プロジェクトで IIOSS を利用しやすくなるように考慮しました。複数の開発者間のネットワーク上での、ソースコードのバージョン管理には、ごく一般的な手法である CVS サーバを用いました。

ソースコード第一主義

一般に、古いウォーターフローモデル信奉者の中には、「自然言語語で記述した仕様書から、実際に動作するプログラムは自動的に生成されない」ことを、直感的に理解できない人が沢山います。また、「自然言語で記述されたドキュメントの内容が、プログラムの内容と一致しているかどうか検査するのは容易ではない」ことも忘れられがちです。われわれは、以下の諸点に対する考察から、IIOSS の開発においては、ソースコードそのものを最重視する方針をとりました。

- 部品として採用した既存のオープンソースシステムのソースコードの量が少なくない。
- 既存のオープンソースシステムのドキュメントは不十分でかつ最新のコードと対応していない。
- 正しく構成されたオブジェクト指向プログラムは、ソースコードそのものに保守に必要な情報が含まれる。
- Java による単一言語の実装であるため、階層間のセマンティックギャップは最小化される。

- ・オープンソースで公開したあとも、開発者にとって最も頼りになるのはやはりソースコードである。

デザインパターンの利用

IIOSS システムにおいては、われわれ独自の新機能の実装も、既存のオープンソースのシステムのクラスに新たな機能のクラスを追加する形で実装されるものが少なくありません。われわれの設計チームは、できるだけデザインパターンを用いて、将来の拡張に備えた設計を試みましたが、実際の実装においては、その設計は十分に活かされず「力づくの実装」となっている部分も少なくありません。

Java の特性を活かす

IIOSS の開発において、われわれは、スーパークラス、サブクラスの正しい構成そのものがシステムの構造をおのずと正しい形に導くとの判断から、Java の特徴を活かし、オブジェクトのたたずまいやクラスの構成が本来あるべき正しい姿を維持するように注意しました。

業界標準への準拠

IIOSS のいくつかの機能には、「業界標準への準拠」という大きな目標があります。われわれは、設計の段階で、OMG や ODMG の英文ドキュメントを注意深く調査し、仕様の内容や、実装の優先順位などについて検討しました。しかし、開発リソースなどの制約から、結果的に「部分的な準拠」に留まっている機能も多く、それが今後の開発の課題のひとつとなっています。

第4章 IIOSS システム

4.1 ファシリティの概要

IIOSS は、オブジェクト指向設計支援ツールおよび開発環境を提供しています。IIOSS は以下の6つの機能(ファシリティ)から構成されています。

- ・ **モデル編集ファシリティ**(Model Editing Facility : MEF)
- ・ **モデル検証ファシリティ**(Model Debugging Facility : MDF)
- ・ **インターフェイス構築ファシリティ**(Interface Building Facility : IBF)
- ・ **フォーマット変換ファシリティ**(Format Conversion Facility : FCF)
- ・ **データベースファシリティ**(Database Facility : DBF)
- ・ **統合環境開発**(Integrated Development Environment : IDE)

図 4-1は、IIOSS のシステム構成を図解したものです。

IIOSS の各ファシリティは、要求分析、設計、設計の検証、プロトタイピングといったソフトウェア開発を支援するものです。IIOSS は、これらの機能をお互いに連携させることによって、ソフトウェア開発のプロセス全体をサポートします。

4.1.1 モデル編集ファシリティ : MEF

MEF(Model Editing Facility)は、UML モデルダイアグラムをグラフィカルに表示 / 編集できるツールで、UML を用いたシステムの設計モデルを記述、編集するエディタ機能を提供します。UML 1.1 および XMI(XML Metadata Interchange)に準拠しており、6種類のダイアグラムを編集することができます(図 4-2)。

モデル編集ファシリティでサポートされているダイアグラムは、クラス図、ユースケース図、状態チャート図、アクティビティ図、コラボレーション図、シーケンス図です。MEF は UML を視覚的に編集する機能だけでなく、以下の機能も提供しています。使い勝手は、Rational Rose など、よく使われている市販 UML モデルエディタとほぼ同等です。

