修 士 論 文

クラス間の通信形態に注目した設計レベルの再構成手法の研究

北陸先端科学技術大学院大学
情報科学研究科情報システム学専攻

大尾　健介

2007 年 3 月

修 士 論 文

クラス間の通信形態に注目した設計レベルの再構成手法の研究

指導教官　鈴木正人 助教授

審査委員主査　鈴木正人 助教授
審査委員　落水浩一郎 教授
審査委員　片山卓也 教授

北陸先端科学技術大学院大学
情報科学研究科情報システム学専攻

510058 大尾　健介

提出年月: 2007 年 2 月

概 要

アプリケーションに対して機能拡張や追加を行う場合、ソースコードの変更方法が問題となっている。機能拡張/追加を実現するために、リファクタリングを用いた手法などが提案されている。しかし、この手法は個々の機能拡張/追加要求に対応するので、繰り返すことで要求の実現が困難になる。また、機能拡張/追加に伴い新しいサービスとの連携を行うことがあるが、従来の小規模かつ局所的な修正では対処できなくなる可能性がある。しかしながら全体的かつ統一性をもった設計変更に対しては有効な手法や支援環境は確立されておらず、設計レベルで機能拡張/追加を支援する手法が必要である。

本研究では、システムの機能拡張/追加の際に必要となる全体的なアーキテクチャの変更 (設計再構成) を支援する手法を提唱する。

目 次

第1章　はじめに

1.1　背景

既存のアプリケーションに対して、機能の拡張や新しい機能の追加要求を行う場合、ソースコードの変更方法が問題となっている。新しいサービスとの連携の必要性も出てくる。機能拡張、機能追加を実現するために、リファクタリングなどの従来の手法を用いることがある。従来の手法は局所的でありその範囲を超えた要求には対応できない。しかしながら全体的かつ統一性をもった設計変更に対しては有効な手法や支援環境は確立されておらず、設計レベルで機能拡張/追加を支援する手法が必要である。

1.2　目的

本研究では、システムの機能拡張/追加の際に必要となる全体的なアーキテクチャの変更 (設計再構成) を支援する手法を提唱する。この手法を設計レベルの再構成手法と呼び、以下の定義を持たせた：

- システムの機能拡張/追加の際に必要となる全体的なアーキテクチャを変更する

さらに、設計レベルの再構成手法の目標として以下の２つを掲げる。

1. システムの既存部分に対する変更を最小にする
2. 新しく作成するクラスの数も最小にする

1.3　論文の構成

本稿の構成は下記に示す通りである。

1 章 本研究の背景と目的について述べ，研究のアプローチを簡単に説明する。

2 章 研究のアプローチで再構成手法を提案するにあたり注目したクラス間の通信 (相互作用) について説明する。

3 章 本研究で提案する再構成手法について述べる。この章の前半では再構成手法として提案する再構成ルールについて述べる。後半では、再構成ルールを作成するにあたり対象とした客室予約管理システムの概要について説明する。

4 章 本研究で作成した再構成ルールについて解説し、それと同時に各再構成ルールの比較・評価の結果について説明する。

5 章 本研究をまとめ、今後の展望を述べる。

第2章　再構成支援手法の提案

本研究で提唱する再構成支援手法は、再構成ルールに基づいている。再構成ルールとは、機能拡張/追加しやすくシステムの設計を変更する指標である。この章では、再構成ルールを作成するにあたり着目したクラス間の通信 (相互作用) について説明する。また、クラス間の通信についてUML(The Unified Modeling Language) 図を用いて調査を行っている。その中で用いたクラス図、シーケンス図、コミュニケーション図を簡単に説明する。

2.1　クラス間の通信 (相互作用)

本研究では再構成ルールの作成にあたり、クラス間の通信 (相互作用) に着目する。クラス間の通信を調査することで、クラスの役割を分析することができる。クラス間の通信を特徴づける値として以下の 3 種類を採用する。

Style:　通信の形態 (1 対 1 /1 対多 同期/非同期)
Amount:　メッセージに含まれるデータ量
Frequency:　メッセージの頻度

Style

クラス間でやり取りされる通信の形態を表す。形態の種類として 1 対 1 /1 対多あるいは同期/非同期通信が挙げられる。

Amount

1 つのメッセージに含まれるデータ量を表す。つまり、メッセージに与えられる引数のデータ量に着目する。

Frequency

クラスとクラスの間でやり取りするメッセージの頻度を表す。

2.2 UML

UML は、ソフトウェアの設計図を描写するための標準言語である。UML は、ソフトウェア全体のシステムの成果物をビジュアライズ化、仕様化、構築、文書化することができる。本研究では、クラス図、シーケンス図、コミュニケーション図を用いている。本研究で用いるクラス、インタフェース、コラボレーションは、以下のように定義している。

- クラス：オブジェクト単位を表しており、属性と操作を持っている。
- インタフェース：クラスとクラスが通信を行なう際に接続する部分を表す。
- コラボレーション：クラスがインタフェースを通して行う一連のメッセージ通信を表す。

以後、クラス図、シーケンス図・コミュニケーション図の順で説明する。

2.2.1 クラス図

クラス図は、オブジェクト指向システムをモデリングする際に最も一般的に利用する図である。クラス図には、クラス、インタフェースおよびコラボレーションとそれら相互の関係を示す。クラス図を用いることで、システムの設計ビューの静的な側面をモデリングできる。

2.2.2 シーケンス図

シーケンス図はシステムの動的な側面をモデリングするためのダイアグラムである。一連のオブジェクトとその関係、およびオブジェクト間で送信される可能性があるメッセージからなる。

2.2.3 コミュニケーション図

コミュニケーション図はシーケンス図と同様にシステムの動的な側面をモデリングするダイアグラムである。一連のオブジェクトとその関係、およびオブジェクト間で送信される可能性があるメッセージからなる。

第3章　再構成手法プロセスと対象システム

この章では、再構成ルールの作成から利用までの全体のプロセスを説明する。その後、再構成ルールの抽出・作成にあたり、対象としたシステムの概要と追加する機能について説明する。

3.1　再構成ルール

図 3.1 は再構成ルールの全体図を表している。再構成ルールのプロセスは以下の 3 つのフェーズから成る。

1. 作成・登録フェーズ
   システムから再構成ルールの候補を抽出し、再構成ルールの作成を行うフェーズである。作成した再構成ルールは、データベースに登録する。

2. 適用・評価フェーズ
   登録してある再構成ルールをシステムに適用し、機能拡張/追加に効果があるか評価を行うフェーズである。評価後、必要ならば再構成ルールの修正を行う。

3. 利用フェーズ
   条件を基に効果のある再構成ルールを選び出し、利用するフェーズである。

次に、各フェーズについて詳しく説明を行っていく。

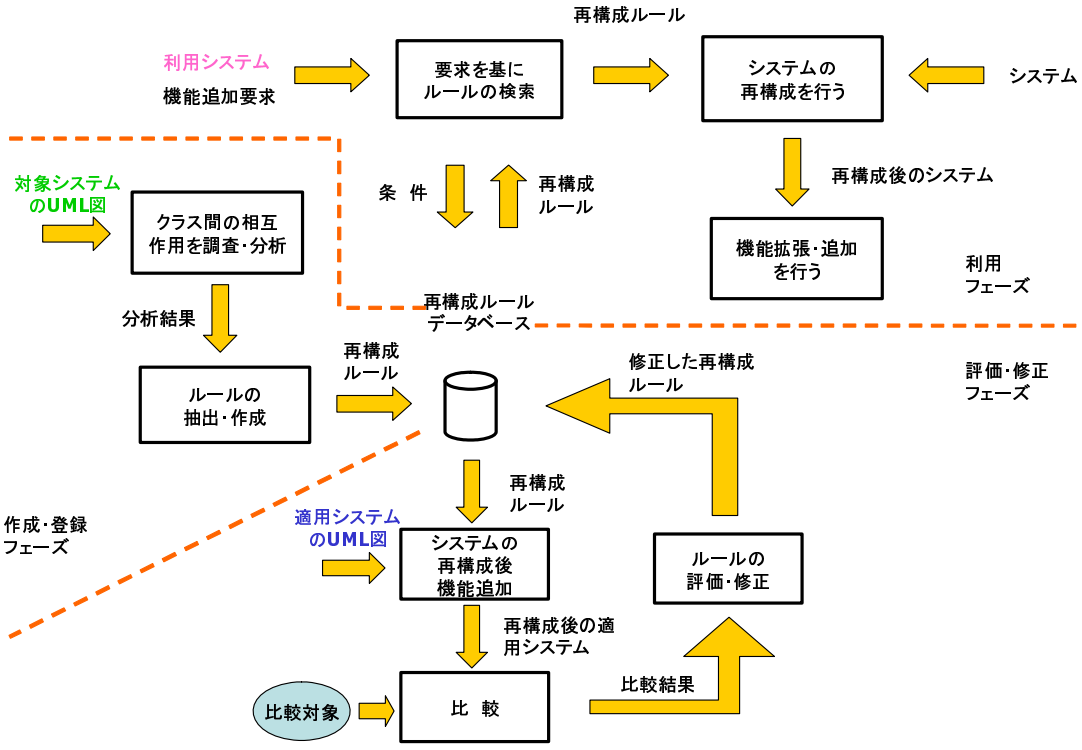

図 3.1: 再構成ルールの全体図

3.1.1 再構成ルールの作成・登録

図 3.2 は、再構成ルールの作成から登録までの一連の流れを表している。
このフェーズは以下の順番で実施する。

1. 対象システムの UML 図とソースコードからクラス間通信の調査・分析を行なう。調査を行う内容は

   - 各サービスで発生するメッセージの数
   - メッセージの内容・役割
   - メッセージのデータ量
   - クラスが持つ役割

   である。調査内容から分析を行ない、サービスで発生するメッセージの一部から再構成ルールの候補を抽出し、その成果物を分析結果とする。

2. 分析結果から再構成ルールを作成する。
   分析結果の例として、以下の様な内容を持っている。

   - Obj1 と Obj2 間のメッセージ数:10 回
   - メッセージの内容:客室の予約
   - Obj1 の役割:予約の作成を行う

   この結果から、機能拡張/追加を行うにあたり、効果のあるクラス間通信の変更を再構成ルールとし、作成する。なお、作成した再構成ルールは、クラス間通信を特徴づける 3 種類の内、いずれかの特徴を持つ。

3. 作成した再構成ルールをデータベースへ登録する。

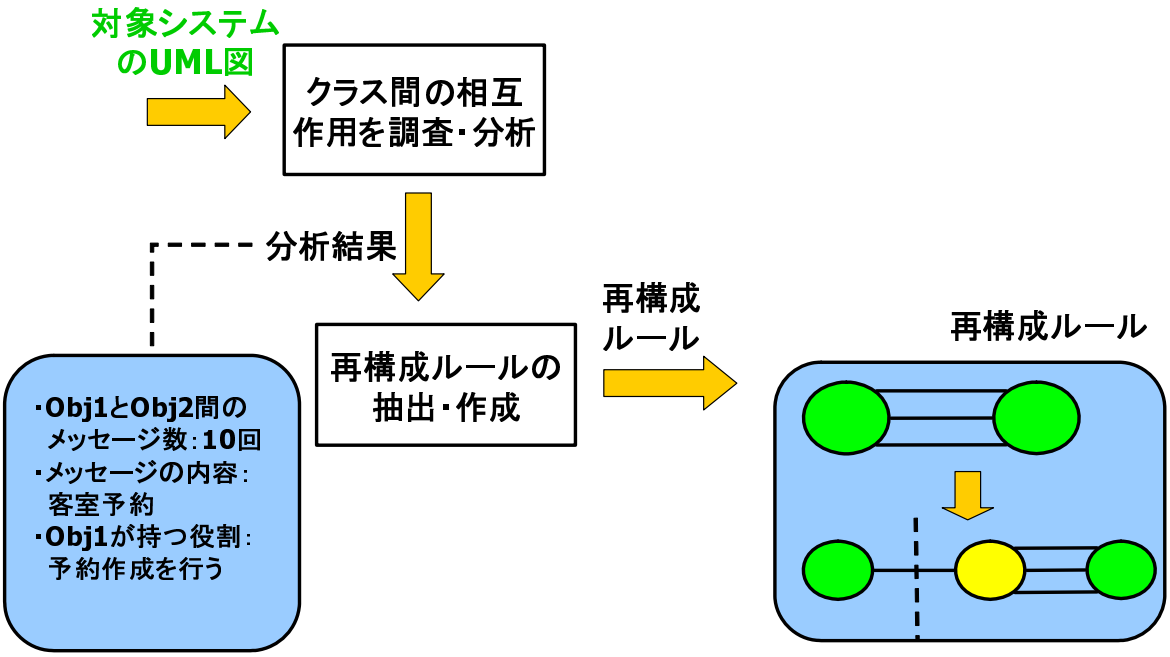

図 3.2: 再構成ルールの作成・登録フェーズ

3.1.2　再構成ルールの適用・評価フェーズ

図 3.3 は、再構成ルールの適用から評価までの一連の流れを表している。
このフェーズは以下の順番で実施する。

1. 再構成ルールを用いて適用システムの再構成を行った後、機能を追加する
2. 通信形態が類似している別のシステムと比較を行い、再構成ルールの評価を行う
3. 評価結果から、再構成ルールの修正を行いより体系的なルールに作り上げていく

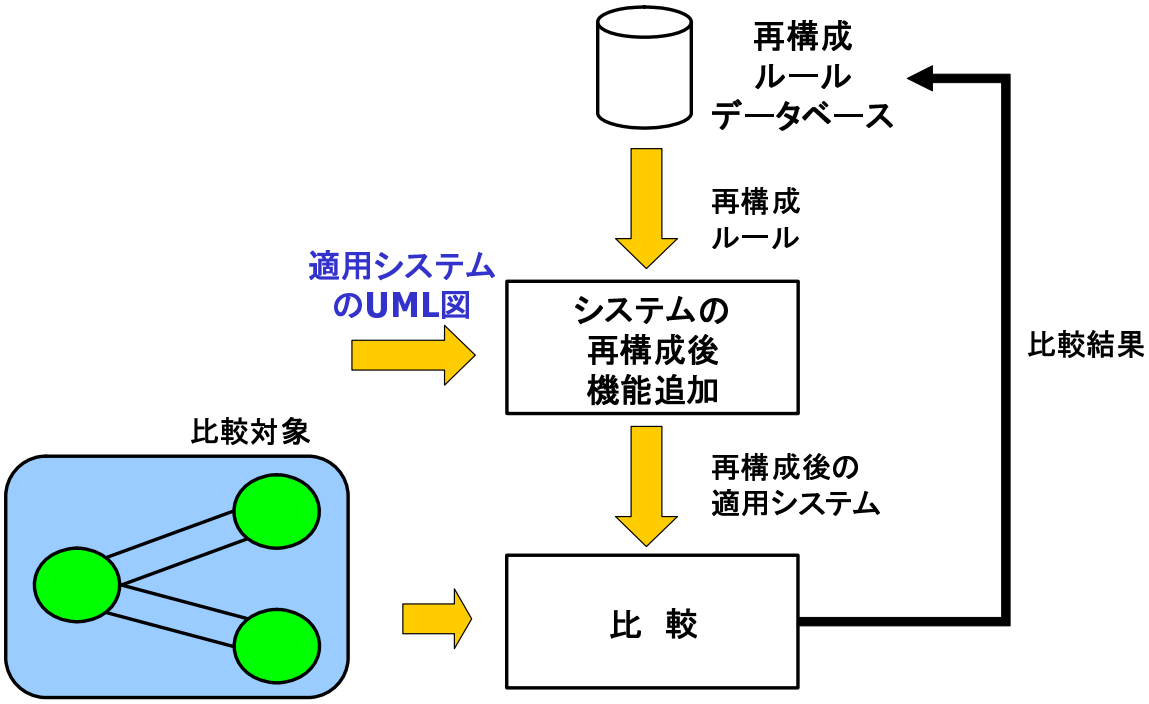

図 3.3: 再構成ルールの適用・評価フェーズ

3.1.3　再構成ルールの利用フェーズ

図 3.4 は、再構成ルールを利用する際の一連の流れを表している。
このフェーズは以下の順番で実施する。

1. 機能拡張/追加要求から、効果的な再構成ルールを検索する
2. 条件に合う再構成ルールを用いて、システムの再構成を行う
3. 再構成後に、機能拡張/追加作業を行う

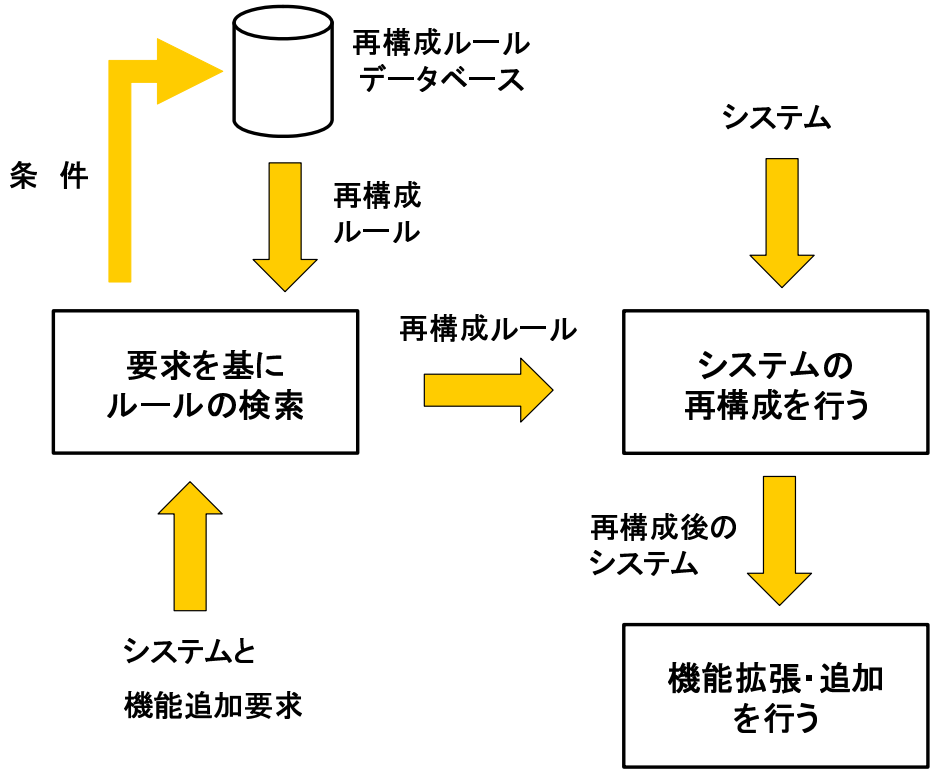

図 3.4: 再構成ルールの利用フェーズ

3.2　客室予約管理システム

再構成ルールの作成と適用実験を実施するにあたり、客室予約管理システムを対象とする。客室予約システムが持つサービスの種類とその内容について説明する。

3.2.1　サービスの概要

このシステムは Web アプリケーションシステムであり、JSP とサーブレットを用いて構築されている。記述言語は Java 言語で記述してあり、13 クラスで構成されている (図 3.6 参照)。このシステムでは以下の 5 種類のサービスを提供する (図 3.5 参照)。

1. 予約作成：予約記録の作成
2. 予約照会：登録されている予約記録を照会する
3. 空室照会：客室の空室状況を照会する
4. 客室管理：客室の状態を管理する
5. 予約取り消し：予約記録を取り消す

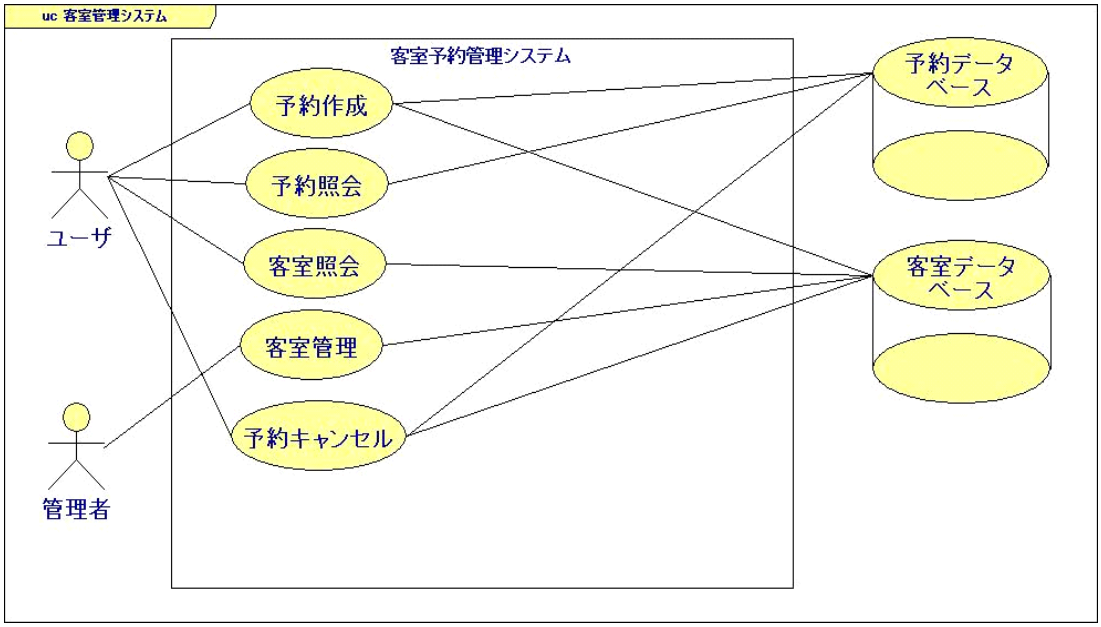

図 3.5: 客室予約管理システム (ユースケース図)

次に客室管理システムが持つ５つのサービスの目的と概要を説明する。

a) 予約作成　目的： 客室の予約を行なう

概要： ユーザは予約管理画面から新規作成をクリックし、開始日・終了日・シングル数・ツイン数・ユーザ名 (予約記録) を入力する。予約記録を受け取って検査を行った後、予約記録の ID 番号を作成する。そして、予約記録として予約データベースへ登録する。最後に、ユーザへ登録した予約記録をユーザへ表示する。

b) 予約照会　目的： 登録した予約を照会する

概要： ユーザは予約管理画面から記録照会をクリックし、予約 ID・ユーザ名（予約照会記録）を入力する。入力された予約照会記録中の予約 ID を基に、予約データベースから該当する予約記録を取得する。取得した予約記録をユーザへ表示する。

c) 空室照会　目的： 部屋の状態を照会する

概要： ユーザは客室管理画面から空室状態表示をクリックし、開始日・終了日・客室番号 (客室照会情報) を入力する。入力された客室照会情報から表を作成し、指定した範囲の客室状態をユーザへ表示する。

d) 客室管理　目的： 客室の状態を変更する

概要： 管理者は客室管理画面から客室管理をクリックし、開始日・終了日・部屋番号・状態 (客室管理情報) を入力する。入力された客室管理情報を基に、指定した客室の部屋状態を変更する。最後に、客室変更記録を管理者へ表示し、客室の状態が変更されたことを知らせる。

e) 予約キャンセル　目的： 予約記録の削除を行なう

概要： ユーザは予約管理画面から予約のキャンセルをクリックし、予約 ID・ユーザ名 (予約キャンセル情報) を入力する。入力された予約キャンセル情報中にある予約 ID 番号を基にデータベースを参照し、予約記録を削除する。削除した予約記録の内容をユーザへ表示する。

次に、客室予約管理システムの構成をクラス図 (図 3.6) で示し、各クラスの役割について述べる。また、客室予約管理システムのサービス「予約作成」のシーケンス図 (図 A.3) を示し、メッセージ通信の概要について説明する。

3.2.2 クラスの役割

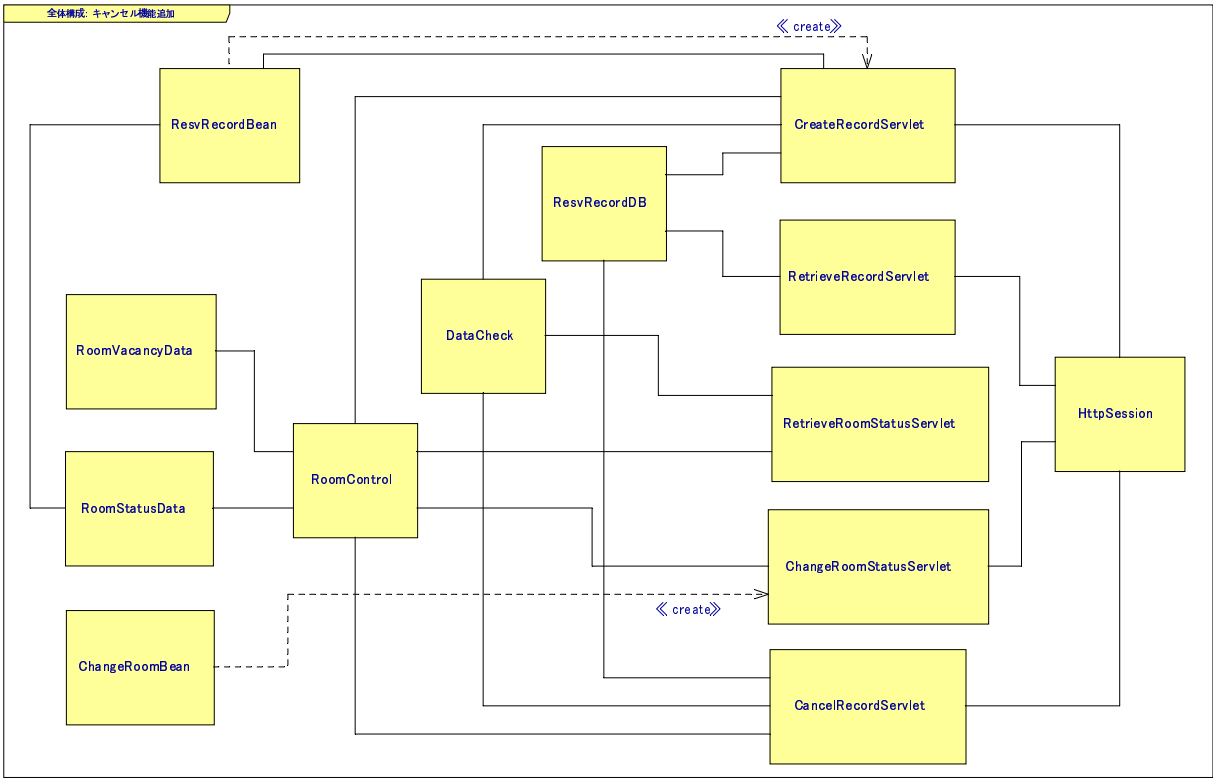

図 3.6: 客室予約管理システム (クラス図)

