平成１３年度　技術研究会論文

UML 入門

大分県情報サービス産業協会
技術研究会　UML 部会

メンバー紹介

部会長	姫野　幸雄	三井造船システム技研(株) ｅ-ビジネス事業部　第二システム部　ＨＲグループ
副部会長	高橋　泰文	大銀コンピュータサービス(株) 業務部
	安部　智行	システムエイジ(株) システムソリューション事業部
	祝出　幸一	大分シーイーシー(株) 第一システム部
	今別府　晋哉	(株)富士通大分ソフトウェアラボラトリ マルチメディアシステム開発部
	広田　朋寛	(株)富士通大分ソフトウェアラボラトリ マルチメディアシステム開発部
	清松　慶一	(株)オーイーシー Ｗｅｂビジネス推進部

部会開催履歴

第 1 回	2001／7／12	15：30～16：30	ＵＭＬ部会の活動方針
第 2 回	2001／7／24	13：00～16：30	ＵＭＬ部会の具体的な活動方針の決議と勉強会
第 3 回	2001／8／8	14：00～18：00	ユースケース図、シナリオ、アクティビティ図の勉強会
第 4 回	2001／8／21	13：00～18：00	クラス図勉強会
第 5 回	2001／9／4	14：00～18：00	シーケンス図、コラボレーション図、ステートチャート図勉強会
第 6 回	2001／9／26	14：00～18：30	パッケージ図、コンポーネント図、配置図勉強会
第 7 回	2001／10／11	14：00～17：40	前回実習問題に対するクラス図の比較検討と論文作成に関する概要検討
第 8 回	2001／10／23	14：00～18：30	論文作成に関するスケジュールの検討と各自持ち寄った下書き論文の検討
第 9 回	2001／11／6	14：00～19：00	論文作成に関する検討
第 10 回	2001／11／28	14：00～17：00	同上
第 11 回	2001／12／5	14：00～18：50	同上
第 12 回	2001／12／19	14：00～21：00	同上
第 13 回	2002／1／16	14：00～18：50	同上
第 14 回	2002／1／23	13：30～16：00	同上
第 15 回	2002／1／30	13：30～15：00	発表内容の検討
第 16 回	2002／2／7	13：30～17：00	同上
第 17 回	2002／2／12	13：30～16：00	同上

～　論文目次　～

第１章　　はじめに
第２章　　テーマの選定理由
第３章　　UML 入門
　３．１　ユースケース図
　３．２　シナリオ
　３．３　アクティビティ図
　３．４　シーケンス図
　３．５　クラス図
　３．６　ステートチャート図
　３．７　コラボレーション図
　３．８　パッケージ図
　３．９　コンポーネント図
　３．１０　配置図
第４章　　まとめ

第1章　はじめに

我々システム開発を担うエンジニアにとっての課題は、そのシステムの利用者の構想やニーズを正確に把握し、これに適合したシステム開発を行うことにある。しかし現実的にはユーザの構想をシステム化する段階において開発者とユーザ間のコミュニケーション上の隔たりがあることによってユーザが意図しないものになっていたり、考慮不足があったり等の問題が欠かせないものになっている。

このような問題を解決する為に近年ＵＭＬによる表記法が注目されるようになってきている。ＵＭＬはユーザおよび開発者間のプロジェクトに対する相互理解を深めながらシステムの設計から開発までのプロセスを総合的に進めていくことができるものである。

我々はこうしたＵＭＬの表記法について学習しどのような利点があるかなどの分析を行うことをテーマとし調査・研究を進めることにした。

第2章　テーマの選定理由

UML とは、Unified Modeling Language(UML：統一モデリング言語)を意味し、オブジェクト指向開発における設計図の標準規格である。

UML が話題になっている要因は、C++ や Java 等のオブジェクト指向言語を使用する機会が増えてきたことにより「オブジェクト指向開発方法論」による開発の必然性が増してきたことが考えられる。

「オブジェクト指向」とはシステムをオブジェクトという単位で考えていく方法であり、システムをオブジェクトの集合と考え、オブジェクトどうしの協調作業でシステムの機能が実現されるようにオブジェクトの構造を作成する。(図 1)

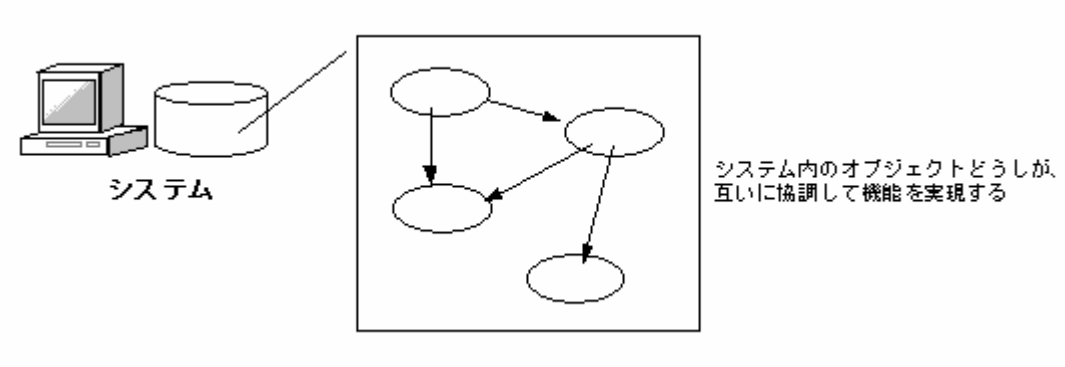


図 1 オブジェクト指向によるシステム構築

「オブジェクト指向開発方法論」というのは、このようなオブジェクト指向を使ってシステムを作っていくときの手順や成果物、管理項目などを定めたものである。

「オブジェクト指向開発方法論」としては OMT 法や OOSE 法、Booch 法や Shlaer&Mellor 法といった多数の方法論が存在する。それぞれの方法論には(表 1)のような特徴があり、作ろうとするシステムの性格や重点を置きたい工程(分析や設計)に応じて好きなものを選ぶことができる。

方法論	特徴
Booch 法	設計にフォーカス
OMT 法	分析にフォーカス、3つの視点(オブジェクト、動的、機能)のモデル
OOSE(Objectory)法	ユースケースの導入、オブジェクトの3カテゴリー(boundary、control、entity)
Coad/Yordon 法	初めての OOAD 手法
Shlaer/Mellor 法	組み込みに特化した方法論
Drop	日本発の OO 開発方法論

表 1 代表的なオブジェクト指向開発方法論

しかし、それぞれの方法論特有の特徴のため「表記法」が問題となっていた。「表記法」とは成果物として作られるモデル(システムの構造や振舞いを単純化して表現した図)の描き方を決めたものであるが、この表記法が方法論によって異なっていた。

どの方法論も、その出発点はオブジェクト指向ということで共通しているので、クラスやオブジェクト、クラスの構造やオブジェクト間での協調関係など、取り扱う対象やモデル化したいものも共通であるが、異なった表記法で表現していた。
そうしたことから、同じ「クラス」や「オブジェクト」でも図にするとその描き方が微妙に異なるという状況が生まれてしまった。(図 2)

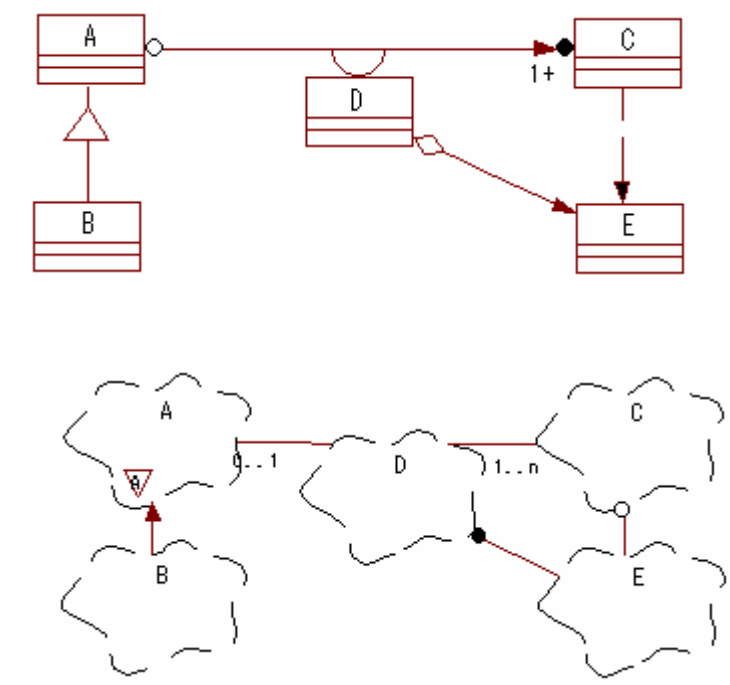


図 2 方法論による表記法の違い(上：OMT 法、下：Booch 法)

