Advanced Responsibility-Driven Design

Rebecca Wirfs-Brock

責務駆動設計の上級テクニック

Rebecca Wirfs-Brock

What Is Responsibility-Driven Design?

A way to design software that…

- –emphasizes modeling of objects' **roles**, **responsibilities**, and **collaborations**
- –uses informal tools and techniques
- –adds responsibility concepts and thinking to any design process

責務駆動設計とは?

以下のようにソフトウェアを設計する方法:

- –オブジェクトの**ロール**、**責務**、**コラボレーション**のモデリングを強調
- –形式にこだわらない道具とテクニックを使用
- –設計プロセスに、責務の概念と考え方を追加

Responsibility-Driven Design Principles

Maximize Abstraction

- Initially hide the distinction between data and behavior. Think of objects responsibilities for “knowing”, “doing”, and “deciding”

Distribute Behavior

- Promote a delegated control architecture
- Make objects smart— have them behave intelligently, not just hold bundles of data

Preserve Flexibility

- Design objects so interior details can be readily changed

責務駆動設計の原則

抽象化を最大限に行う

- データと振る舞いの違いを最初は隠す. オブジェクトの責務を「知っている」、「行う」、「判断する」と考える

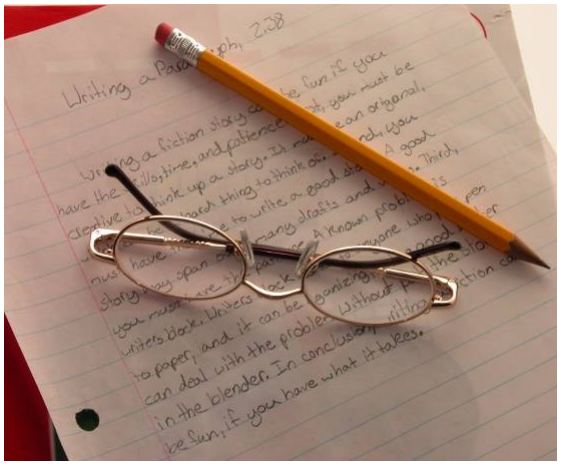
振る舞いを分散する

- 制御を委譲するアーキテクチャを推奨する
- オブジェクトを賢くする— 知的に振舞うように、データを束で持つだけにしない

柔軟性を保つ

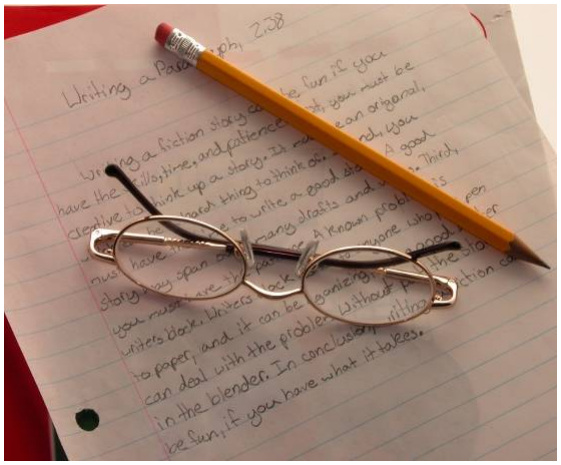
- オブジェクトの内部の詳細をいつでも変えられるように設計する

A Designer's Story: A tool for seeing what's important



Designer's story—a quickly written paragraph or two description of important ideas, what you know, and what you need to discover

設計ストーリー: 何が大事かを理解するための道具



設計ストーリー—何が分かっており、何を見つけないといけないかなどの重要なアイデアをすばやく、1，2段落で記述したもの

Elements of a story…

What is your design supposed to do?

Is there something similar you can draw upon or emulate?

What will make it a success?

What are the most challenging parts?

ストーリーの要素…

設計したものが行うだろうことは何か?

何か利用したり、まねることができるような似たものが存在するか?

設計を成功させるためのものは何か?

もっとも手ごわい部分は何か?

Why tell a designer's story?

To state your perspective about what's important

Describing the problem helps you own it

Sharing your story with your team builds understanding and a common vision

Metaphors are hard to come by…identifying themes and key responsibilities from designer stories is one alternative

なぜ、設計ストーリーを語るか?

何が重要かについての自分の見方を述べる

問題を記述すれば、対応するのに役立つ

チームがストーリーを共有することで、理解や共通のビジョンを形成できる

メタファーを思いつくのは難しい…その代わりに設計ストーリーからテーマと中心的な責務を識別する

An Online Banking Story—I

The online banking application provides internet access to banking services. It should be easily configured to work for different banks. A critical element in the design is the declaration of a common way to call different backend banking systems. We will define a common set of banking transactions and a framework that will call into banking-specific code that "plugs into" the standard layer implementing the details. The rest of our software will only interface with the bank-independent service layer.

オンラインバンキングの設計ストーリー—I

このアプリケーションは、バンキングサービスをインターネットから利用できるようにするものである。このアプリケーションでは、異なる銀行に対応して機能するように簡単に構成できなければならない。設計において重要なのは、異なるバックエンドの銀行システムを呼び出す共通の方法と、限りのあるリソースを共有するための信頼性の高い手段の両方を宣言することである。そのため、共通の銀行取引の集合と、銀行取引に特化したプログラムコードを呼び出すためのフレームワークを定義することになるだろう。このフレームワークは、共通の取引処理の詳細を実装している標準レイヤに、この固有のプログラムコードを「組み込む」ことになるだろう。ソフトウェアの残りの部分は、単に、この個別の銀行に依存しないサービスを提供するレイヤとやり取りするものになるだろう。

An Online Banking Story—cont.

At the heart of our system is the ability to rapidly configure our application to work for different backends and to put a different user interface on each. This includes customizing screen layouts, messages and banner text. The online banking functions are fairly simple: customers register to use the online banking services, then log in and access their accounts to make payments, view account balances and transaction histories, and transfer funds.