CreateRecordServlet

役割: ユーザが入力した予約記録を受け取り、予約記録を作成する。作成した予約記録を予約データベースへ登録する。予約が作成され、登録されたことをユーザへ知らせる。

RetrieveRecordServlet

役割: ユーザが入力した予約記録照会要求データを受け取り、該当する予約記録を予約データベースから取り出してくる。取り出してきた予約記録の内容をユーザへ表示する。

RetrieveRoomServlet

役割: ユーザが入力した客室状態要求データを受け取り、客室の空室状況を表示する表の作成を行わせる。作成した表をユーザに表示させる。

ChangeRoomServlet

役割: ユーザが入力した客室管理要求データを受け取り、客室管理を行う。客室管理の内容は、ある客室の状態を”busy”に変更し、客室の予約ができないようにする。客室の管理を行った記録をユーザへ表示する。

CancelRecordServlet

役割: ユーザが入力した予約記録キャンセル要求データを受け取り、予約記録のキャンセルを行う。キャンセルした予約記録をユーザへ表示する。

RoomControl

役割: 予約記録を作成する際に、客室の状態を参照し予約記録が作成可能か検査を行う。予約記録要求データを元に、客室の予約を行う。ユーザから入力された客室状態参照要求データをもとに、客室状態の表を作成する。予約記録がキャンセルされた際に、予約していた客室の状態を空室に変更する。

DataCheck

役割: 予約記録作成要求データの日付・客室数・ユーザの検査を行う。入力された日付が今日から何日後か計算を行う。２つの入力された予約記録が同じものか検査を行う。

ResvRecordBean

役割: 予約記録作成要求データを基に作成される予約記録を表す。予約記録は開始日・終了日・シングル数・ツイン数・ユーザ・予約 ID の６つの文字列の変数を持っている。

ChangeRoomBean

役割: 客室管理要求データを基に作成される客室管理記録を表す。客室管理記録は、開始日・終了日・客室番号・要求する客室状態の 4 つの文字列の変数を持っている。

RoomVacancyData

役割: 客室の状態を文字列の配列形式で保持している。予約記録の作成・キャンセルに伴い客室を予約または空室の状態へ変更する。

RoomStatusData

役割: 客室の状態を整数型の配列形式で保持している。予約記録の作成・キャンセルに伴い空室数を減少または増加する操作を行う。

ResvRecordDB

役割: 予約記録を登録するデータベースを保持している。予約記録の作成・参照・キャンセルに伴い予約の登録・読み込み・削除を実行する。

HttpSession

役割: 作成・参照・削除を行った予約記録、客室管理記録をユーザへ表示するためにセッションに登録する。登録した記録をユーザへ表示する。

次にサービス「予約作成」のシーケンス図を示し、各メッセージの役割について説明する。

3.2.3　メッセージ通信順序

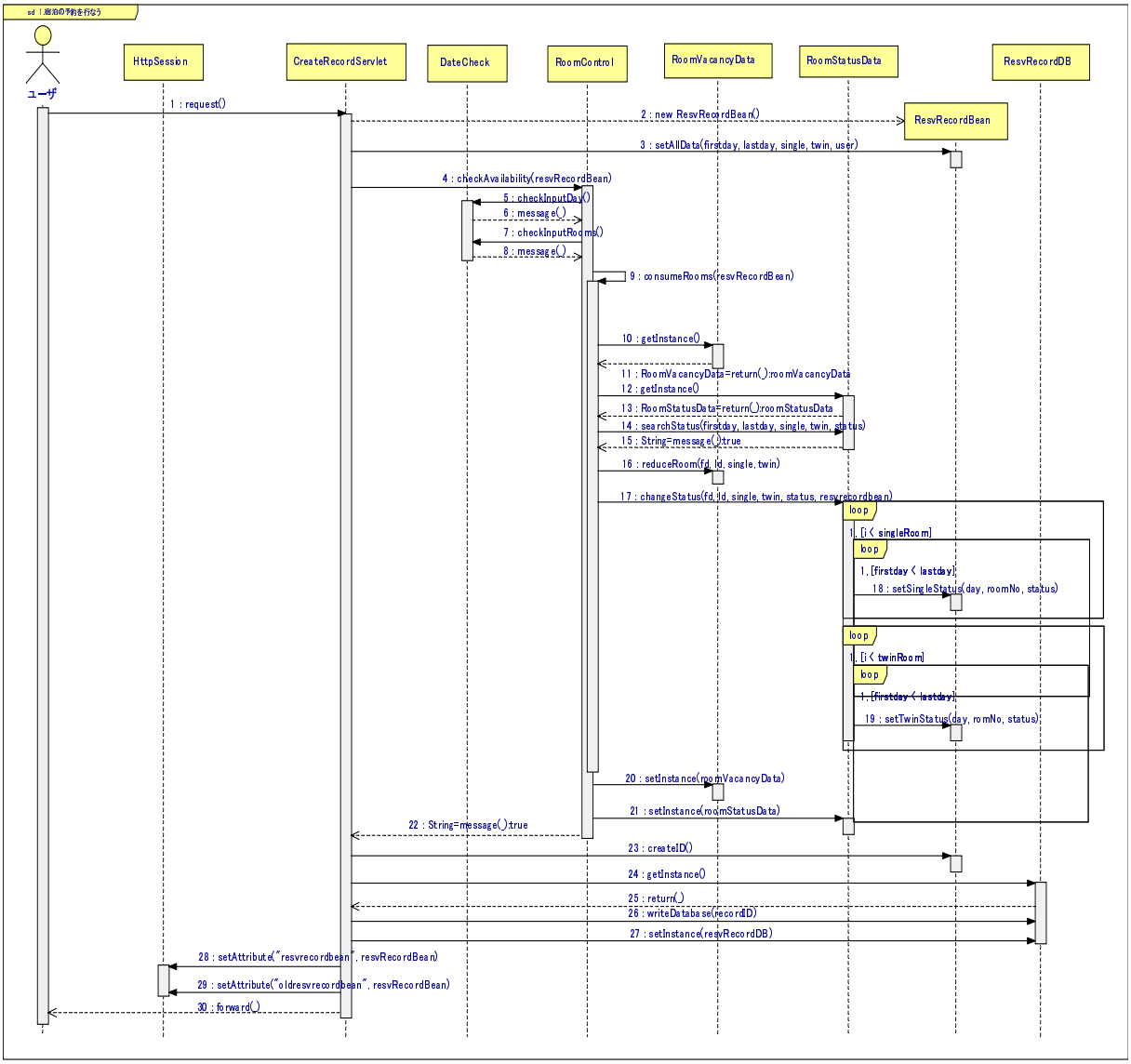

図 3.7:「予約作成」のシーケンス図

図 3.7 は、「予約作成」のシーケンス図を表しており、オブジェクト間のメッセージ通信を表している。主要なメッセージの目的と概要について説明する。

`1:request()`

目的：　ユーザから `CreateRecordServlet` へ予約記録作成要求データを送る。

概要：ユーザが `Web` 上で入力した予約記録作成要求データをまとめて送信する。

2:new ResvRecordBean()

目的： 予約記録作の作成にあたり、予約記録を登録するインスタンスを生成する。

概要：ユーザが入力した予約記録作成要求データを基に予約記録として登録するインスタンスを生成する。

3:setAllData(firstday, lastday, single, twin, user)

目的： 2 で作成したインスタンスに予約記録が必要なデータを登録する。

概要：予約記録としてデータベースに登録するために、2 で作成したインスタンスに必要なデータを登録する。登録するデータとして、開始日 (firstday)・終了日 (lastday)・シングル数 (single)・ツイン数 (twin)・ユーザ (user) の 6 つのデータを登録する。 6 つのデータはすべて文字列型で作成したものである。

4:checkAvailability(resvRecordBean)

目的：予約記録を基に客室の予約を行えるか検査を行い、客室の予約を行う。

概要：予約記録 (resvRecordBean) を引数として、CreateRecordServlet から RoomControl へメッセージが送られる。このメッセージを送ることで、予約記録のデータを検査するメッセージ (メッセージ番号 5,7) と客室の予約を行えるか検査をし、予約を行うメッセージ (メッセージ番号 9) が RoomControl から送信される。このメッセージの戻り値として、文字列が CreateRecordServlet に帰る。

5:checkInputDay(firstday, lastday)

目的： 予約記録が持つ日付の検査を行う。

概要：予約記録が持っている開始日・終了日のデータが指定した形で入力されているか検査を行う。本研究で指定した形は、[西暦/月/日] の形に指定している。問題がなければ戻り値として文字列を返す。

7:checkInputRooms(single, twin)

目的： 予約記録が持つ客室数の検査を行う。

概要：予約記録が持っているシングル数・ツイン数のデータが指定した範囲内に正しく入力されているか検査を行う。本研究で指定したシングル数・ツイン数の範囲は [0-5] である。問題がなければ戻り地として文字列を返す。

9:consumeRooms(resvRecordBean)

目的： 予約記録を元に客室の予約を行う。

概要：予約記録を元に、指定された日付の範囲の客室を予約する。客室の予約にあたり、RoomVacancyData、RoomStatusData に対して客室の予約を行うメッセージ (メッセー番号 14,16) を送信する。これらのメッセージが全て実行することで客室の予約が完了する。

14:searchStatus(firstday, lastday, single, twin, status)

目的： 客室が予約できるか検査を行う。

概要：予約記録を元に RoomStatusData が保持する客室が予約できるか検査を行う。検査を行う際に、客室の状態が"resv"でないか検査をする。問題がなければ戻り値として文字列を返す。

16:reduceRoom(fd, ld, single, twin)

目的： 予約に伴い客室の空室数を減らす。

概要：予約記録を元に RoomVacancyData が保持する空室数を減らしている。

17:changeStatus(fd, ld, single, twin, status, resvRecordBean)

目的： 客室の予約を行う。

概要：予約記録を元に RoomStatusData が保持する客室を予約する。予約を行う際に、客室の状態を"resv"に変更し次の予約記録が予約できないようにしている。

23:createID()

目的： 予約 ID を作成する。

概要：客室が予約され、予約記録としてデータベースに登録する前に予約 ID 番号を作成する。予約 ID 番号を作成することで、予約記録の照会・削除で呼び出すことができるためである。

26:writeDatabase(recordID, resvRecordBean)

目的： 予約記録を予約データベースへ登録する。

概要：予約 ID(recordID) と予約記録 (resvRecordBean) を引数としてメッセージに与え、予約記録を登録する。登録の際に、予約 ID をインデックスとして登録することで予約の参照・削除において予約 ID を元に取り出すことができる。

第4章　再構成ルールの作成と適用実験

4章では、客室予約管理システムからクラス間の通信に着目し、作成した再構成ルールについて説明を行う。そして、再構成ルールの適用実験を行い、再構成ルールの評価を行った。

4.1　作成した再構成ルール

客室予約管理システム各サービスで発生するクラス間の通信分析を行なった。分析を行なう際に、通信の特性（Style/Amount/Frequency）それぞれ着目した結果、以下の３つの再構成ルール (R1,R2,R3) を抽出した。() 内は関連する特性である。4.1 では、作成した再構成ルールの説明を行う。

R1：クラスの役割の委譲 (Frequency)

　　クラスが持つ役割の一部を新しいクラスへ委譲する

R2：クラスの従属化 (Style)

　　あるクラスとあるクラスの間に従属関係を持たせる

R3：オブジェクトとデータの不整合の解消 (Amount)

　　オブジェクトが利用するデータとメッセージで送るデータとの間の不整合を解消する

以後の節で、各再構成ルールの内容・効果・前提条件・シーケンス図・対象部分のソースコードについて説明を行う。

4.1.1 R1：クラスの役割の委譲

- 内容

  クラスが持つ役割の一部を、新しく作成したクラスに委譲するルールである。図 4.1 において、オブジェクト OBJ1(以後 OBJ1) とオブジェクト OBJ3(以後 OBJ3) 間のメッセージ通信が頻繁にやり取りされている。さらに、メッセージ messageB() からメッセージ messageE() はメッセージ add(OBJ2, OBJ3) を送る前処理としてメッセージ通信を行っている。

  そこで OBJ1 と OBJ3 の間に仲介するオブジェクト OBJ4(以後 OBJ4) を新しく作成し、OBJ1 が行っていた前処理のメッセージを送るという役割を OBJ4 に委譲する。そうすることで OBJ4 と OBJ3 の間でメッセージ add(OBJ2, OBJ3) の前処理のメッセージ交換を行なわれる。さらに、OBJ4 と OBJ3 間のメッセージ交換を内部通信に変更すれば、外部通信が減少する効果を得られる。(図 4.2)

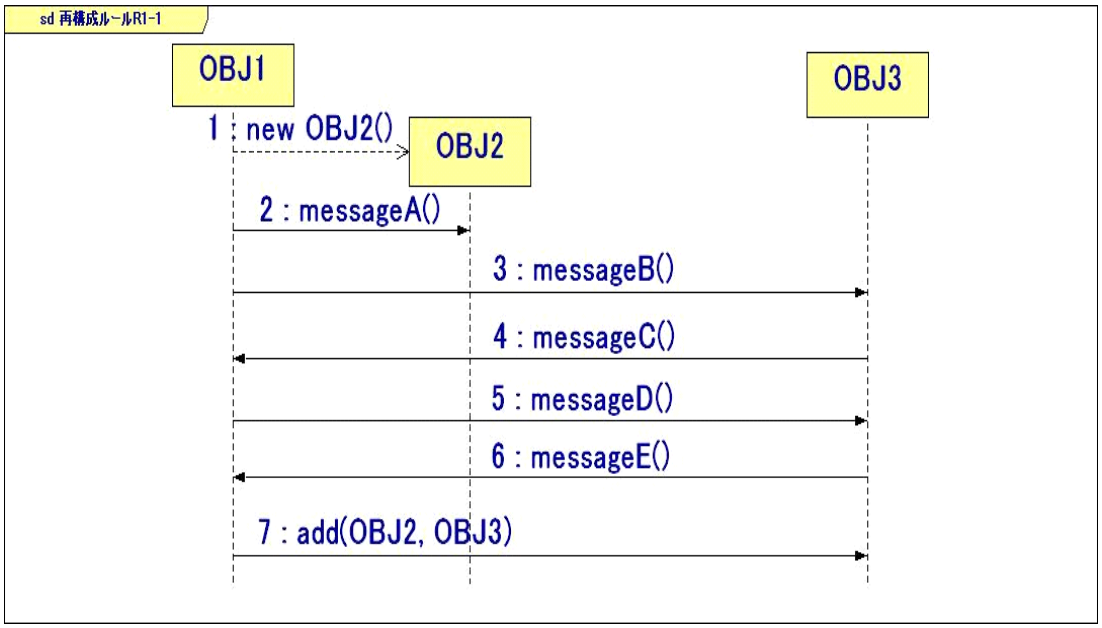

図 4.1: R1 適用前：3 つのインスタンス間のメッセージ通信

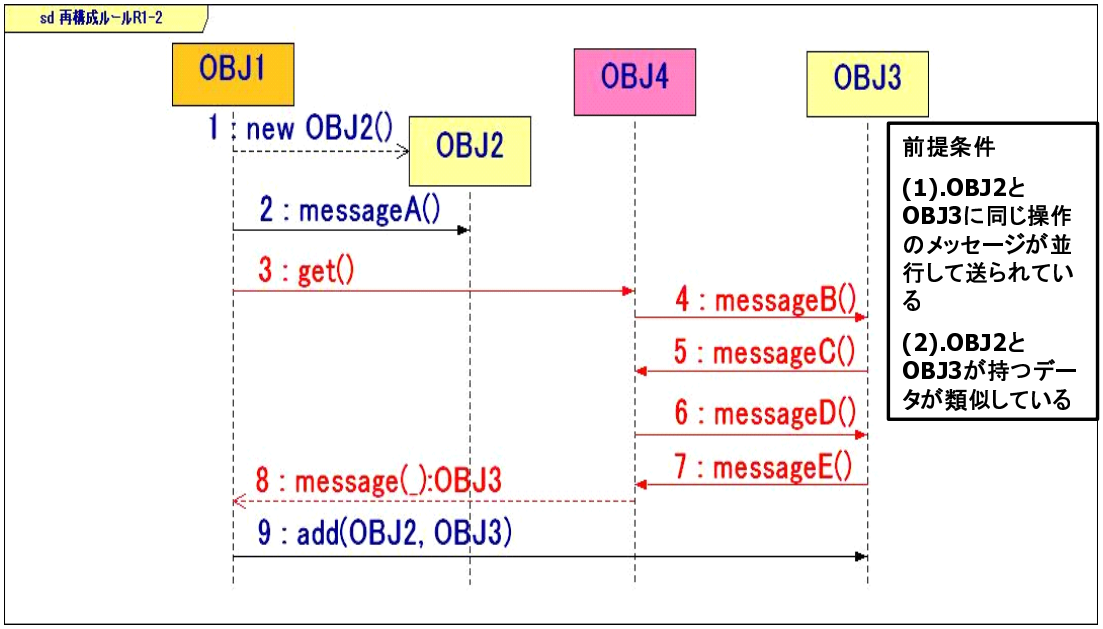

図 4.2: R1 適用後： 4 つのインスタンス間のメッセージ通信

- 効果

  R1 を適用することで、「メッセージを実行するために必要な前処理を実行する」という役割が新しいクラスへ委譲される。委譲もとのクラスの役割が減少し、機能拡張/追加時に変更する箇所を最小にする効果が得られる。

- 前提条件

  以下の前提条件を満たす必要がある。

  1. オブジェクト間でメッセージ交換が頻繁に行われている。
  2. 1 つのメッセージを実行するための前処理メッセージが存在する。

- R1 の抽出部分

R1 を客室予約管理システムから抽出した部分の説明を行う。図 4.3 は、サービス「予約作成」で発生するメッセージの一部を抜き出したものである。メッセージ番号 1 から 7 は、ユーザから入力されたデータを基に予約記録を作成し、予約データベースへ登録するプロセスを表している。CreateRecordServlet クラスは、

1. ResvRecordBean クラスのインスタンスを生成し、予約記録を作成する
2. 作成した予約記録を ResvRecordDB クラスが持つ予約データベースへ登録する
3. 予約記録をユーザーに表示する

という役割を持っている。そのため、CreateRecordServlet クラスから予約データベースへ登録するメッセージ (writeDatabase()) を ResvRecordDB クラスへ送信する前に、前処理のメッセージ（番号 4,5）を ResvRecordDB クラスへ送信している。つまり、CreateRecordServlet クラスが予約記録を作成し、予約データベースへ登録する役割を持っているので、CreateRecordServlet クラスと ResvRecordBean クラス間のメッセージ通信回数が増加する原因となっている。

R1 を適用することで、作成した新しいクラス ResvRecordControl へ CreateRecordServlet クラスが持つ前処理のメッセージを実行する役割を委譲する。委譲することで CreateRecordServlet クラスが持つ役割が減り、機能拡張/追加要求時に変更する箇所を少なく抑えることができる。

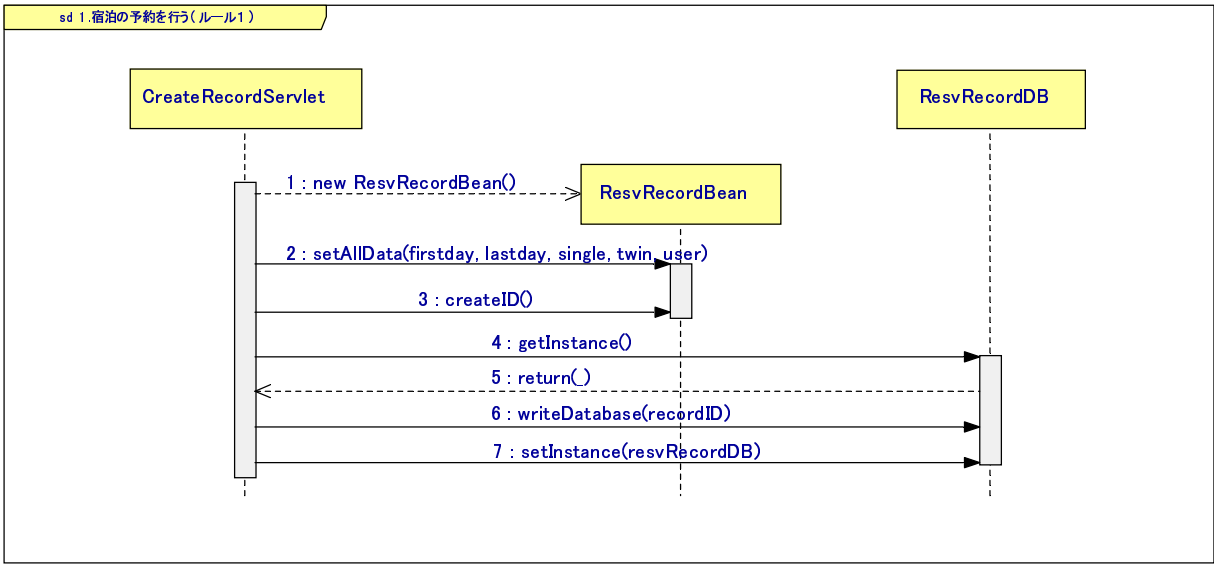

図 4.3: R1 抽出部分：3 つのオブジェクト間のメッセージ通信

CreateRecordServlet クラスにおいて R1 を抽出した部分のソースコードを示す。(プログラム 1：CreateRecordServlet.java)

