i219 ソフトウェア設計論 期末レポート

s2425004 成田晴香

対象システム概要

分注機を取り付けたロボットアームを用いて、試薬を混合し新しい試薬を作製する「自動分注システム」をモデリングの対象とした（Figure 1）。本システムは一次元座標上に、分注機を取り付けたロボットアームと 3 つの容器を並べている。ロボットアームは座標 0 にあり、 3 つの容器がそれぞれ座標 10，20，30 にある。ロボットアームと分注機はそれぞれの通信方法でプログラムから制御されている。容器 1 、容器 2 に混ぜたい試薬を予め入れておき、容器 3 は空とする。試薬は試薬データベースに登録されているものを使用する。ロボットアームで分注機を移動させ、容器 1 と容器 2 の内容物を容器 3 に移動し、容器 3 の内容物を新しい試薬とし、試薬データベースに登録する。消費した試薬は試薬データベースから消去する。また、キーボード入力により、本システムの動作中に危険があったときのために動作を停止、再開ができるものとする。本課題におけるロボットアームや分注機は疑似的に実装したものであり実体は持たない。

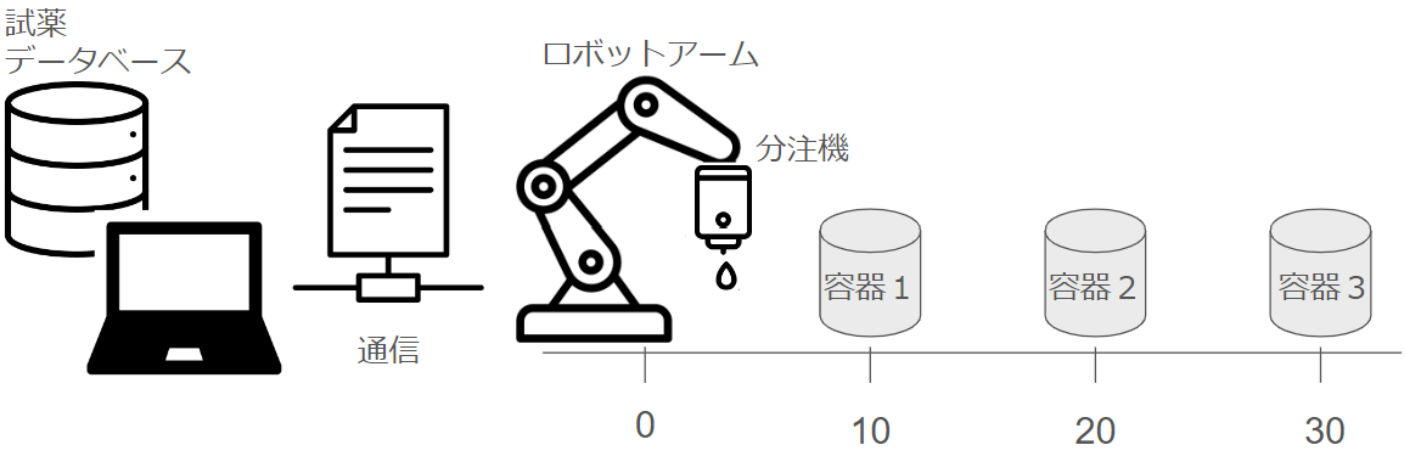


Figure 1. 自動分注システム

クラス図とその説明

クラス図

対象システムを AutoDispenserSystem クラスとした。AutoDispenserSystem クラスは、試薬データベース (ReagentDatabase クラス)、分注機（Dispenser クラス）、ロボットアーム（RobotArm クラス）、容器（Container クラス）の集約から成る (Figure 2)。