オンラインバンキングの設計ストーリー—続き

このシステムで最も大切なのは、異なるバックエンドとともに機能するようにすばやく構成できることと、各バックエンドのインターフェースをすっきりしたものにすることである。これには、画面レイアウト、メッセージ、バナーテキストのカスタマイズが含まれる。オンラインバンキング機能は、次のようなかなりシンプルなものである。「顧客は、オンラインバンキングサービスを使用するための手続きを行う。次に、ログインして、自分の口座にアクセスし、支払や残高照会、取引履歴照会、振込を行う」。

Identify Story Themes

Themes are key areas of system activity and design focus

- Online Banking themes
  - Common banking functions
  - Managing scarce resources
  - Flexible configuration

Themes can be explored to find candidate objects

設計ストーリーのテーマを識別する

テーマは、システムの活動の中心的な部分と設計の焦点である

- オンラインバンキングのテーマ
  - 共通のバンキング機能
  - 乏しいリソースを管理する
  - 柔軟な構成

テーマを検討することで、オブジェクトの候補を見つけることができる

What To Look For

Look for inventions that represent:

- The ***work*** your software performs
- The ***things*** your software affects or is connected to
- The ***information*** that flows through your software
- Your software's ***decision-making***, ***control*** and ***coordination*** activities
- Ways to ***structure*** and ***manage*** groups of objects
- ***Representations*** of real world things your software needs to know something about

探すべきもの

以下のようなものを表現する設計概念を探す:

- ソフトウェアが行う「***仕事***」
- ソフトウェアが影響を与えたり、接続する「***もの***」
- ソフトウェアを通じて流れる「***情報***」
- ソフトウェアが行う「***判断***」、「***制御***」、「***調整***」の活動
- オブジェクトの集合を「***構造化***」し、「***管理***」する方法
- ソフトウェアが知らなければならない実世界の物事を「***表現***」したもの

Finding Online Banking Objects

Banking Activities and Common Services

Theme	Perspective	Candidate Objects
Common Banking Functions	What work needs to be done?	Performing financial transactions, Querying accounts, verifying users (**transactions, queries, users**)
	What things does the software affect or is it connected to?	Connection to backend bank systems (**bank connection**)
	What information flows through the software system?	Convey information about **financial transactions**, **account**s, **user**s
	What real world things does the software need to know about?	Representing **customer**s, **user**s and the **bank account**s they access

オンラインバンキングのオブジェクトを見つける

バンキング活動と共通サービス

テーマ	観点	オブジェクトの候補
共通バンキング機能	何をなさねばいけないか?	金融取引を実行する, 口座を問い合わせる, ユーザを検証する (**トランザクション, 問い合わせ, ユーザ**)
	ソフトウェアが影響を与えたり、接続するものは何か?	バックエンドのバンキングシステムへのコネクション (**銀行コネクション**)
	ソフトウェアシステムを通じて流れる情報は何か?	**金融取引**、**口座**、**ユーザー**に関する情報が運ばれる
	ソフトウェアが知らなければならない実世界の物事は何か?	**顧客**、**ユーザー**、とそれらの人がアクセスする**銀行口座**を表すもの

Role Stereotypes: A tool for seeing and shaping object behaviors

stereotype—A conventional, formulaic, and oversimplified conception, opinion, or image

ロールステレオタイプ: オブジェクトの振る舞いを理解し、形作るための道具

ステレオタイプ—型にはまった紋切り型で過度に単純化された概念、意見、イメージ

Object Role Stereotypes

Information holder - knows and provides information

Structurer - maintains relationships between objects and information about those relationships

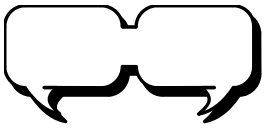
オブジェクトロールステレオタイプ

情報保持役 – 情報を知っており、提供する

構造化役 – オブジェクト間の関係と、それらの関係についての情報を維持する

Object Role Stereotypes

Interfacer - transforms information and requests between distinct parts of a system



Service provider - performs work on demand

オブジェクトロールステレオタイプ

インタフェース役 – システムの異なる部分の間でリクエストの受け渡しや情報の変換を行う

サービス提供役 – 求めに応じて、仕事を行う

Object Role Stereotypes

Coordinator – mechanically reacts to events

Controller - makes decisions and closely directs others' actions

Wirfs-Brock Associates www.wirfs-brock.com

オブジェクトロールステレオタイプ

調整役 – イベントに機械的に対応する

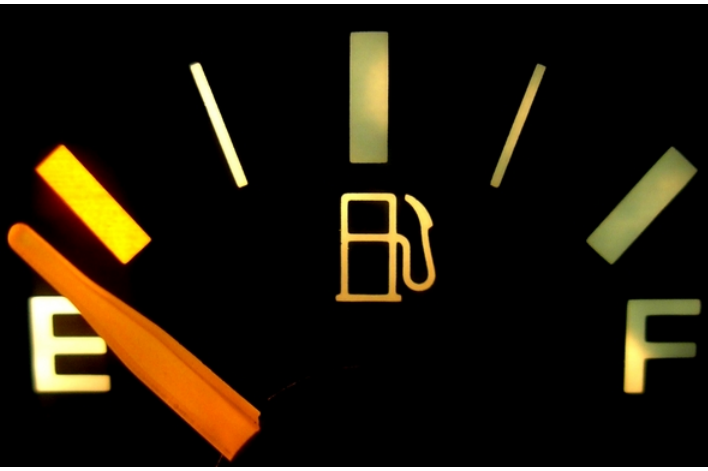
制御役 – 判断を行い、他のオブジェクトのアクションを指示する

Wirfs-Brock Associates www.wirfs-brock.com

Adjusting Control

Skilled designers can tune an object to be more or less controlling by relocating decision-making responsibilities

制御を調整する



熟練した設計者は、判断を行う責務の配置を変えることでオブジェクトが制御する度合いを多くも少なくも調整できる

Three Uses for Object Role Stereotypes

1. Think about design objects needed
2. Study a design to learn what types of roles predominate and how they interact
3. Blend stereotypes to make objects more responsible
   - information holders that compute
   - service providers that maintain information
   - structurers that derive facts
   - interfacers that transform

オブジェクトロールステレオタイプの3つの用途

1. 必要な設計オブジェクトを考える
2. 設計を調べて、どの種類のロールが多く、それらがどのように相互作用しているかを知る
3. ステレオタイプを併用し、オブジェクトにもっと責任を持たせる
   - 情報保持役が演算する
   - サービス提供役が情報を維持する
   - 構造化役から情報を得る
   - インターフェイス役が変換する