このような方法論乱立という状況を打破するため、方法論の統一を図る大胆な試みが始まったが、方法論者には方法論者ごとに「これがベストの方法論だ」と思うものがあるので、それを統一しようというのは非常に難しいことだった。その結果、方法論の統一はあきらめ、とりあえず表記法だけを統一しようということでできたのがこの UML である。(図 3)

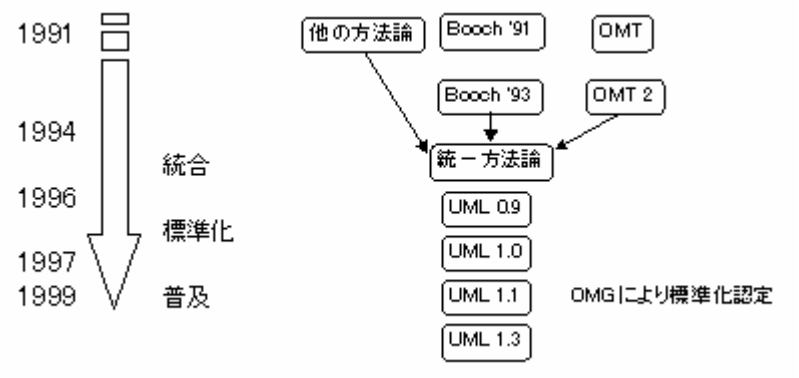


図 3 UML ができるまでの道のり

UML の登場により、さまざまな方法論の表記法は統一され、モデルの翻訳という問題はなくなり、UML はオブジェクト指向言語を使う開発者の間ではモデルの記述言語として標準規格の地位を得た。(図 4)

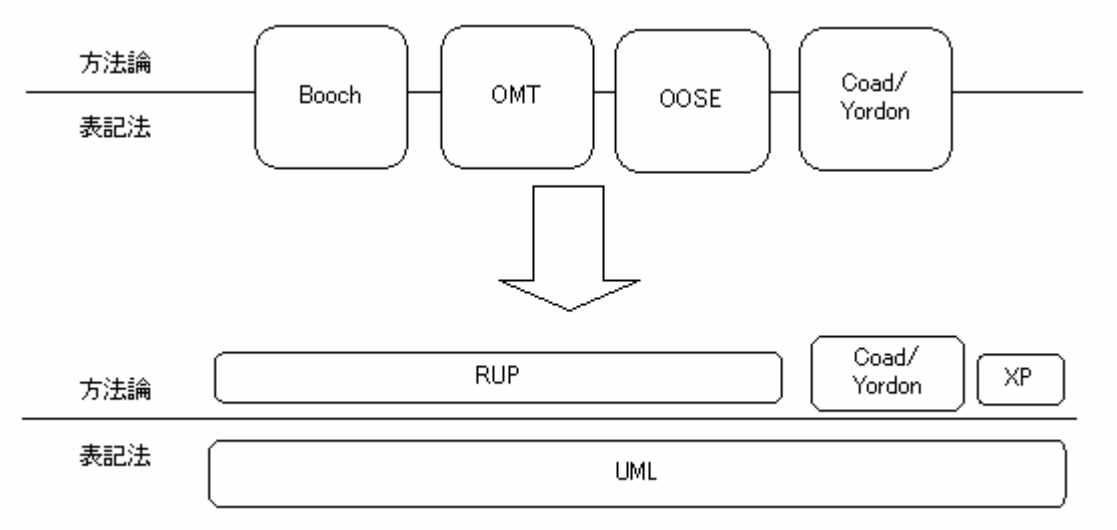


図 4 UML がもたらした世界

UML を用いる最大の利点は、それがエンジニアの間の共通言語になることである。共通言語になるということは、エンジニア間のコミュニケーションの道具として UML が使えるということで、

1. 意味が正確に伝えられる
2. ノウハウが残せる

といったメリットを生む。

意味が正確に伝わるというのは、UML は図を描くための表記法が厳密に決められているので、曖昧性の無い図を描くことができるからである。曖昧性の無い図を使ってコミュニケーションができるということは、言葉や文章を使ったコミュニケーションよりも大量の情報を正確に伝えることがでることを意味する。こうように自分の考えや認識を相手にきちんと伝えられるというのは、ソフトウエア開発では非常に重要なことである。

また、様々なノウハウをモデルという抽象的な形で残すことができるというのもメリットとなる。UML でモデルを作っておけば、システムの構造や振舞いを、モデルという理解しやすく、プログラム言語に依存しない形で残すことができる。つまり、自分の持っているシステム開発に関する様々なノウハウを、ソースコードよりも利用しやすい形で残すことができる。

オブジェクト指向で開発したシステムなら、システム内部の構造や振舞いは全て UML で記述できる。UML によるビジュアルなモデルはソースコードよりも圧倒的に理解し易いということを意味し、理解し易いシステムのモデルがあるということは、

1. 拡張・修正しなければいけない部分がすぐに分かる
2. どのように変更すればいいのかをモデルを元に検討できる
3. 変更の影響がどこまで及ぶかが分かる

といったメリットがある。

本部会では、上記のような理由によりオブジェクト指向開発における設計図の標準規格として広がりを見せている UML について学び、各「表記法」を作成できるようにすることを目標に活動を行なった。

第3章　UML入門

この章では以下に示すＵＭＬ表記法に対し本部会で学習した結果をもとに各表記法について記述する。また記述内容については下記要件定義をもとに作成した。ただし表記法によっては下記要件定義とは別に例をあげたものもある。

定義内容	視点	ダイアグラム名
要求図	モデル化対象に対する要求を表す	ユースケース図
構造図	モデル化対象の静的な構造を表す	クラス図
		オブジェクト図
		パッケージ図(クラス図の一種)
振舞い図	モデル化対象の動的な振舞いを表す	シーケンス図(相互作用図)
		コラボレーション図(相互作用図)
		アクティビティ図
		ステートチャート図
実装図	モデル化対象の物理的な実装を表す	コンポーネント図
		配置図

[要件定義]
会議室の利用予約を管理するシステムを考える。
対象となる会議室は大(50 名収容)、小(20 名収容)の 2 つ。
予約は「目的」「利用開始時刻」「利用終了時刻」「予約者」を決めることによって行われる。
希望した時間に既に予約が入っている場合、予約はできない。
システムに対しては予約状況の確認、予約、予約のキャンセルが可能。
予約のキャンセルは予約を行った人しかできないが、
システム管理者は自由に予約のキャンセルが可能。

３．１　ユースケース図

３．１．１　概要

(1) ユースケース図とは

(a) ユーザからの要件定義をもとに構築するシステムと利用する側との関連性を明確に示すための図。

(2) ユースケース図を書く意味

(a) ユーザからもシステムの内容が理解しやすい。

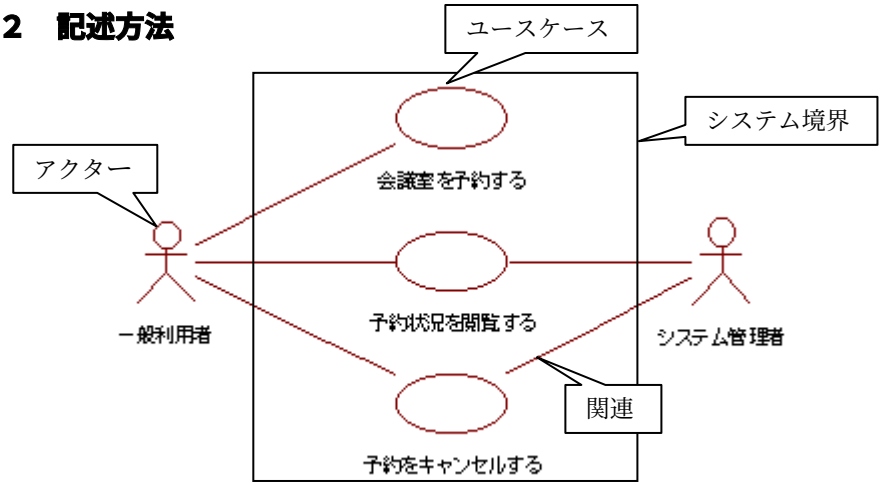
(b) システムがどのようなサービスを提供するかを示す。

(c) 要求に対する相互理解を得ることができる。

(3) 他図との関連

(a) ユースケース図を元にシナリオ・シーケンス図・コラボレーション図を作成することができる。

３．１．２　記述方法



(1) 語句の説明

(a) アクター　：アクターは人間の図で表記する。
：アクターはシステムの一部ではない。
：アクターはシステムと情報をやり取りするものであり、システムに情報を渡したり、システムから情報を受け取ったりするものである。
：アクターとして定義できるものとしてシステムを操作する人間・ハードウェア機器・外部システムが考えられる。