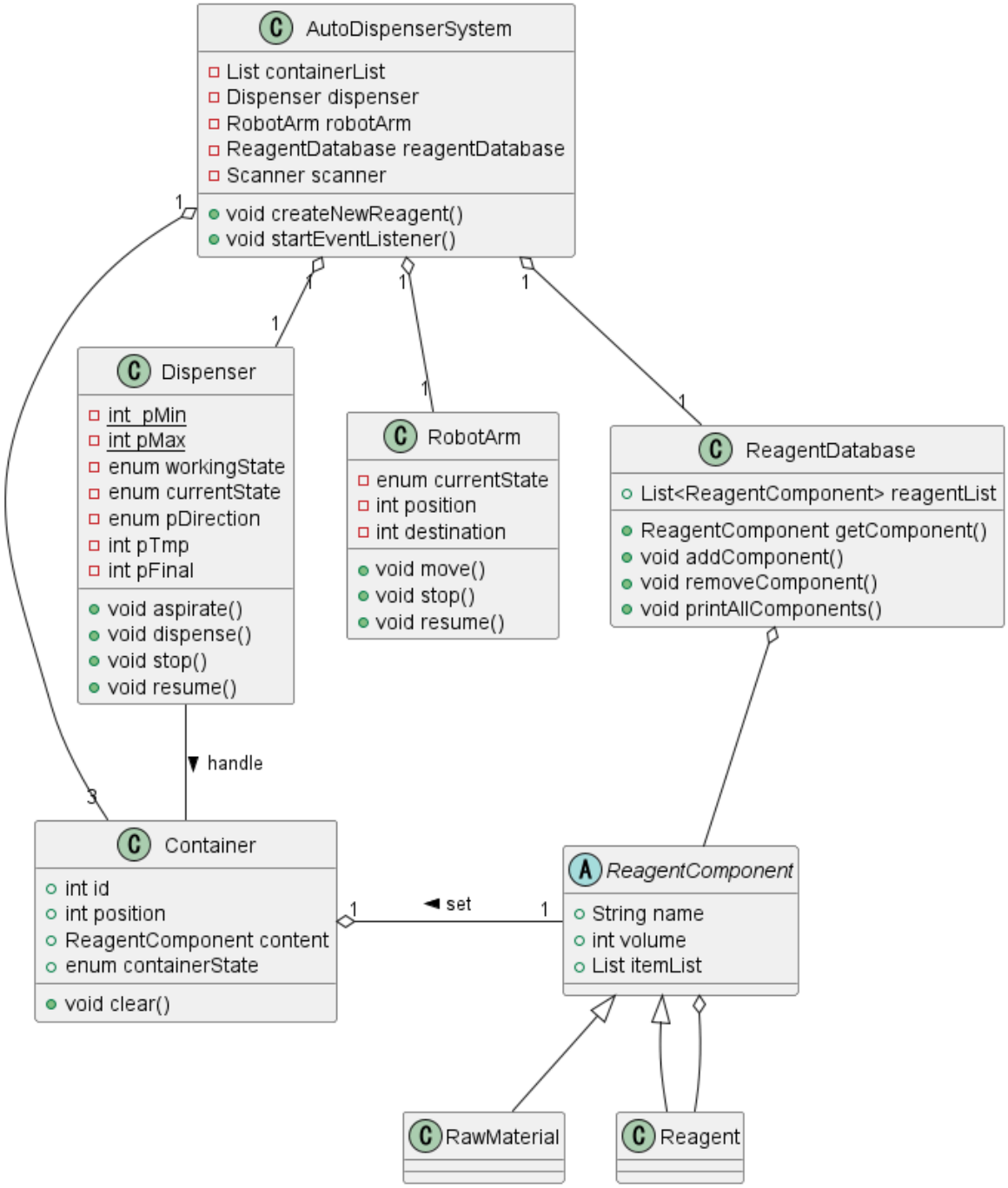


Figure 2.　自動分注システムのクラス図

AutoDispenser クラス

システム全体を指す。新しい試薬を作製するための createNewReagent メソッドを持つ。また、ロボットアームや分注機の作動中における動作の停止、再開をキーボード入力で受け付けるためのイベントリスナーを開始する startEventListener メソッドを持つ。

Dispenser クラス

分注機の動作を制御するクラス。分注機は内部のプランジャーの位置を変化させることで吸引・吐出を行う。

- プランジャー位置の最低値、最高値が決まっておりそれを超える動きはできない。最低値と最高値を pMin, pMax というクラス変数とした。また、吸引時にはプランジャーを上げ、吐出時には下げるのでその方向を変数 pDirection とした。吸引・吐出したい液量をプランジャー位置の変化量に換算し、最終的なプランジャー位置を pFinal とした。また、現在のプランジャー位置を pTmp とした。
- 分注機の状態を Idle と Working の二つに大別し、変数 currentState とした。さらに、Working のとき、プランジャーの移動方向に応じて Aspirating、Dispensing という状態を取りうる workingState という変数を用意した。
- 本クラスは容器の内容物を取り扱うためのメソッドを持つ(handle)。吸引を aspirate メソッド、吐出を dispense メソッド、一時停止を stop メソッド、再開を resume メソッドにて実装した。

Container クラス

容器の状態を制御するクラス。本システムでは 3 つのインスタンスを生成する。

- ID、座標、内容物を変数 id, position, content とした。content には ReagentComponent クラスのインスタンスをひとつ格納する(set)。また、分注機によって試薬の吸引・吐出がされているかを示す containerState 変数を持つ。

RobotArm クラス

ロボットアームの動作を制御するクラス。

- 現在地、目的地をそれぞれ 変数 position、 destination とした。
- 状態として Idle と Moving を持つ。
- 目的地までの移動を move メソッド、一時停止を stop メソッド、再開を resume メソッドにて実装した。