Pulling up a level…to compare

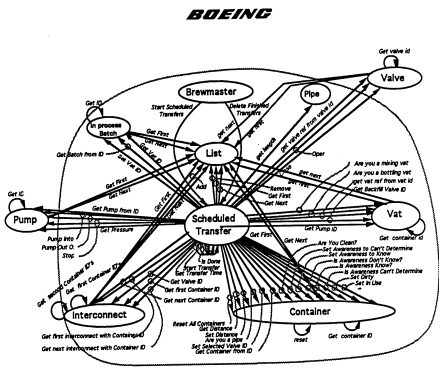


FIGURE 6.3-10 TRANSFER A BATCH

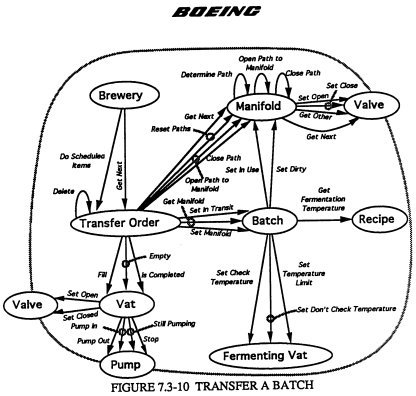


FIGURE 7.3-10 TRANSFER A BATCH

"The Object-Oriented Brewery: A Comparison of Two Object-Oriented Methods,"R. Sharble and S. Cohen, Boeing Technical Report BCS-G4059, 1992.

"How Designs Differ", R. Wirfs-Brock, Smalltalk Report, vol. 1, no. 4

比較するために…1つレベルを上げる

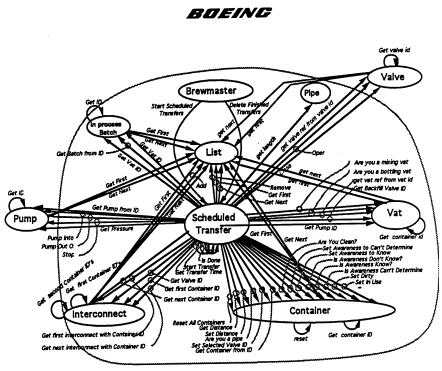


FIGURE 6.3-10 TRANSFER A BATCH

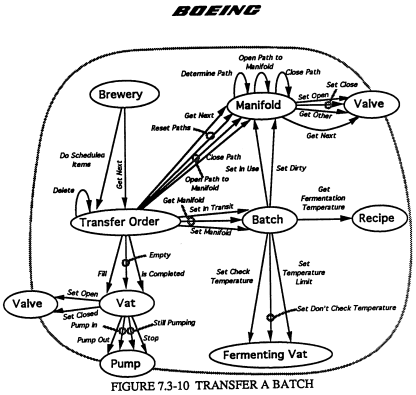


FIGURE 7.3-10 TRANSFER A BATCH

"The Object-Oriented Brewery: A Comparison of Two Object-Oriented Methods,"R. Sharble and S. Cohen, Boeing Technical Report BCS-G4059, 1992.

"How Designs Differ", R. Wirfs-Brock, Smalltalk Report, vol. 1, no. 4

…and characterize

Data-Driven Design Approach	Responsibility-Driven Design Approach
centralized control	delegated control
controllers	coordinators
inherited attributes	inherited behavior
many low-level messages	fewer, higher-level messages
lots of simplistic information holders	a few smart objects that blend role stereotypes

…そして特徴をまとめると

データ駆動型設計	責務駆動型設計
集中型制御	委譲型制御
制御役	調整役
属性の継承	振る舞いの継承
多数の低レベルなメッセージ	少数のハイレベルなメッセージ
多数の単純な情報保持役	複数のロールステレオタイプが並存する少数の賢いオブジェクト

n-tier web applications

Layer	Functionality	Role	Technique
Client	User Interface	**Interfacer**	HTML, JavaScript
Presentation	Page Layout	**Interfacer**	JSP
Control	Command	**Coordinator**	Servlet
Business Logic	Business Delegate	**Controller**	POJO, Session EJB
Data Access	Domain Model	**Information Holder, Structurer**	JavaBean, Entity EJB
Resources	Database, Enterprise Services	**Service Provider**	RDBMS, Queues, Enterprise Service Bus

N層のwebアプリケーション

層	機能	ロール	テクニック
クライアント	ユーザインターフェース	**インタフェース役**	HTML, JavaScript
プレゼンテーション	ページレイアウト	**インタフェース役**	JSP
制御	コマンド	**調整役**	Servlet
ビジネスロジック	ビジネスの代表	**制御役**	POJO, Session EJB
データアクセス	ドメインモデル	**情報保持役, 構造化役**	JavaBean, Entity EJB
リソース	データベース, 企業サービス	**サービス提供役**	RDBMS, Queues, Enterprise Service Bus

Finding Role Stereotypes in Patterns

A Mediator is a coordinator

A Strategy is a service provider

State objects are service providers

An Adapter is an interfacer

…

パターンにおいてロールステレオタイプを探す

Mediatorは調整役

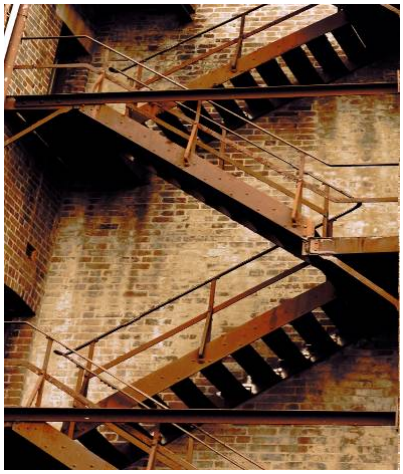
Strategyはサービス提供役

Stateオブジェクトはサービス提供役

Adapterはインターフェース役

…

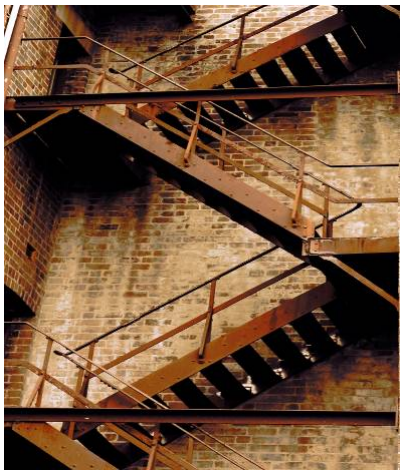
Technique: Seeing at different abstract levels