(b) ユースケース：ユースケースは楕円形で表記する。
：ユースケースはシステムとアクターとの間の対話をモデル化したものである。
：ユースケースはアクターから開始され、システムのある機能を実行するものである。
：ユーザがシステムを利用して遂行する単位業務のひとつを抽象化した

ものである。
：ユースケースをすべて集めたものが、システム全体の機能を網羅した形になる。

(c) システム境界：システム境界はユースケースを長方形で囲むように表記する。
：システム境界は今回構築するシステムの影響範囲を明確にするものである。

(d) 関連：関連はアクターとユースケースを実線で結ぶ。
：関連はアクターから開始されるユースケースやユースケースからアクターへ出力される関係のあるもの示す。

(2) 作成方法・手順

(a)ユーザと入念なヒアリングを行いユーザが今回のシステム構築で実現したいことを分析する。

(b)ユーザの要件定義から人がシステムに投入する情報・システムから得たい情報を解析しアクターを定義する。

(c)今回構築するシステムと既存のシステムおよび同時に作成する他のシステムとの境界を分析し外部システムをアクターとして定義する。

(d)アクターとシステムの関連から上記で分析した単位業務をユースケースとして定義する。

(e)最後に定義したアクターとユースケースの関連付けを行う。

３．１．３作成時の注意点

(1) ユーザからの要件定義がすべて網羅されているかチェックする。

(2) システム内のユースケースとアクターの関連がすべて定義されているかチェックする。

(3) 作成後にシステム設計者とユーザの双方にて確認しシステムの認識に間違いがないか？漏れがないかを検証する。

３．１．４評価

(1) ユースケース図という表記法を用いてユーザの要件定義とシステム設計者間の食い違い・漏れが発見しやすくなるなどシステム設計時には欠かせないものではないかと思われる。

(2) （２）システム設計者から見ても構築システムとユーザおよび外部システムとの関連・システムの機能概要が視覚的に把握できる為、システム全体が把握しやすい。

(3) システム完成後の変更追加においてもユースケース図の見直しをすることによってシステム全体への影響把握がしやすい等の利点が考えられる。

(4) ユースケース図作成にあたってはシステム分析・ユーザとのレビュー経験など担当 SE のスキルが要求される為､必ずしも最適なユースケース図が一回で作成されるとは限らない。よってスパイラルモデルのように何度か繰り返し見直しをしていく必要があると思われる。

(5) 上記の要因によりユースケース図に不備があった場合は、先の工程で再作成する必要も出てくるため、再作成した場合は、全ての作業に対し見直しを実施する必要が出てくる。

３．２　シナリオ

３．２．１　概要

(1) シナリオとは

(a)ユースケースを具体的に実行したときの流れを文章に表現したもの。

(2) シナリオ作成する意味

(a)シナリオを作成することにより実例を通して具体的なシステムの動きや流れを表現することができる。

(3) 他図との関連

(a)シナリオを作成することによりこれから先に必要となるシーケンス図やクラス図を作成する際のオブジェクトやクラスの抽出元になる。

３．２．２　記述方法

シナリオ１　：　会議室の利用予約をおこなう。
１．一般利用者は会議室予約システムを立ち上げる。
２．会議室管理画面より会議室予約画面を呼び出す。
３．会議室予約画面より予約対象となる大会議室・小会議室を選択する。
４．予約内容として「目的」「利用開始時刻」「利用終了時刻」「予約者」を入力する。
５．予約入力を終了する。
６．予約入力された時間に会議室が空いていることを確認する。
７．予約情報を設定し会議室に追加予約登録を行う。

(1) 基本シナリオ

全て問題なく動作する事例を表現するものであり本来のシステムが処理する全ての機能が表現されるものである。ユースケースに対応する業務処理を複数作成する必要がある。

(2) ２次シナリオ

基本シナリオ以外の例外的な処理を表現するものである。在庫切れ・オペミスなど例外処理として想定させる処理を全て表現する。

(3) 作成方法・手順

(a)ユーザーの要求定義およびユースケース図から想定される全ての標準処理ケースを洗い出し処理の流れが把握できるようわかりやすく記述する。

(b)同様に業務運用の中で考えられるエラー処理・イレギュラー処理を洗い出し

３．２．３　作成時の注意点

(1) 処理の流れが把握できるようわかりやすく記述する。
(2) ユーザーの要件定義・ユースケース図から全ての業務処理を洗い出す。
(3) 洗い出されたシナリオが業務の流れから無理なく整理されているか注意する。業務間の複合により１つのシナリオから複数の流れを記述していないかなど。
(4) エラーケースについてはエラー処理時の業務の流れをユーザーと協議しシステム対応するか、ユーザー運用にて対応するかをシナリオ上にわかりやすく表現する。
(5) シナリオ作成後、要求定義やユースケース図に変更があった場合は、新たにシナリオを追加すると共に以前作成したシナリオに対して整合性がとれているか、漏れがないかなど再度検討する。

３．２．４　評価

(1) ユーザーの要件定義やユースケースから文章にて処理を記述する為、設計者が見たときに抽象的な記述ではなく、具体的な記述がなされており理解しやすい。
(2) 個々に基本シナリオ・２次シナリオと分けて記述されている為、漏れが発見しやすくまとめる際にもまとめやすい。
(3) シナリオ作成自体においてはユーザーとのヒアリングにおける要件定義およびユースケースを元に作成するため SE のスキルにあまり依存しない。
(4) シナリオ作成者が個人判断で処理ケース作成している為、漏らさず記述できているかチェックがしにくい。
(5) また作成者自身の文章表現にて作成される為、わかりづらい、勘違いがあるなどの可能性がある。よって作成後のシナリオについてはユーザーに検証してもらう、プログラマーとレビューするなど確認が必要。
(6) シナリオ記述例などなるべくサンプルを用意して、個人の判断に依存しないような基準を作成するほうが望ましいのではないか？