ReagentDatabase クラス

試薬データベースのクラス。

- 登録されている試薬が格納されたリスト itemList を変数に持つ。itemList の要素は ReagentComponent クラスのインスタンスである。
- itemList からインデックスをもと試薬を返す getComponent メソッド、試薬をデータベースに追加する addComponent メソッド、試薬をデータベースから消去する removeComponent メソッド、試薬一覧を表示する printAllComponents メソッドを持つ。

ReagentComponent クラス、Reagent クラス、RawMaterial クラス

試薬同士を混ぜて新しい試薬とすることから、試薬をコンポジットパターンで表現した。ReagentComponent クラスを抽象クラス、RawMaterial クラスを Leaf、Reagent クラスを Composite とした。

- 試薬名、液量、内容物リストをそれぞれ name, volume, itemList という変数に持つ。

オブジェクト図

ここで、試薬データベースには初期値として reagent1, reagent2, reagent3, reagent4 の 4 つの試薬が登録されているものとする。reagent2, reagent3 をそれぞれ容器 1, 容器 2 に入れる。この初期状態をオブジェクト図に示した（Figure 3）。また、自動分注システムの稼働により新しい試薬を容器 3 に作製した結果の最終状態のオブジェクト図を Figure 4 に示す。

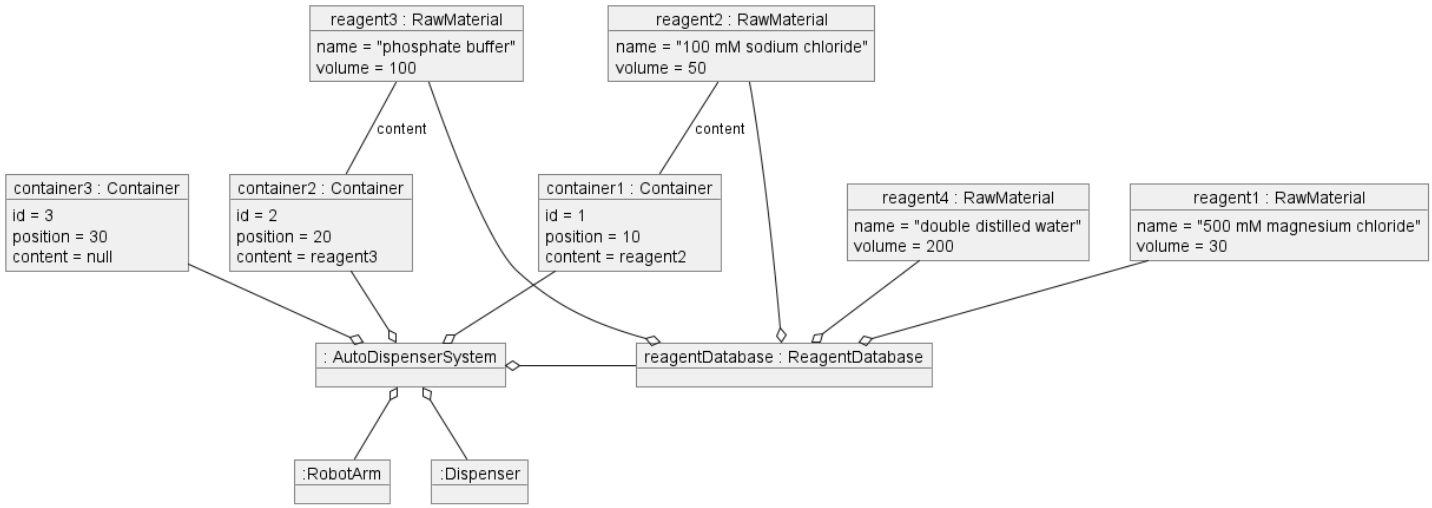


Figure 3. 初期状態のオブジェクト図

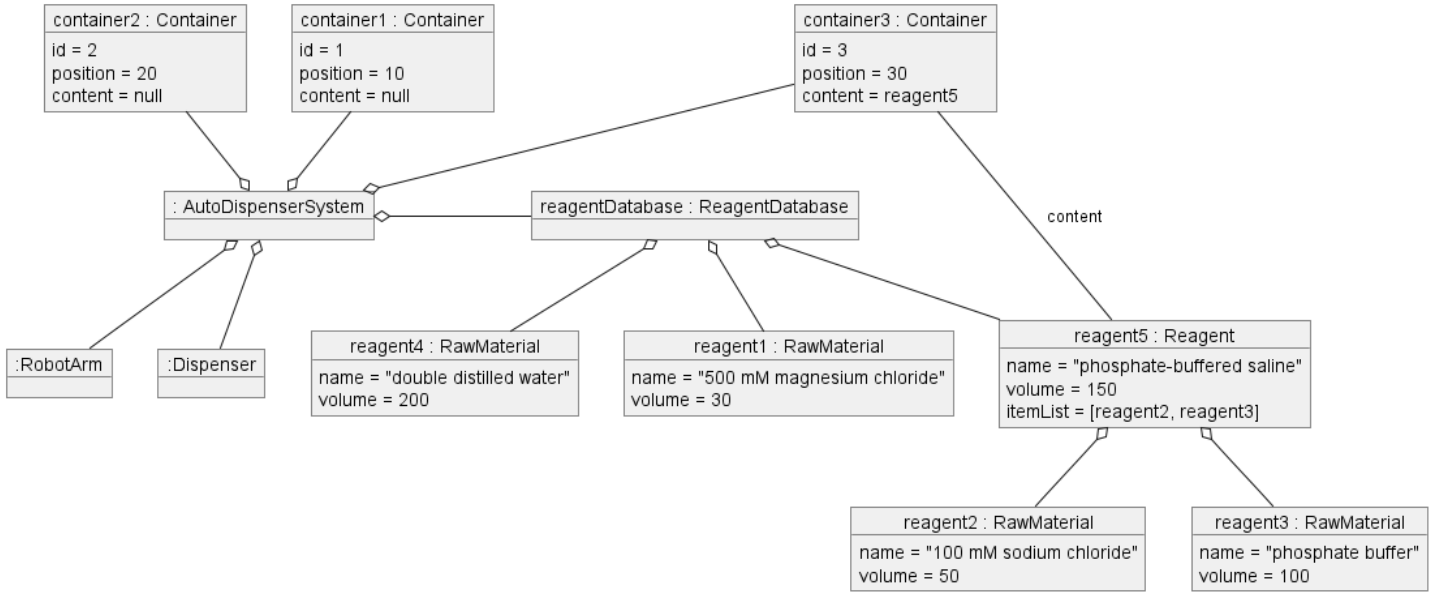


Figure 4. 最終状態のオブジェクト図

ステートマシン図とその説明

ステートマシン図

1. ロボットアームの状態遷移