We can see objects and behavior at different levels:

- At the *conceptual* level- a set of responsibilities
- At the *specification* level- set of methods that can be invoked
- At the *implementation* level- code and data

テクニック: 異なる抽象化レベルで見る



オブジェクトや振る舞いを異なる抽象化レベルで見ることができる:

- *概念*レベル- 責務の集合
- *仕様*レベル- 呼び出すことができるメソッドの集合
- *実装*レベル- コードとデータ

Technique: Pull Up a Level
Reverse engineer a class into responsibilities

The Java Calendar class

Internally, Calendar keeps track of a point in time in two ways. First, a "raw" value is maintained, which is simply a count of milliseconds since midnight, January 1, 1970 GMT, or, in other words, a Date object. Second, the calendar keeps track of a number of fields, which are the values that are specific to the Calendar type. These are values such as day of the week, day of the month, and month. The raw millisecond value can be calculated from the field values, or vice versa.

Calendar also defines a number of symbolic constants. They represent either fields or values. For example, MONTH is a field constant. It can be passed to get() and set() to retrieve and adjust the month. AUGUST, on the other hand, represents a particular month value. Calling get(Calendar.MONTH) could return Calendar.AUGUST.

Calendar Methods

public int getFirstDayOfWeek()

This method returns the day that is considered the beginning of the week for this Calendar. This value is determined by the Locale of this Calendar. For example, the first day of the week in the United States is Sunday, while in France it is Monday.

public abstract int getGreatestMinimum(int field)

This method returns the highest minimum value for the given time field, if the field has a range of minimum values. If the field does not have a range of minimum values, this method is equivalent to getMinimum().

public abstract int getLeastMaximum(int field)

This method returns the lowest maximum value for the given time field, if the field has a range of maximum values. If the field does not have a range of maximum values, this method is equivalent to getMaximum(). For example, for a GregorianCalendar, the lowest maximum value of DATE_OF_MONTH is 28.

public abstract int getMaximum(int field)

This method returns the maximum value for the given time field. For example, for a GregorianCalendar, the maximum value of DATE_OF_MONTH is 31.

public final void set(int year, int month, int date)

This method sets the values of the year, month, and day-of-the-month fields of this Calendar.

public final void set(int year, int month, int date, int hour, int minute) This method sets the values of the year, month, day-of-the-month, hour, and minute fields of this Calendar.

public final void set(int year, int month, int date, int hour, int minute, int second)

This method sets the values of the year, month, day-of-the-month, hour, minute, and second fields of this Calendar.

public void setFirstDayofWeek(int value)

This method sets the day that is considered the beginning of the week for this Calendar. This value should be determined by the Locale of this Calendar. For example, the first day of the week in the United States is Sunday; in France it's Monday.

public void setLenient(boolean lenient)

This method sets the leniency of this Calendar. A value of false specifies that the Calendar throws exceptions when questionable data is passed to it, while a value of true indicates that the Calendar makes its best guess to interpret questionable data. For example, if the Calendar is being lenient, a date such as March 135, 1997 is interpreted as the 135th day after March 1, 1997.

public void setMinimalDaysInFirstWeek(int value)

This method sets the minimum number of days in the first week of the year. For example, a value of 7 indicates the first week of the year must be a full week, while a value of 1 indicates the first week of the year can contain a single day. This value should be determined by the Locale of this Calendar.

public final void setTime(Date date)

This method sets the point in time that is represented by this

Calendar.

public void setTimeZone(TimeZone value)

This method is used to set the time zone of this Calendar.

テクニック: 1レベル引き上げる
クラスを責務にリバースエンジニアリングする

The Java Calendar class

Internally, Calendar keeps track of a point in time in two ways. First, a "raw" value is maintained, which is simply a count of milliseconds since midnight, January 1, 1970 GMT, or, in other words, a Date object. Second, the calendar keeps track of a number of fields, which are the values that are specific to the Calendar type. These are values such as day of the week, day of the month, and month. The raw millisecond value can be calculated from the field values, or vice versa.

Calendar also defines a number of symbolic constants. They represent either fields or values. For example, MONTH is a field constant. It can be passed to get() and set() to retrieve and adjust the month. AUGUST, on the other hand, represents a particular month value. Calling get(Calendar.MONTH) could return Calendar.AUGUST.

Calendar Methods

public int getFirstDayOfWeek()

This method returns the day that is considered the beginning of the week for this Calendar. This value is determined by the Locale of this Calendar. For example, the first day of the week in the United States is Sunday, while in France it is Monday.

public abstract int getGreatestMinimum(int field)

This method returns the highest minimum value for the given time field, if the field has a range of minimum values. If the field does not have a range of minimum values, this method is equivalent to getMinimum().

public abstract int getLeastMaximum(int field)

This method returns the lowest maximum value for the given time field, if the field has a range of maximum values. If the field does not have a range of maximum values, this method is equivalent to getMaximum(). For example, for a GregorianCalendar, the lowest maximum value of DATE_OF_MONTH is 28.

public abstract int getMaximum(int field)

This method returns the maximum value for the given time field. For example, for a GregorianCalendar, the maximum value of DATE_OF_MONTH is 31.

public final void set(int year, int month, int date)

This method sets the values of the year, month, and day-of-the-month fields of this Calendar.

public final void set(int year, int month, int date, int hour, int minute) This method sets the values of the year, month, day-of-the-month, hour, and minute fields of this Calendar.

public final void set(int year, int month, int date, int hour, int minute, int second)

This method sets the values of the year, month, day-of-the-month, hour, minute, and second fields of this Calendar.

public void setFirstDayofWeek(int value)

This method sets the day that is considered the beginning of the week for this Calendar. This value should be determined by the Locale of this Calendar. For example, the first day of the week in the United States is Sunday; in France it's Monday.

public void setLenient(boolean lenient)

This method sets the leniency of this Calendar. A value of false specifies that the Calendar throws exceptions when questionable data is passed to it, while a value of true indicates that the Calendar makes its best guess to interpret questionable data. For example, if the Calendar is being lenient, a date such as March 135, 1997 is interpreted as the 135th day after March 1, 1997.

public void setMinimalDaysInFirstWeek(int value)