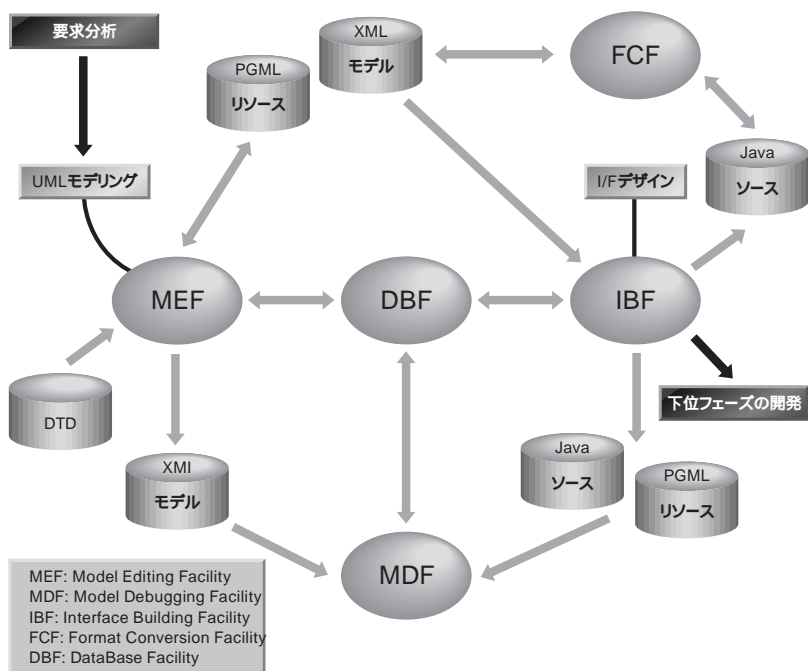


図 4-1 IIOSS システム構成図

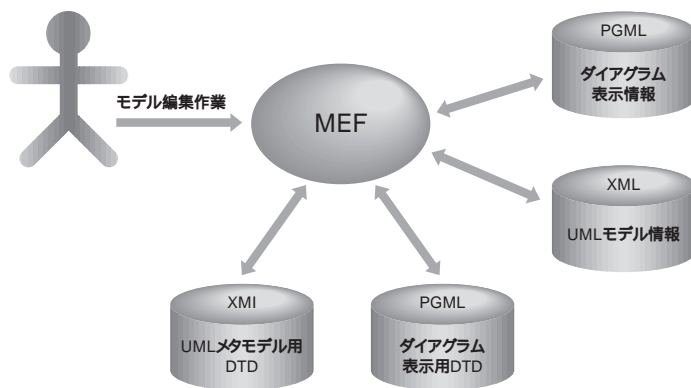


図 4-2 MEF のデータフロー図

- ダイアラムのグループ化機能
- 関連する UML モデル要素間のハイパーリンク機能
- ダイアグラムの静的診断機能
- XMI ファイルからの表示情報生成機能

MEF は、Argo/UML をベースとしています。Argo/UML は、カルフォルニア大学アーバイン校で開発されたオープンソースソフトウェアです(Argo/UML に関する詳しい情報は <http://argouml.tigris.org/> を参照してください)。MEF の機能、具体的な操作方法については 5 章で詳しく説明 / 解説しています。

4.1.2 モデル検証ファシリティ : MDF

MDF(Model Debugging Facility)は、UML モデルのシミュレーション / デバッグをインタラクティブに行うツールです。MEF と有機的に統合されているので、モデルを編集しながらのデバッグや、モデルエディタと同じ画面を使ったブレークポイントの設定などを行うことができます。UML モデルの可動性を検証できる MDF のような機能は、市販の CASE ツールにもあまり見られません。MDF は、HIOSS プロジェクトがスクラッチから実装した機能です。

MDF はモデルの可動性を検証できるので、ソフトウェア開発における初期の段階で問題点を見つけ出すことができます(図 4-3)。

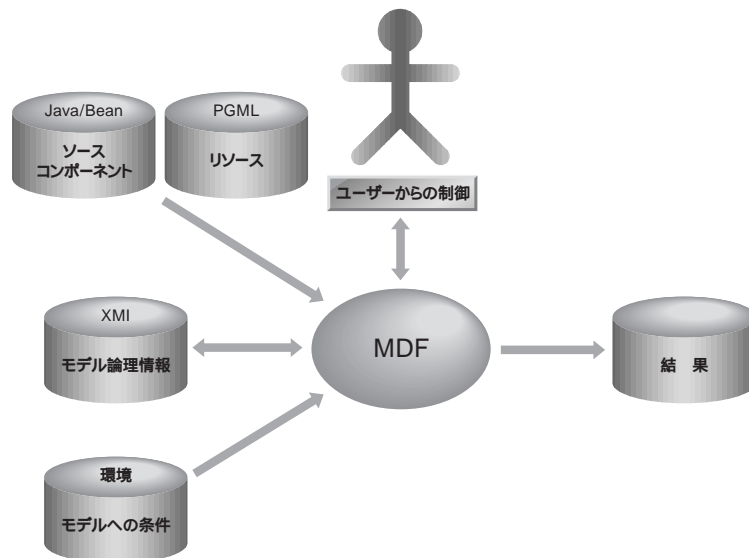


図 4-3 MDF のデータフロー図

MDF では、システムの動的な部分をモデリングする振る舞い図をシミュレートの対象とすることができます。対象にできる振る舞い図の種類は以下のとおりです。

- ・シーケンス図
- ・コラボレーション図
- ・アクティビティ図
- ・ステートチャート図

MDF は、モデルの実行過程の視覚的な表示、ブレークポイントの設定、状態変化の表示、トレース、ステップ実行、ユーザーによる条件指示など、通常のプログラムのデバッグと同じような機能を備えています。特にステートチャート図を対象に、イベントとして他のオブジェクトとメッセージ交換ができるので、複数のステートチャート図にまたがるシミュレーションを行うことができます。なお、MDF の機能や操作方法については、6 章で詳しく説明 / 解説しています。

4.1.3 インターフェイス構築ファシリティ : IBF

IBF(Interface Building Facility)は、Java 言語の各インターフェイスを作成するツールです。GUI だけでなく、データベース、ネットワークといったインターフェイスを編集 / 生成することができます。MDF で活用できるインターフェイスの試作、および UML モデルの情報(モデルレベルの設計)を踏まえたインターフェイスの構築もできます(図 4-4)。

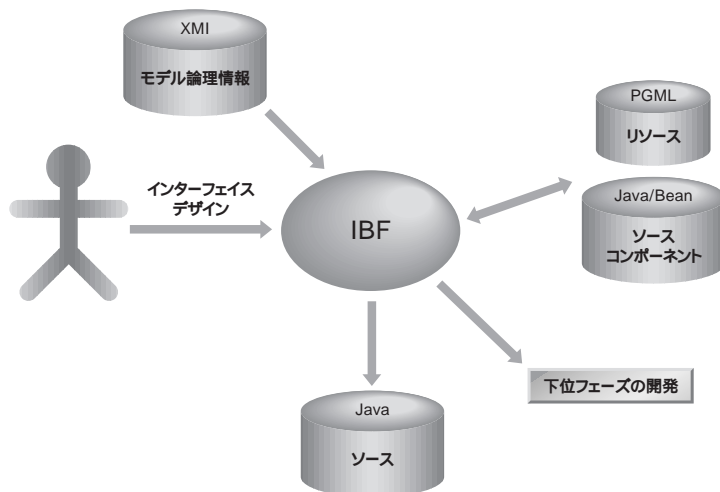


図 4-4 IBF のデータフロー図

IBF は以下の機能を提供しています。

- MEF で生成された UML モデル情報の参照 / 表示機能
- X Window 環境で動作するエディタの呼び出し機能
- Java Swing GUI の編集、生成機能
- RDB、OODB などのデータベースインターフェイスの編集 / 生成機能
- RMI、IDL などのネットワークインターフェイスの編集 / 生成機能
- 編集、生成した Java のソースプログラムのコンパイル / 実行管理機能