まず、状態を待機状態である Idle、移動中である Moving とした (Figure 5)。また、途中停止、再開という挙動を取り入れるため、待機状態とは別に目的に到達したことを示す Finished を用意した。初期状態からはまず Idle に入る。move メソッドが呼び出されると Moving に遷移する。目的地が現在地より遠い場合、position をインクリメントしその逆ではデクリメントすることで移動する。目的地に達すると Finished に遷移し sendFinishedSignal メソッドを実行、目的地の初期化を行った後そのまま Idle に戻る。目的地に到達するまでに stop メソッドが呼び出されると、目的地の座標を保持したまま Idle に遷移し、resume メソッドが呼び出されるのを待つ。resume メソッドが呼び出されると、再び Moving に遷移し、再開する。

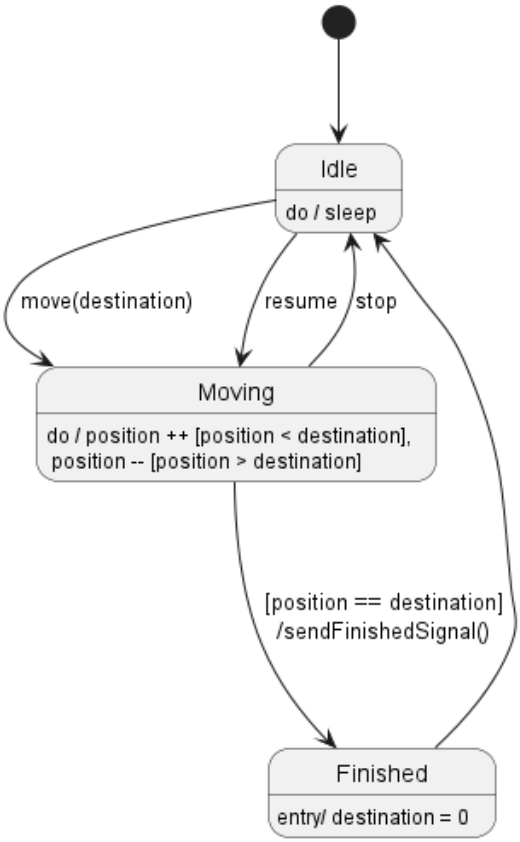


Figure 5.ロボットアームのステートマシン図

1.1 Java で実装したプログラムの抜粋とその説明

move メソッド, stop メソッドの実装を以下に示す。

```
public void move(int newDestination) {
        destination = newDestination;
        currentState = State.MOVING;
        System.out.println("Moving to destination: " + destination);
        update();
    }

public void stop() {
        if (currentState == State.MOVING) {
            currentState = State.IDLE;
            System.out.println("Stopped at position: " + position);
        }
    }
```