This method sets the minimum number of days in the first week of the year. For example, a value of 7 indicates the first week of the year must be a full week, while a value of 1 indicates the first week of the year can contain a single day. This value should be determined by the Locale of this Calendar.

public final void setTime(Date date)

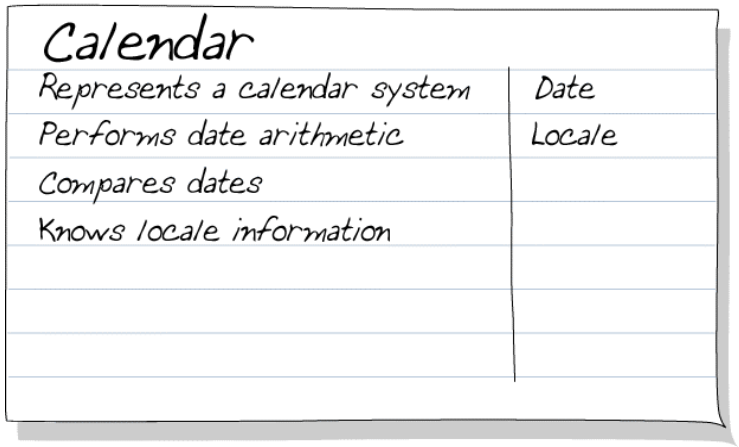
This method sets the point in time that is represented by this

Calendar.

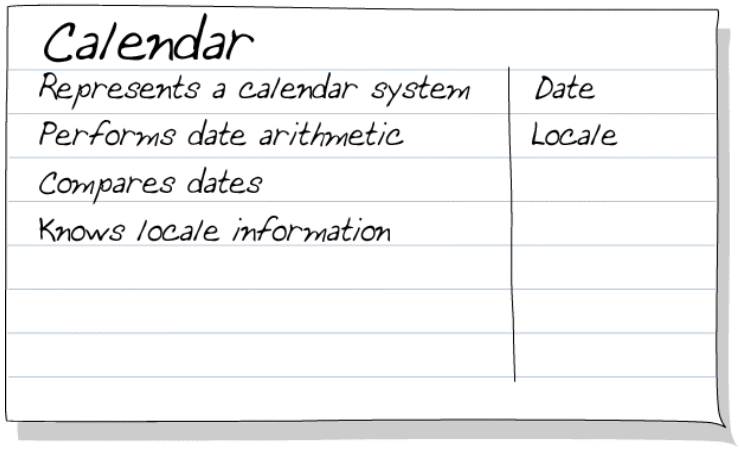
public void setTimeZone(TimeZone value)

This method is used to set the time zone of this Calendar.

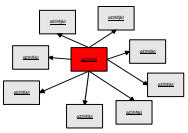
To get a general picture: Calendar revealed



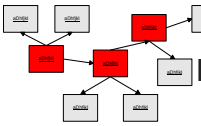
一般的な姿が得られる: 現れたCalendarクラス



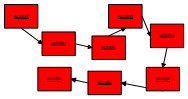
Control centers and collaboration styles: Tools for shaping solutions



Centralized



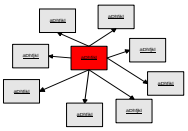
Delegated



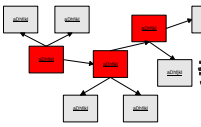
Dispersed

control center—a place where objects charged with controlling and coordinating reside

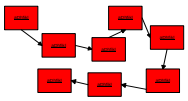
制御センターとコラボレーションスタイル: 解決策に形を与えるための道具



集中型



委譲型



分散型

制御センター—制御や調整を行うオブジェクトが住む場所

Control Design

Deciding on and consciously choosing a particular control style is one of the most important design decisions.

There are many control centers in your design, each may have a different style

Involves decisions about

- how to control and coordinate tasks,
- where to place responsibilities for making domain-specific decisions (rules), and
- how to manage unusual conditions (the design of exception detection and recovery)

Goal: Make similar parts of a design consistent

制御の設計

特定の制御スタイルに決めたり、意図的に選択したりすることはもっとも大事な設計判断の1つである.

1つの設計において複数の制御センターが存在し、その各々が異なるスタイルを取ることもある

以下についての判断も含まれる

- タスクをどのように制御し、調整するか,
- ドメイン固有の判断 (ルール)をどこに置くか,
- 異常な状況にどのように対応するか (例外の検出と復旧の設計)

目指すべきこと: 設計の類似した部分に一貫性を持たせる

Characteristics of Centralized Control

Generally, one object (the controller) makes most of the important decisions and figures out what to do.

Tendencies to avoid:

- Overly complex control logic
- No work done by other objects

Benefit:
Easy to follow what's going on

Drawback:
Changes can ripple among controlling and controlled objects

集中型制御の特徴

一般的に、1つのオブジェクト(制御役)が重要な判断の大半を行い、何をすべきかを考える

避けるべき傾向:

- 複雑すぎる制御ロジック
- 他のオブジェクトが仕事をしない

長所:
何が行われているのか追うのは簡単

短所:
変更が制御するオブジェクトと制御されるオブジェクトに波及する

Characteristics of Delegated Control

A coordinator passes some decision making and actions off to objects surrounding a control center. Each delegate has a significant role:

- Coordinators tend to know about fewer objects
- Messages are higher-level

Tendency to avoid:

- Helper objects that don't do enough

Benefit:

- Changes typically localized and simpler
- Easier to divide interesting design work among a team

委譲型制御の特徴

調整役が制御センターの周囲のオブジェクトにある程度の判断やアクションを委ねる。委ねられたオブジェクトは、まとまった役割を持つ:

- 調整役は、より少数のオブジェクトを知っていることが多い
- メッセージは、ハイレベル

避けるべき傾向:

- 行うことがそれほどないヘルパーオブジェクト

長所:

- 変更は、一般的に局所化し、簡単になる
- やり応えがある設計作業をチーム内で分割しやすい

Characteristics of Dispersed Control

Decision-making and action are spread among a chain of objects who each perform a small, well-defined task

Tendencies to avoid:

- Little or no value-added by an object in the chain
- Hardwired dependencies

Opportunity:
Design so responsibilities can be chained in novel ways

分散型制御の特徴

判断とアクションは、小さな、適切に定義されたタスクを行うオブジェクトの連鎖に分散する

避けるべき傾向:

- 連鎖中に価値が少なかったり、無かったりするオブジェクトがいる
- 固定した依存性

可能性:
今までにないやり方で責務をつなげることができる

A Strategy For Handling Exceptions For A Key Collaboration

- List most likely exception cases. Name and describe them first
- Then address how to resolve easy-to-recover from ones
- Explore alternatives for more difficult cases. Test for usability and feasibility

中心的なコラボレーションで 例外に対処する戦略

- 起こりそうな例外のケースのリストを作成する。それらのケースに名前を与え、説明を書く
- 簡単に回復できるケースの解決に取り組む
- さらに難しいケースについて複数の対処策を検討する。使いやすさと実現性を確かめる

Assign Exception-Handling Responsibilities to Decision Makers

There are many different ways to “handle” an exception. It could be logged and rethrown (possibly more than once), until someone takes corrective action

Who naturally might handle exceptions?