IBF は、Korfe をベースにさまざまな機能拡張、追加を行う形で開発されました。Korfe は、Java Lobby によって開発されたオープンソースの GUI Builder です (Korfe に関する詳しい情報は <http://jfa.javalobby.org/projects/korfe.html> を参照してください)。IBF の機能や操作方法については 7 章で詳しく説明 / 解説しています。

4.1.4 フォーマット変換ファシリティ : FCF

FCF (Format Conversion Facility) は、UML モデル (実際には UML モデルの XMI 表現) と他の表現形式との間の相互変換を行うツールです。FCF を使うことで、UML の一部として表現されているモデルオブジェクトと、Java 言語で表現された実装オブジェクト間のフォーマットを相互変換ことができます (図 4-5)。

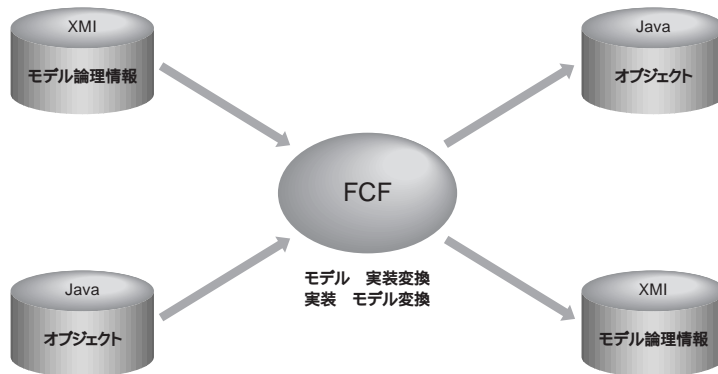


図 4-5 FCF のデータフロー図

FCF は以下の機能を提供します。

- ・モデル情報(XMI)からプログラムソースコード(Java)への変換機能
- ・既存の Java プログラムからモデル情報(XMI)を抽出する機能

FCF を用いることで、MEF でモデリングされたシステムを Java の(スケルタルな)ソースコードに変換できるため、モデリングからプロトタイピングや実装段階へのシームレスな移行が実現できます。また、すでに作成された Java のソースコードから UML のモデル情報を作成し、UML モデルから生成したソースコードの変更点などをモデルに反映させることができます。FCF の機能や操作方法については8章で詳しく説明/解説しています。

4.1.5 データベースファシリティ : DBF

DBF(DataBase Facility)は、オブジェクトのリストを格納することができる汎用オブジェクトデータベースです。ODMG 準拠(現在は一部)のオブジェクト指向データベースサーバとそのライブラリで構成され、IIOSS 全体のリソースを管理する機能を提供しています。データベースを管理するための GUI も用意されています。また、リモートクライアントからのアクセスも可能です(図 4-6)。

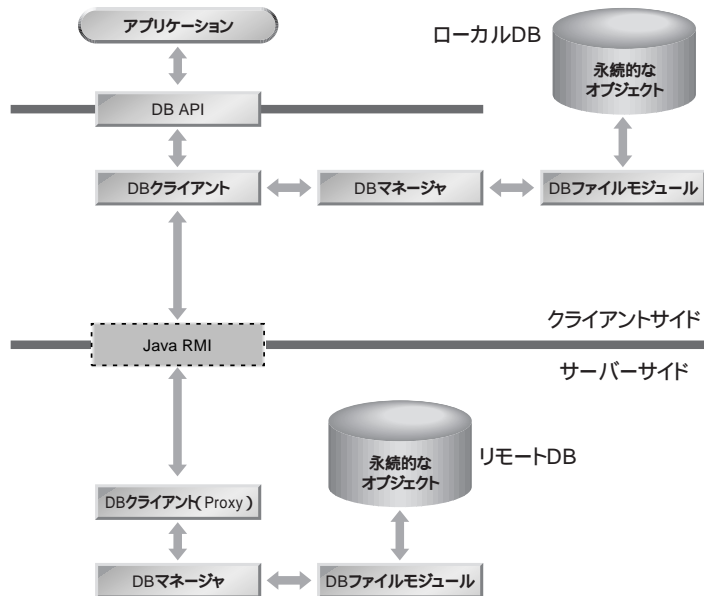


図 4-6 DBF のデータフロー図

DBF は以下の機能を提供しています。

- オブジェクト単位のバージョンング機能
- XML ハンドリング機能
- ロケーショントランスアペンシ機能
- 各種管理ツール機能

DBF を用いることで、他のファシリティのデータを永続的なデータとして、オブジェクトの関係を正しく保ったまま保存、更新することができます。

DBF は、sO(storedObjects)をベースに、IIOSS 独自の機能拡張を施し、構築されたものです。sO は、Gerhard Paulus 氏が構築したオープンソース OODB です(sO についての詳しい情報は、<http://storedobjects.sourceforge.net/>を参照してください)。DBF の機能や操作方法については9章で詳しく説明 / 解説しています。

4.1.6 統合開発環境 : IDE

IDE(Integrated Development Environment)は、すべての IIOSS ツールを統合する環境です。各ファシリティが生成したリソースを管理できます。実現されている機能は、ファイル管理、プロジェクト管理、各ファシリティの起動、ヘルプ機能などです。IDE を用いることで、これらのリソースを一元管理できます(図 4-7)。