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class CreateRecordServlet extends HttpServlet{
 HttpSession session;
    ...
 public void doPost(HttpServletRequest request, HttpServletResponse response)
 throws ServletException, IOException {
    ...
       ResvRecordBean resvRecordBean = new ResvRecordBean();
       resvRecordBean.setAllData(firstday, lastday, single, twin, creator);
    ...
       resvRecordBean.createID();
       ResvRecordDB resvRecordDB = ResvRecordDB.getInstance();
       resvRecordDB.writeDatabase(resvRecordBean.getReserveID(), resvRecordBean);
       ResvRecordDB.setInstance(resvRecordDB);
    ...
 }
}
```

プログラム 1：R1 抽出部分 (CreateRecordServlet.java)

4.1.2 R2：クラスの従属化

- 内容

  類似したデータを持つ複数のクラス間に、従属関係を持たせる。従属関係持たせることで、マスタークラスが全てのスレーブクラスへメッセージ通信を行う。

  図 4.4 では、オブジェクト OBJ1(コントローラクラス) からオブジェクト OBJ2 とオブジェクト OBJ3 へそれぞれ同じ手順でメッセージ通信が行なわれる。さらに、OBJ2 と OBJ3 が持つデータは類似している。

  そこで、R2 を適用し、OBJ2 と OBJ3 の間に従属関係を持たせることで、マスタークラス (図 4.5 では OBJ2) がスレーブクラス (図 4.5 では OBJ3) へのメッセージ通信を全て行う。結果、コントローラクラスが呼び出すメッセージの数が減少し、OBJ1 と OBJ2,OBJ3 との依存度が低くなる効果を得られた。(図 4.5 参照)

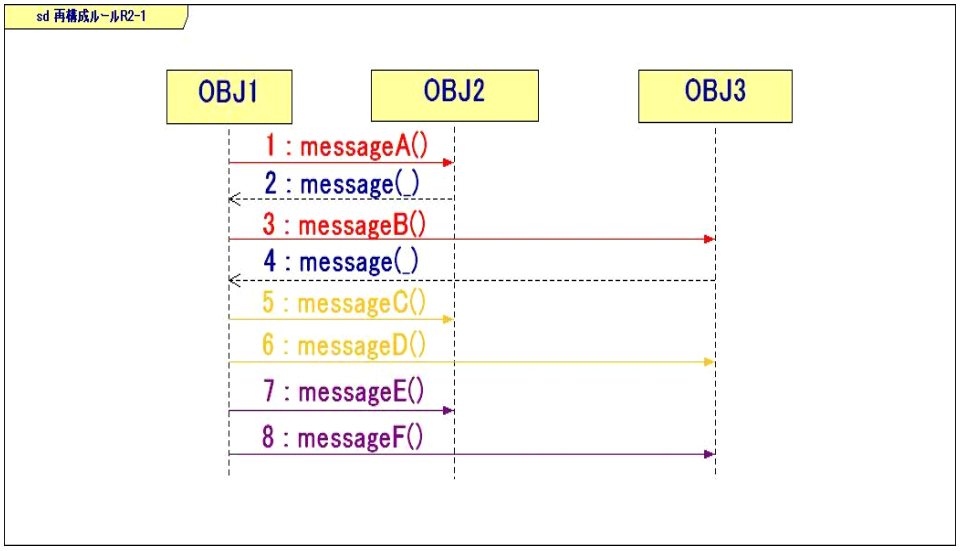

図 4.4: R2 適用前：3 つのインスタンス間のメッセージ通信

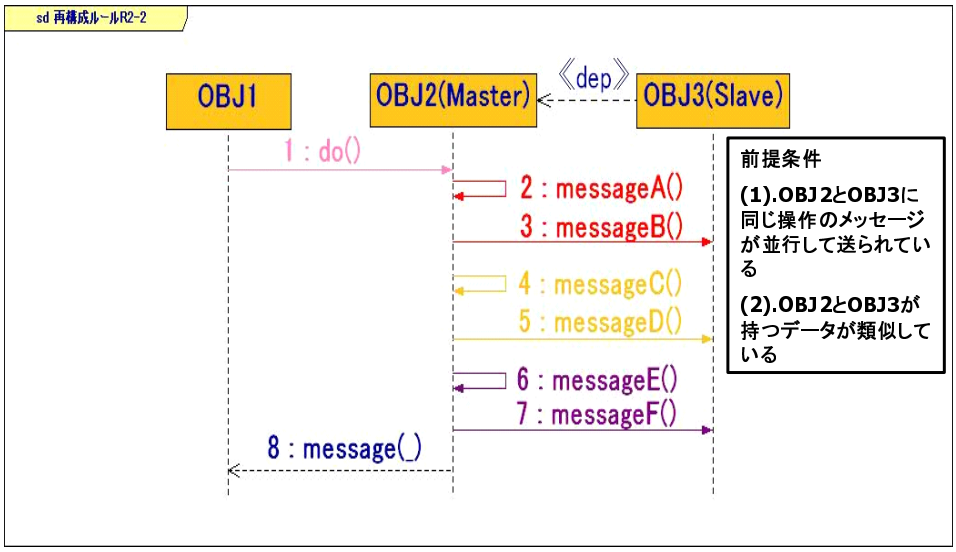

図 4.5: R2 適用後：3 つのインスタンス間のメッセージ通信

- 効果

  コントロールクラス (図 4.5 では OBJ1) から呼び出すメッセージの数が減少する。OBJ2 と OBJ3 に従属関係を持たせるので、クラス間 (OBJ1 と OBJ2,OBJ3) の依存度を低くすることができる。

- 前提条件

  以下の前提条件を満たす必要がある。

  1. コントロールクラスから複数のクラスに同じ操作内容のメッセージが並行して送られている。
  2. 複数のクラスが持つデータが類似している。

- R2 の抽出部分

  R2 は客室予約管理システムの以下の 3 つのサービスから抽出し、作成した。

  - 予約記録

- 客室管理
- 予約キャンセル

図 4.6 は、サービス「予約作成」、図 4.7 は、サービス「客室管理」、図 4.8 は、サービス「予約キャンセル」で行うメッセージ通信の一部を抜き出したシーケンス図である。この中から、「予約作成」の抽出部分について説明を行う。

図 4.6 のメッセージ通信は、作成した予約記録から客室を予約する操作を表す。はじめに、CreateRecordServlet クラスは RoomControl クラスへ客室の空室数が満たしているか検査を行うメッセージ (checkAvailability()) を送信する。RoomControl クラスは、そのメッセージを受け取ると自分自身へ要求された数の客室を予約するメッセージ (consumeRooms()) を実行する。consumeRooms() が実行されると、RoomVacancyData クラスと RoomStatusData クラスへ交互に部屋の状態を変更するメッセージを送信する。

図 4.6 のメッセージ番号 3 から 12 は、客室の状態を変更する際に RoomVacancyData クラスと RoomStatusData クラスに並行してメッセージ通信を行っている。さらに、RoomVacancyData クラスが持つデータと RoomStatusData クラスが持つデータは類似しており、それぞれ客室状態を保持したデータを持つ。

図 4.7 と図 4.8 についても同様な状態となっている。

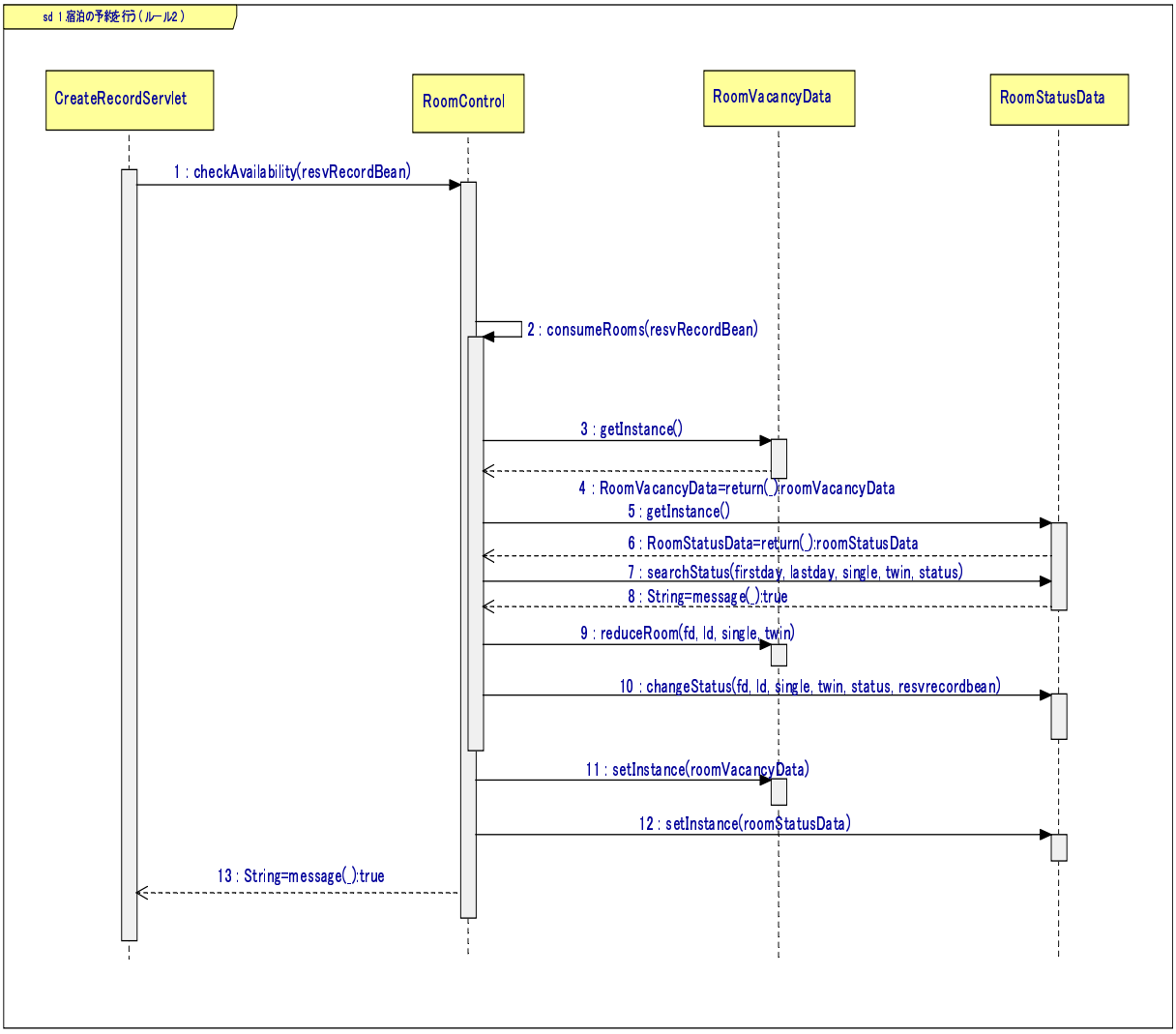

図 4.6: R2 抽出部分:「予約作成」のメッセージ通信 (一部)

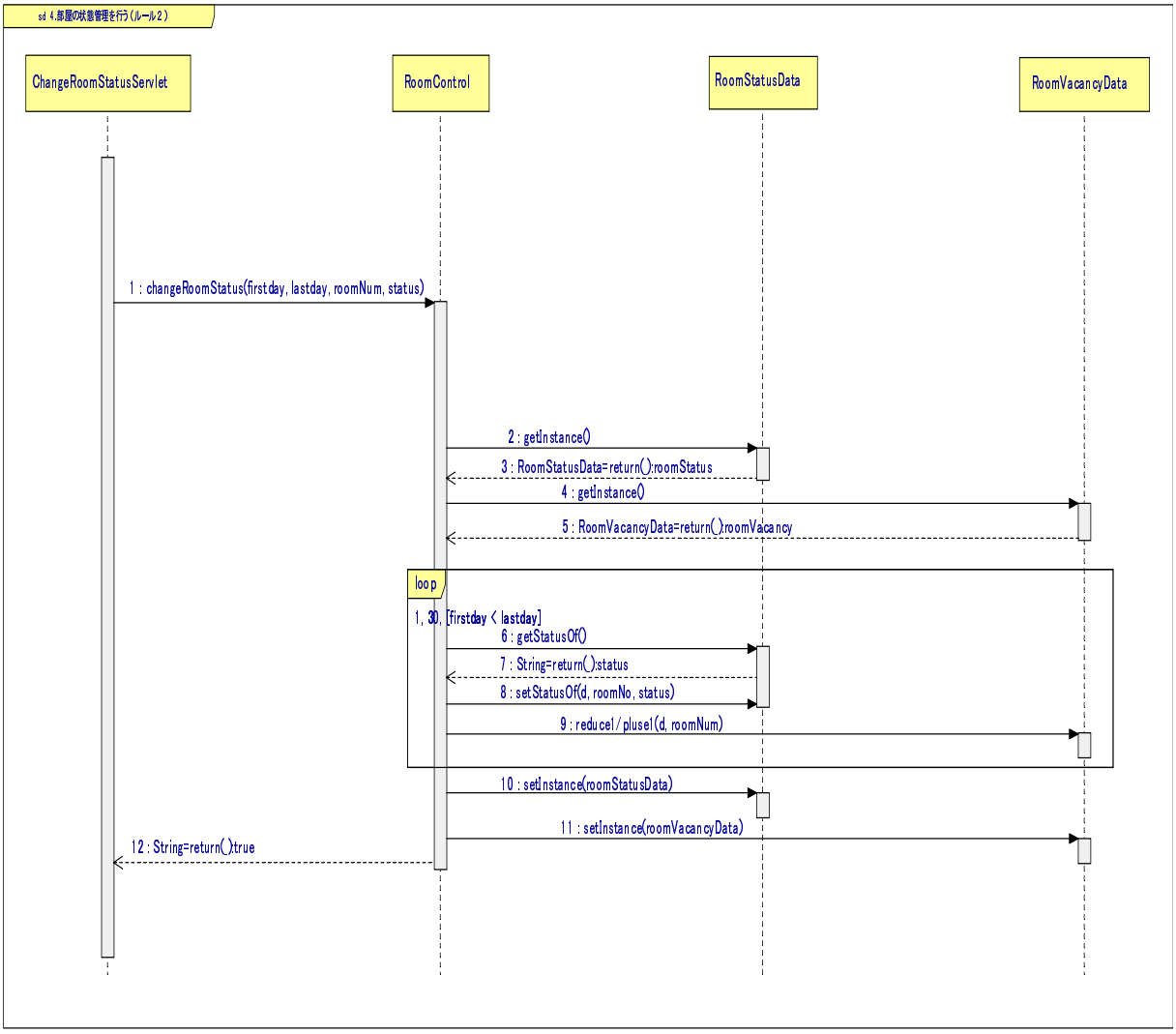

図 4.7: R2 抽出部分:「客室管理」のメッセージ通信 (一部)

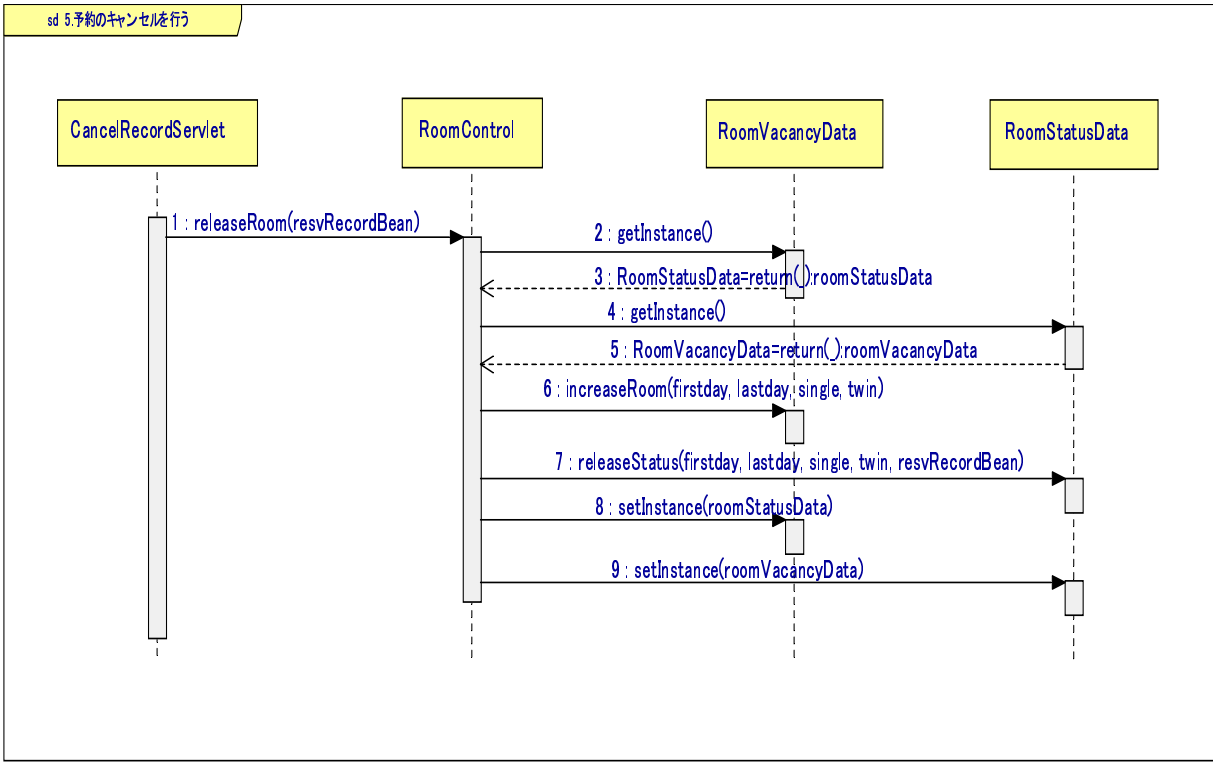

図 4.8: R2 抽出部分:「予約キャンセル」のメッセージ通信 (一部)

次に、R2 を抽出したソースコード (RoomControl.java に定義しているメソッド consumeRooms(),changeRoomStatus(), releaseRooms()) を示す (プログラム 2,3,4)。

```
import java.io.*;
import java.util.*;
public class RoomControl implements Serializable {
    public String consumeRooms(ResvRecordBean rrb){
        RoomStatusData roomStatusData = RoomStatusData.getInstance();
        RoomVacancyData roomVacancyData = RoomVacancyData.getInstance();
        message = roomStatusData.searchStatus(first, last, single, twin, status);
         ...
        roomVacancyData.reduceRoom(first, last, single, twin);
        roomStatusData.changeStatus(first, last, single, twin, status, rrb);
         ...
        RoomStatusData.setInstance(roomStatusData);
        RoomVacancyData.setInstance(roomVacancyData);
        return "true";
    }
}
```

プログラム 2：R2 抽出部分 (RoomControl.java のメソッド consumeRooms())

```
import java.io.*;
import java.util.*;
public class RoomControl implements Serializable {
   public String changeRoomStatus(String fday, String lday,
                                  String roomNum, String status){
       RoomStatusData roomStatus = RoomStatusData.getInstance();
       RoomVacancyData roomVacancy = RoomVacancyData.getInstance();
        ...
      for(int d = first; d <= last; d++){
      if(roomStatus.getStatusOf(d, roomNo).equals("resv")){
      reserved = true;
      break;
      }
      }
      for(int d = first; d <= last; d++){
      roomStatus.setStatusOf(d, roomNo, status);
      if(req_status.equals("busy"))
     roomVacancy.reduce1(d, roomNo);
      }
      RoomStatusData.setInstance(roomStatus);
      RoomVacancyData.setInstance(roomVacancy);
      return "true";
  }
}
```

プログラム 3：R2 抽出部分 (RoomControl.java のメソッド changeRoomStatus())

```
import java.io.*;
import java.util.*;
public class RoomControl implements Serializable {
  public void releaseRooms(ResvRecordBean rrb) {
        ...
        RoomStatusData roomStatus = RoomStatusData.getInstance();
        RoomVacancyData roomVacancy = RoomVacancyData.getInstance();
        roomVacancy.increaseRoom(first, last, single, twin);
        roomStatus.releaseStatus(first, last, single, twin, status, rrb);
        RoomStatusData.setInstance(roomStatus);
        RoomVacancyData.setInstance(roomVacancy);
  }
}
```

プログラム 4：R2 抽出部分 (RoomControl.java のメソッド releaseRooms())

4.1.3 R3：受け手と引数の境界不整合の解消

- 内容

  クラスが利用したいデータだけをメッセージの引数に与えることで、データ境界とオブジェクト境界の不整合を解消するルールである。

  図 4.9 では、オブジェクト Servlet からオブジェクト Control へ引数 d1，d2 が与えられたメッセージ messageA() を送信する。Control からオブジェクト OBJ1，OBJ2，OBJ3 へそれぞれメッセージ通信を行うが、メッセージごとに利用する引数は異なる。

  例えば、Control から OBJ1 へのメッセージ messageB() が引数として必要なデータは、a1，a2 の 2 つである。同様にメッセージ messageC() が引数として必要なデータは、a1，a3 の 2 つである。メッセージ messageD() が引数として必要なデータは、a2，a4 である。つまり、メッセージ messageA で与えた引数の一部がメッセージ messageB,C,D で利用されている。

  表 4.1，4.2 は、メッセージが供給するデータとオブジェクトが使用するデータをまとめたものである。OBJ1 が使用するデータは、メッセージ messageA に受け渡したデータ d1，d2 の一部を使用する。OBJ2，OBJ3 において使用するデータは、メッセージ messageA に受け渡したデータ d1，d2 の一部であり、受け手が使用するデータとメッセージ引数との間で不整合が起こっている。

  R3 適用後の図 4.10 では、メッセージ messageB,C,D が必要とするデータをそれぞれまとめ、メッセージ messageA() の引数として与える。メッセージ messageA() の引数として与えられたデータ b1，b2，b3 はそれぞれ messageB，C，D の引数として利用される。R3 適用前では受け手と引数との不整合の問題が起こっていたが、R3 適用後では不整合が解消され、各メッセージで利用するデータごとに整理し、引数に与えている。(表 4.3，4.4 参照)

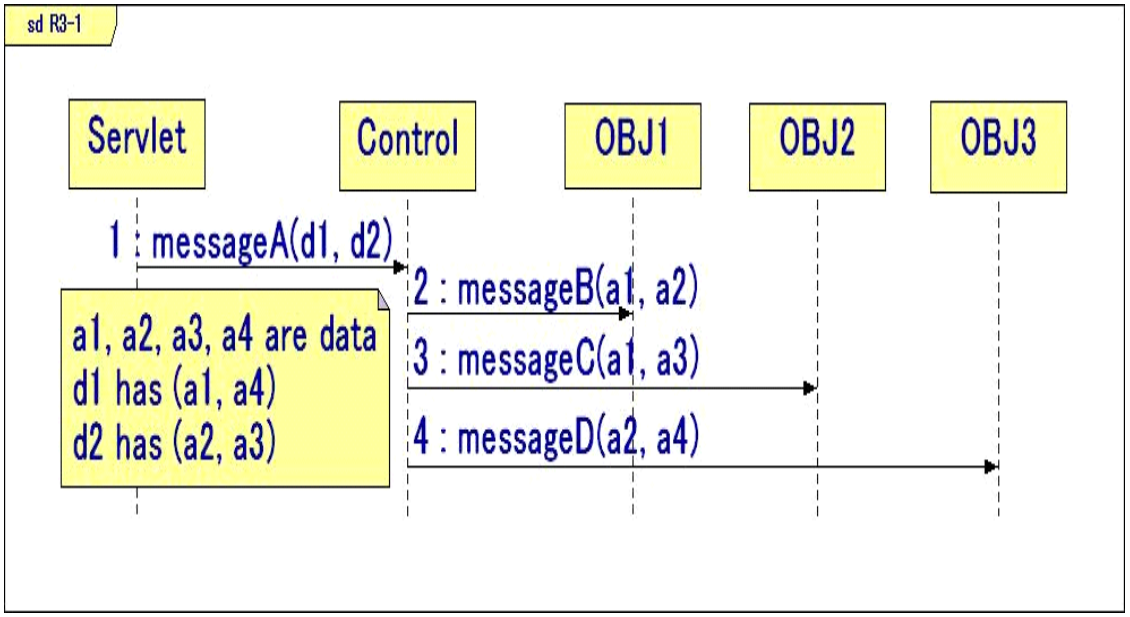

図 4.9: R3 適用前：5 つのインスタンス間のメッセージ通信

表 4.1: R3 適用前:メッセージが供給するデータ

メッセージに与えたデータ	データの中身
d1	a1, a4
d2	a2, a3

表 4.2: R3 適用前:オブジェクトが使用するデータ

オブジェクト	オブジェクトが使用するデータ
OBJ1	a1, a2
OBJ2	a1, a3
OBJ3	a2, a4

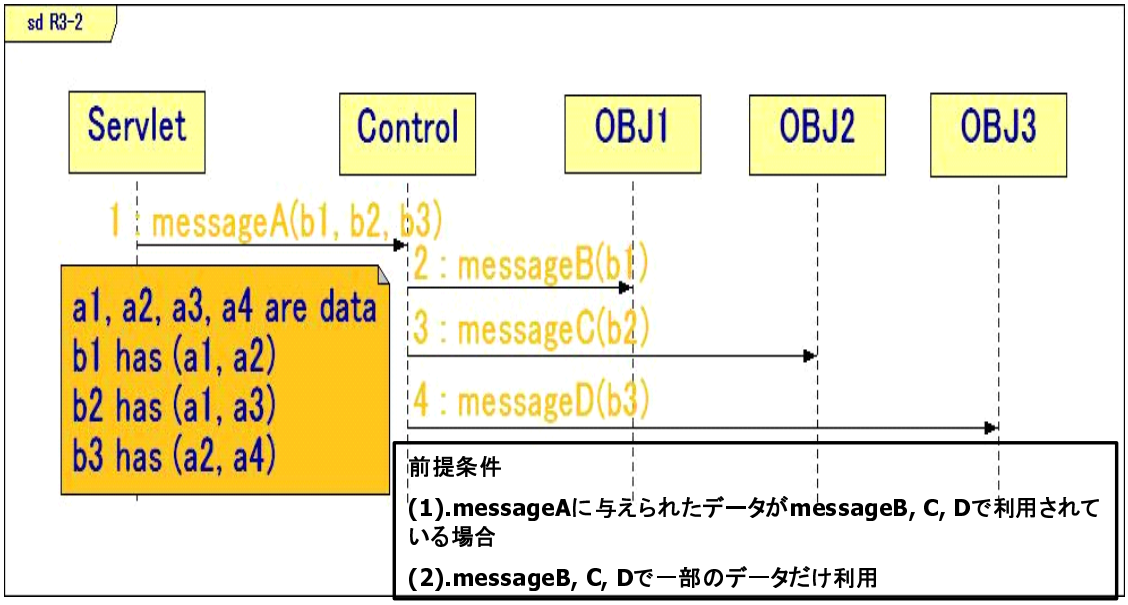

図 4.10: R3 適用後：5 つのインスタンス間のメッセージ通信

表 4.3: R3 適用後:メッセージが供給するデータ

メッセージに与えたデータ	データの中身
b1	a1, a2
b2	a1, a3
b3	a2, a4

表 4.4: R3 適用後:オブジェクトが使用するデータ

オブジェクト	オブジェクトが使用するデータ	メッセージに与えられた引数
OBJ1	a1, a2	b1
OBJ2	a3, a4	b2
OBJ3	a5, a6, a7	b3

- 効果

  受けてが必要とするデータのみをメッセージ引数として送るので、不必要なデータを削減することができる。

- 前提条件

  1. メッセージに与えられたデータが他のメッセージで利用されている場合
  2. 他のメッセージ中で一部のデータだけ利用している場合

- R3 の抽出部分

  R3 を客室予約管理システムから抽出した部分について説明する。図 4.11 は、R3 を抽出した部分のメッセージ通信を表す。図 4.11 中のメッセージ番号 1 から 17 は、作成した予約記録を基に客室の状態を操作するプロセスを表している。

  はじめに、CreateRecordServlet クラスは RoomControl クラスへ客室の状態を変更するメッセージ (checkAvailability()) を送信する。checkAvailability() メッセージが呼び出されると、DataCheck クラスに対して、日付の検査を行うメッセージ checkInputDay() と要求された客室数の検査を行うメッセージ checkInputRooms() を実行する。それらの検査を行った後、RoomControl クラスは客室の状態を変更するメッセージ consumeRooms() を呼び出し客室の状態を変更する。客室の状態を変更するメッセージ consumeRooms() が実行されると、RoomControl クラスから RoomVacancyData クラス、RoomStatusData クラスへ部屋の状態を変更するメッセージ searchStatus(),reduceRoom(),change-

Status() が実行され、客室の状態が変更される。

ここで各メッセージが持つ引数に注目する。CreateRecordServlet クラスから RoomControl クラスへ送信されるメッセージ checkAvailability() は resvRecordBean を引数として持つ。resvRecordBean は ResvRecordBean クラスのインスタンスであり、変数として String 型の変数を 6 つ、ArrayList 型（その中には String の変数が入る）の変数を２つ持っている。

次に実行される RoomControl クラスから CheckData クラスへのメッセージ checkInputDay(), checkInputRooms() が引数として利用する変数はインスタンス resvRecordBean が持つ String 型の変数２つである。

RoomControl クラスが実行するメッセージ consumeRooms() はインスタンス resvRecordBean を引数としてそのまま利用している。

RoomControl クラスから RoomVacancyData クラスに向かって部屋の空室数を変更するメッセージ reduceRoom() では、Integer 型の変数 4 つを新しく引数として利用している。

RoomControl クラスから RoomStatusData クラスに向かって部屋の状態を検査するメッセージ searchStatus() では、consumeRooms() の引数 resvRecordBean から String 型の変数を 5 つ利用している。

さらに RoomControl クラスから RoomStatusData クラスに向かって部屋の状態を変更するメッセージ changeStatus() では、consumeRooms() の引数 resvRecordBean と Integer 型の変数 4 つ、String 型の変数 1 つを新しく引数として利用している。

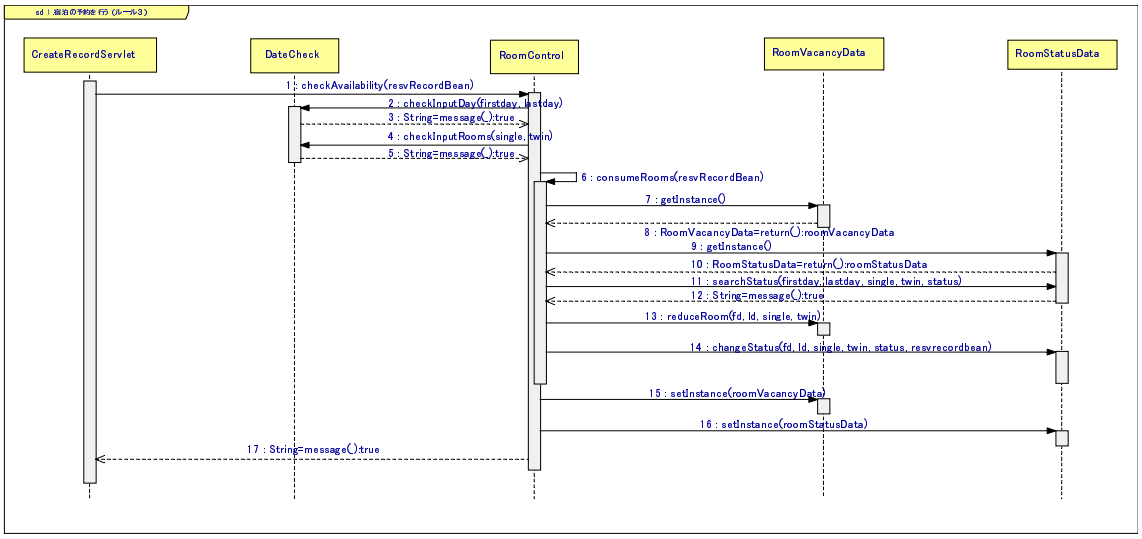

図 4.11: R3 適用後：5 つのインスタンス間のメッセージ通信

表 4.5: R3 抽出部分: メッセージが供給するデータ

メッセージ名	メッセージに与えたデータ	データの中身
checkAvailability()	resvRecordBean	String:6, ArrayList:2
checkInputDay()	firstday, lastday	String:2
checkInputRooms()	single, twin	String:2
consumeRooms()	resvRecordBean	String:6, ArrayList:2
reduceRoom()	fd, ld, single, twin	Integer:4
searchStatus()	dirstday, lastday, single, twin, status	String:5
changeStatus()	fd, ld, single, twin, status, resvRecordBean	Integer:4, String:7, ArrayList:2

表 4.6: R3 抽出部分:オブジェクトが使用するデータ

オブジェクト	オブジェクトが使用するデータ	メッセージに与えられた引数
RoomControl	fd, ld, single, twin, status, resvRecordBean	resvRecordBean
DataCehck	firstday, lastday, single, twin	firstday, lastday, single, twin
RoomVacancyData	fd, ld, single, twin	fd, ld, single, twin
RoomStatusData	fd, ld, single, twin, status	fd, ld, single, twin, status, resvRecordBean

次に、R3 を抽出したソースコード (RoomControl.java から呼び出すメソッド checkAvailavility(), checkInputDay(), checkInputRooms(), consumeRooms()) を示す (プログラム 5)。