- The initial requestor. If it knows enough to perform corrective action, then the exception can be taken care of and not be propagated
- As a fallback position, let some object who takes responsibility for controlling the action handle the exceptions
- Controllers and objects located within a control center are natural places for handling exceptions

判断を行うオブジェクトに 例外処理の責務を割り当てる

1つの例外に「対処する」方法は多数ある。誰かが修整するためのアクションを取るまで、その例外をログに記録したり、再び（たぶん2回以上）投げる。

例外を誰が対処するのが自然か?

- 最初にリクエストしたオブジェクト。そのオブジェクトに修整アクションを行うだけの知識があれば, 例外は対応され、伝播しない
- 次善策として、アクションの制御を行う責務を担うオブジェクトに、例外への対処を行わせる
- 制御役と制御センター内に位置するオブジェクトが、例外に対処するのは自然である

Trust Regions: A tool for seeing where "defensive" behavior is needed

Collaborate:

1. To work together, especially in a joint intellectual effort

2. To cooperate treasonably, as with an enemy occupation force

信頼領域:「防衛的」な振る舞いが必要なところを理解するための道具

コラボレートする:

1. 協調すること。特に共同の知的活動において

2. 敵の占領部隊に対するように、腹の中では裏切りながら表向きは協力すること

Trust regions

A trust region is a design area where trusted collaborations occur

Components and objects at the “edges” may take on extra responsibilities

Within a trust region, collaborations can be more collegial

- Check once, then proceed…
- Code deep inside a trust region need not check for well-formed or timely requests