３．３　アクティビティ図

３．３．１　概要

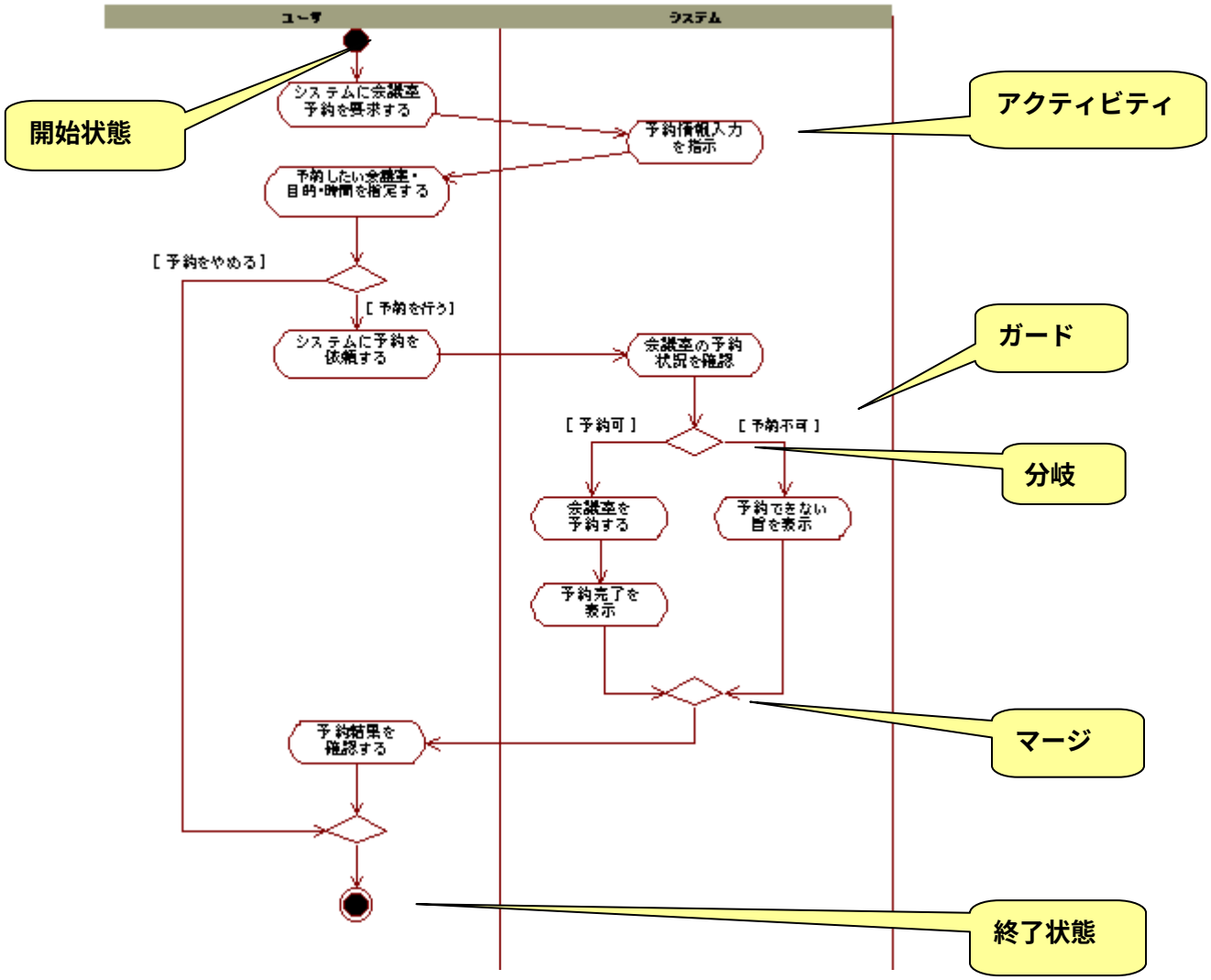
(1) アクティビティ図とは

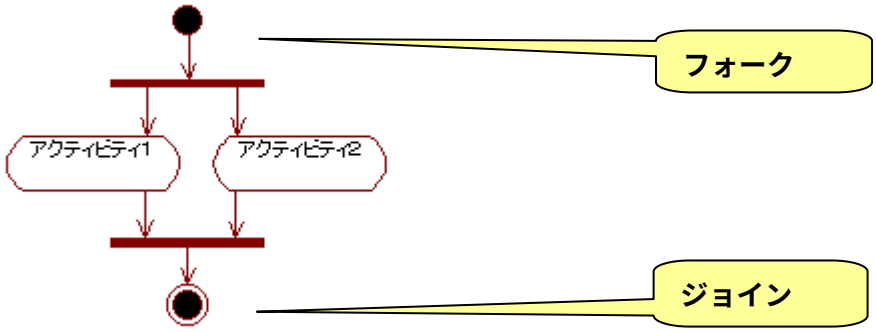
(a) ワークフローのような、手順的なもののモデリングに使われる図である。

(2) 他図との関連

(a) ユーズケースの詳細定義を行う際に用いることができる。

３．３．２　記述方法





⑴ 語句の説明

(a) 開始状態：状態遷移の開始を表す。
(b) フォーク：１つの入力遷移に対して複数の出力遷移を引き起こす。
(c) ガード：分岐での処理を行う内容を表す。
(d) アクティビティ：何かを行っている状態を表す。
(e) 分岐：条件付き振る舞いの開始を示す。
(f) マージ：分岐によって開始された条件付き振る舞いの終了を示す。
(g) ジョイン：フォークで開始された各スレッドを集める。
(h) 終了状態：状態遷移の終了を表す。

３．３．３　作成上の注意点

特になし

３．３．４　評価

(1) フローチャートと異なり、並行処理や待ち合わせ同期といった並行表現ができる。
(2) 企業全体や業務全体のモデルにおける一連のワークフローの記述ができる。
(3) ユースケースごとに対応する処理フローの記述ができる。
(4) あるオブジェクトの持つ１メソッドの内部のアルゴリズムを記述ができる。
(5) アクションとオブジェクトの間の結び付きをあまり明確にすることができない。

３．４　シーケンス図

３．４．１　概要

(1) シーケンス図とは

(a) シナリオをもとにオブジェクト間のやりとりを、時系列に沿って表現した図である。処理の流れを時間順に１つずつ記述できるため、シナリオと相応させて具体的な内容を示すのに便利。

(b) ユースケースを実現するのに必要なオブジェクトの集合と、その関連性を明確に表現できる。

(2) シーケンス図を書く意味

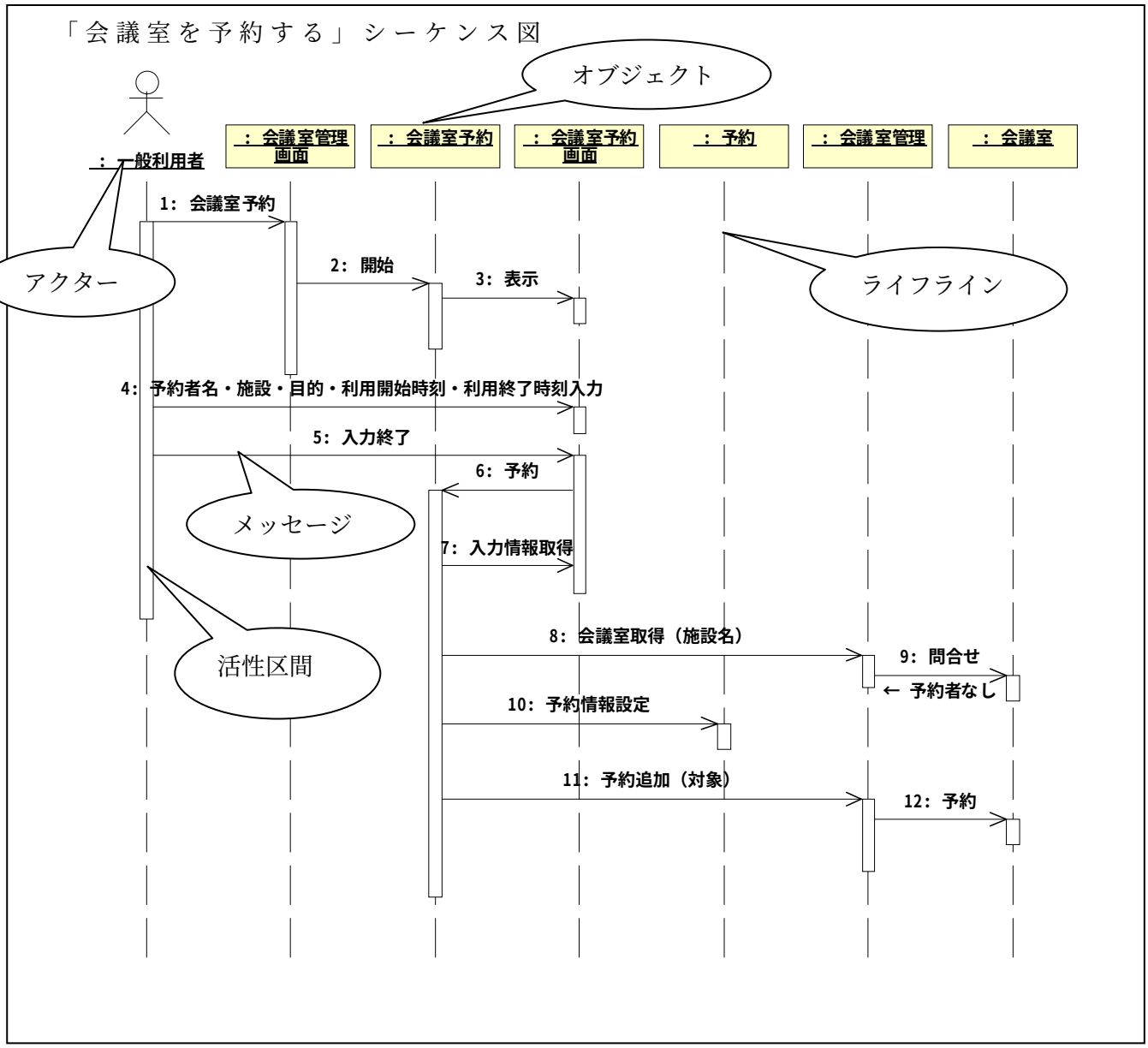
(a) シーケンス図はオブジェクト間の協調の順序が分かりやすいため、シナリオとの対応性をみたり、処理の流れに不備が無いかを調べたりする際に用いられる。

(3) 他図との関連

(a) シナリオをもとに作成する。

(b) システムの同じ振る舞いをコラボレーション図で表現することもできる。

３．４．２　記述方法



(1) 語句の説明

(a) アクター：ユースケース図参照。

(b) オブジェクト：クラスが実体化したもの。

(c) ライフライン：オブジェクトが生存している期間を表す。

(d) メッセージ：オブジェクト間のやりとり。

(e) 活性区間：そのオブジェクトに制御が移っていることを示す。

(2) 作成方法・手順

(a) ユースケースのシナリオごとに作成する。

(b) オブジェクト候補として、シナリオから全ての名詞・名詞句を抜き出す。

(c) オブジェクトを抽出する。

＜判別ポイント＞

(イ)同じオブジェクトを指していないか

(ロ)一つの単語が複数の概念を指していないか

(ハ)状態を指していないか

(ニ)今回のシステムには関係ない外部のものやアクターではないか

(ホ)あるオブジェクトの属性値を指していないか

(d) オブジェクトを横軸、時間の経過を縦軸にとり、各オブジェクトから下方に向けて、このオブジェクトの生存期間を示す縦破線(オブジェクトのライフライン)を引く。

(e) ライフラインによりオブジェクトのライフサイクル(生成、消去のタイミング)を知ることができる。

(f) 各メッセージは、２つのオブジェクトのライフライン間にある矢印で表され、メッセージの送信順序は、ページの上から下の順に新しくなる。

(g) 図に活性区間を含めることにより、オブジェクトがアクティブな期間であることを示すことができる(アクティブな期間とは、プロシージャによる相互作用の場合は、プロシージャがスタックに積まれている期間を意味する)。活性区間は省略でき、その方がより簡単に図を作成できるが、分かりやすさは低減する。