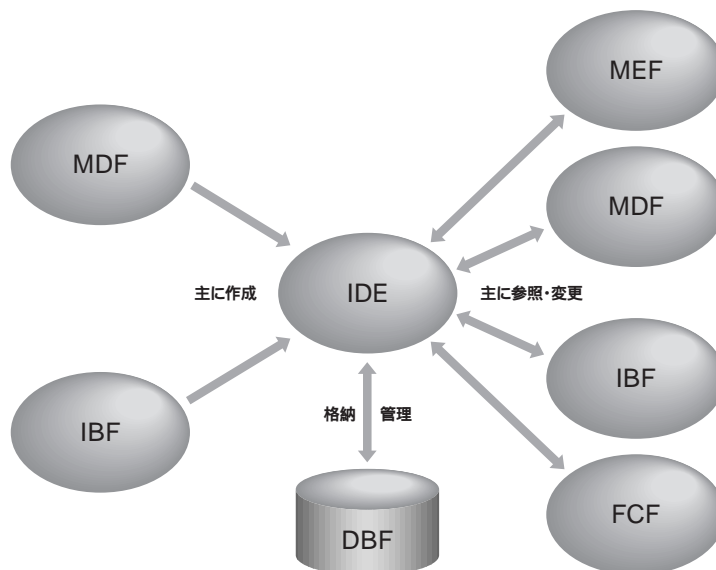


図 4-7 IDE のデータフロー図

IDE は以下の機能を提供しています。

- ファイル管理機能
- プロジェクト管理機能
- 各ファシリティの呼び出し機能
- ユーティリティ機能
- オンラインヘルプ機能

IDE を利用し、複数のファシリティに参照される可能性のあるファイルを統一的に管理し、プロジェクト全体の管理を効率良く行うことができます。IDE は IBF と同じく Korfe をベースに構築されています。IDE の機能や操作方法については 10 章で詳しく説明 / 解説しています。

4.2 IIOSSで扱うUMLダイアグラム

MEF では、UML 1.1 の中の以下の 6 種類のダイアグラムの作図ができます。

- ユースケース図
- クラス図
- シーケンス図
- コラボレーション図
- ステートチャート図
- アクティビティ図

ここでは、MEF を使ってこれらのダイアグラムの作図方法を説明する前に、これらのダイアグラムが記述する内容について、簡単な例題をもとに説明します。

4.2.1 ユースケース図

インターネットで利用できるオンラインショップを想定しましょう。オンラインショップのためのシステムを構築するためには、まず、このシステムが提供すべきサービスや機能の概要を設計する必要があります。通常はショップのオーナーの要求をヒアリングし、その内容を整理します。そして、それらの内容をショップのオーナーと確認した上で、実際のシステムの構築に取り掛かることとなります。

ユースケース図は、このような開発の段階で、システムが提供すべきサービスや機能を洗い出し、整理するのに使います。

ユースケース図は、システムのユーザーの視点から見た機能やサービスをまとめたものです。最も単純なユースケース図は、図 4-8 のようになるでしょう。

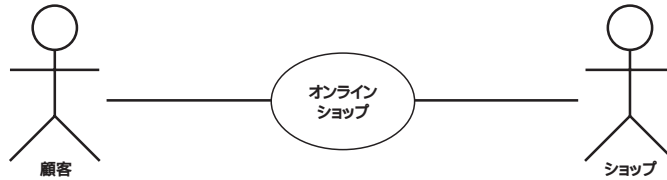


図 4-8 ユースケース図(最も単純な例)

この図は、顧客とショップの担当者がオンラインショップを利用する関係を示しています。左右の人形の記号は、それぞれ、オンラインショップの顧客とショップの担当者を表します。中央の楕円は、オンラインショップのシステムを表します。UML では、左右の記号をアクター、中央の記号をユースケースといいます。

オンラインショップのシステムでは、商品、注文、顧客を管理する必要があります。図 4-9 は、オンラインショップがこの 3 つのサブシステムで構成されていることを示します。

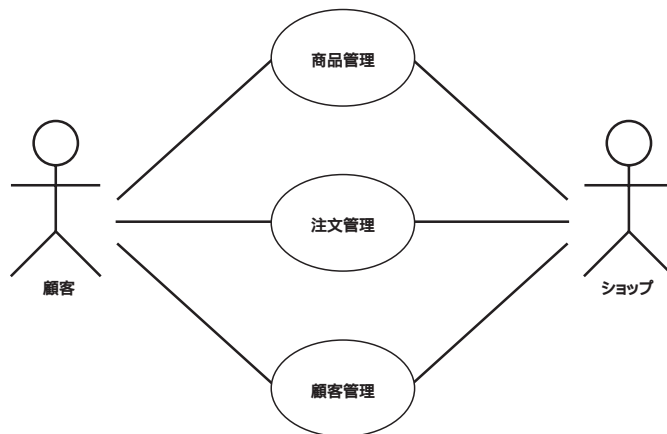


図 4-9 ユースケース図(3つのサブシステムで構成されるオンラインショップ)

それでは、商品管理サブシステムは、どのような機能で構成されるのでしょうか。商品管理サブシステムでは、商品の登録、削除、更新をショップの担当者が実行します。削除と更新を行うときは、商品を検索する必要があります。一方、顧客は、商品の検索は行いますが、登録や削除、更新は行うことができません。図 4-10 は、商品管理サブシステムをさらに機能ごとに分解して、それぞれの機能の関係を示しています。

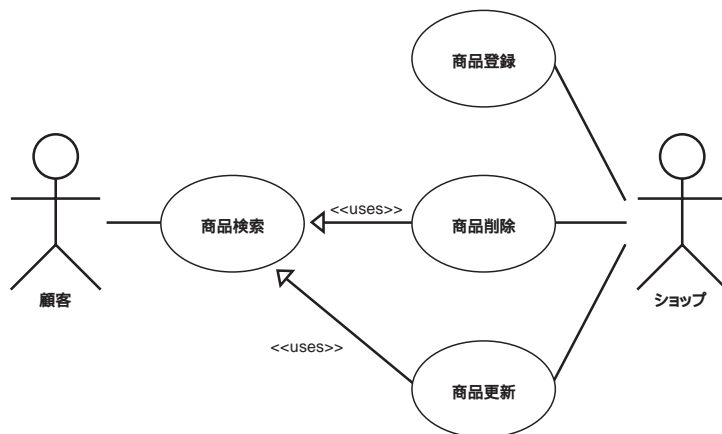


図 4-10 ユースケース図(商品管理サブシステム)