```
import java.io.*;
import java.util.*;
public class RoomControl implements Serializable {
   public String checkAvailavility(ResvRecordBean resvRecordBean){
        DataCheck datecheck = new DataCheck();
        message = datecheck.checkInputDay(resvRecordBean.getFirstday(),
                                          resvRecordBean.getLastday());
        ...
        message = datecheck.checkInputRooms(resvRecordBean.getSingle(),
                                            resvRecordBean.getTwin());
        ...
        message = consumeRooms(resvRecordBean);
        ...
        return message;
   }
}
```

プログラム 5：R3 抽出部分 (RoomControl.java のメソッド checkAvailability())

```
import java.io.*;
import java.util.*;
public class RoomControl implements Serializable {
        public String consumeRooms(ResvRecordBean resvRecordBean){
        int first = firstday.difference(today);
        int last = lastday.difference(today);
        int rsingle = Integer.parseInt(rrb.getSingle());
        int rtwin = Integer.parseInt(rrb.getTwin());
        String status = "resv";
        RoomStatusData roomStatusData = RoomStatusData.getInstance();
        RoomVacancyData roomVacancyData = RoomVacancyData.getInstance();
        message = roomStatusData.searchStatus(first, last, rsingle, rtwin, status);
        ...
        roomVacancyData.reduceRoom(first, last, rsingle, rtwin);
        roomStatusData.changeStatus(first, last, rsingle, rtwin, status, resvRecordBean);
        ...
        RoomStatusData.setInstance(roomStatusData);
        RoomVacancyData.setInstance(roomVacancyData);
        return "true";
    }
}
```

プログラム 6：R3 抽出部分 (RoomControl.java のメソッド consumeRooms())

4.2 再構成ルール適用実験と評価

本節では、再構成ルール適用実験を実施した際に、追加したキャンセル待ち機能について説明する。次に、再構成ルール抽出実験を実施し、得られた結果から再構成ルールの評価を行う。この 4.2 節では、以下の定義を用いて説明を行う。(図 4.12 参照)

- 再構成前の客室予約管理システム (SB) にキャンセル待ち登録機能 (f1) を追加した状態を SB+f1 として取り扱う。
- SB+f1 にキャンセル時の利用可能記録検索機能 (f2) を追加した状態を SB+f1+f2 として取り扱う。
- R1 を適用し再構成を行った (SA1) 後に f1 を追加した状態を SA1+f1 として取り扱う。
- SA1+f1 に f2 を追加した状態を SA1+f1+f2 として取り扱う。
- R2 を適用し再構成を行った (SA2) 後に f1 を追加した状態を SA2+f1 として取り扱う。
- SA2+f1 に f2 を追加した状態を SA2+f1+f2 として取り扱う。
- R3 を適用し再構成を行った (SA3) 後に f1 を追加した状態を SA3+f1 として取り扱う。
- SA3+f1 に f2 を追加した状態を SA3+f1+f2 として取り扱う。

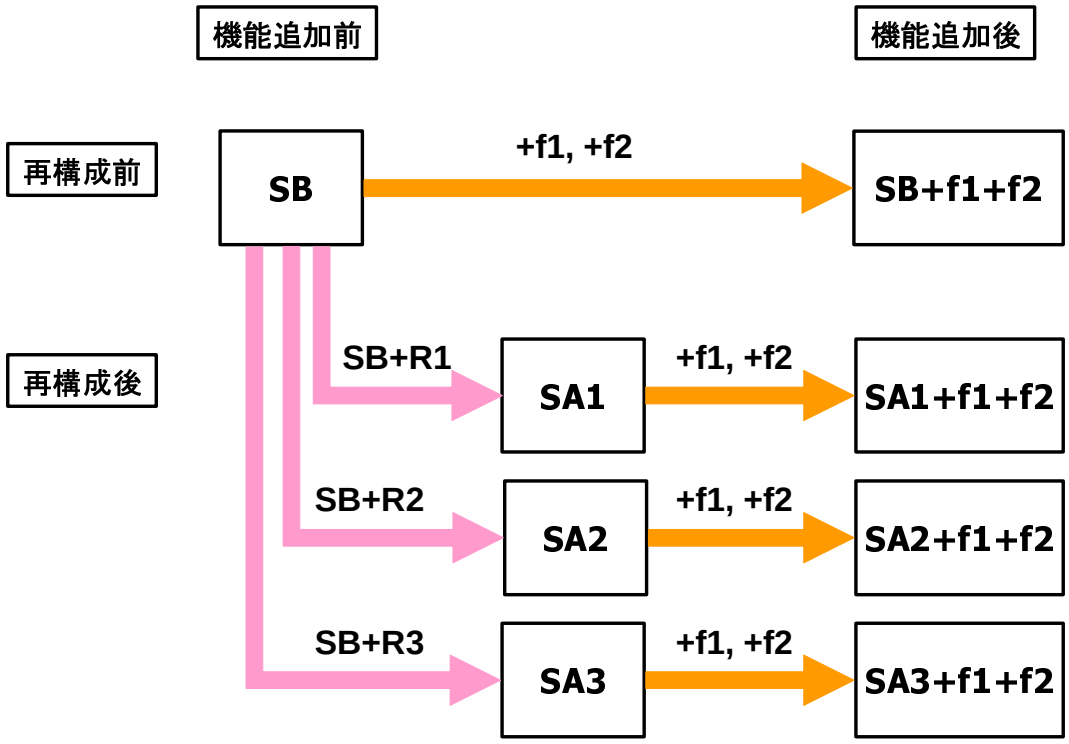

図 4.12: 再構成ルール適用実験概要図

4.2.1　システムに対する追加要求機能

客室予約管理システムに追加するキャンセル待ち機能 (f1,f2) について説明する。図 4.13 は f1,f2 を追加した客室予約管理システムのユースケース図である。f1,f2 の目的・概要はそれぞれ以下の通りである。

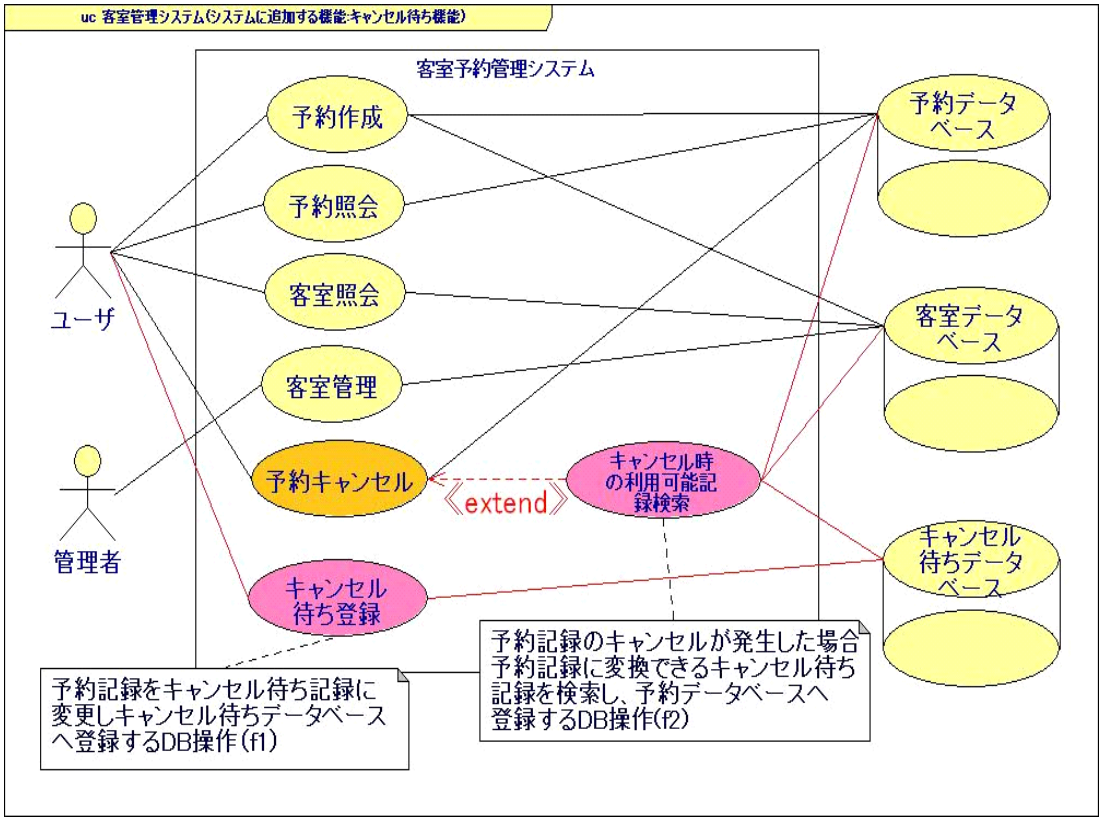

図 4.13: キャンセル待ち機能

1. キャンセル待ち登録機能 (f1)

目的　客室が満席で予約できなかった予約記録をキャンセル待ち記録に変更し、キャンセル待ちデータベースへ登録する。

概要　f1 は以下の DB 操作を実行する。予約の作成時に指定した期間の客室が満席の場合、予約記録をキャンセル待ち記録に変換しキャンセル待ちデータベースへ登録する。ユーザが客室の予約を行うが、すでに客室が満室で予約できない場合、キャンセル待ちに登録するかと尋ねる。

登録する場合は予約記録をキャンセル待ち記録に変換し、キャンセル待ちデータベースへ登録する。

2．キャンセル時の利用可能記録検索機能 (f2)

目的　予約記録のキャンセルが発生した場合、予約記録に変換できるキャンセル待ち記録を検索し、予約データベースへ登録する DB 操作。予約記録へ変更し、登録されたことをユーザへ知らせる。

概要　f2 は以下の DB 操作を実行する。キャンセル待ち記録を予約記録へ変換し、予約データベースへ登録した後にユーザへ通知する操作を行う。ユーザが予約記録のキャンセルを行なった際に、その予約記録に基づいて該当する日の客室を開放する。そこで、空室数を増やした日の範囲内にキャンセル待ち記録が存在しないか、キャンセル待ちデータベースから 1 つずつキャンセル待ち記録の検査を行う。
検査の内容は、開始日と終了日の範囲と要求している客室数が空室数以下かどうかである。そして、開始日と終了日の範囲がキャンセルした予約記録の範囲内であり、かつ要求する客室数も満たしている場合に予約記録へ変換を行い、変換した新しい予約記録を予約データベースに登録する。最後に該当するユーザにキャンセル記録から予約記録に変更したことを通知する。