move メソッドでは currentState を Moving にした後、一時停止や再開などのイベントに対応するため update メソッドを用意した。ここでステートマシン図に記述した状態遷移を行うため、currentState の条件分岐をしている。

```java
public void update() {
        while (true) {
            switch (currentState) {
                case IDLE:
                    // 待機する
                    try {
                        Thread.sleep(1000);
                    } catch (InterruptedException e) {
                        e.printStackTrace();
                    }
                    break;
                case MOVING:
                    System.out.println("Current position: " + position);
                    // 目的地が現在地より遠いとき
                    if (position < destination) {
                        position++;
                    // 目的地が現在地より近いとき
                    } else if (position > destination) {
                        position--;
                    }
                    // 疑似的にロボットアームの動きを再現するためのスリープ
                    try {
                        Thread.sleep(700);
                    } catch (InterruptedException e) {
                        e.printStackTrace();
                    }
                    // 目的地に到達したとき
                    if (position == destination) {
                        currentState = State.FINISHED;
                        System.out.println("Reached destination: " + position);
                    }
                    break;
                case FINISHED:
                    // 目的地を初期化してIdleに戻る
                    sendFinishedSignal();
                    currentState = State.IDLE;
                    destination = 0;
                    return;
            }
        }
    }
```

1.2 一次停止、再開のイベントにおけるシーケンス図

1.1 の条件分岐を考慮して、1.移動開始、2.キーボード入力による一時停止、3. 待機、4. キーボード入力による再開、5. 目的地到達の流れをシーケンス図で表現した (Figure 6)。

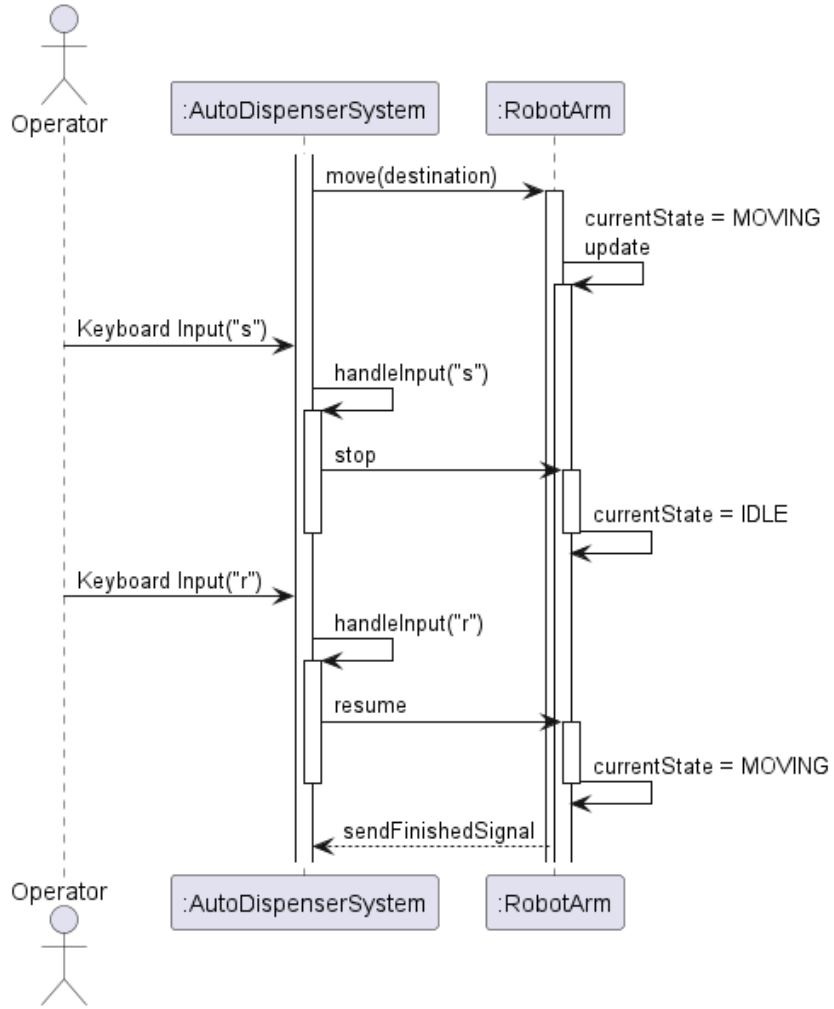


Figure 6. 一時停止、再開イベント発生におけるロボットアームのシーケンス図

2. 分注機の状態遷移

まず、ロボットアームと同様に状態を Idle、Working、Finished の 3 つに大別した(Figure 7)。また Working を吸引中、吐出中に対応する Aspirating、Dispensing に分類した。まず、aspirate メソッドが呼び出されると、Working の中の Aspirating に遷移する。このとき、プランジャーの位置が取れる範囲には制限があるため、pTmp が pMax を超えていないという条件を付ける。pMax を超えないように pFinal を吸引液量から計算し、そこに到達するまで pTmp をインクリメントする。dispensing メソッドは全てその逆を行う。吸引・吐出中に stop メソッドが呼び出されると、その時の履歴を保存し、Idle に遷移し、待機する。待機中に resume メソッドが呼び出されると、保存していた履歴状態に再び戻る。ロボットアームと同様に pTmp が pFinal と同じになるまで移動すると、Finished に遷移、sendFinishedSignal メソッドを実行し、pFinal や pDirection の初期化を行ったあと Idle に戻る。