図 4-11は、注文管理サブシステムの機能の構成を示します。顧客が商品を注文するとき、注文登録サブシステムは、顧客が選んだ商品に関する情報、たとえば価格や在庫の有無、発送予定日などを得るために、商品検索を実行します。また、顧客は注文するとき、発送先や支払い方法など、個人情報を登録する必要があります。注文管理システムに登録された情報は、注文通知機能によって、ショップの担当者に連絡されます。

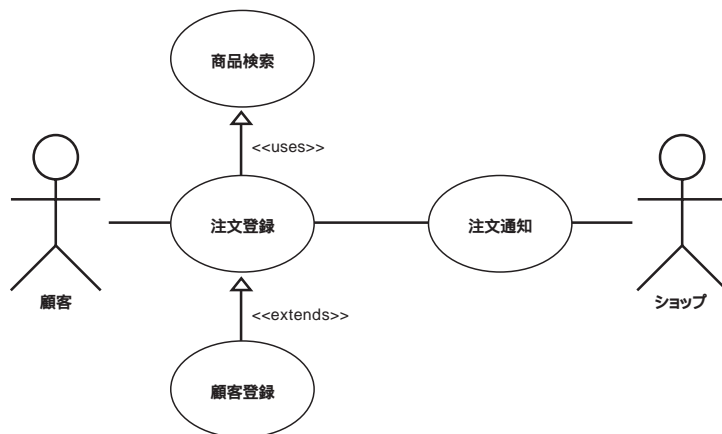


図 4-11 ユースケース図(注文管理サブシステム)

注文登録と顧客登録のユースケースは、<<extends>> が指定されています。これは、注文登録時に発生する顧客情報の登録が、注文登録にとって付加的な機能であることを示します。

ユースケース図は、開発プロセスにおける要求定義段階でまとめます。ユースケース図では、あくまでユーザーから見たときのシステムの構成要素の機能やサービスとそれらの関係だけを記述します。

4.2.2 クラス図

ユースケース図をまとめる過程で、オンラインショップは商品管理、注文管理、顧客管理の3つのサブシステムで構成されることがわかりました。要求仕様の概略がまとまったら、次にシステムの構造をより詳細に記述する必要があります。

システムをオブジェクト指向プログラミング言語で実装する場合には、プログラムの基本単位はクラスになります。クラスとは、システムを構成する個々の要素の持つべき属性と操作を1つの単位にまとめたものです。オブジェクト指向のアプローチでシステムの分析と設計を行う場合、クラスを定義することが、その後の作業の前提になります。

ユースケース図をまとめる過程で、オンラインショップのシステムには、少なくとも、商品、注文、顧客という3つの要素が必要とされることがわかりました。この3つの要素は、クラスとして定義するのに適しているように思われます。そこで、まずこの3つの要素をクラス図に定義してみましょう(図4-12)。



図 4-12 クラス図

クラス図では、クラスおよびクラス間の関係を記述します。図4-12の四角で示した記号がクラスに対応します。

関連

商品と注文と顧客の間には、互いに関連性があります。注文は、誰が何を注文したかを示します。したがって、この3つのオブジェクトの間には、次のような関連が設定できます。

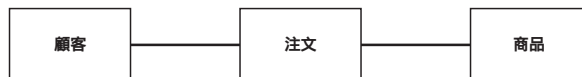


図 4-13 クラス間の関連

1人の顧客は、そのオンラインショップで1個の商品だけでなく、いくつかの商品をまとめて複数個、注文することが多いでしょう。次の図は、このような1対多の関係を、より明確に記述したものです。

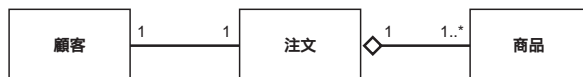


図 4-14 クラス間の関連 多重度と集約

この図では、関連を示す線の両端に数字が指定されていますが、UML では、これを関連の多重度と呼びます。また、菱形の記号は、注文がいくつかの商品を持つことを示します。このような関連を UML では集約と呼びます。

汎化と特化

このようなクラス間の横の関係だけでなく、縦の関係も考えられます。たとえば、このオンラインショップが、会員制のサービスを実施した場合を想定してみましょう。会員制に加入した顧客を会員顧客と呼ぶことにします。クラス図では、顧客と会員顧客の関係を、次のように記述します。



図 4-15 汎化と特化

オブジェクト指向では、一般にこのような関係を拡張と呼びます。顧客クラスを拡張したのが会員クラスになります。また、会員顧客クラスは顧客クラスのサブクラスであるともいえます。逆に、顧客クラスは会員顧客クラスのスーパークラスであるともいえます。UML では、スーパークラスに対する関係を汎化、サブクラスに対する関係を特化といいます。

属性と操作

最初に述べたように、クラスとは、システムを構成する個々の要素の持つべき属性と操作を1つの単位にまとめたものです。クラスを定義するときには、その属性と操作を定義する必要があります。図 4-16は、属性と操作を記述した商品クラスを示します。

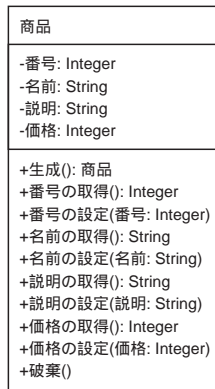


図 4-16 商品クラスの属性と操作

商品クラスでは、番号、名前、説明、価格を属性として定義しました。また、それらの属性に対する取得操作と設定操作を定義しています。

属性名と操作名の先頭にあるマイナス記号(-)は、それらの可視性がプライベートであることを示します。可視性とは、クラスの属性やメソッドに対して、クラスの外部からアクセスできるかどうかを示します。可視性がプライベートの場合、クラスの外部からはアクセスできません。