次に SB に f1 と f2 を追加した結果を報告する。

4.2.2　キャンセル待ち機能 (f1,f2) 追加における変更箇所の分析結果

まず、SB にキャンセル待ち登録操作 (f1) とキャンセル時の利用可能記録検索機能 (f2) の追加を行った。機能追加の際に新しく作成した部分と修正・変更した既存部分の調査をソースコードを用いた。さらに、キャンセル待ち機能の際に実行されるメッセージの流れを調べるためシーケンス図で f1 を図 4.14 で、f2 を図 4.15 で表す。

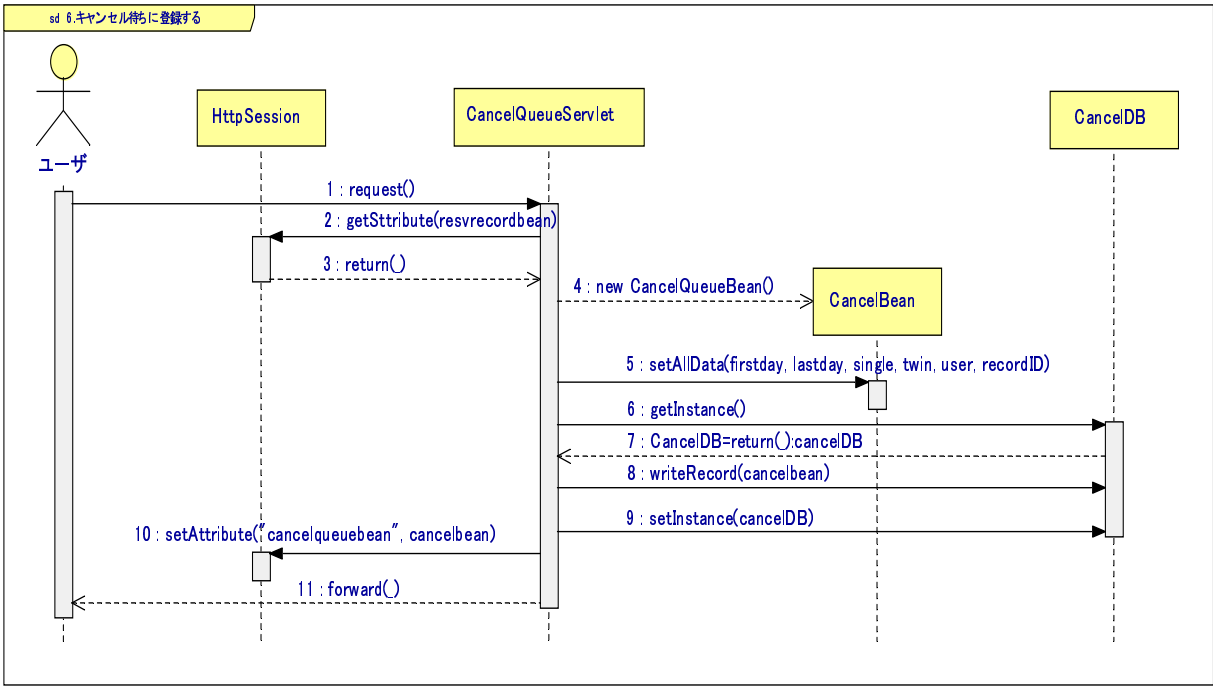

図 4.14: SB+f1 : キャンセル待ち登録機能追加後のシーケンス図

図 4.14, 図 4.15 を調査するにあたり、f1, f2 の中で発生するメッセージの総数をカウントした。カウントしたメッセージは同期/非同期のみであり、同期の戻りのメッセージはカウントしていない。キャンセル待ち登録操作・キャンセル時の利用可能記録検索機能追加の際に分析を行なった項目は、

- メッセージの総数:キャンセル待ち登録操作機能・キャンセル時の利用可能記録検索機能で発生するメッセージの合計
- 新しく作成したクラス:キャンセル待ち登録操作機能・キャンセル時の利用可能記録検索機能で作成した新しいクラスの数
- 新しく作成したメソッド:キャンセル待ち登録操作機能・キャンセル時の利用可能記録検索機能で作成した新しいメソッドの数
- 修正・変更した既存クラス:キャンセル待ち登録操作機能・キャンセル時の利用可能記録検索機能で修正・変更した既存クラスの数

である。

SB+f1 は表 4.7 の結果に、SB+f1+f2 は表 4.8 の結果となった。表 4.7 の分析結果から、SB+f1 では既存の部分に対する修正・変更はなく、全て新しく作成したクラス・メソッドだけで f1 を実現している。

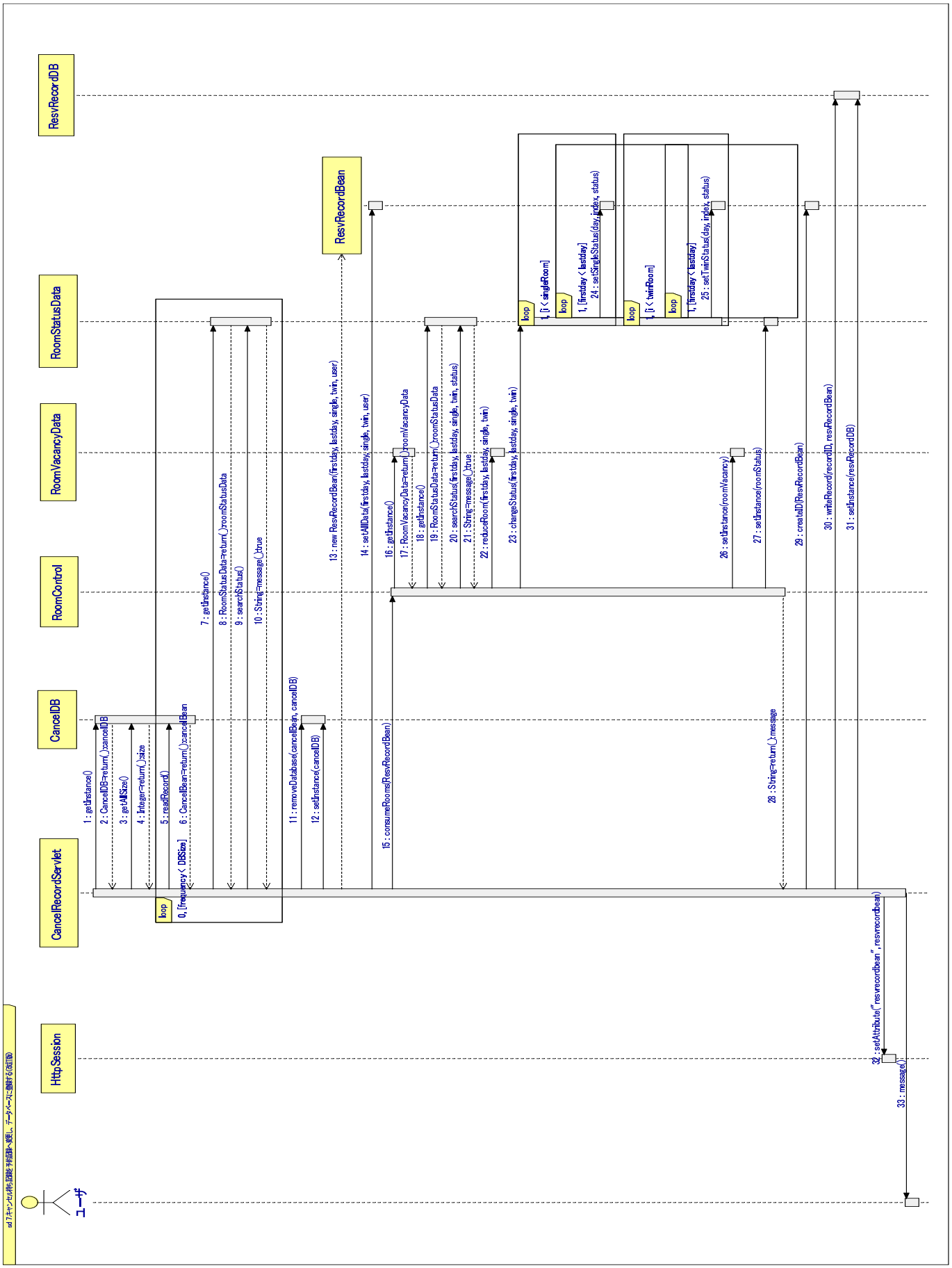

図 4.15: SB+f1+f2：キャンセル時の利用可能記録検索機能追加後のシーケンス図

図 4.14 から新しく作成したクラスはCancelQueueServlet クラス，CancelBean クラス，CancelDB クラスの 3 クラスである。CancelQueueServlet クラスは、指定した範囲の日の客室が満室で予約できなかった時、予約記録を受け取りキャンセル待ち記録として変換する。そして、キャンセル待ち記録をキャンセル待ちデータベースへ登録するメッセージを CancelDB クラスへ送信するという 2 つの役割を持っている。CancelBean は、予約記録をキャンセル待ち記録に変更する時に、CancelQueueServlet クラスから生成する Bean である。変数として開始日・終了日・シングル数・ツイン数・ユーザ・登録番号の文字列を持っている。CancelDB は予約記録から変換したキャンセル待ち記録を CancelQueueServlet クラスから受け取り登録する役割を持っている。

表 4.8 の分析結果から、f2 では既存の部分に修正・変更を加えて実現している。図 4.15 から修正・変更した既存クラスは、CancelRecordServlet クラス，RoomStatusData クラスの 2 クラスである。

特に CancelRecordServlet クラスの変更箇所が多く、14 メソッドが新たに追記されている。f2 全体のメッセージ総数は 24 メソッドとサービス単位としては一番多いメッセージ総数となった。

新しく作成したメソッドは、CancelDB クラスに 4 メソッド、RoomStatusData クラスに 1 メソッド作成した。RoomStatusData クラスに新しく作成したメソッド searchStatus() は、CancelRecordServlet クラスから呼び出される。キャンセル待ち記録で要求している客室数が、空室数の範囲内かどうかを検査する役割を持っている。

表 4.7: SB+f1:キャンセル待ち登録操作機能の分析結果

メッセージ総数	7 回
新しく作成したクラス	3 クラス
新しく作成したメソッド	4 メソッド
変更した既存クラス	0 クラス
既存クラスに追記したメソッド数	0 メソッド

表 4.8: SB+f1+f2:キャンセル時の利用可能記録検索機能の分析結果

メッセージ総数	24 回
新しく作成したクラス	0 クラス
新しく作成したメソッド	5 メソッド
変更した既存クラス	2 クラス
既存クラスに追記したメソッド数	14 メソッド

次に、各再構成ルールの方針に従ってシステムの再構成を行った後、f1,f2 をそれぞれ追加した。その結果をそれぞれまとめ、SB+f1, SB+f1+f2 と比較し、評価を行った。

4.2.3 再構成後のシステムに機能追加した結果の分析と比較・評価

R1：クラスの役割の委譲ルール適用結果と比較評価

SA1+f1 は図 4.16 のような通信順序となり、SA1+f1+f2 は図 4.17 のような通信順序となった。まず、SA1+f1 について SB+f1 と比較したときの違いと、SA1+f1 を実現する際に行なった作業について説明する。そして SB+f1 と比較・評価を行った結果について述べる。SA1+f1+f2 についても同じ手順で説明を行っていく。

- キャンセル待ち登録操作機能追加 (f1)

  図 4.16 は、R1 を適用後 f1 を追加したオブジェクト間の通信形態をシーケンス図で描写した。R1 適用前では、CancelQueueServlet オブジェクトから CancelDB オブジェクトへキャンセル待ち記録をキャンセルデータベースへ登録するメッセージを送信する前に CancelQueueServlet オブジェクトから CancelDB オブジェクトへ前処理となるメッセージを送信していた。R1 を適用することで、新しく作成した CancelControl オブジェクトが前処理のメッセージを CancelDB オブジェクトへ送信し、キャンセル待ちデータベースが必要なデータを CancelQueueServlet オブジェクトへ返す操作を行う。R1 を適用することにより CancelQueueServlet オブジェクトが持っていたキャンセル待ちデータベースへ登録するにあたり前処理のメッセージを実行する役割を CancelControl オブジェクトへ委譲している。さらに、CancelControl オブジェクトと CancelDB オブジェクト間のメッセージ通信を内部通信に変更することで、f1 で行われるメッセージ通信数を減少することが可能となった。

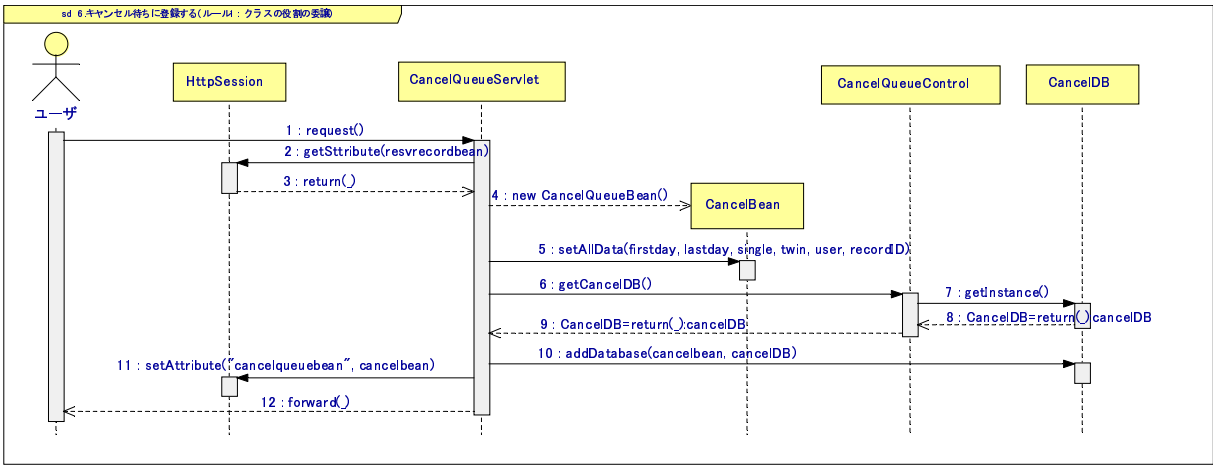

図 4.16: R1 適用：キャンセル待ち登録操作機能追加 (SA1+f1)

次に、図 4.16 を用いて、R1 適用後のキャンセル待ち登録操作機能実現のために新しく作成したクラスや変更したクラスを調査し、表 4.9 にまとめ、再構成前と比較・評価を行った。