信頼領域

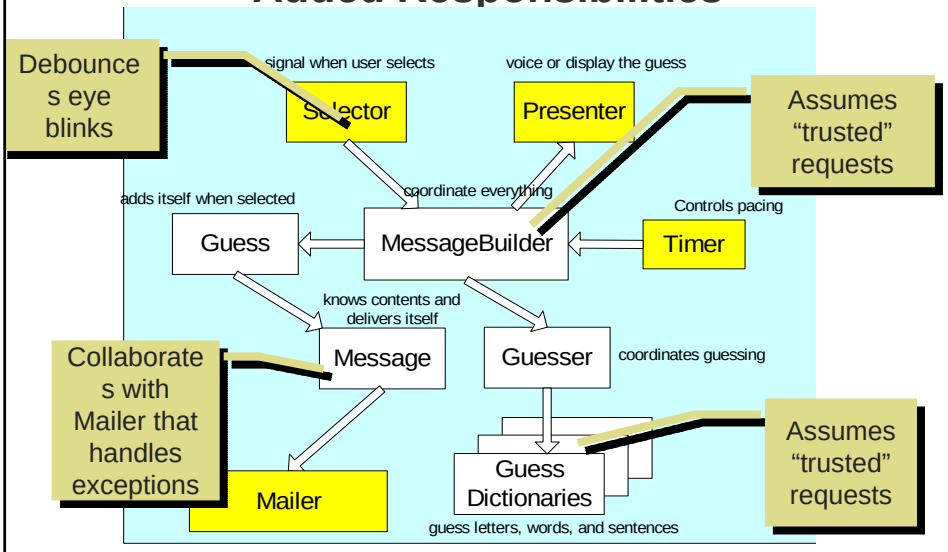
信頼領域は、信頼するコラボレーションが起きる設計部分

「端」のコンポーネントやオブジェクトは、余分な責務を持つことがある

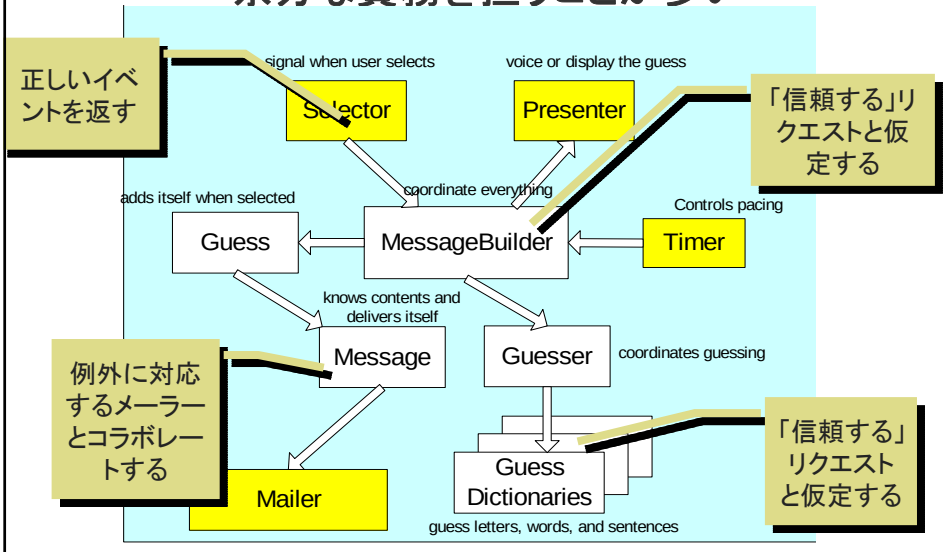
信頼領域の内部では, コラボレーションはもっと打ち解けたものになる

- 1回チェックし、進む…
- 信頼領域の奥にあるコードは、リクエストの形式やタイミングをチェックする必要はない

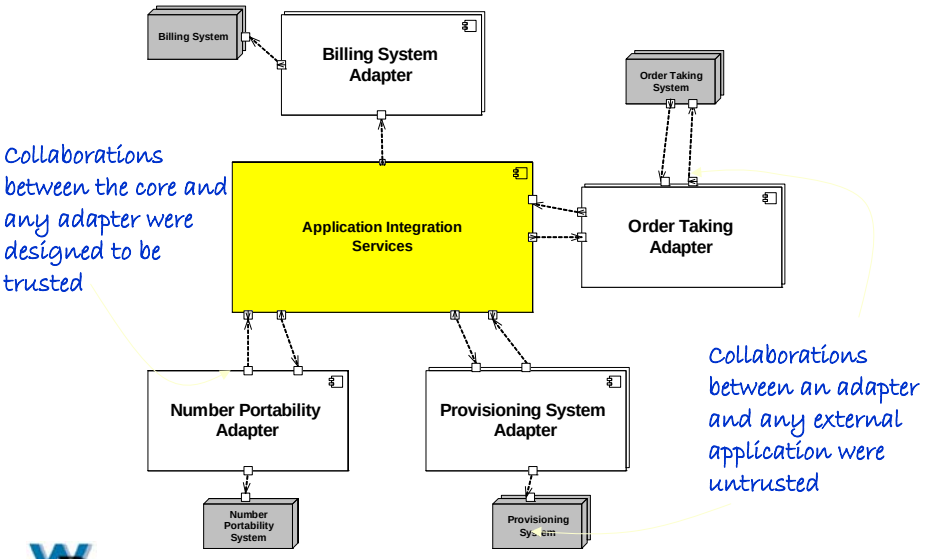
Objects At The “Edges” Often Take On Added Responsibilities



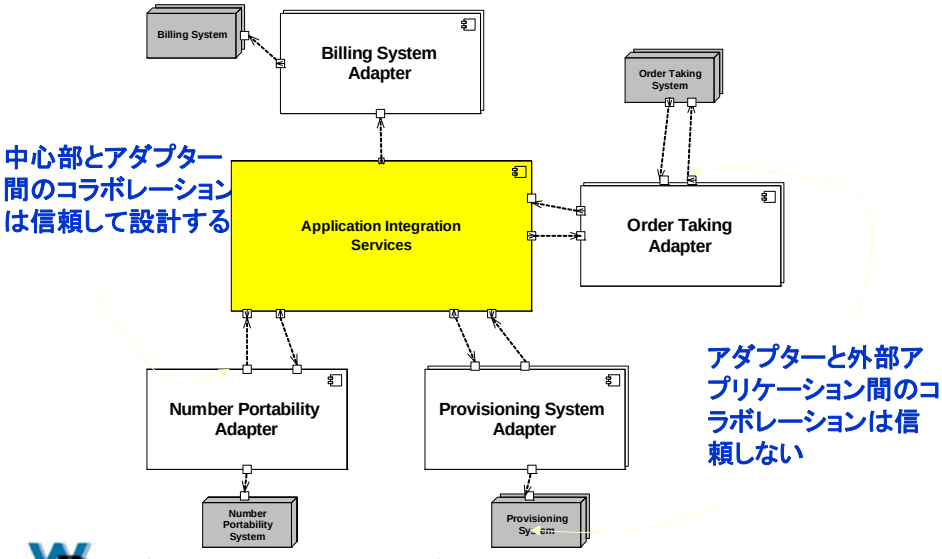
「端」のオブジェクトは、余分な責務を担うことが多い



Trust In A Telco Integration Application



電気通信統合アプリケーションにおける信頼



Collaboration Cases To Consider

Collaborations between objects or components…

- that interface to the user and the rest of the system
- in different layers or subsystems
- inside your system that interface to external systems
- you design and those designed by someone else

There are degrees of trust

Don't design every object to behave defensively

- Redundant checks are hard to keep consistent and lead to brittle code and poor performance

考えるべきコラボレーションのケース

以下のオブジェクトやコンポーネント間のコラボレーション…

- ユーザとシステムの残りの部分とのインターフェースを取る
- 異なるレイヤやサブシステムにいる
- 開発対象のシステム内で外部システムとのインターフェースを取る
- 自分で設計したものと誰か他の人が設計したもの

信頼には、複数の度合いがある

すべてのオブジェクトを防衛的に設計しないこと

- 冗長なチェックは一貫性を保つのが難しく、壊れやすいコードと低い性能を招いてしまう

Principles and Strategies of Adding Flexibility to Your Design

Flexible: Capable of responding or conforming to new or changing situations.
—Webster's Seventh New Collegiate Dictionary

設計に柔軟性を追加する原則と戦略

Flexible（柔軟な）: 新しい状況や変化する状況に対応する能力、もしくは順応する能力があること
—Webster's Seventh New Collegiate Dictionary

When to Consider Flexibility

As soon as you partition responsibilities into related chunks

Monolithic software is hard to change

Early decisions dramatically increase or inhibit your software's ability to flex. So…

- Structure your system so that points of potential change and variation are insulated from the rest of the system
- Divide responsibilities among objects so you encapsulate any behaviors that might change
- Add explicit "hooks" that allow responsibilities to be modified or collaborators to be replaced without affecting pre-existing code

柔軟性を考慮すべきタイミング

責務をいくつかの固まりに分割した直後

一枚岩のソフトウェアは変更が難しい

早期に決断することで、ソフトウェアが柔軟なものになるかならないかが劇的に変わる…

- 変化したり、バリエーションに対応する箇所をシステムの残りの部分から切り離すように、システムの構造を作る
- 変化するかもしれない振る舞いをカプセル化するために、責務を複数のオブジェクトに分ける
- 既存のコードに与えることなく、責務を変更したり、コラボレータを置き換えたりすることを可能にする明示的な「フック」を追加する

Techniques for Adding Flexibility

Refactor and redistribute responsibilities

Design new places where behavior can be “tuned” and services can be replaced

Decide on the best “tuning” strategy given what you know

Use factoring strategies and known patterns

Apply inheritance (when variations are relatively few) or composition (when variations are many or need to change at runtime)