可視性には、プライベートの他に、パブリックとプロテクトがあります。パブリックは"+"で表記し、クラスの外部からアクセスできることを示します。プロテクトは"#"で表記し、クラスの外部からはアクセスできないけれども、そのクラスのサブクラスからはアクセスできることを示します。

属性名には、そのタイプを指定できます。商品クラスの例では、番号や価格に Integer(整数)を、名前や説明に String(文字列)を指定しています。属性名とタイプの間はコロン(:)で区切ります。

操作名は、括弧の中にその操作のパラメータを指定します。パラメータは、パラメータ名とそのタイプを指定し、コロンで区切ります。パラメータを複数扱う操作の場合には、パラメータをカンマで区切ります。パラメータ指定の後にその操作が返すタイプを指定します。操作が何も返さないときは、省略できます。

インターフェイス

ユースケース図でまとめたオンラインショップの要求定義によれば、商品の注文が登録されたとき、その内容をショップの担当者に通知する必要があります。このような通知の機能は、注文に限らず、それ以外のクラスにもあると便利です。しかし、通知の機能は、クラスによって方法や内容が異なるかもしれません。このような問題を解決するのが、インターフェイスと

呼ばれる特殊なクラスです。図 4-17は、注文クラスが通知可能というインターフェイスを実装している関係を示しています。

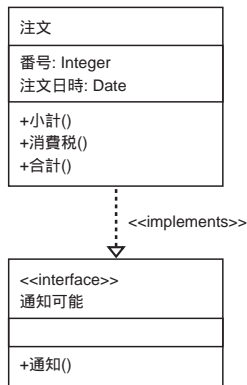


図 4-17 インターフェイス

インターフェイスには、インターフェイス名の前に <<interface>> というキーワードを指定しています。インターフェイスはクラスと異なり、属性を定義せずに、操作だけを定義します。また、インターフェイスの操作は、すべて可視性がパブリックでなければなりません。

最後に、これまで説明してきたクラス図の個々の要素をひとつにまとめたものを示します。各クラスの属性に対する取得操作と設定操作は省略していますので、注意してください(図 4-18)。

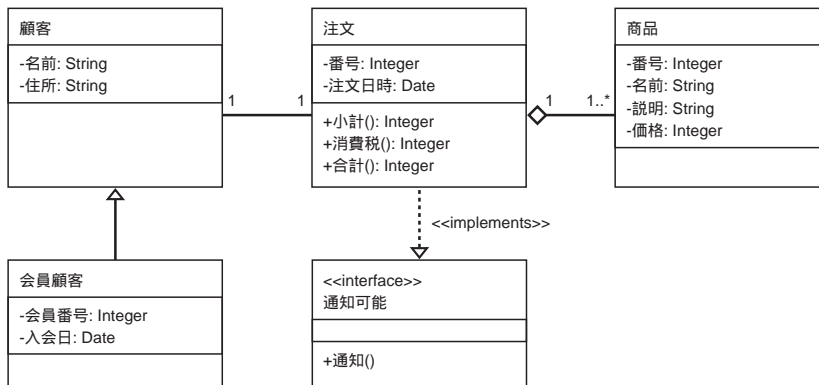


図 4-18 クラス図

4.2.3 振る舞い図

クラス図の作成は、システムの分析 / 設計段階の中心的作業に位置付けられますが、しかし、あくまでシステムの静的な構造を記述しているにすぎません。静的という意味は、時間の要素が考慮されていないということです。クラス図は、システムがどのようなクラスで構成されていて、それらのクラスがどのような属性や操作を持ち、またクラス間にどのような関連があるかを記述します。しかし、実際にシステムが稼動しているときに、これらのクラスを基に作成されるオブジェクトの具体的な振る舞いについては、何も記述されていません。

これから説明するシーケンス図、コラボレーション図、ステートチャート図、アクティビティ図は、オブジェクトの振る舞いを記述するためのダイアグラムであり、ユースケース図を含めてこれらを振る舞い図と呼ぶこともあります。振る舞い図は、システムの動的な構造を記述します。動的という意味は、時間の要素が考慮されているということです。

以下、振る舞い図の各ダイアグラムの目的と特徴について、オンラインショップの例を基に説明します。

4.2.4 シーケンス図

シーケンス図と次に説明するコラボレーション図は、オブジェクト間の相互作用を記述するためのダイアグラムです。この2つの図は、相互作用図と呼ばれることもあります。オブジェクト指向のパラダイムでは、オブジェクト間の相互作用は、オブジェクトからオブジェクトへのメッセージの通信として表現されます。そのため、相互作用図では、オブジェクト間のメッセージの通信関係を記述することになります。

図 4-19 に示すように、シーケンス図は通信のプロトコルを説明するときに使うセッション図に似ています。シーケンス図を記述する場合には、対象となる相互作用に参加するアクターとオブジェクトを横に並べます。アクターについては、ユースケース図で使ったのと同じ記号で表記しますが、相互作用図では、アクターもオブジェクトの一種と見なすことができます。

シーケンス図では、アクターおよびオブジェクトの名前には下線がついている点に注意してください。これは、クラスではなくオブジェクト(クラスのインスタンス)であることを示します。

オブジェクトの下から伸びている点線は生存線(LifeLine)といいます。この生存線にそって上から下に時間が経過することを示します。生存線の上の四角は、そのオブジェクトの活動期間を示します。メッセージはオブジェクトが活動しているときに送受信されるので、活動期間(UMLでは活性化(Activation)と呼びます)の間を結ぶ線を使ってメッセージ通信を記述します。どのオブジェクトからどのオブジェクトへ送信されるのかを明記するために、線の終端には矢印を付けます。