SB+f1 と比較すると、メッセージ総数が 1 つ減少し、新しく作成したクラスとメソッドはそれぞれ 1 クラス、1 メソッド増加する結果となった。メッセージ総数の減少は、R1 の特徴である前処理のメッセージを新しいクラスがすべて請け負う。さらに、通信を行っているクラスと内部通信に変更することで、効果が得られる。

新しく作成したクラスと新しく作成したメソッドがそれぞれ SB ＋+f1 と比較して増加しているが、メソッドについては 5 つのメソッドの内、2 つのメソッドが内部通信用に作成されているので作成する新しいメソッドが増加してもメッセージ総数が減少している。f1 では、キャンセル待ちデータベースに登録するメッセージの前処理メッセージ数が 1 だったため R1 の効果が低い。今後は、前処理メッセージ数の数も考慮にいれた R1 の実験を行いたい。

表 4.9: SA1 と SA1+R1 の比較結果

比較した項目	SA1	SA1+R1
メッセージ総数	7 回	6 回
新しく作成したクラス	3 クラス	4 クラス
新しく作成したメソッド	4 メソッド	5 メソッド (内部メソッド 2 つ)
変更した既存クラス	0 クラス	0 クラス
変更した既存メソッド	0 クラス	0 メソッド

- キャンセル時の利用可能記録検索機能追加 (f2)

図 4.17 は、SA1+f1+f2 をシーケンス図で描写した図である。SA1+f1+f2 では、CancelRecordServlet オブジェクトはキャンセル待ちデータベースから予約記録に変換できるキャンセル待ち記録を探し出し、変換するという役割を持っている。R1 の適用により、CancelControl オブジェクトを新しく作成し、そのオブジェクトにデータベースから予約記録に変換できるキャンセル待ち記録を検索する役割を委譲した。

CancelRecordServlet オブジェクトと CancelDB オブジェクトの間に CancelControl オブジェクトを配置している。この配置により、CancelRecordServlet オブジェクトから予約記録に変換できるキャンセル待ち記録の検索を始めるメッセージ searchCancel() を受け取る。そして、CancelDB オブジェクトから該当するキャンセル待ち記録を探し出してくるという新しいプロセス手順に変更することができた。

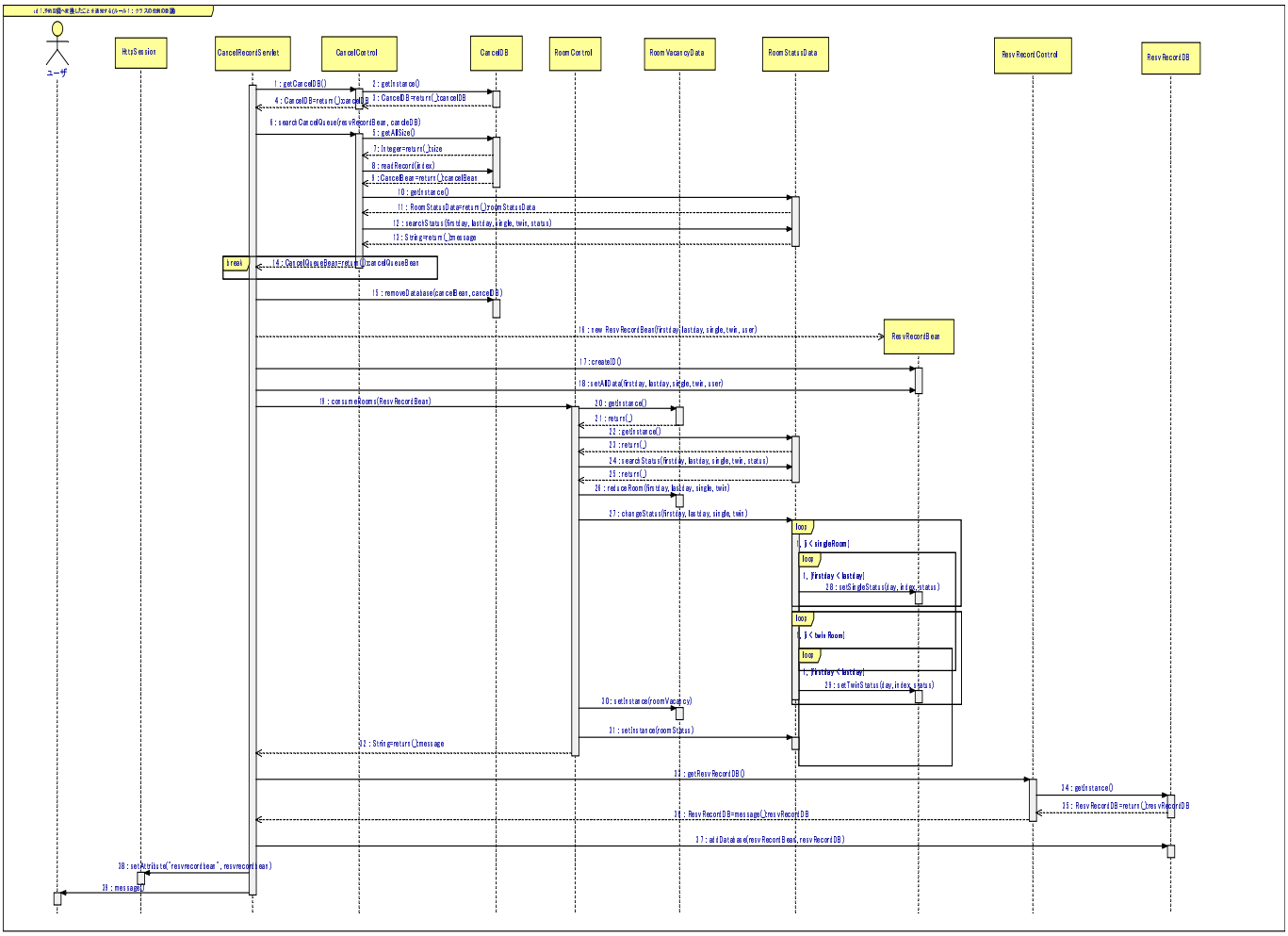

図 4.17: SA1+f1+f2：キャンセル時の利用可能記録検索機能追加

次に、図4.17からSA1+f1+f2実現のために新しく作成したクラスや変更したクラスを調査し表4.10にまとめ、SB+f1+f2と比較・評価を行った。R1を適用することで、SB+f1+f2に比べメッセージの総数、新しく作成したクラス、既存クラスに追記するメソッド数が減少した。CancelControlクラスに役割を委譲したことで、f2追加時のCancelRecordServletクラスの変更箇所が減少した。さらに、新しく作成したクラスCancelControlで作成したメソッド数は、SB+f1+f2と比べて1つ増加しただけで、変化はない。

表 4.10: SB+f1+f2 と SA1+f1+f2 を比較した結果

比較した項目	SB+f1+f2	SA1+f1+f2
メッセージ総数	24回	21回
新しく作成したクラス	0クラス	1クラス
新しく作成したメソッド	5メソッド	6メソッド
変更した既存クラス	2クラス	2クラス
変更した既存メソッド	0メソッド	0メソッド
既存クラスに追記したメソッド数	7メソッド	4メソッド

R1のまとめ

R1では、クラスの役割を新しいクラスに委譲し、メッセージの通信順序を変更した。SA1+f1では、キャンセル待ち記録をキャンセル待ちデータベースへ登録する前に実行されるメッセージの処理を行う役割を新しいクラスに委譲させメッセージ通信順序を変更した。その結果、メッセージの総数が1つ減少する効果が得られた。

SA1+f1+f2では、予約記録に変換できそうなキャンセル待ち記録をキャンセル待ちデータベースから探し出す処理を行う役割を新しいクラスに委譲させメッセージ通信順序を変更した。その結果、メッセージの総数が3つ減少し、既存クラスへ追記するメソッド数も3つ減少する効果が得られた。

R2：クラス間の従属関係ルール適用結果と比較評価

R2 を適用し再構成を行った後に f1,f2 を追加した結果、SA2+f1 の変化はなかったが、SA2+f1+f2 は図 4.18 のような通信順序となった。そこで、SA2+f1+f2 について SB+f1+f2 と比較したときの違いと、SA2+f1+f2 を実現する際に行なった作業について SB+f1+f2 と比較・評価を行った結果について述べる。

- キャンセル時の利用可能記録検索機能追加

  図 4.18 は、SA2+f1+f2 をシーケンス図で描写した図である。SB+f1+f2 では、客室の状態を変更するメッセージが RoomControl クラスから RoomStatusData クラス、RoomVacancyData クラスへ個別に送信されていた。R2 を適用し、RoomVacancyData クラスと RoomStatusData クラスの間に従属関係を持たせることで、RoomStatusData クラスがマスタークラスとなり RoomVacancyData クラスがスレーブクラスという関係に変更した。

  従属関係を結ぶことで RoomControl クラスから客室の状態を変更するメッセージがマスタークラスの RoomStatusData クラスだけに送信される。そして、そのメッセージを受け取った RoomStatusData クラスは、自分自身とスレーブクラスの RoomVacancyData クラスへ客室の状態をそれぞれ変更するメッセージを送信する。R2 の適用により客室状態を変更するメッセージの通信順序が大きく変更され、通信回数を減少することができた。

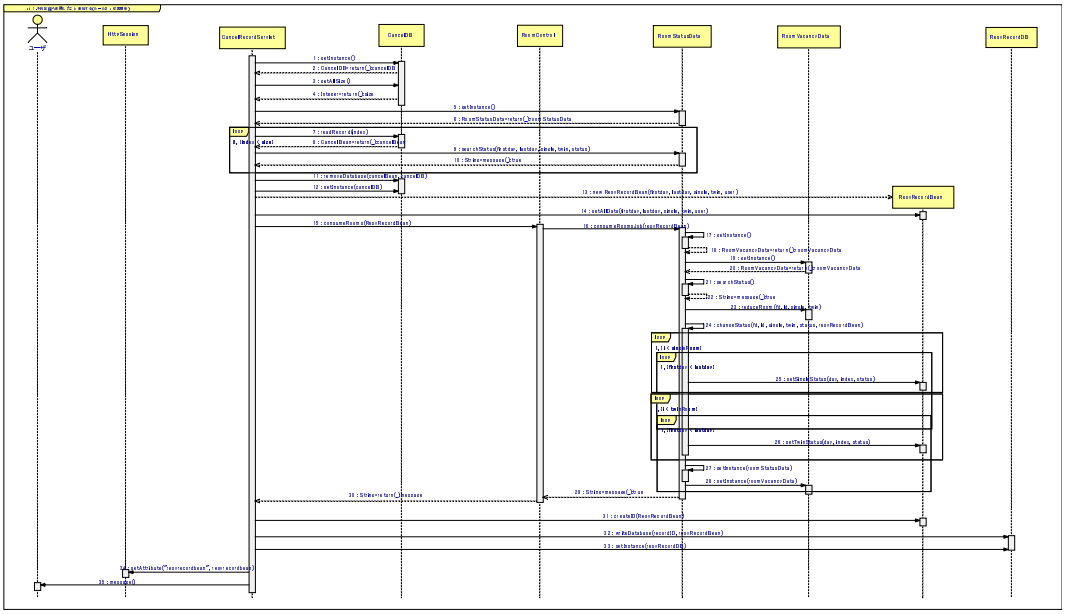

図 4.18: SA2+f1+f2：キャンセル時の利用可能記録検索機能追加後

次に、図 4.18 から SA2+f1+f2 のために新しく作成したオブジェクトや変更したオブジェクトの調査を行い表 4.11 にまとめ、SB+f1+f2 と比較・評価を行った。

R2 を適用することで SB+f1+f2 に比べメッセージの総数が減少した。それ以外の分析項目での変化は見られなかった。これは、SA2+f1+f2 の適用対象が客室状態を変更する一連のメッセージ中だけを対象としているためである。オブジェクトの関係が従属関係に変化したことで再構成前には、外部メッセージだったものが内部メッセージに変化した。メッセージ総数が減少する効果は得られているが、既存部分の追加作業の改善については R2 の対象となっておらずキャンセル待ち機能追加の場合、R2 の効果はメッセージ総数の減少のみとなった。

表 4.11: SB+f1+f2 と SA2+f1+f2 の比較結果

比較した項目	SB+f1+f2	SA2+f1+f2
メッセージ総数	24 回	21 回
新しく作成したクラス	0 クラス	0 クラス
新しく作成したメソッド	4 メソッド	4 メソッド