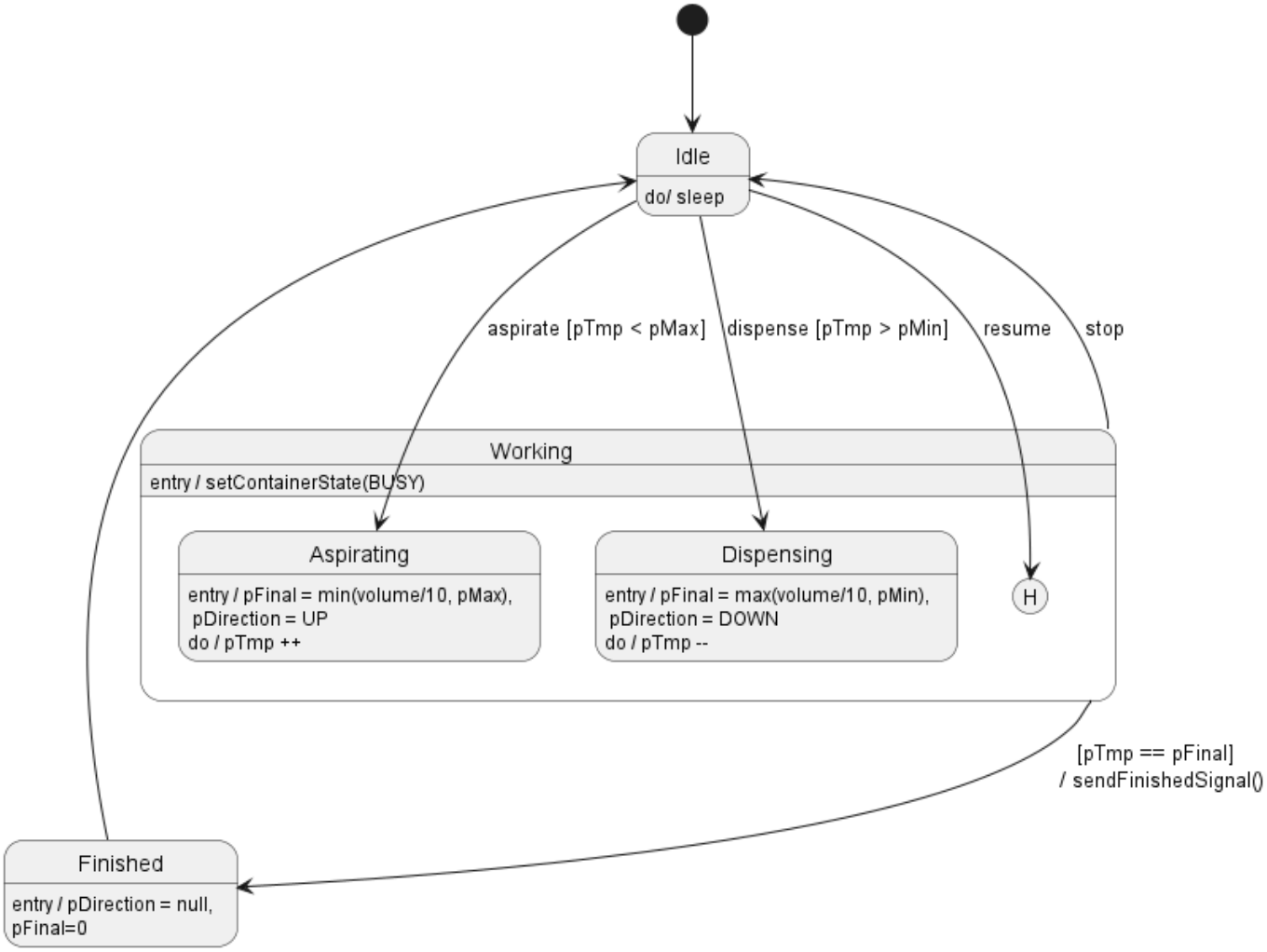


Figure 7. 分注機のステートマシン図

2.1 Java で実装したプログラムの抜粋とその説明

aspirate メソッド、stop メソッドを以下に示す。説明はコメントアウトの通り。

```
public void aspirate(int volume, Container container) {
    // 吸引できる状態はIdleに限る
    if (currentState == State.IDLE) {
        currentState = State.WORKING;
        workingState = WorkingState.ASPIRATING;
        container.setState(Container.ContainerState.BUSY);
        pDirection = plungerDirection.UP;
        // 現在のプランジャー位置に吸引液量をプランジャー位置変化量に換算した値と
        // pMaxを比較し、小さい方をpFinalとする
        pFinal = Math.min(pTemp + getPlungerDelta(volume), pMax);
        System.out.println("Aspirating volume: " + volume + " mL from container: " + container.position);
        update(container);
    }
}

public void stop() {
    // 一時停止できる状態はWorkingに限る
    if (currentState == State.WORKING) {
        currentState = State.IDLE;
        workingState = null;
        System.out.println("Stopped at pTemp: " + pTemp);
    }
}
```