柔軟性を追加するためのテクニック

責務をリファクトしたり、再分配する

振る舞いを「調整」したり、サービスを置き換えることがえdきるような新たな場所を設計する

知っていることにもとづいて、もっともよい「調整」戦略を決める

分解の戦略や既知のパターンを使う

（バリエーションが比較的少ない場合は）継承を使い、（バリエーションが多かったり、実行時に変化しなくてはならない場合は）コンポジション（集約）を使う

Telling How to Adapt a Design

Explain three things:

1. the current design
2. what aspects are adaptable
3. how to make these adaptations

Wirfs-Brock Associates

設計を適合させる方法を説明する

3点を説明する:

1. 現在の設計
2. 適合可能な側面
3. 適合させるための方法

Wirfs-Brock Associates

Recipe for Adding a Guess

To Add a New Kind of Guess

1. Define a class that implements the Guess interface.
   This type of object must know contents, formatted for both display and speech, know how long to wait before continuing with another guess, and be able to add itself to a message.
   Specifically, it must implement these methods:

```
public String displayableText()
public String speakableText()
public String getContent()
public Duration waitTime()
void processMessage(Message m)
```

2. Define a class that implements the Bidder interface.
   This type of object will contain all of the corresponding Guess objects and determine which is most relevant to the current message and how relevant they are. Specifically, it must implement:

```
Bid bidOn(Message m)
```

適合させるためのレシピ

レシピ名: 常に「～する方法」で終える

意図: このレシピを使う理由

設計記述:どのクラスとどのインターフェースが関与していてどれを理解しなければならないか、それらがどんなロールを演じるか、どんなコラボレーションが関与するか。どんな責務がバリエーションを通して適応されるか……。これらは、UML の図と、その他の記述で補強される。

関連するレシピ:類似したバリエーションを達成する複数の代替手段、あるいは関連する下位レシピ。このレシピが複雑な場合は、複数の下位レシピへ分割しなければならないかもしれない。

ステップ: 1.最初に、…のxyz インターフェースを実装するクラスを作成する

2.インターフェースに…という名前のメソッドを定義する

3. …という名前の別のメソッドを定義する

議論:ここには、発生しうる問題、バリエーションが正しくインストールされたことを検証する方法、このアプローチを使用する際に行ってはならないことを記述できる

A Recipe for Making an Adaptation

Recipe Name: Usually starts with “How to”

Intent The reason to use this recipe

Design Description Which classes and interfaces are involved and need to be understood, what roles do they play and what collaborations are involved. What responsibilities are adapted via the variation... backed up by supporting UML diagrams and other descriptions.

Related Recipes: Alternative ways to accomplish a similar variation; or related sub-recipes. A complex recipe may be broken down into several sub-recipes.

Steps 1. First create a class that implements the xyz interface...

2. In it define a method named...

3. And another method named...

Discussion: Mention problems that might crop up, how to test that a variation is correctly installed, or what should not be attempted using this approach.

「推測」クラスを追加するためのレシピ

To Add a New Kind of Guess

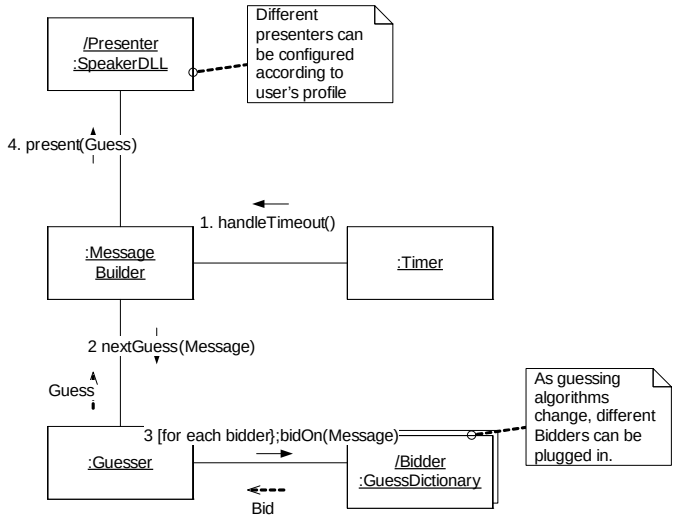
1. Define a class that implements the Guess interface.
   This type of object must know contents, formatted for both display and speech, know how long to wait before continuing with another guess, and be able to add itself to a message.
   Specifically, it must implement these methods:

```
public String displayableText()
public String speakableText()
public String getContent()
public Duration waitTime()
void processMessage(Message m)
```

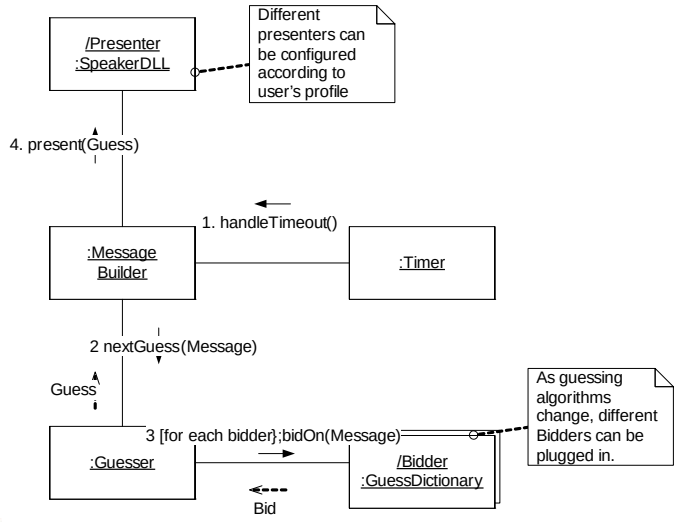
2. Define a class that implements the Bidder interface.
   This type of object will contain all of the corresponding Guess objects and determine which is most relevant to the current message and how relevant they are. Specifically, it must implement:

```
Bid bidOn(Message m)
```

What Can Be Configured



何が構成できるか



Designing Responsibly

Use the best tool for the job
- Tools for thinking, abstracting, analyzing, simplifying
- Tools for making your application flexible

Learn design techniques, and practice, practice, practice

Keep learning

責任をもって設計する

もっともよい道具を仕事に使う
- 考え、抽象化し、分析し、シンプルにするための道具
- 開発しているアプリケーションを柔軟にするための道具

設計のテクニックを学び、練習、練習、練習

学び続ける

The best designers never give up, they just know when to call it a day!

最も優秀な設計な設計者は決してあきらめない，今回はここまでということをわきまえているだけだ!