変更した既存クラス	2 クラス	2 クラス
変更した既存メソッド	0 メソッド	0 メソッド
既存クラスに追記したメソッド数	14 メソッド	14 メソッド

R2 のまとめ

R2 は、類似したデータを持つクラスが複数存在し、データを操作するメッセージが並行して送信する場合、クラス間に従属関係を持たせる。そうすることで、マスタークラスが自分自身とスレーブクラスに対してメッセージを送信するように変更し、メッセージ総数を減少させる効果が得られる。SA2+f1 では SB+f1 と同じ結果となった。SA2+f1+f2 ではメッセージ総数を 3 つ減少することができた。しかし、既存クラスに追記した数についての変化はなく、SA2+f1+f2 における効果はメッセージ総数の減少だけとなった。

R3：受け手と引数の境界不整合の解消ルール適用結果と比較評価

R3 を適用し再構成を行った後にキャンセル待ち機能 f1,f2 を追加した結果、f1 の追加において、R3 を適用するメッセージが存在しなかった。f2 の追加時には R3 を適用し、あるメッセージのデータ量が変化した。(図 4.19 参照) そこで、SA3+f1+f2 について SB+f1+f2 と比較したときの違いと SA3+f1+f2 を実現する際に行なった作業について述べる。

- キャンセル時の利用可能記録検索機能追加

  図 4.19 は、SA3+f1+f2 におけるオブジェクト間通信をシーケンス図で描写した図である。SB+f1+f2 と比較しても SA3+f1+f2 は、メッセージ通信順序は変化していない。

  CancelRecordServlet オブジェクトから RoomControl オブジェクトへ送信されるメッセージ consumeRooms() の引数が変更されている。SB+f1+f2 の場合は、引数として resvRecordBean が与えられていたが SA3+f1+f2 適用後の引数は dataA, dataB, dataC に変更された。dataA は Integer 型の変数を 4 つ持ち、dataB は dataA と String 型の変数を 1 つ持ち、dataC は dataB と resvRecordBean の変数を持っている。

  引数をメッセージが必要とするデータにまとめたことで、オブジェクト境界とデータ境界の不整合が解消した。

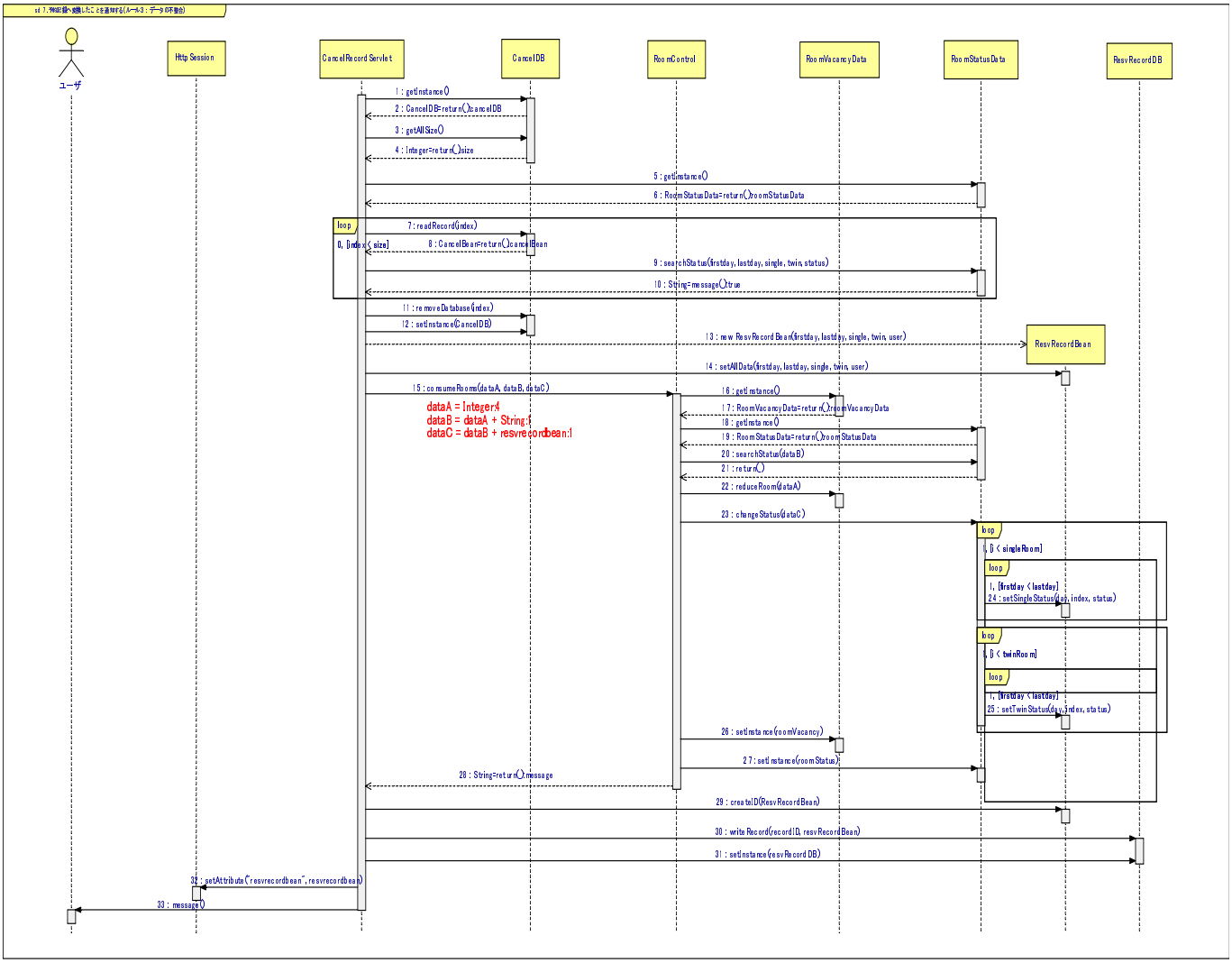

図 4.19: SA3+f1+f2：キャンセル時の利用可能記録検索機能追加後

次に、図4.19からSA3+f1+f2のために新しく作成したクラスや変更したクラスの調査を行い表4.12、4.13にまとめ、SB+f1+f2と比較・評価を行った。SA3+f1+f2は、各メッセージが必要とするデータごとに引数として与えるよう変更したことで無駄なデータの削除という効果を得ることができた。しかし、表4.13においてメッセージのデータ量を変化させたCancelRecordServlet クラスからRoomControlクラスのメソッドconsumeRooms()のデータ量がSB+f1+f2よりも増加している。各メッセージで利用するデータごとにまとめてメッセージ中に流したことでデータの不整合は解消されたがデータ量が増加してしまう欠点が露呈した。

表 4.12: SB+f1+f2 と SA3+f1+f2 の変更箇所比較結果

比較した項目	SB+f1+f2	SA3+f1+f2
メッセージ総数	24 回	24 回
新しく作成したクラス	0 クラス	0 クラス
新しく作成したメソッド	4 メソッド	4 メソッド
変更した既存クラス	2 クラス	4 クラス
変更した既存メソッド	0 メソッド	4 メソッド
既存クラスに追記したメソッド数	14 メソッド	14 メソッド

表 4.13: SB+f1+f2 と SA3+f1+f2 のデータ量比較結果

データ量が変化したメソッド	consumeRooms()	
	SB+f1+f2	SA3+f1+f2
変化したデータ量	resvRecordBean:1 resvRecordBean = String:6, ArrayList:2	dataA:1, dataB:1, dataC:1 dataA = Integer:4 dataB = dataA + String:1 dataC = dataB + resvRecordBean:1

R3 のまとめ

R3 では、受け手と引数の境界不整合を解消するために、メッセージで利用するデータごとをまとめて先頭のメッセージの引数として渡すようにデータ量を変更した。SB+f1+f2 と比較を行った結果、SA2+R3 を適用した一連のメソッドにおける不整合が解消した。しかし、キャンセル時の利用可能記録検索機能を追加するにあたり R3 を適用することで f2 追加における機能追加容易性の面では、SB+f1+f2 と変化はなかった。よって R3 は他の機能追加要求の場合に効果が得られるものだと考察する。

4.3 適用実験のまとめ

最後に、本章で行った再構成ルール適用実験をまとめる。各再構成ルールごとにわけ、評価結果から利点・欠点を述べる。

4.3.1 利点

- R1

  SB+f1 と SA1+f1 を比較した結果、R1 を適用することでキャンセル待ち記録登録機能のメッセージ総数が 1 つ減少した。
  SB+f1+f2 と SA1+f1+f2 を比較した結果、R1 を適用することで SA1+f1+f2 のメッセージ総数が 3 つ減少した。既存クラスに追記するメソッドについても 4 つ減少した。

- R2

  SB+f1+f2 と SA2+f1+f2 を比較した結果、R2 を適用することで SA2+f1+f2 のメッセージ総数が 3 つ減少した。

- R3

  SB+f1+f2 と SA3+f1+f2 を比較した結果、R3 を適用することで SA3+f1+f2 で発生するメッセージの引数と受け手オブジェクトの境界不整合が解消した。

4.3.2 欠点

- R1

  SB+f1 と SA1+f1 を比較した結果、R1 を適用することで新しく作成したクラス/メソッドが 1 つ増加した。
  SB+f1+f2 と SA1+f1+f2 を比較した結果、R1 を適用することで新しく作成したクラス/メソッドが 1 つ増加した。

- R2

  SB+f1+f2 と SA2+f1+f2 を比較した結果、R2 は機能追加における既存部分の修正箇所に対する効果を得られなかった。

- R3

  SB+f1+f2 と SA3+f1+f2 を比較した結果、R3 は機能追加における既存部分の修正箇所に対する効果を得られなかった。

4.3.3 改善案

- R1

  R1:キャンセル待ち記録登録機能では、前処理となるメッセージが１つしか存在しないのでR1の効果が得られない。最低２つ以上のメッセージが発生する場合にR1を適用するべきである。(パラメータの設定) キャンセル時の利用可能記録検索機能でも同様な改善案となった。

- R2

  今回の機能追加実験で複雑になった部分とR3が着目した複雑さの部分が異なっていた。今後さらに詳しい分析を行ない、R2を適用することで効果のある機能拡張・追加を見つける必要がある。

- R3