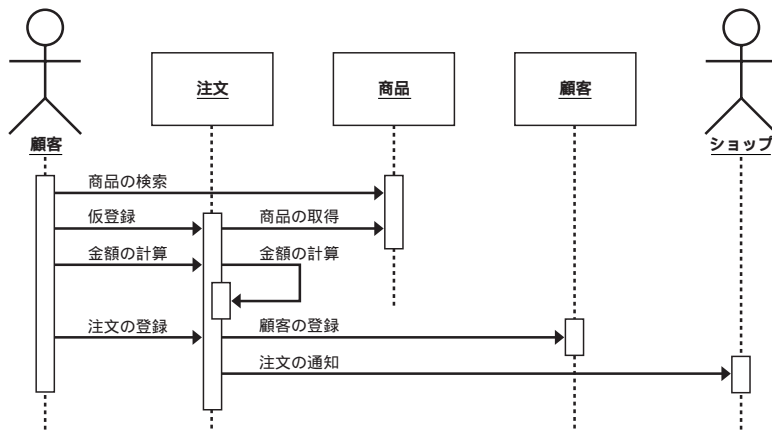


図 4-19 シーケンス図

矢印の上に記述されているテキストは、メッセージの内容を示します。通常は、矢印の先のオブジェクトの持つ操作を記述します。また、テキストの先頭に、そのメッセージが実行される順序を示す番号を指定することもできます。ただし、シーケンス図では、左から右に、上から下にメッセージがやり取りされるので、この順序番号は省略できます。メッセージのフォーマットは、以下のとおりです。

順序番号 [ガード] * [繰り返し] リターンリスト := 操作名 (引数リスト)

ガードとはその操作を実行する場合に満たすべき条件です。

図 4-19では、注文オブジェクトに「金額の計算」というメッセージがあります。このメッセージは、注文オブジェクトがそれ自体に対してメッセージを送信していることを示します。つまり、注文オブジェクト自身の持つ操作を実行していることを示します。

4.2.5 コラボレーション図

シーケンス図では、図の縦方向が時間軸に相当しました。これに対して、コラボレーション図では、オブジェクト間の関係を空間的に表現します。そしてオブジェクト間でメッセージ通信が発生する場合には、オブジェクトを実線で結びます。実際に発生するメッセージについては、実線に並行する矢印で表現します。この矢印と一緒に、メッセージの発生する順序とその内容をテキストで記述します(図 4-20)。

矢印の脇にあるテキストは、シーケンス図の場合と同様に、メッセージの内容を示します。先頭にある順序番号は、シーケンス図では省略できましたが、コラボレーション図では必須になります。

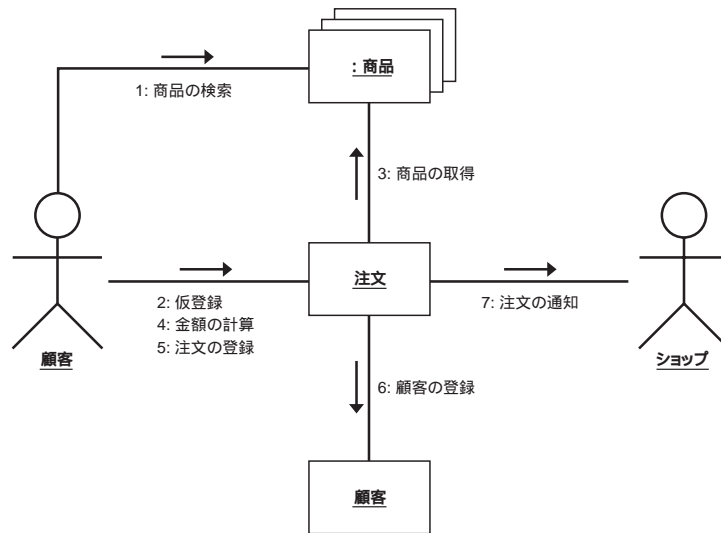


図 4-20 コラボレーション図

4.2.6 ステートチャート図

相互作用図は、オブジェクト間のメッセージ通信を記述するものであり、オブジェクトを外側から見てそれらの関係を記述します。これに対して、ステートチャート図とアクティビティ図は、オブジェクトを内側から見た記述になります。

ステートチャート図では、あるオブジェクトが、開始状態から始まり終了状態で終わるまでの、そのオブジェクトの状態のライフサイクルを記述します。開始状態から終了状態までの間、オブジェクトの状態はさまざまに変化します。この変化のことを、ある状態から別の状態への遷移といいます。

図 4-21は、注文オブジェクトのライフサイクルを示したものです。ステートチャート図では、開始状態を黒塗りの円で、終了状態を中黒の円で記述します。各状態は角の丸い四角形で記述し、状態間の遷移を矢印付きの実線で記述します。

状態の遷移は、なんらかのイベントによって発生します。ステートチャート図では、状態間の遷移を示す矢印に、その遷移が発生する契機となったイベントをテキストで記述します。遷移のフォーマットは以下のとおりです。

イベント [ガード] / アクション

イベントは、その遷移を発生させる引き金(トリガー)になるものを示します。イベントのフォーマットは次のようになります。

イベント名(パラメータリスト)