ロボットアームと同様に一時停止、再開に対応すべく update メソッドを用意した。

```
public void update(Container container) {
    while (true) {
        switch (currentState) {
            case IDLE:
                // 待機する
                try {
                    Thread.sleep(700);
                } catch (InterruptedException e) {
                    e.printStackTrace();
                }
                break;
            case WORKING:
                System.out.println("Current plunger position: " + pTemp);
                // 吸引するとき
                if (workingState == WorkingState.ASPIRATING) {
                    pTemp++;
                    // 疑似的に分注機の動きを再現するためのスリープ
                    try {
                        Thread.sleep(1000);
                    } catch (InterruptedException e) {
                        e.printStackTrace();
                    }
                    if (pTemp == pFinal) {
                        finishWorking();
                    }
                // 吐出するとき
                } else if (workingState == WorkingState.DISPENSING) {
                    pTemp--;
                    // 疑似的に分注機の動きを再現するためのスリープ
                    try {
                        Thread.sleep(300);
                    } catch (InterruptedException e) {
                        e.printStackTrace();
                    }
                    if (pTemp == pFinal) {
                        finishWorking();
                    }
                }
                break;
            case FINISHED:
                // pFinalとpDirectionを初期化してIdleに戻る
                // 容器の状態をIdleに戻す
                pFinal = 0;
                pDirection = null;
                sendFinishedSignal();
                currentState = State.IDLE;
                container.setState(Container.ContainerState.IDLE);
                return;
        }

    }
}
```

2.2 一次停止、再開のイベントにおけるシーケンス図

2.1 の条件分岐を考慮して、1.吸引開始、2.キーボード入力により一時停止、3. 待機、4. 再開、5. 吸引の流れをシーケンス図で表現した (Figure 8)。