３．４．３　作成時の注意点

(1) 検討を進める過程で、クラスの細分化、追加、削除等を行うため、シーケンス図はその都度変化する。

(2) 一般的に、メッセージの時間的順序は縦軸上の位置によって示されるため、メッセージ番号は省略してもよい。

３．４．４　評価

(1) コラボレーション図と同じく相互作用図の 1 種。

(2) オブジェクトの相互作用を時系列に見るのに適している。

(3) シーケンス図は大変シンプルな表記法なため、一目で制御の流れを把握することができる。制御の全体的な流れは、オブジェクト指向プログラムでは最も理解しにくいことの１つに数えられる。オブジェクト指向流の優れた設計は、通常、さまざまなクラスに分かれた多数の小さなメソッドで構成されており、振る舞いの全体的なシーケンスをつかむのがとても難しい場合がよくある。シーケンス図は、こうした問題を解消してくれる。

(4) オブジェクトの抽出は、シーケンス図を作成するＳＥのスキルに依る。

(5) メッセージの種類には何種類かあるようだが、本部会ではそこまで掘り下げることができなかった。

３．５　コラボレーション図

３．５．１　概要

(1) コラボレーション図とは

(a) オブジェクト間のつながりを詳細に表現したい場合に用いる。

(b) オブジェクト間のつながりに着目して、メッセージやデータの流れを表現した図。

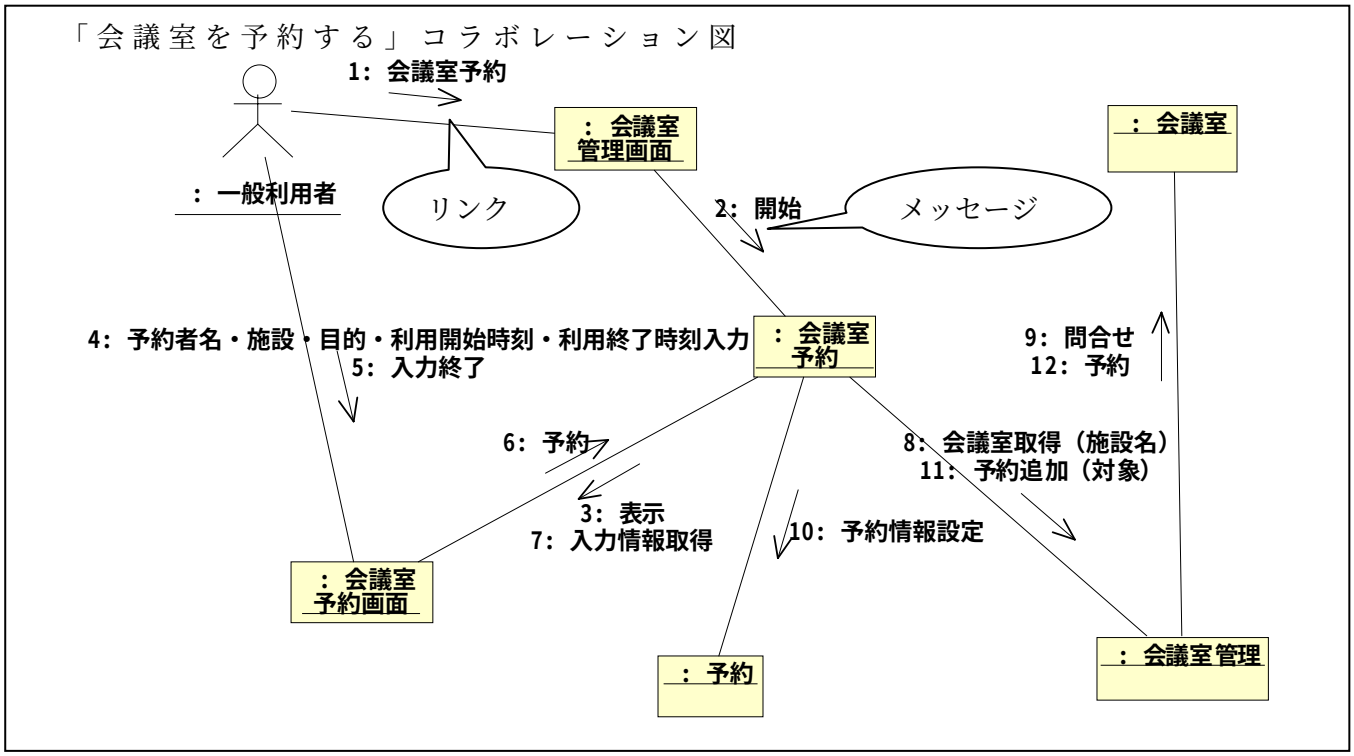
(2) コラボレーション図を書く意味

(a) オブジェクト間やクラス間の関係が分かりやすくなることが特徴。そのためコラボレーション図は、クラス間の関係の洗い出しや検証に用いられる。

(3) 他図との関連

(a) システムの同じ振る舞いをシーケンス図で表現することもできる。

３．５．２　記述方法



(1) 語句の説明

(a) リンク：他のオブジェクトとの参照関係を示す。

(b) メッセージ：シーケンス図参照

(2) 作成方法・手順

(a) オブジェクトはオブジェクト名にアンダーラインの付いたシンボルで表される。それらを関連で結び、各オブジェクトの参照・つながりを示す。

(b) メッセージは、２つのオブジェクトをつなぐリンクの近くに付加した矢印で表す。

３．５．３　作成時の注意点

(1) メッセージの順番を示す、メッセージ番号が必須である。
(2) シーケンス図のメッセージの内容と同じである。

３．５．４　評価

(1) シーケンス図と同じく相互作用図の１種。
(2) コラボレーション図とシーケンス図は基本的に同じ記述能力を持っていて、同じ情報を表現することができる。ただ着目点が異なっているため、組み合わせて使用することでより正確なモデルを作成することができるようになる。最近のモデリングツールでは、どちらかの図を作ると、もう一方の図を自動的に生成できるものもある。
(3) 「オブジェクトの相互作用を時系列的にとらえる場合」はシーケンス図、「オブジェクト間のつながりを詳細に表現したい場合」はコラボレーション図が適している。
(4) コラボレーション図でリンクがあるオブジェクトはクラス図においても関連があることが考えられる。
(5) コラボレーション図が動的な図であるのに対し、クラス図は静的な図である。
　動的：処理の流れを表す。PG で言えば実行している状態
　静的：プログラムで言えばソース。

３．６　クラス図

３．６．１　概要

(1) クラス図とは

(a) 対象領域やシステムの静的な構造を表した図で、「クラス」と「クラス間の静的な関係」を表現できる。

(2) クラス図を書く意味

(a) UML の表記法の中で最もプログラムに近いので、プログラミングのために作成する。

(3) 他図との関連

(a) 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を元に作成する。

３．６．２　記述方法

(1) クラス図で使われるモデリング要素

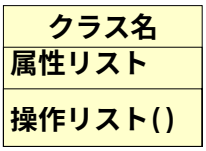
クラス図は情報量が多い図なので、理解するためには様々なモデリングの要素を知っている必要がある。本節ではクラス図で使われるモデリングの要素について説明する。

⒜ クラス

(ｲ) クラスはオブジェクト指向で最も重要なもの

(ﾛ) 全てのオブジェクトは必ずなんらかのクラス定義に基づいて生成されるため、クラスが無ければオブジェクトは作れない

(ﾊ) UML ではクラスを下記のように表現する



・クラスを表す枠の中は 3 つの部分に分かれていて、それぞれ下記が入る
一番上の部分にはクラス名
真中の部分には、そのクラスで定義される属性(データのこと)のリスト
最下段にはそのクラスで定義される操作