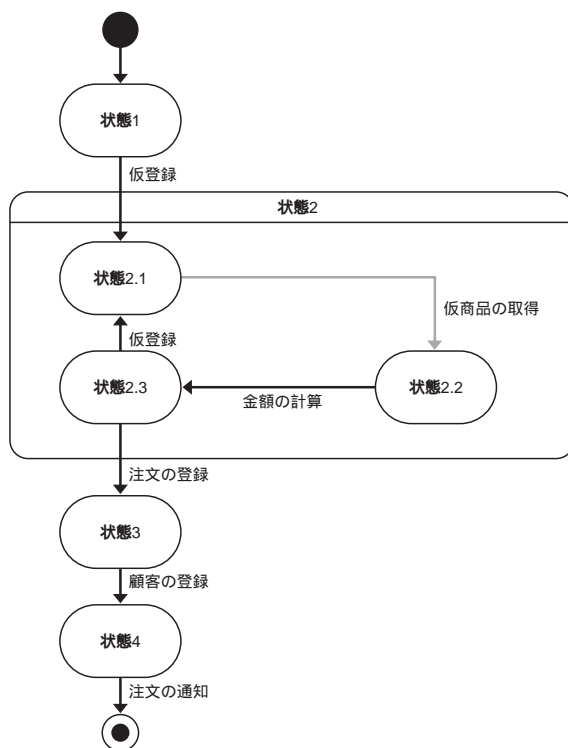


図 4-21 ステートチャート図

相互作用図のメッセージと同様、ガードにはその遷移が発生するための条件を記述します。イベントが発生しても、その条件が満たされなければ、状態は遷移しません。

アクションは、その遷移によって実行される処理内容を記述します。アクションのフォーマットは次のとおりです。

アクション句 ^ 送信句

アクション句と送信句は、それぞれ以下ようになります。

アクション句 リターンリスト := 操作名(引数リスト)
 送信句 リターンリスト := 終了オブジェクト.操作名(引数リスト)

図 4-21では、状態 2 はその内部にさらに 3 つの状態を持っています。このように、いくつかの状態間での遷移を 1 つの状態にまとめることができます。状態 2 のように、内部にネストされた状態の集合を持つ状態を、複合状態(Composite State)といいます。また複合状態の中にネストされた状態を、内部状態といいます。複合状態を使えば、状態遷移を階層的にまとめることができるので、複雑な状態遷移をよりわかりやすく記述することができます。

4.2.7 アクティビティ図

状態チャート図では、オブジェクトの状態遷移を記述しますが、アクティビティ図は状態チャート図の一種と見なすことができます。そのためアクティビティ図は、状態チャート図とほぼ同じ記号を使って記述します。

状態チャート図とアクティビティ図では、何を状態として記述するのか、その観点が異なります。アクティビティ図では、オブジェクトが実行するなんらかのアクションをひとつの状態とみなします。アクティビティ図が一見するとフローチャートに似ているのは、このためです。フローチャートは、プログラムが実行する手続きを単位に、それらの手続きの実行のフローを記述します。オブジェクトが実行するアクションを、ある種の手続きと見なすともできるでしょう。

また、アクティビティ図で記述するアクションは、状態チャート図における遷移を発生させるイベントに対応するので、状態チャート図の矢印を状態記号に置き換えたものがアクティビティ図におけるアクションになっていると見なすこともできます。

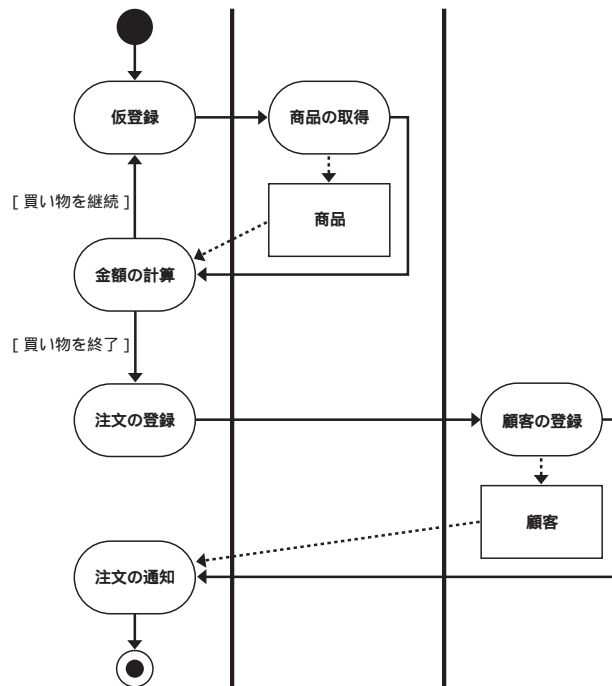


図 4-22 アクティビティ図

図 4-22は、注文オブジェクトのアクションフローを記述したものです。注文オブジェクトで実行されるアクションには、注文オブジェクト以外のオブジェクトの操作も含まれます。した

がって、アクティビティ図では、関連するオブジェクトに対するアクションも記述することができます。

アクティビティ図においても、ステートチャート図と同様に、アクションフローに対してガードとアクションを指定することができます。アクションフローのフォーマットは以下のとおりです。

[ガード] / アクション

ステートチャート図の遷移と違って、イベントは省略できます。なぜなら、アクションフローが発生するトリガーになるのは、図で示されているアクションそのものだからです。

図 4-22 中の 2 本の縦線は、レーン(あるいはパーティション)と呼ばれるもので、それぞれのオブジェクトの管轄範囲を明記するために使います。

アクティビティ図では、アクション間のフローだけでなく、アクションにともなうオブジェクトのフローもあわせて記述することができます。これは、他のオブジェクトの操作を実行した結果発生する、あるアクションから別のアクションへのオブジェクトの受け渡しを示します。アクション間の遷移と区別するために、アクションとオブジェクトの間の関連は、点線の矢印で表記します。

4.2.8 MEFについて

ここでは、簡単なサンプルをもとに、UML のダイアグラム図の概要を紹介しました。MEF でダイアグラムを作図する場合、表示される記号の表記や用語が必ずしも UML 標準と一致していません。また、UML で規定されている各ダイアグラムの表現法のすべてが MEF でサポートされているわけでもありません。MEF は UML ダイアグラムのサブセットをサポートしていると考えたほうがいいでしょう。MEF でダイアグラムを作成する具体的な方法については、本書の第 11 章「MEF を使ったモデリング」に掲載されている押しボタン式歩行者用信号システムを参考にしてください。

4.3 インストールと起動

本章の最後に、IIOSS のインストール方法と起動方法について説明します。

4.3.1 IIOSSのインストール方法

文章中に出てくるコマンドタイプの例において、'% 'や'# 'はシェルにおけるプロンプトを示しています。実際にはタイプする必要はありません。