  今回の機能追加実験で複雑になった部分とR3が着目した複雑さの部分が異なっていた。今後さらに詳しい分析を行ない、R3を適用することで効果のある機能拡張・追加を見つける必要がある。

第5章　まとめと今後の課題

最後に研究を総括し、今後の課題について述べる。

5.1　まとめ

本研究では、機能拡張・追加要求に対応できる設計レベルでの再構成手法について提案を行った。クラス間の通信 (相互作用) に着目し、各サービスのメッセージを調査した。調査結果から分析を行ない、以下の再構成ルールを抽出、作成した。

- R1:クラスの役割の委譲 (Frequency)
- R2:クラスの従属化 (Style)
- R3:受け手と引数の境界不整合の解消 (Amount)

作成した再構成ルールを既存のシステムに適用する実験を行い、再構成前と後では機能追加作業にどのような影響を与えるのか比較・評価を行った。その結果、R1 をシステムに適用した場合に、既存システムの追加箇所が減少する効果を得られ、機能追加がしやすくなった。さらにサービス全体のメッセージ総数も減少した。

R2,R3 を適用することで、機能追加作業は軽減しなかった。今後、R2,R3 についてはさらに詳しい分析が必要だと考えている。

5.2　今後の展望

本研究で提案した再構成ルールをより汎用的に利用するに当たり、以下の課題に取り組む必要がある。

5.2.1　再構成ルールの洗練

今回作成した再構成ルールは同じシステムを用いて適用実験を行っている。より汎用的に再構成ルールが利用できるように、同じドメインを持つ既存システムに対して再構成ルールの適用実験を行い、再構成ルールを洗練する。他のシステムで適用実験を行うことで、今回作成した再構成ルールの修正や条件追加が発生するはずである。そのようなパ

ラメータを作成することで再構成ルールがより洗練されたものとなるはずである。また、今回作成した再構成ルールは相互作用の特徴値 Style についてあまり触れなかった。そこで、Style を変化させる再構成ルールを作成する必要がある。

5.2.2 システムのレイヤー化

再構成ルールを適用した部分のコンポーネント化を行う。今回の適用実験でシステムの再構成を行い、機能を実現した。そこで、適用後のシステムからコンポーネントとなる部分を抽出し、コンポーネント化を行いたいと考えている。

謝辞

本研究を行うにあたり終始御指導賜りました鈴木正人助教授に心より深く感謝申し上げます。

本研究を行うにあたり大変有益な御助言をいただきました落水浩一郎教授に心より感謝申し上げます。

また研究を進めるに当たり貴重な意見をいただきました研究室の皆様に心より感謝いたします。

最後に、大学院での生活を援助を行ってくれた家族ならびに生活面でお世話になった友人に感謝いたします。

付 録A　付属資料

A.1　クラス図

図 A.1 キャンセル機能追加前 (SB)
図 A.2 キャンセル機能追加後 (SB+f1+f2)

A.2　シーケンス図

A.2.1　再構成前

図 A.3 予約記録作成
図 A.4 予約記録照会
図 A.5 空室照会
図 A.6 客室管理
図 A.7 予約キャンセル
図 A.8 キャンセル待ち登録
図 A.9 予約記録変換

A.2.2　再構成後

図 A.10 予約記録作成 (R1)
図 A.11 予約キャンセル (R1)
図 A.12 予約記録変換 (R1)
図 A.13 予約記録作成 (R2)
図 A.14 客室管理 (R2)
図 A.15 予約キャンセル (R2)
図 A.16 予約記録変換 (R2)
図 A.17 予約記録作成 (R3)
図 A.18 予約記録変換 (R3)

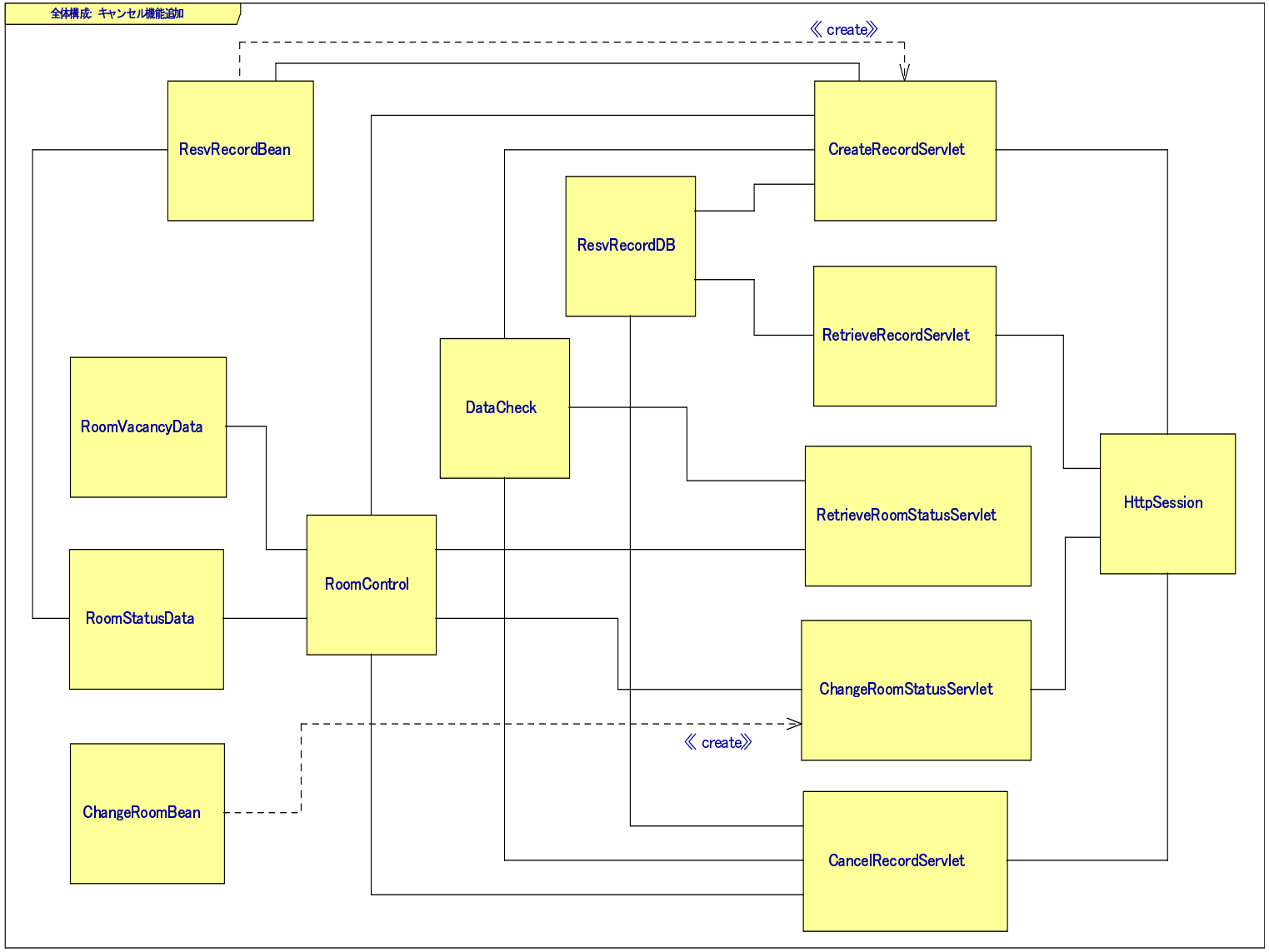

図 A.1: 客室予約管理システム（SB）

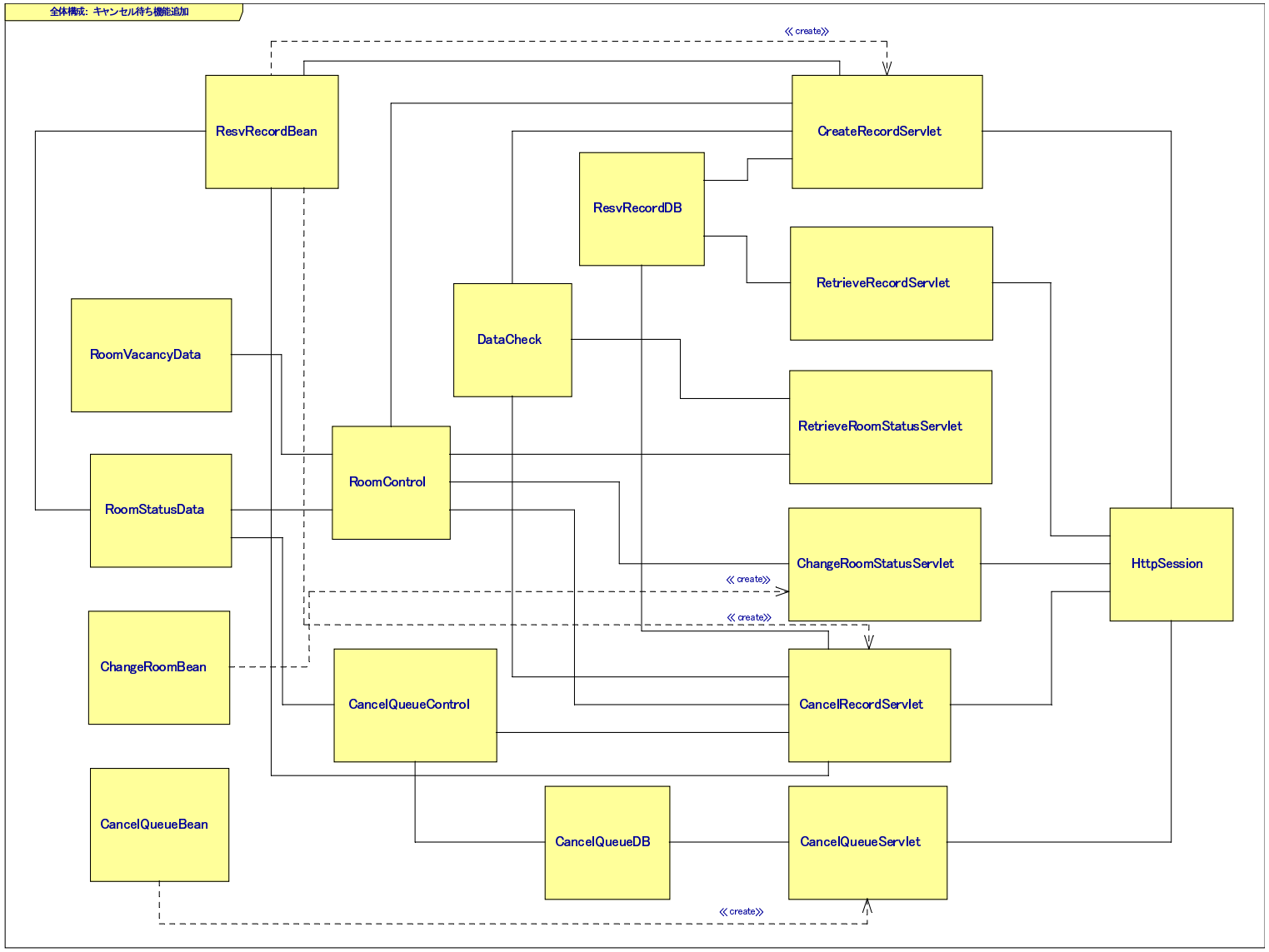

図 A.2: 再構成前の客室予約管理システム（SB+f1+f2）

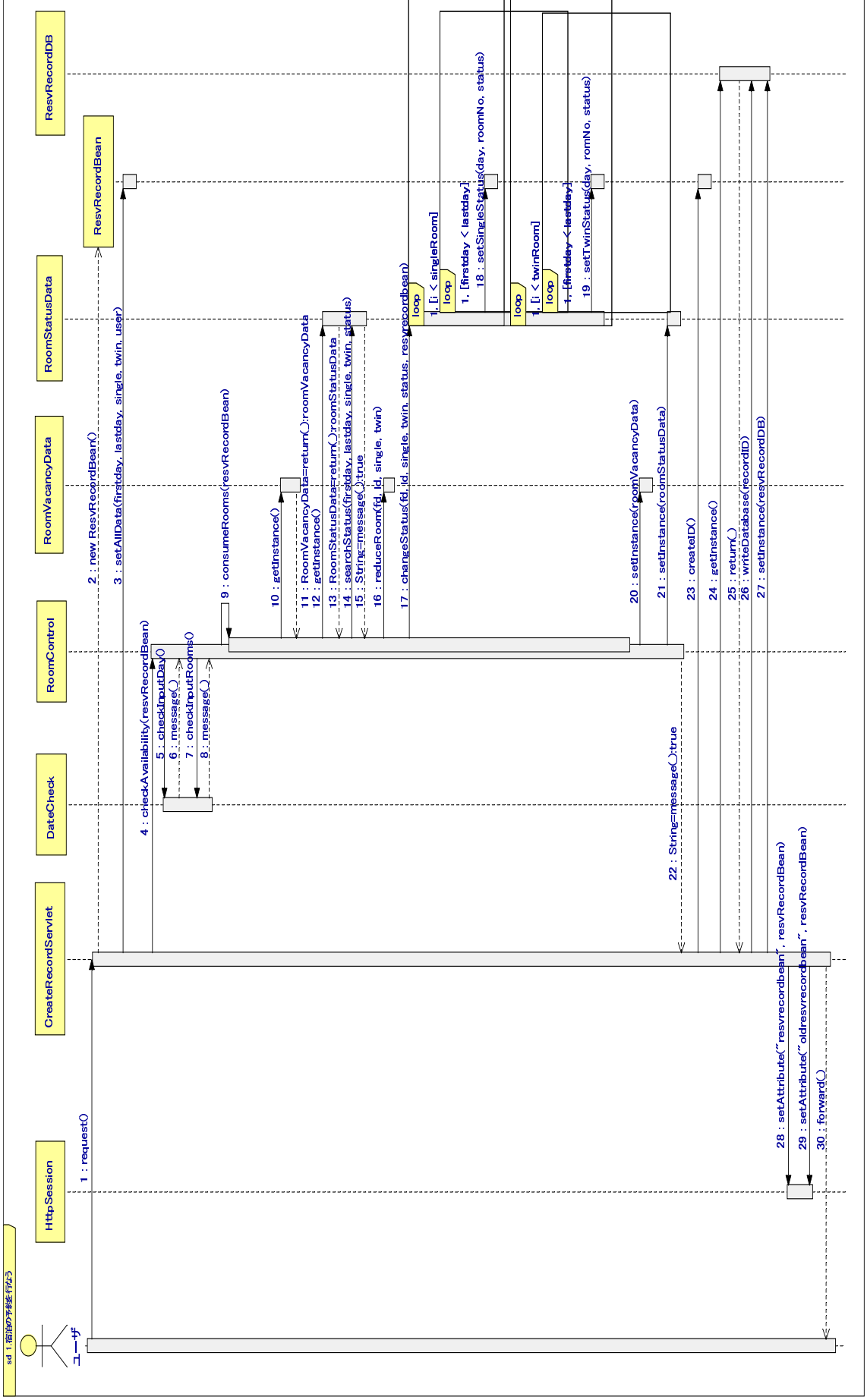

図 A.3: 予約記録を作成する

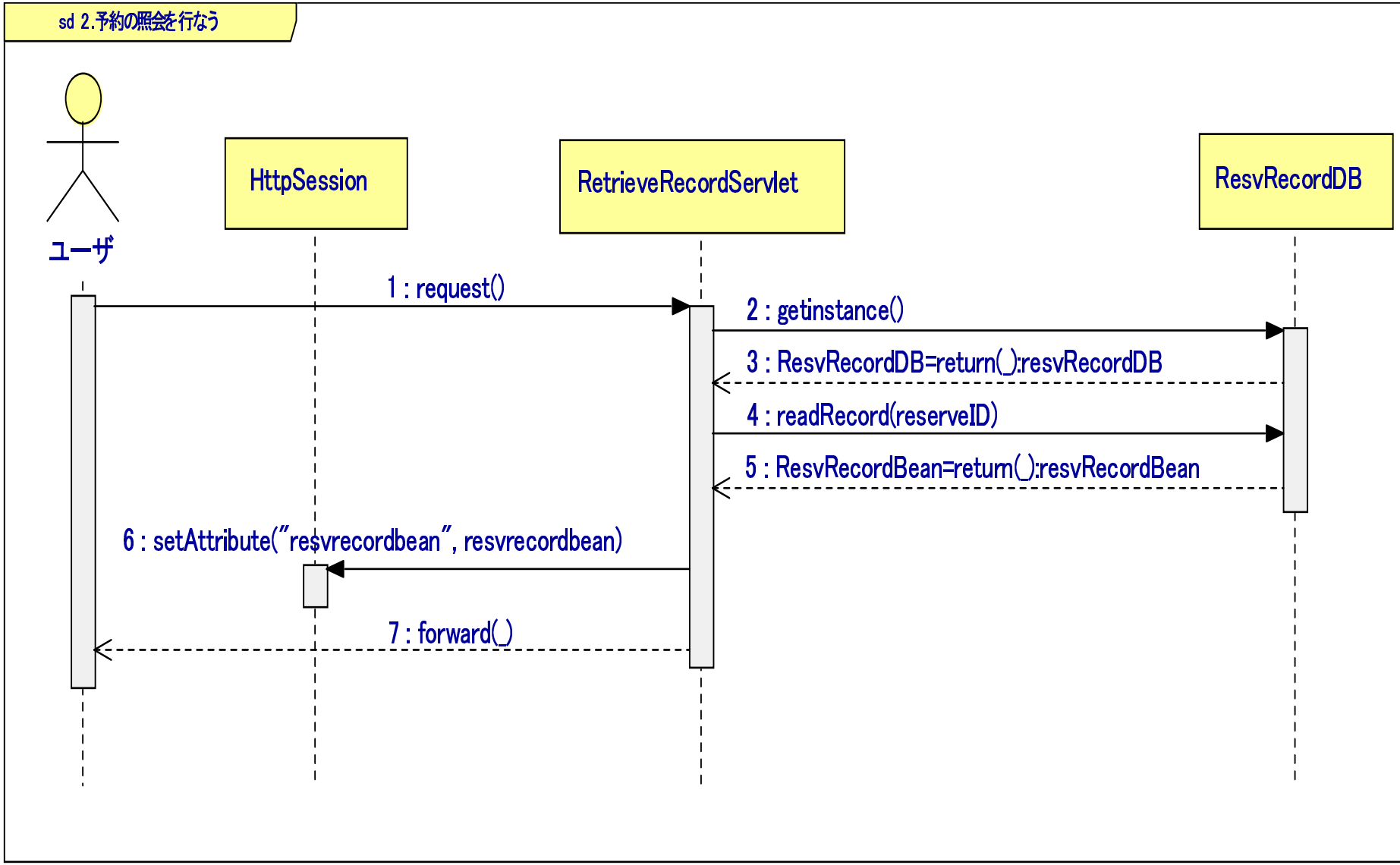

図 A.4: 予約記録を照会する

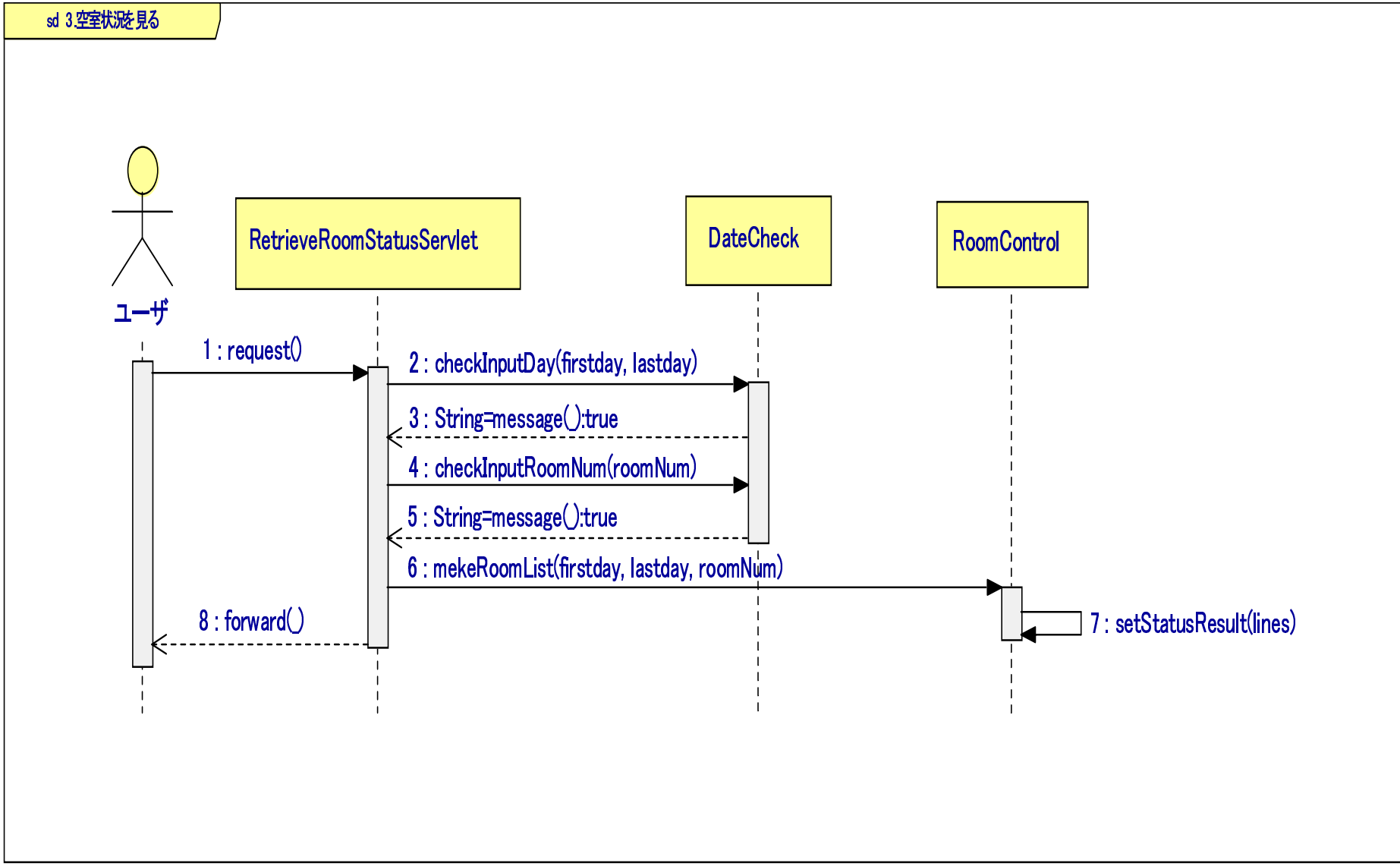

図 A.5: 空室状況を照会する

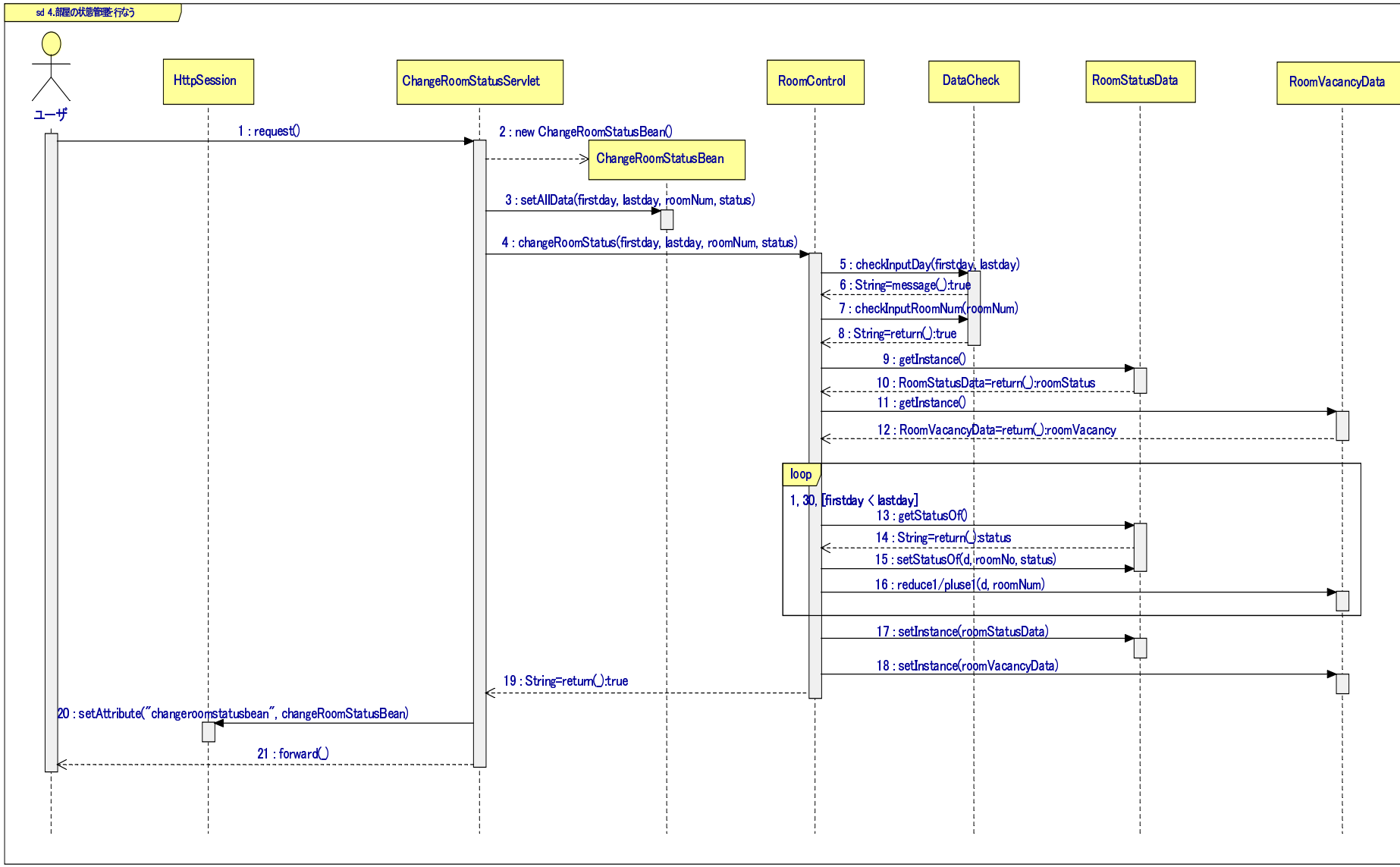

図 A.6: 客室の管理を行なう

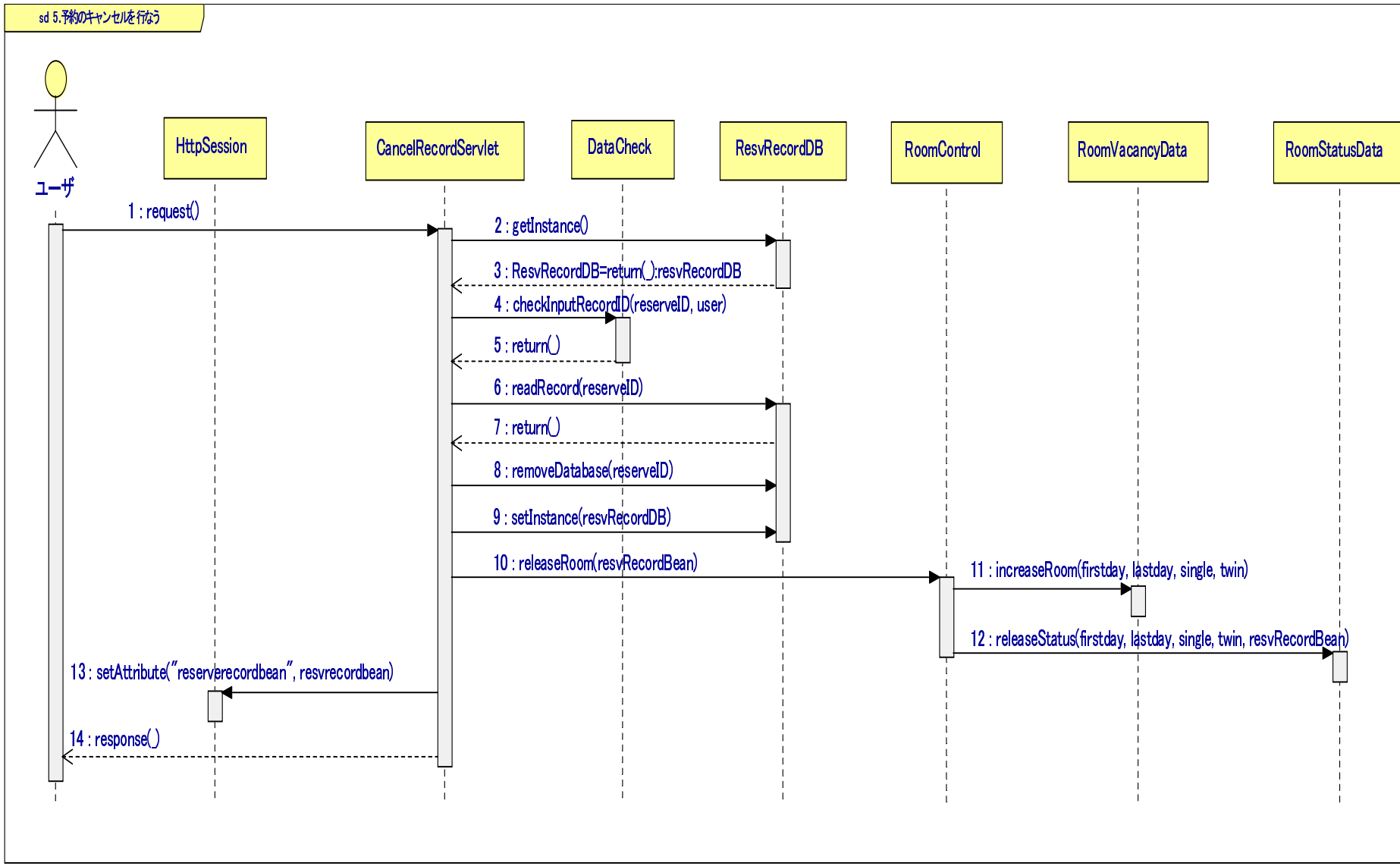

図 A.7: 予約をキャンセルする

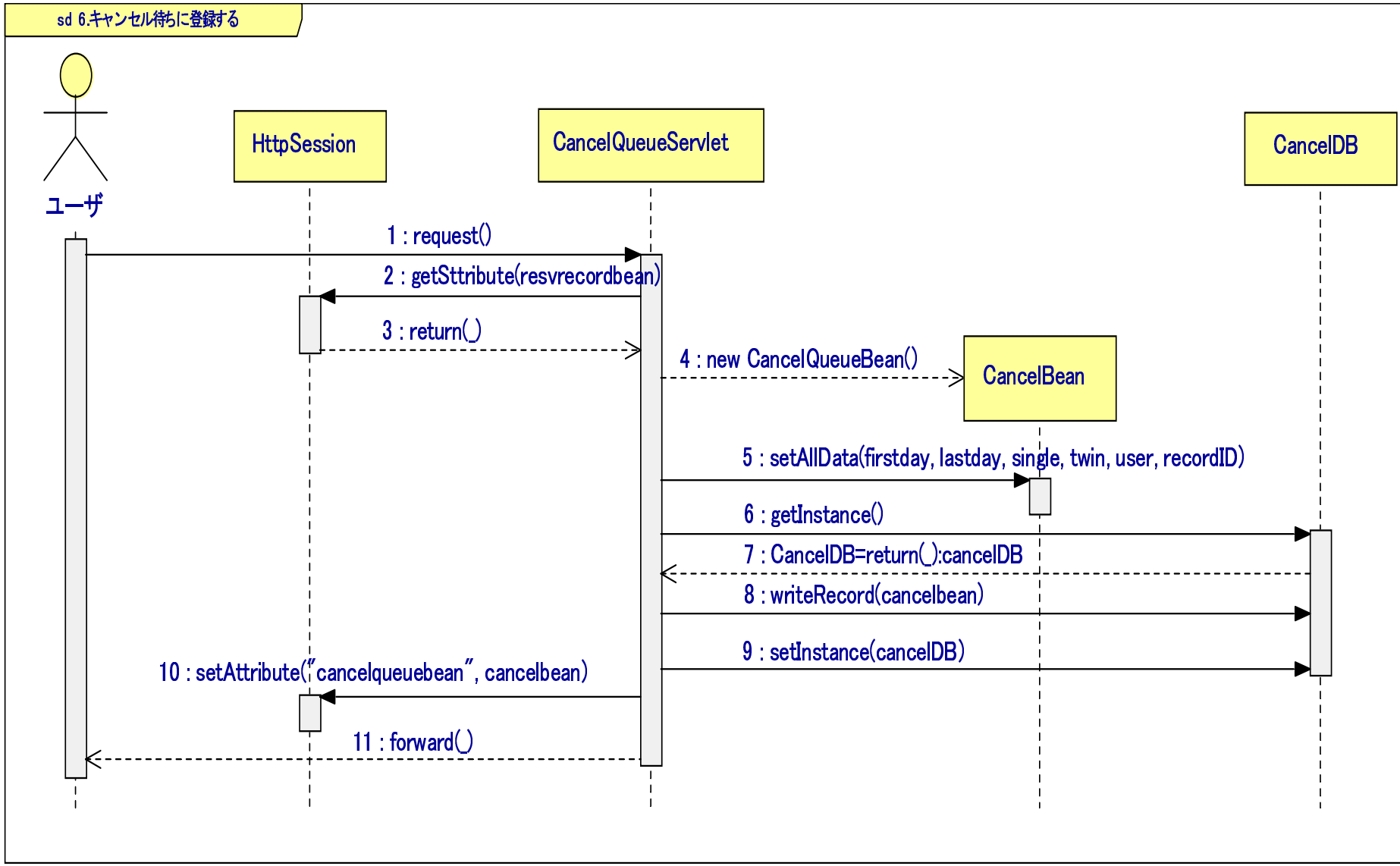

図 A.8: キャンセル待ちに登録する

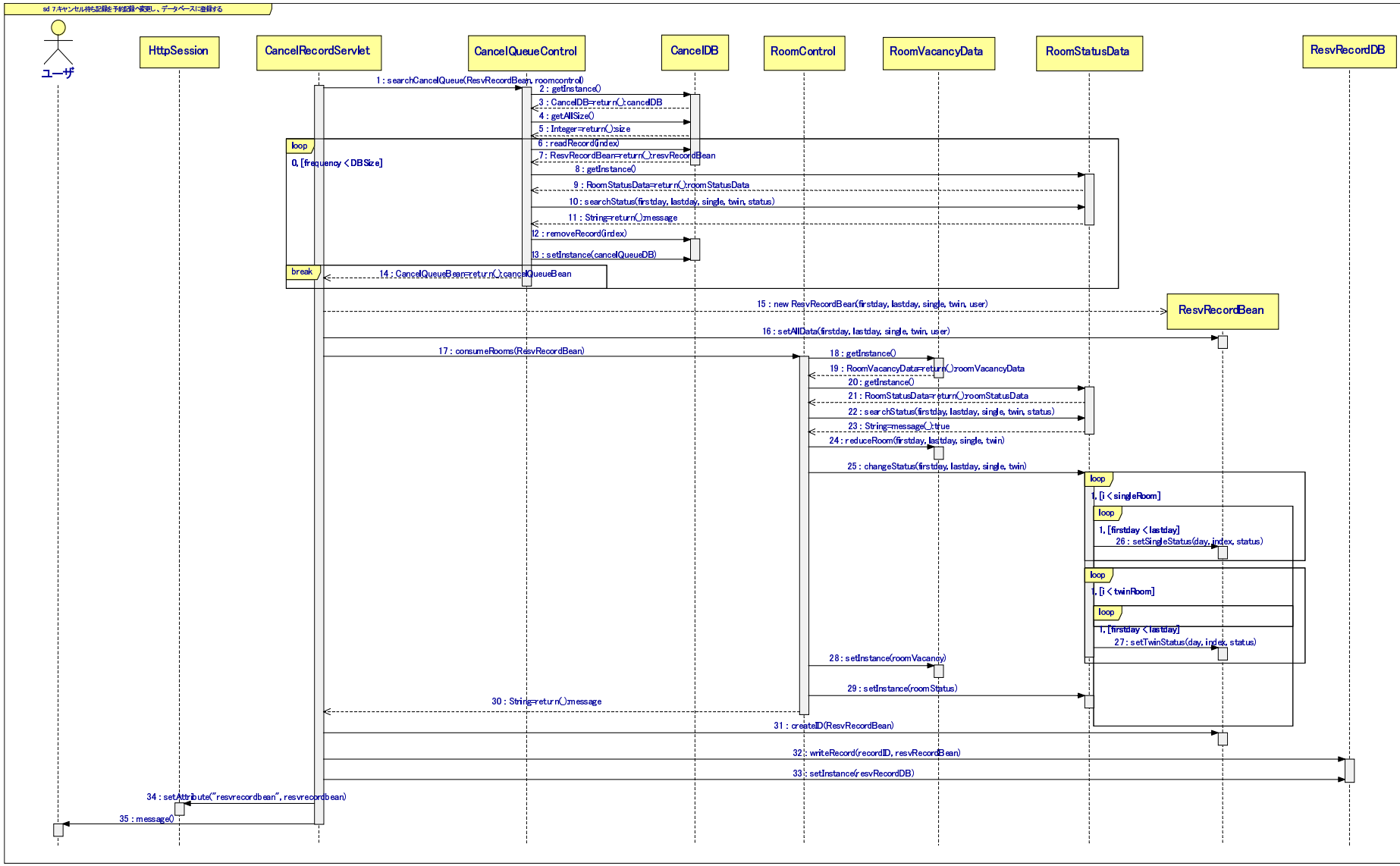

図 A.9: 予約記録へ変換する

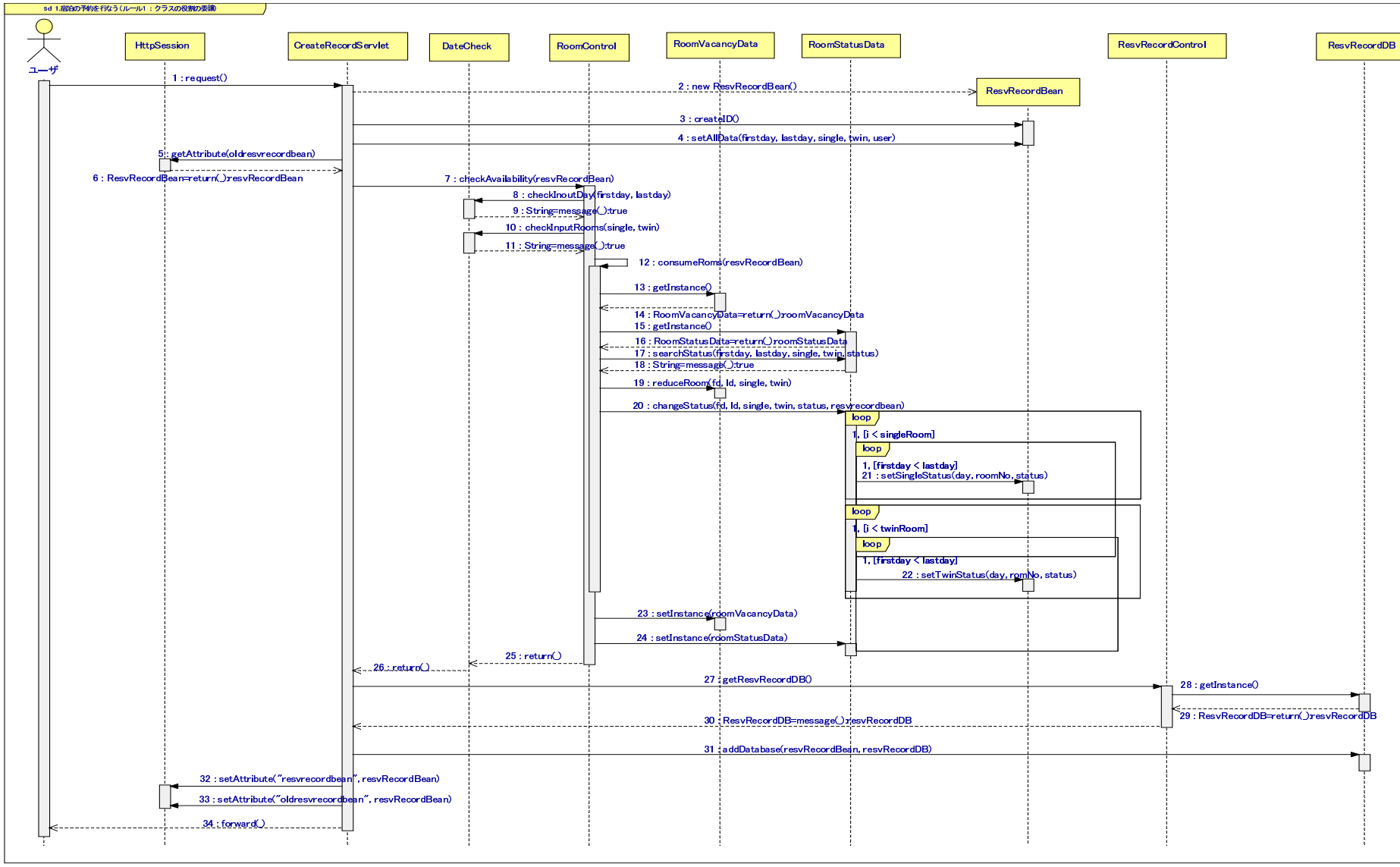

図 A.10: 予約記録作成 (R1 適用)

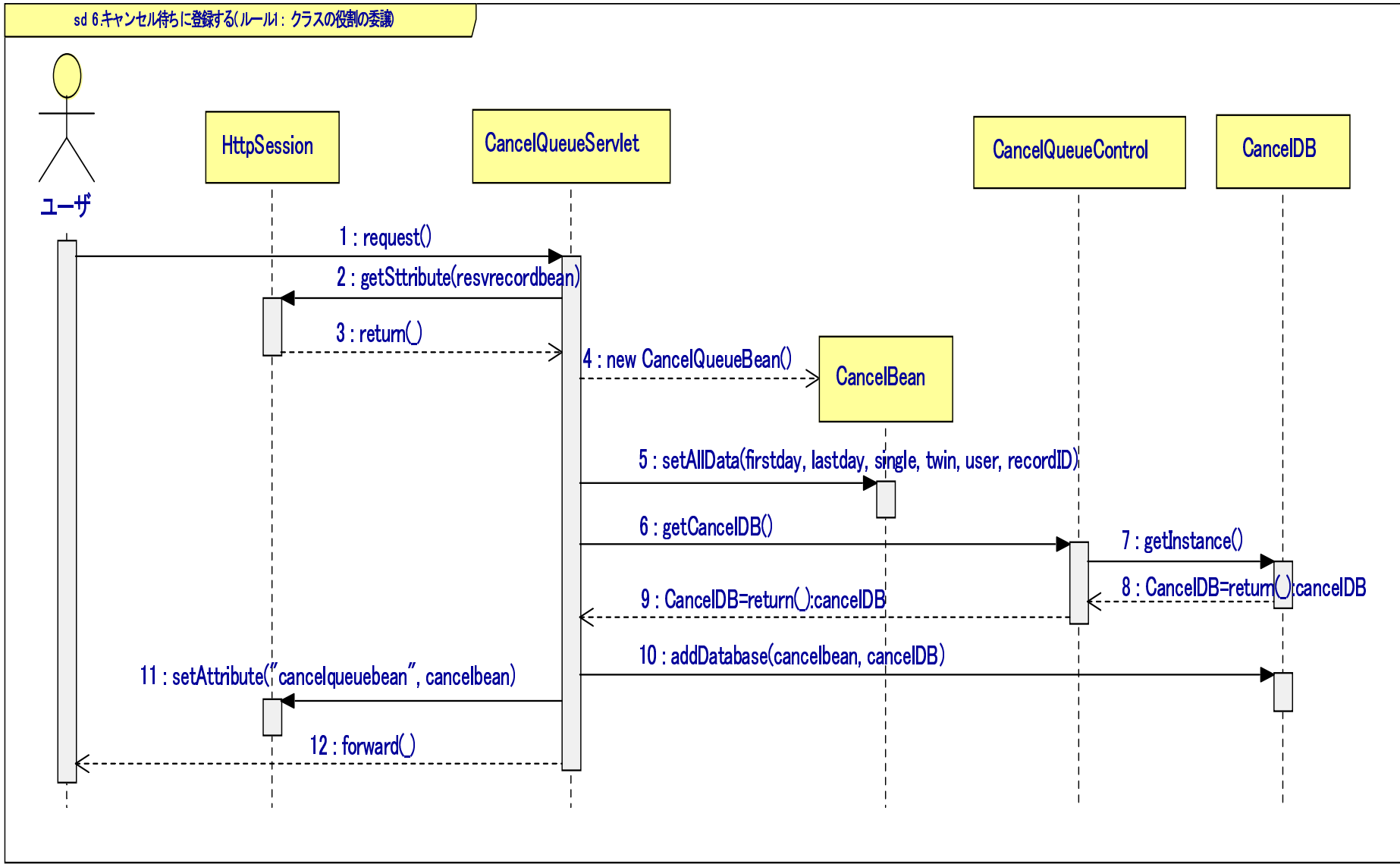

図 A.11: 予約のキャンセル (R1 適用)

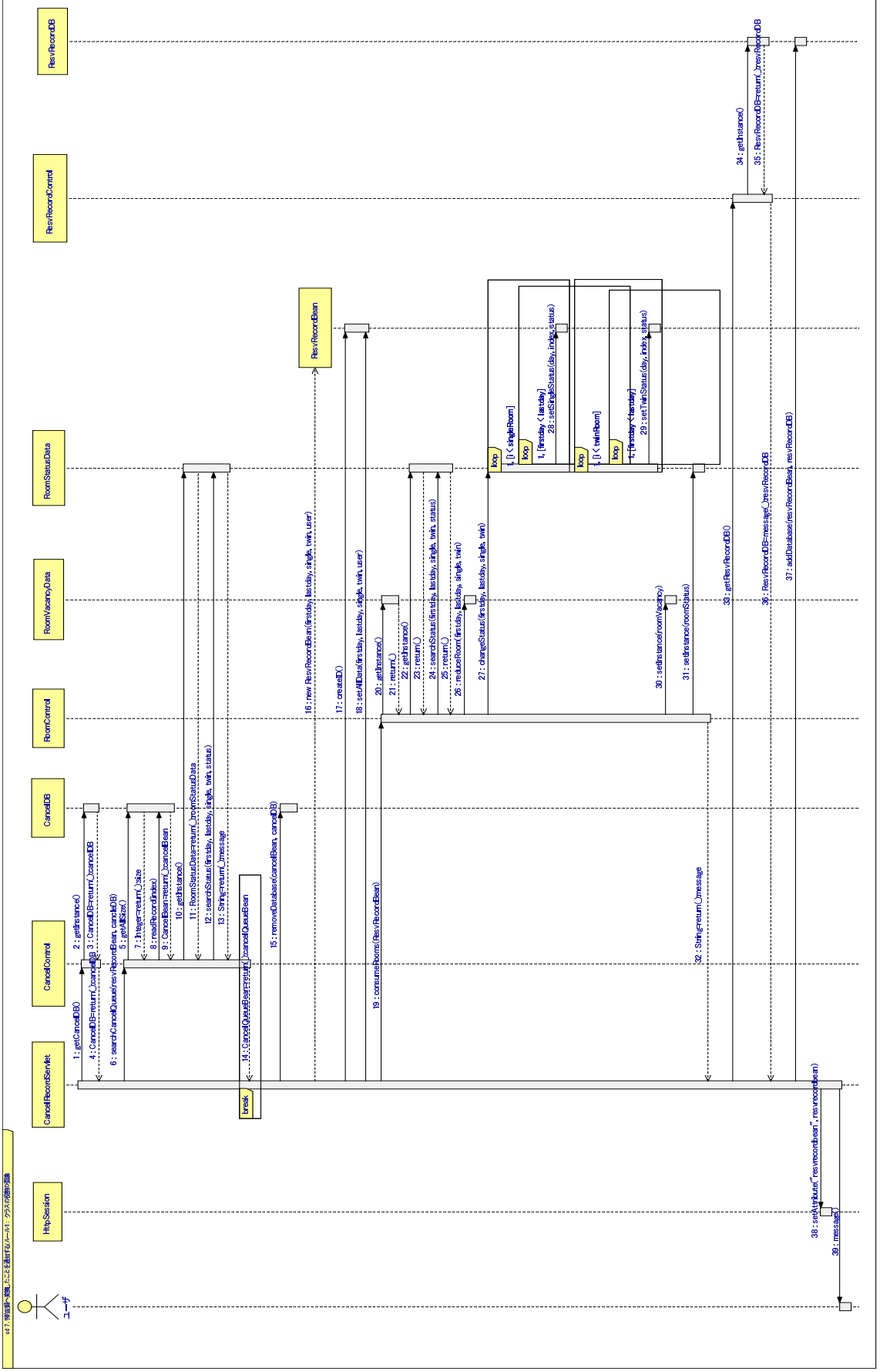

図 A.12: 予約記録へ変換する (R1 適用)

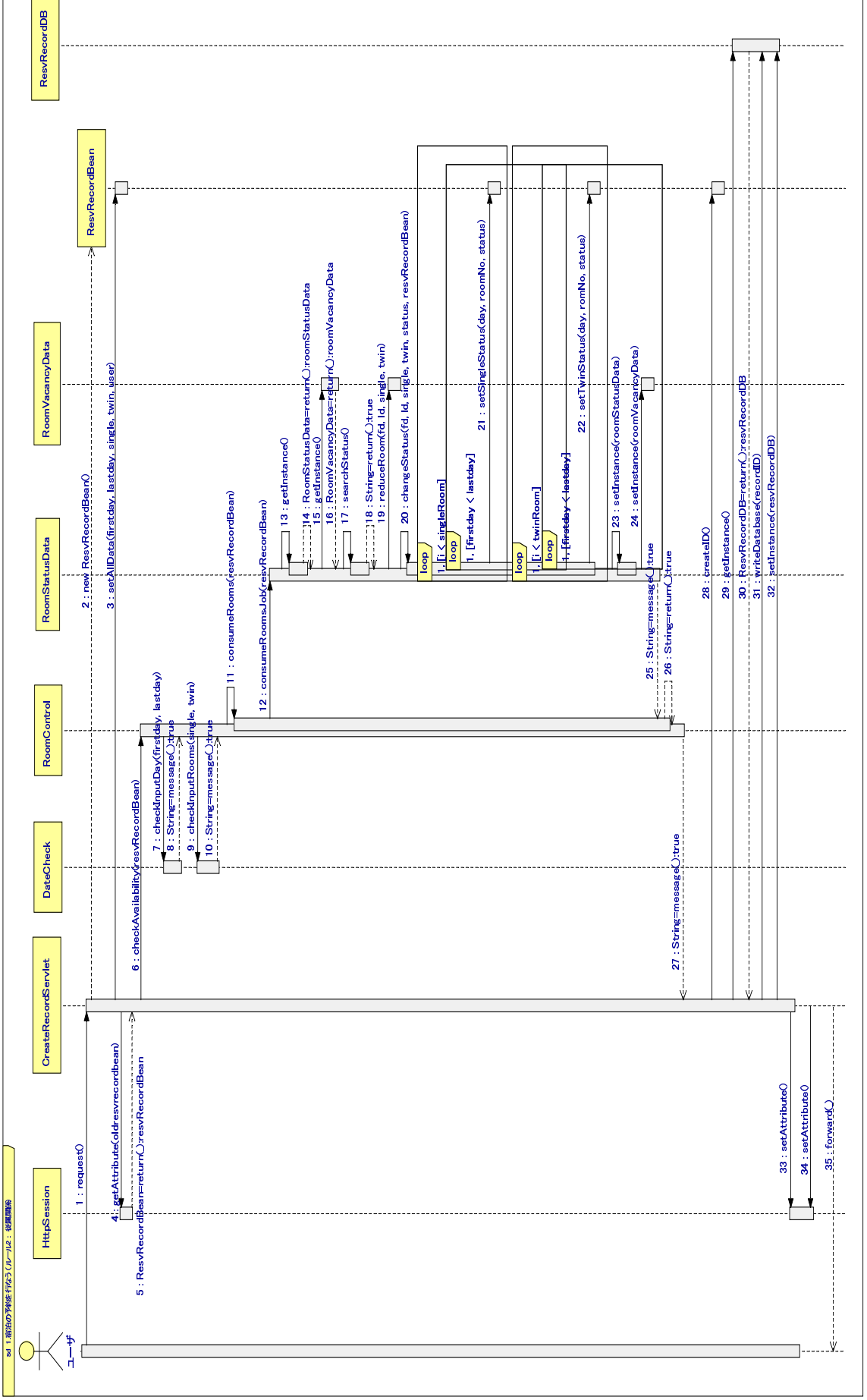

図 A.13: 予約記録作成 (R2 適用)

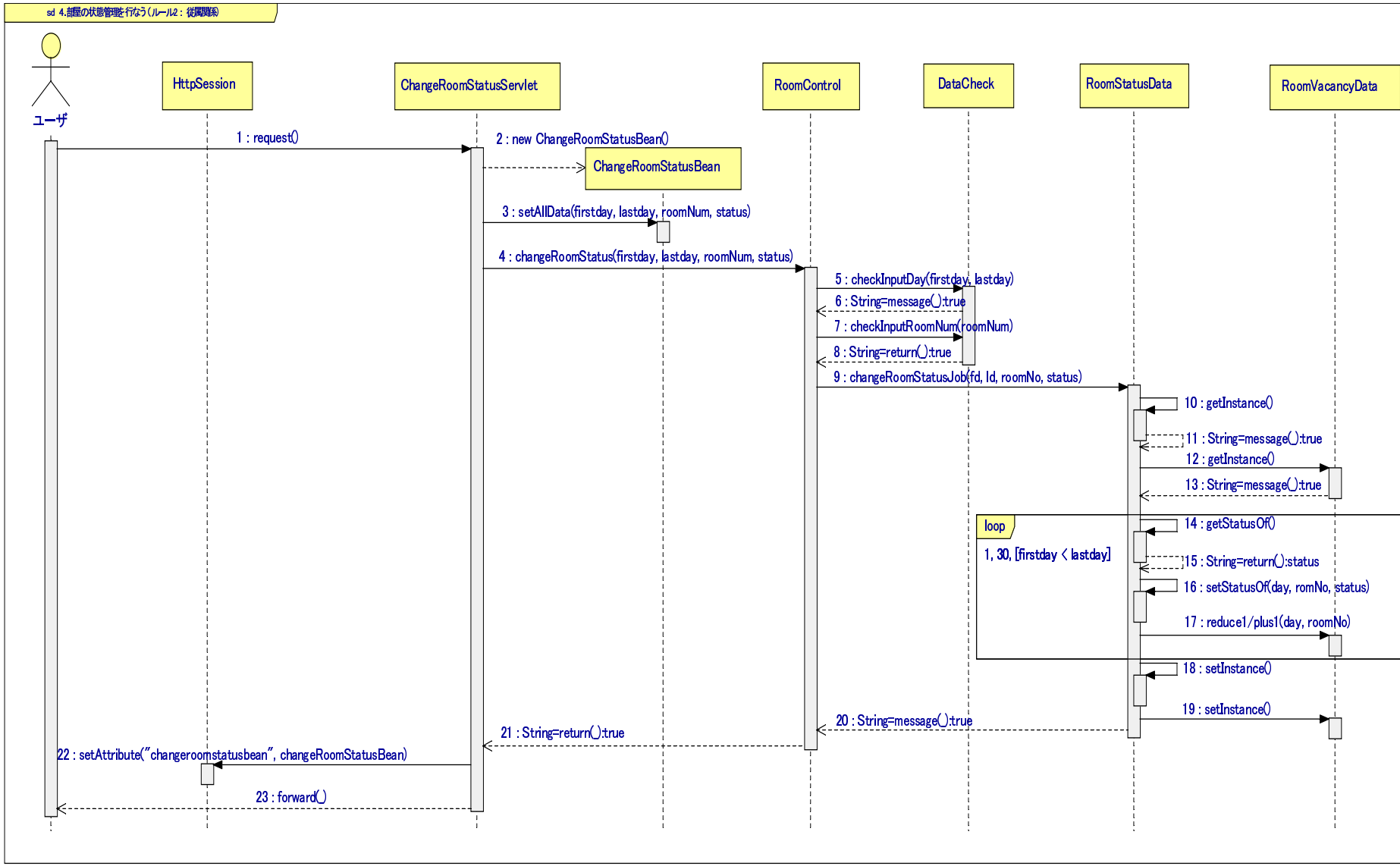

図 A.14: 客室の管理を行なう

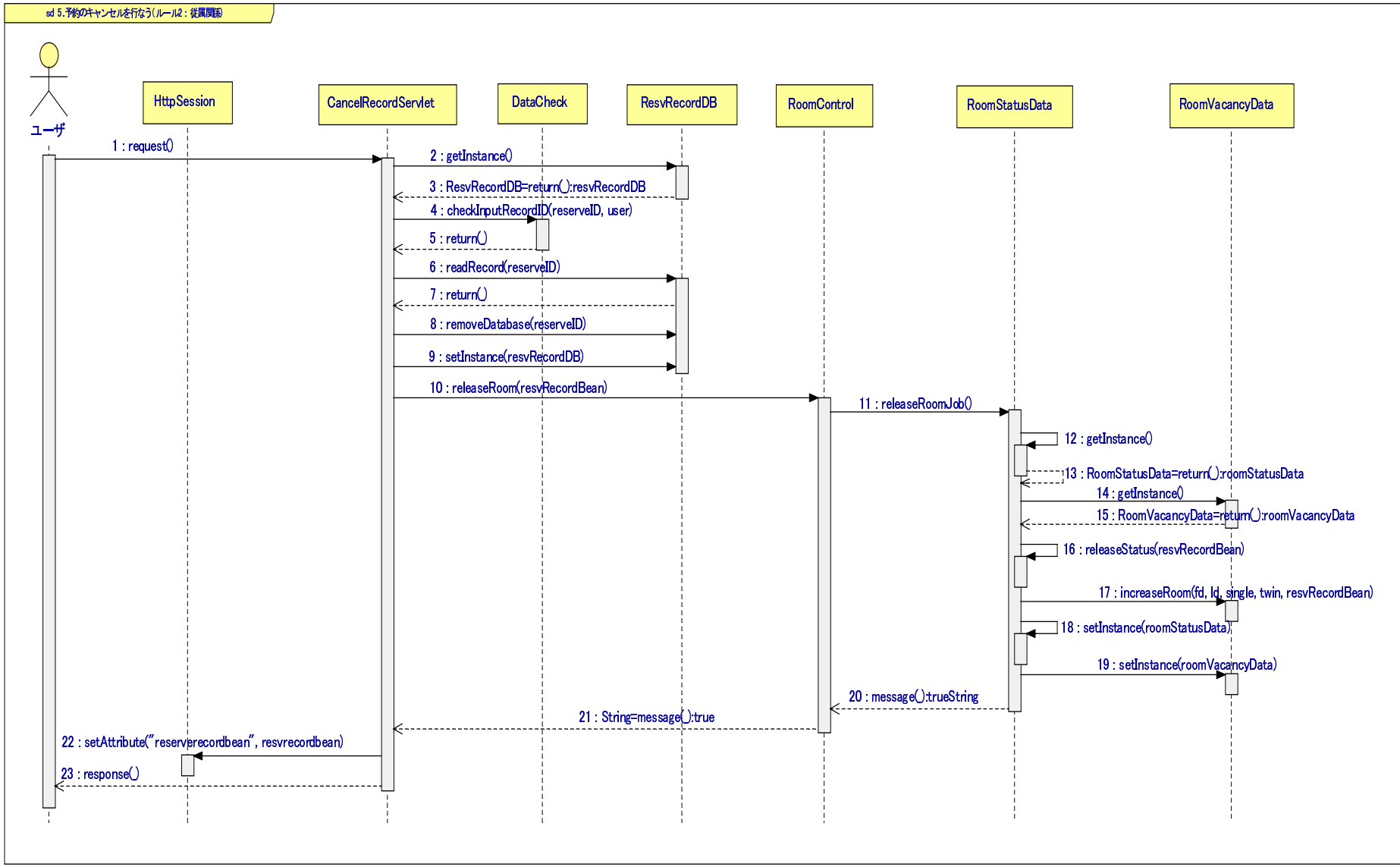

図 A.15: 予約をキャンセルする

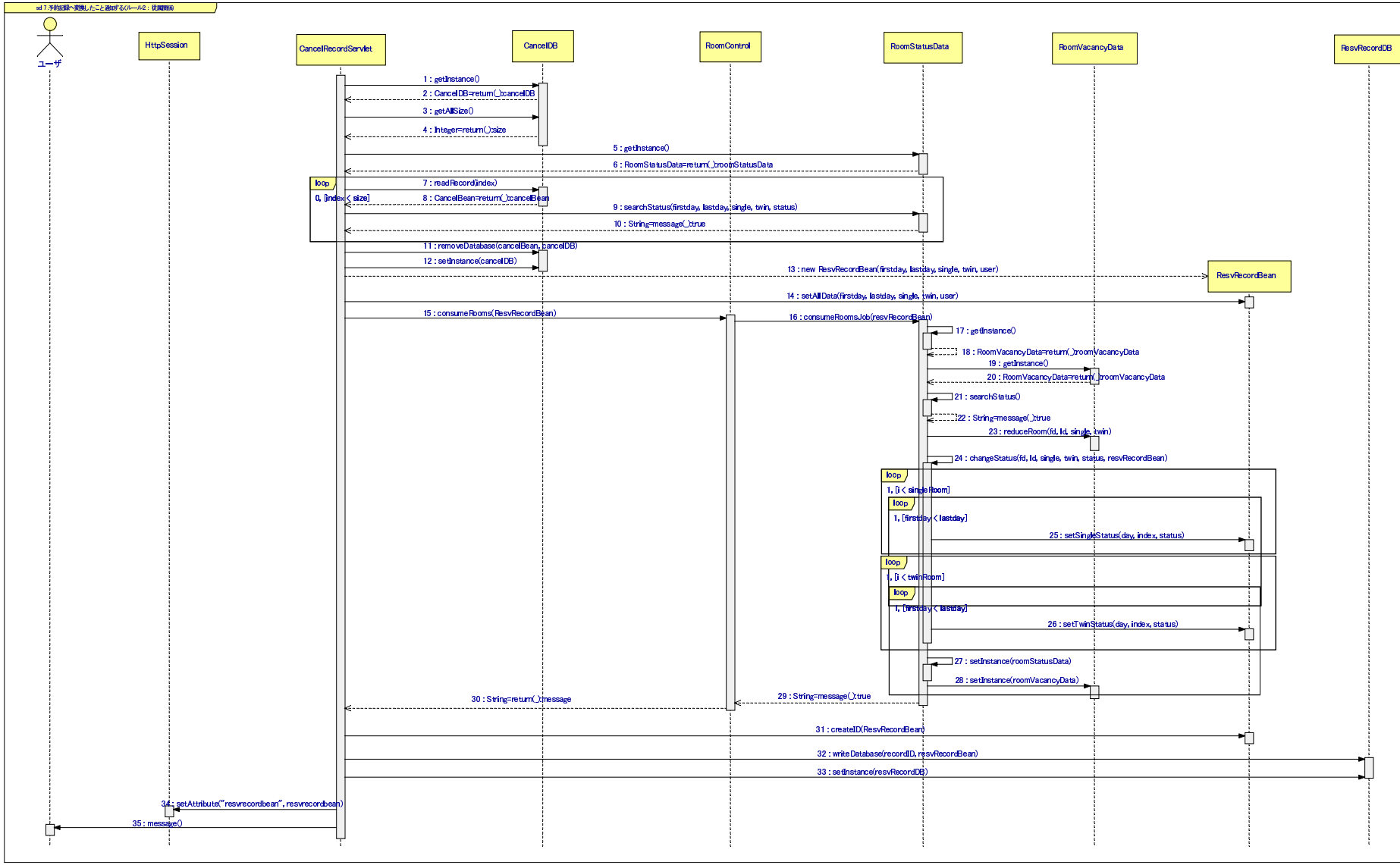

図 A.16: 予約記録へ変換する (R2 適用)

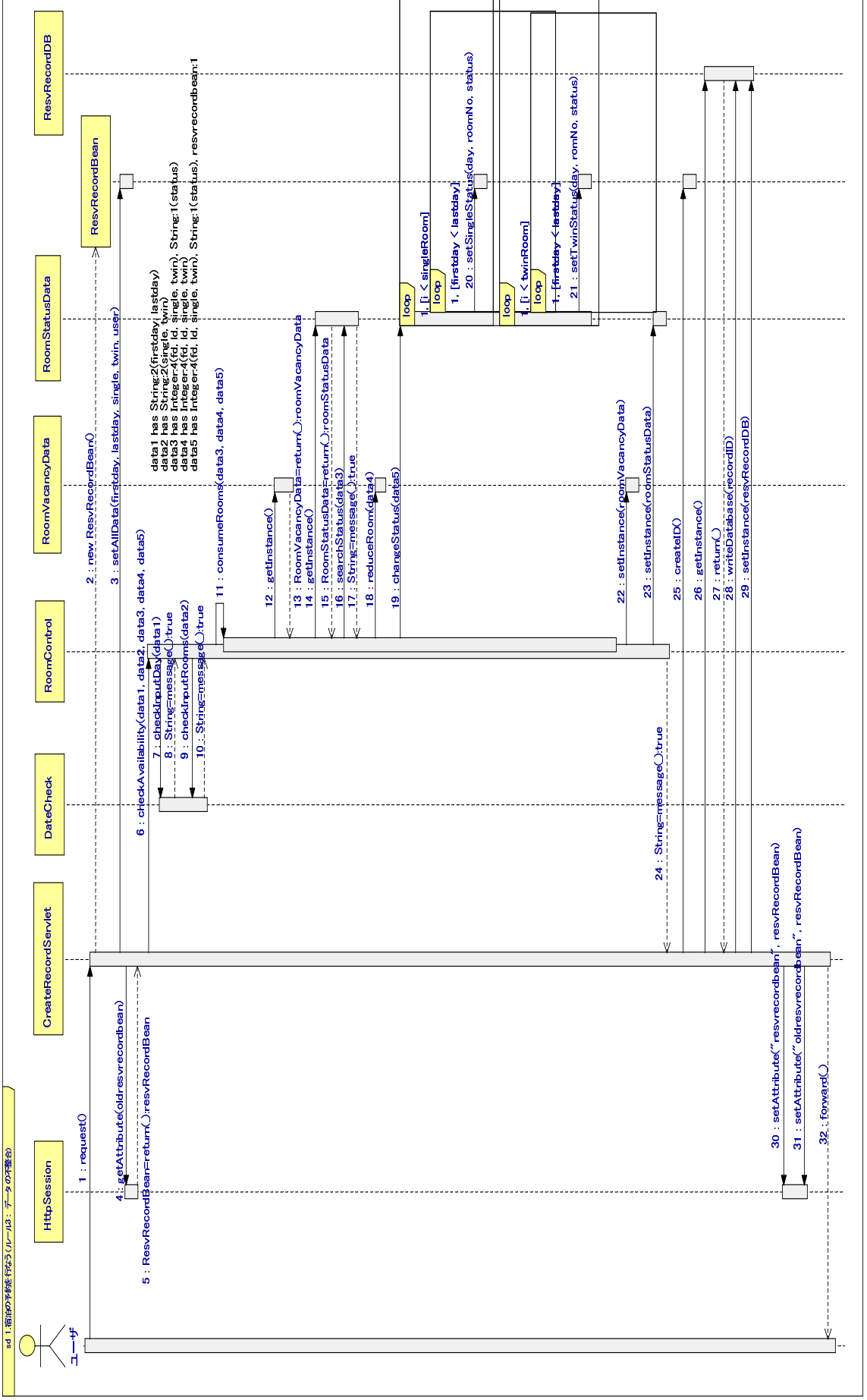

図 A.17: 予約記録作成 (R3 適用)

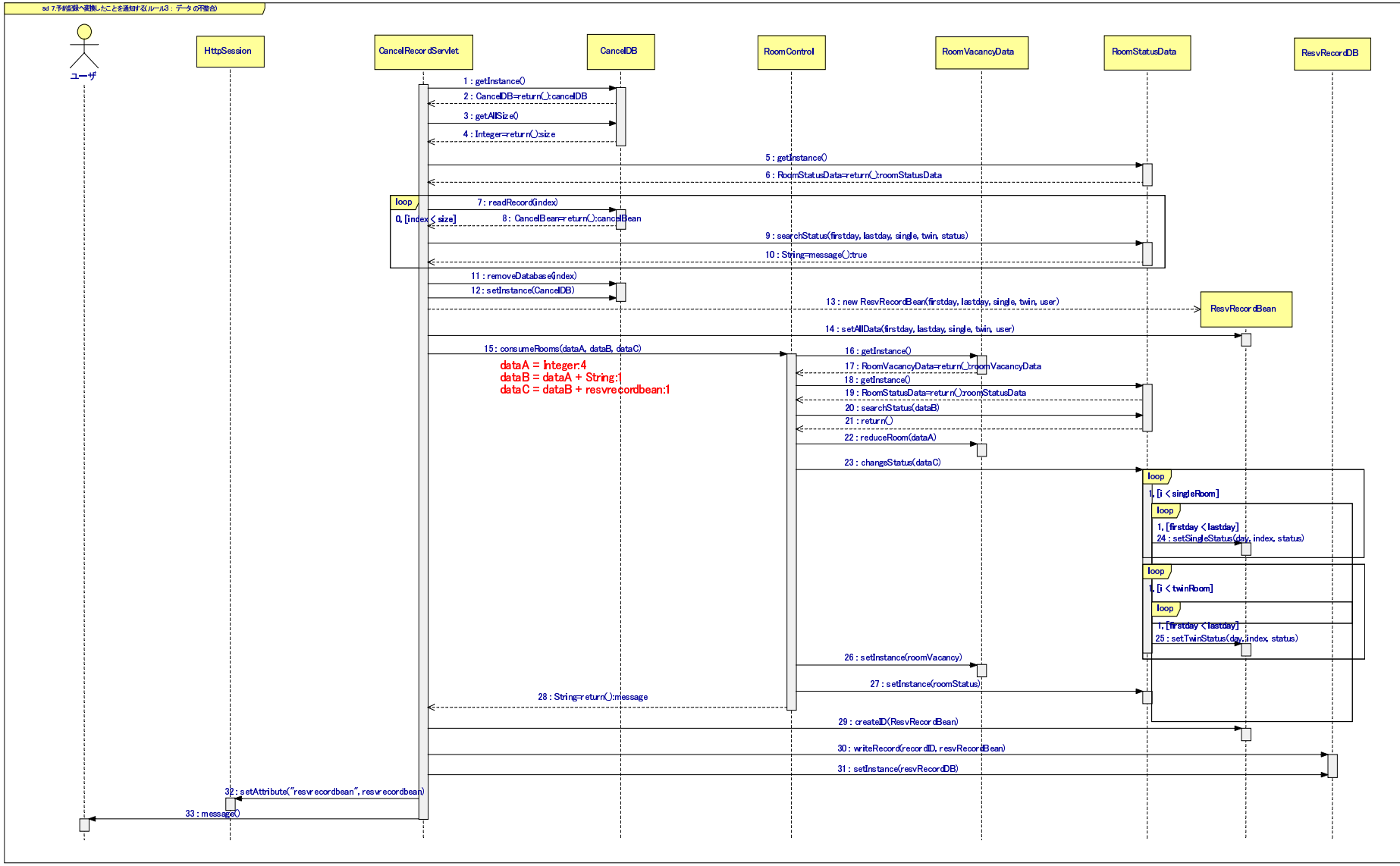

図 A.18: 予約記録へ変換する (R3 適用)

参考文献

[1] マーチン・ファウラー，リファクタリング　プログラミングの体質改善テクニック，ピアソン・エデュケーション，2000.

[2] グラディ・ブーチ，UML ユーザーガイド，ピアソンエデュケーション，　1999.

[3] 株式会社テクノロジックアート，UML2 ハンドブック，翔泳社，　2004.

[4] F. ブッシュマン，R. ムニエ，H. ローネスト，P. 損目ラード，M. スタル，ソフトウェアアーキテクチャ　ソフトウェア開発のためのパターン体系，凸版，1999.

[5] ペルディタ・スティーブン，ロブ・プーリー，オブジェクト指向とコンポーネントによるソフトウェア工学 -UML を使って-，ピアソンエデュケーション，2000.

[6] 木下　是理科系の作文技術，中公新書，1981.

[7] 結城　浩 Java 言語で学ぶデザインパターン入門，ソフトバンク，　パブリッシング，2001.