⒝ 関連

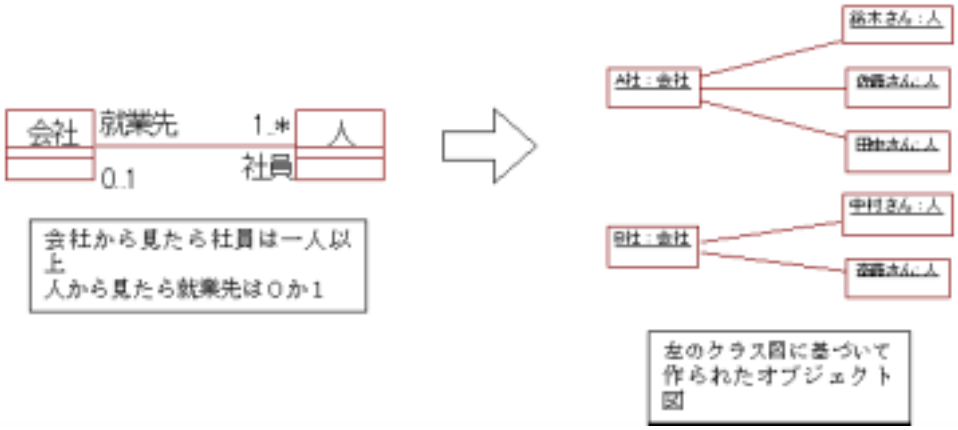
(ｲ) 関連は、関連の両端にあるクラスから作られたインスタンスの間に、何らかの協調関係があるということを表す

(ﾛ) 関連には、様々な情報が付加される。下記はクラス図を作成するときに最低限必要なもの

<関連名>

(ハ) 関連名はその関連がどのような関係なのかを表すもの
(ニ) 下図の例は「人は会社で働く」という関連を表している
(ホ) 通常は動詞系の名前をつける

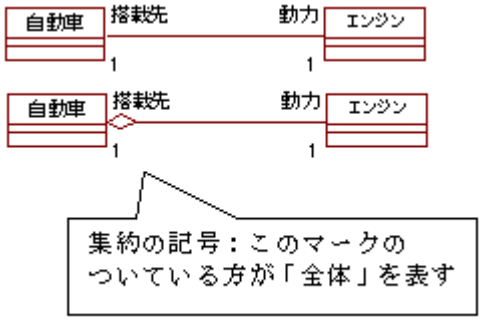
<多重度>
(ヘ) 多重度はその関連に基づいて、いくつのインスタンスが関係するかを表す



書き方	意味	備考
1	厳密に 1	1 関連元のオブジェクトが存在している限り関連先のオブジェクトは必ず 1 つある
0..1	0 か 1(有か無)	関連先のオブジェクトは有っても無くてもいい
0..*	0 以上	"0.."を省略して"*"だけでも同じ意味
1..*	1 以上	関連先のオブジェクトが最低でも 1 つは存在する
1,3,5	離散値指定	指定された数以外の関連にはならない
5..15	範囲指定	関連数の下限値と上限値を設定

<集約>
(ト) 「部分」と「全体」というような構成関係を表すための特殊な関連
(チ) 部分と全体の関係は「集約」を使わなくても、関連名やロール名から分かるが、

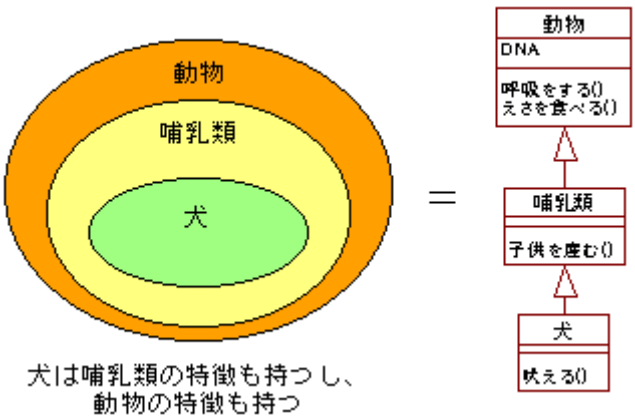
部分－全体の関係を強調したいときに、「集約」が使われる



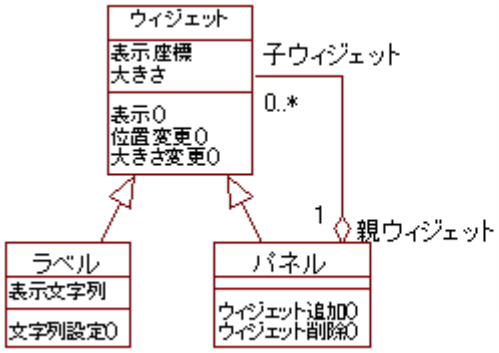
(c) 汎化

(イ) 汎化はクラス間の抽象関係を表すためのモデリング要素。抽象関係というのは、例えば「犬」と「哺乳類」と「動物」という概念の関係のようなもの。この例だと、「動物」という概念が一番抽象的で、次が「哺乳類」、一番具体的なのが「犬」となる。

(ロ) 「汎化の考え方」と「UML を使ったときの表記法」は下記



(ハ) 汎化の実現手段のひとつに「継承」がある。継承とは、すでに用意してあるクラスの属性と操作を引き継いで新しいクラスを作成すること。すでに用意してあるクラスとはクラスライブラリにあるようなもののこと (Java ならパッケージにいろいろと用意されている)。継承の具体例は下記。



ラベルもパネルも GUI 部品に共通して必要な属性と操作を持つウィジェットクラスを継承して作られている。
つまり、ラベルもパネルも「表示座標」と「大きさ」という属性を持ち、「表示」「位置変更」「大きさ変更」という操作を持っている。

(2) クラス図記述手順

(a) コラボレーション図のオブジェクトを検討し、クラスを抽出する。

(b) 各クラスの関連を考慮し、クラス図を作成する。

＜関連を考察する上でのポイント＞

(ニ) コラボレーション図のリンクを、クラス図の関連として見直す。

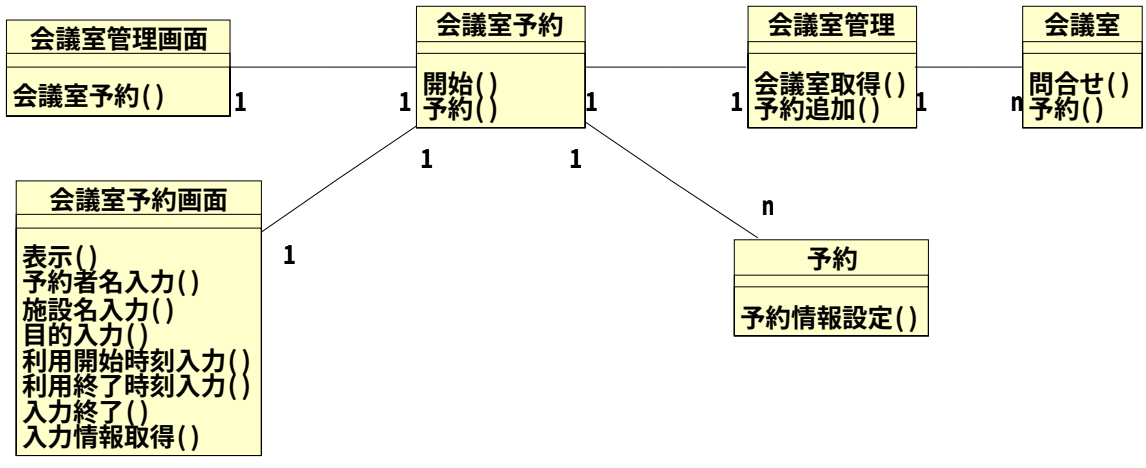
(c) 各クラスの属性（データ定義）を定義する。

(d) モデルの詳細化

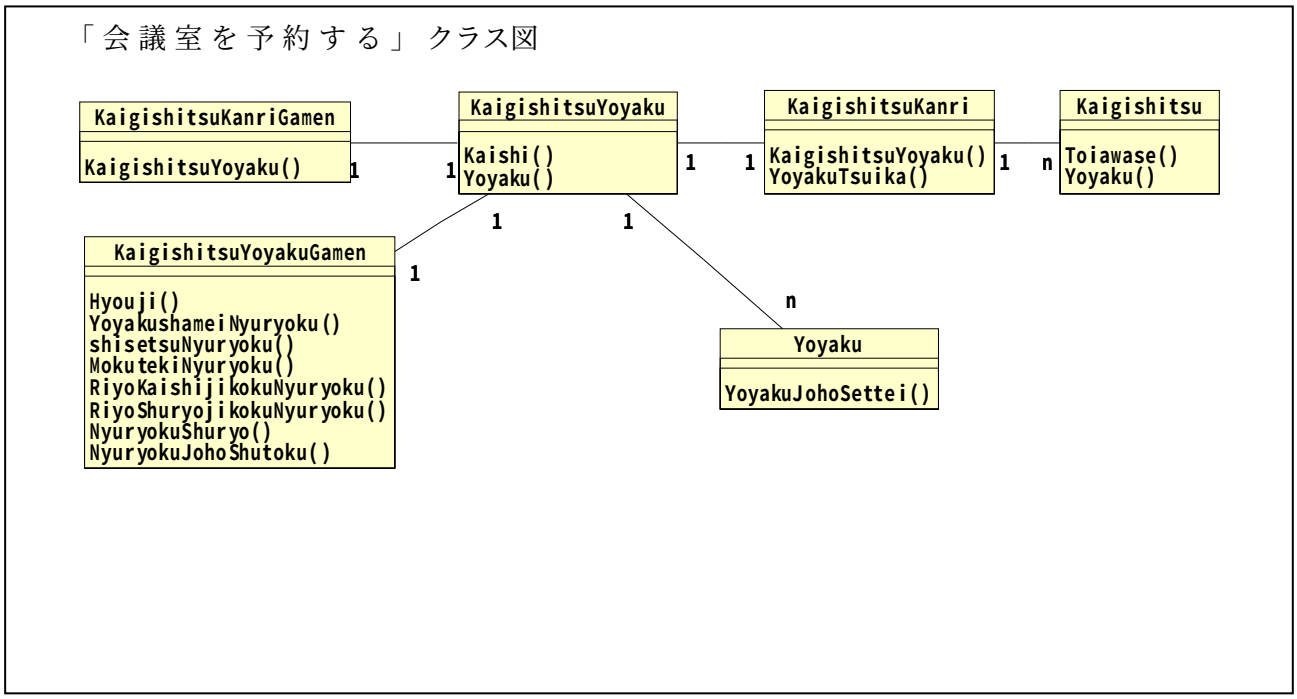
(ホ) 作成したクラス図に対し、各図との整合性・関連付けを行い“ 操作 ”を定義する。

(e) 各シナリオごとに作成したクラス図を統合する。その際に新たなクラスが必要となった場合は追加する。

「会議室を予約する」クラス図



上のクラス図を、さらにプログラミングに近いものに書き換えたのが下記のクラス図である。



３．６．３　作成時の注意点

(1) クラス図は個々のシナリオから作られた相互作用図より作成される。相互作用図の数だけ作成されたクラス図をマージする際には、関連クラスの作成等の問題が生じることがあるため注意が必要。

(2) クラス図の詳細を詰める際には、アクティビティ図やステートチャート図を用いると良い。

３．６．４　評価

(1) クラス図は静的な構造を示す図であり、UML の表記法の中でも最もプログラムに近いものである。そのためクラス図は UML を使った開発において最も重要な図であるといえる。実際の開発はクラス図を中心に行われ、クラスの分類や関連を洗練させながら進めていくと思われる。UML を理解する上で、クラス図を理解することが最も重要であろう。

(2) 初心者だけでは、正確にクラスを抽出したり関連を定義することは難しいため、有識者によるレビューが望ましい。

３．７　ステートチャート図

３．７．１　概要

(1) ステートチャート図とは

(a) １つのオブジェクト（システムの一部分）に注目し、オブジェクトの状態変化（状態遷移）を表すための図

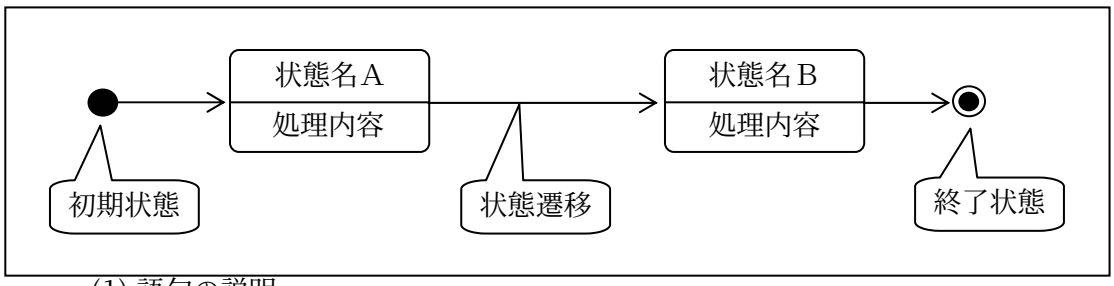
(2) ステートチャート図を書く意味

(a) イベントの発生や時間の経過とともに、オブジェクトの状態が変化する様子を表現できる。

(3) 他図との関連

(a) シーケンス図、クラス図を元に作成する。

３．７．２　記述方法



(1) 語句の説明

(a) 初期状態：オブジェクトが最初にとる状態を表す。

(b) 終了状態：オブジェクトの終了を表す。

(ｲ) オブジェクトが永続的に存在する場合、終了状態が無い場合がある。

(ﾛ) 終了する条件が複数ある場合、終了状態も複数ある場合がある。

(c) 状態遷移：イベントの発生により状態が変化することを表す。

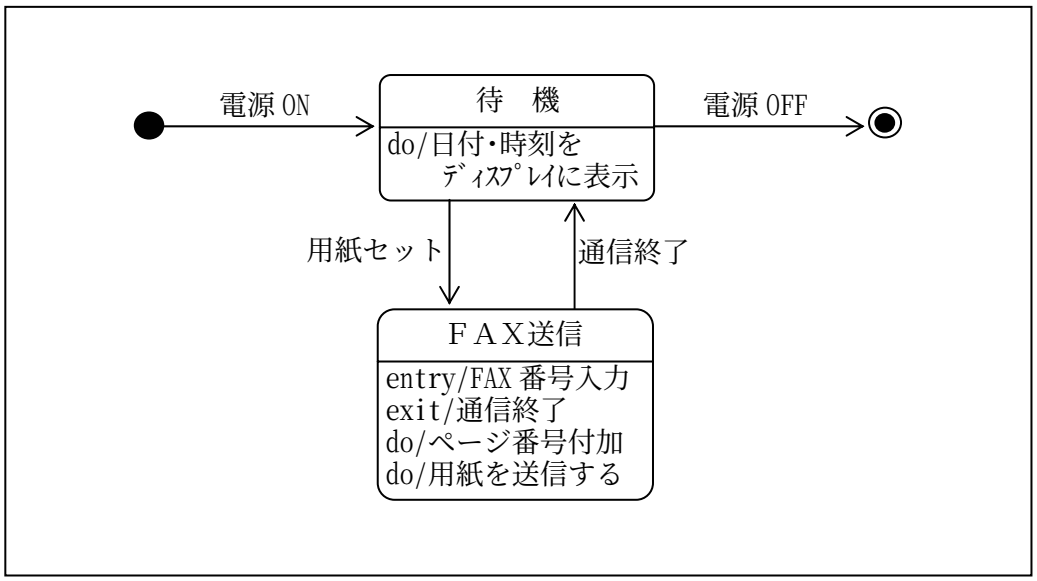
(d) 処理内容：次の三つを記述できる。

(ｲ) ntry/　この状態になった時、実行する処理

(ﾛ) exit/　この状態を抜ける時、実行する処理

(ﾊ) do/　この状態に留まっている間に、実行する処理

(2) ＦＡＸでの送信処理を例に、具体例を示します。



３．７．３　作成時の注意点

(1) 次の箇所でオブジェクトの状態変化が起きる可能性があり、ステートチャート図作成時の判断材料とする。

(a) シーケンス図の「メッセージ」がイベントとなり、状態変化が発生する場合。

(b) クラス図の「操作」によって、状態変化が発生する場合。

３．７．４　評価

(1) オブジェクトの状態変化を表すステートチャート図は、システムを構築する上で必要不可欠な要素だと考えられる。システムの開発段階で考えて見ると、個々のオブジェクトの振る舞いについて、ステートチャート図で記述しておくと、設計者・開発者などグループ内での共通認識ができ、開発がスムーズに行えると思われる。

(2) ステートチャート図に対して、今回の部会内で十分に検討できたとは言えず、　　現状は「状態の変化を表す」ための図であるという認識しかなく、他図との関連が明確ではない。

３．８　パッケージ図

３．８．１　概要

(1) パッケージ図とは

(a) システムの全体構造をパッケージ（グループ）化して表現する。

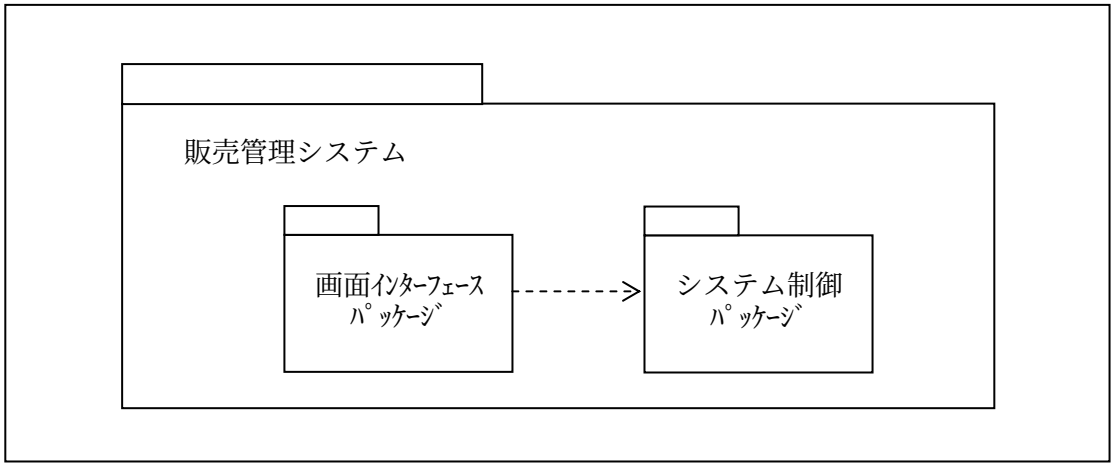
(b) 階層化して表現することも可能。

(c) クラス図の表現方法の１種であり、ＵＭＬでの正式な定義ではない。

(2) パッケージ図を書く意味

(a) クラス図をグループ化し、システム全体構造を理解し易くする。

３．８．２　記述方法



(1) フォルダの形をした図で、パッケージを表す。

(2) 各パッケージは依存関係で関連付けられ、破線矢印で表す。

３．８．３　作成時の注意点

(1) グループ化の方法として、次の２つの方法がある。

(a) クラスを機能別に分類する。

(ｲ) バウンダリクラス　　画面等のインターフェース部分

(ﾛ) コントロールクラス　システム制御部分

(ﾊ) エンティティクラス　内部処理（ＤＢアクセス等処理結果を求める）部分

(b) クラスを業務別に分類する。

３．８．４　評価

(1) システムの全体構成を表すことが可能な為、比較的大きなシステムの理解を深める上で重宝な図だと考える。しかし、必ず必要な図ではなく、必要に応じて作成すれば良いと思われる。

(2) 設計者の経験・考え方によってグループ化の方法が異なることが想定でき、設計者が変わると、パッケージ図も異なってくると思われる。一概にどの分類方法が正しいなどとは言えないが、パッケージ図の作成者は、そのシステムの理解を助けるような図を作成するよう心がける必要がある。

３．９　コンポーネント図

３．９．１　概要

(1) コンポーネント図とは

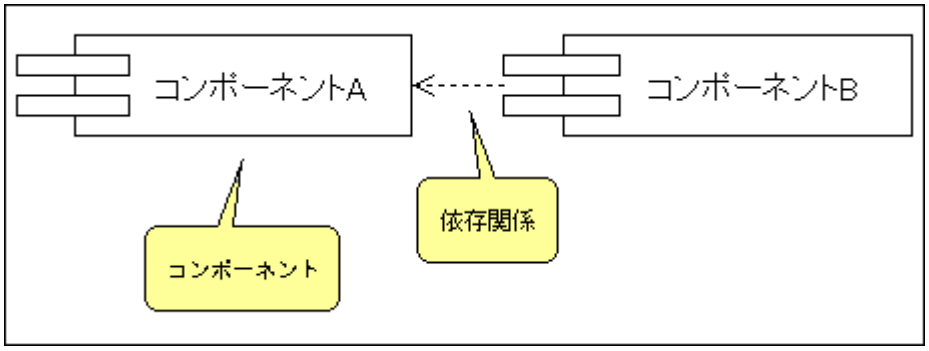
(a) 開発環境内のソフトウェアモジュール構成を表現する。

(b) クラスやオブジェクトの実装コンポーネントへの割り当てを表示したり、ソースコードや実行モジュールの間の依存関係（コンパイルやリンクの順序など）を表現したりする。

(1) 他図との関連

(a) 配置図上にコンポーネント図を描くことにより、各コンポーネントがどのノードで実行されるかが明らかになる。

３．９．２　記述方法



(1) 語句の説明

(a) コンポーネント：・コンパイルやリンクや実行の単位で、システムの物理的構造を組み立てるブロックの役目を果たす。

・いろいろな種類のコンポーネントを定義することができる。

（exe ファイル、dll ファイル、メインプログラム、ヘッダファイルなど）

(b) 依存関係：コンポーネント間の使用関係を表す。

３．９．３　作成上の注意点

特になし

３．９．４　評価

(1) 論理的次元ではなく物理的な次元で示す必要があるときに有効である。

(2) どのコンポーネントどうしが互いに通信するのかを示すときに、依存関係が役立つ。

３．１０　配置図

３．１０．１　概要

(1) 配置図とは

(a) システム全体の物理的な構成を表す図。

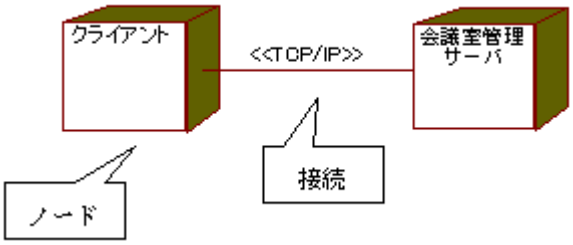
(2) 配置図を作成する意味

(a) システムに関連するハードウェアの接続状況が確認できる。

(3) 他図との関連

(a) ハードウェアにコンポーネントが存在するかを示すこともあるためコンポーネント図とあわせて表記することもできる。

３．１０．２　記述方法



(1) 語句説明

(a) ノード：ノードとは立方体で表記されコンピュータ資源を表すオブジェクトの総称であり、プロセッサとデバイスの 2 種類がある。プロセッサはコンポーネントを実行できるノード、デバイスはコンポーネントを実行できないノードを表す。
ノード内には資源の種類を示すステレオタイプを追加することもできる。

(b) 接続：接続とはノード間を結ぶ実線で表記され各コンピュータ資源がケーブル等で接続していることを表す。

(2) 作成方法・手順

(a) システム構成しているハードウェアを洗い出しノードとして表記する。

(b) 各構成機器の接続状況を把握しノード間を実線で結ぶ。

３．１０．３　作成時の注意点

(1) 機器構成を洗い出す際漏れのないよう注意する。

３．１０．４　評価

(1) ＵＭＬ技法では唯一ハードウェア間の関連を定義した図であり、どのコンポーネントをどの機器に持つかなどを定義することができる。

(2) コンポーネント図とあわせて表記することによりどのコンポーネントがどのハードウェアに配置されているかを確認することができる。

(3) コンポーネントを配置した際、クラス図等に示された処理が現在のシステム構成で無理なく実現できるかなど検討することができる。

(4) ネットワーク構成を表記することによりシステム構成上の配置・台数等も把握できる。

第４章　まとめ

今回の部会を通じてＵＭＬ技法ではどのような表記法がありどのように利用されているか調査することができた。以下に今回の部会を通じて理解できたことを記述する。

- ＵＭＬの利用メリットとして大きく①設計者がプログラマとの意思疎通に利用できる、②設計者がユーザーとの意思疎通に利用できる点の 2 つがあげられる。
- またＵＭＬのメリットにはお互いの考えを正確にコミュニケーションできることに加え、問題発見にも役立つ。従来の大量のソースコードに比べクラス図という一覧性のよい抽象化した図を利用するとことにより問題の発見が容易になると思われる。
- しかしＵＭＬは所詮表記法にしかすぎずどのように記述すればメリットが得られるかはＵＭＬを記述する人の経験・スキルに依存する。ＵＭＬの特長としていろいろな表記法が目的別に準備されているためシステムの規模・内容に応じて必要なものを必要に応じて作成することが大事である。以下の表参照

ユースケース図	分析
アクティビティ図	分析/設計
ステチャート図	分析/設計
クラス図	分析/設計
パッケージ図	分析/設計
シーケンス図	分析/設計
コラボレーション図	分析/設計
オブジェクト図	分析/設計
コンポーネント図	設計
配置図	設計

- 作成に関しては、設計工程のどのタイミングでどの図を使うかの判断もＵＭＬは規定しておらず記述する人の裁量に任されている。今回のわれわれのように順次作成できるものから作成ルール（今回部会内で考えた方法）に従って作成することは可能であるが、必ずしも 1 回で適切な図を作成することはできないと思われる。一般的には①ユーザーが要求する機能を表現するユースケース図を作成し、②静的なデータ構造を示すクラス図にまとめ、③動的なアルゴリズムを表現するシーケンス図・コラボレーション図・ステートチャート図・アクティビティ図で補足するケースが多いと思われる。作成した図においても作業工程が進むにつれ図の見直しを行い不足箇所の追加・変更等も必要であると思われる。

また今回の部会においては各表記法の概要は理解できたものの、本来の実務にて使用したわけではない為、利用価値・利用方法が明確でない点が上げられる。しかし、各表記法の有効性が十分理解できれば適切に利用しシステム開発時に効果的に利用できるものではないかと推測される。

参考文献

１．かんたんＵＭＬ
　　翔泳社　　(株)オージス総研
２．独習ＵＭＬ
　　翔泳社　　Joseph Schmuller
３．ＵＭＬ　ＰＲＥＳＳ
　　技術評論社

参考サイト

基礎編　：　UML 入門
http://www.mamezou.com/tec/Tips/umlForBeginner/index.html

技術情報：知恵の蔵
http://www.mamezou.com/tec/