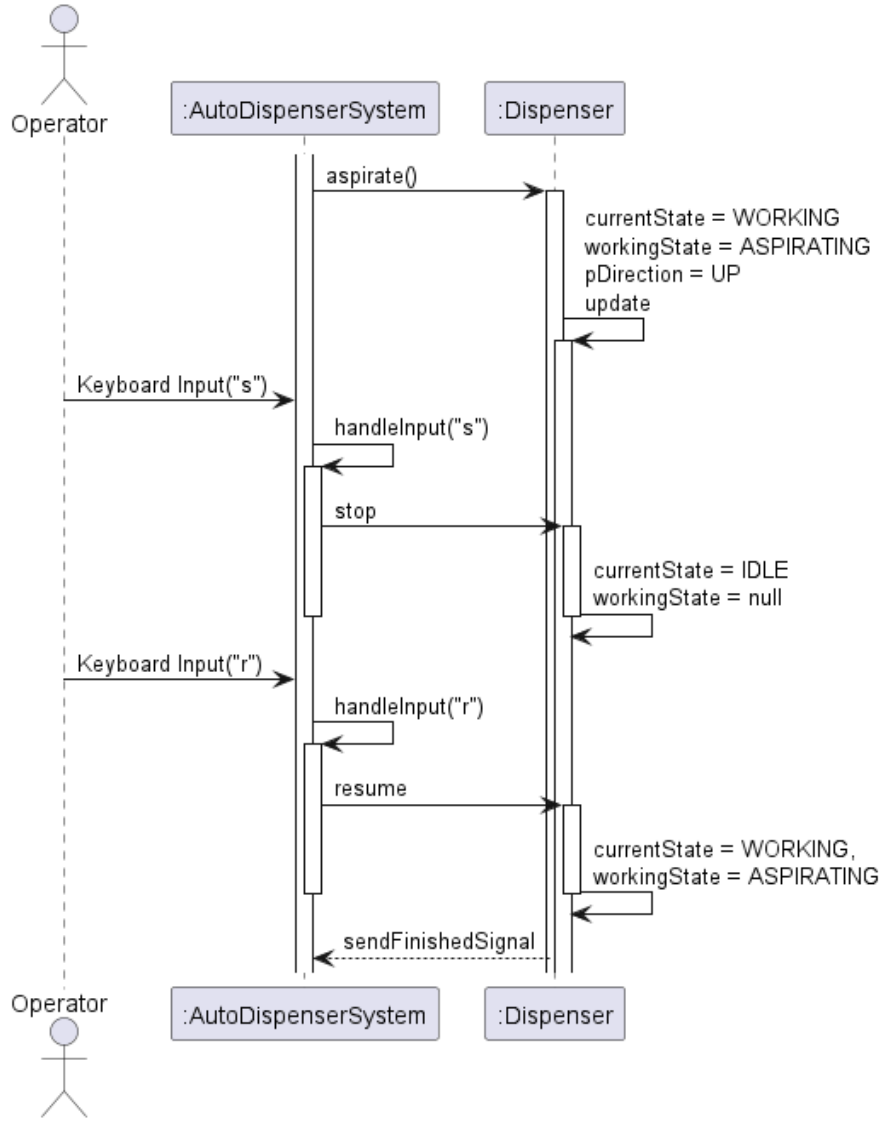


Figure 8. 一時停止、再開イベント発生における吸引中の分注機のシーケンス図

3. システム全体の流れ

システム全体を実行するには、Main.java を実行する。

3.1　データベース登録

4 種類の試薬をデータベースに登録する (Figure 9)。

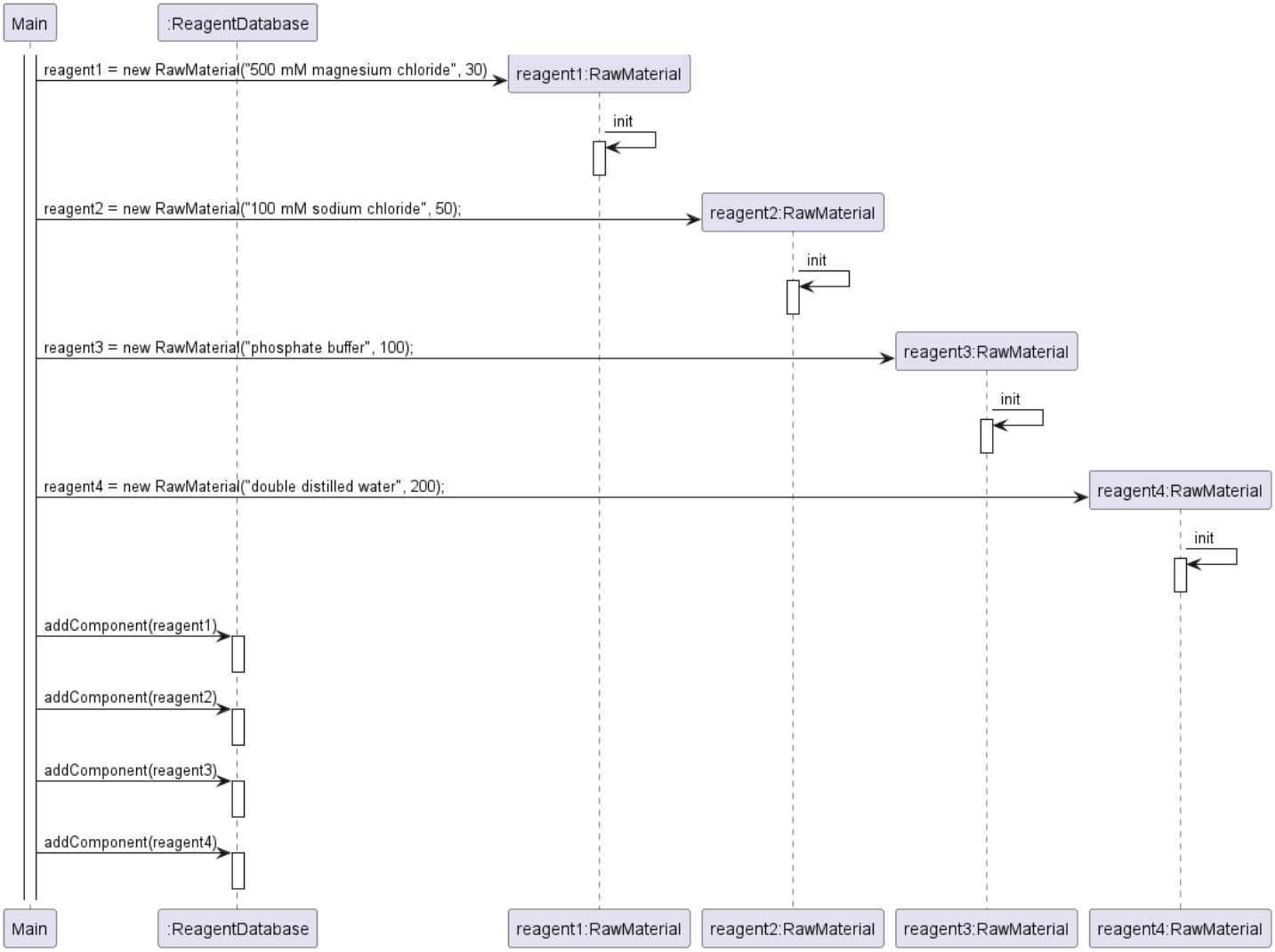
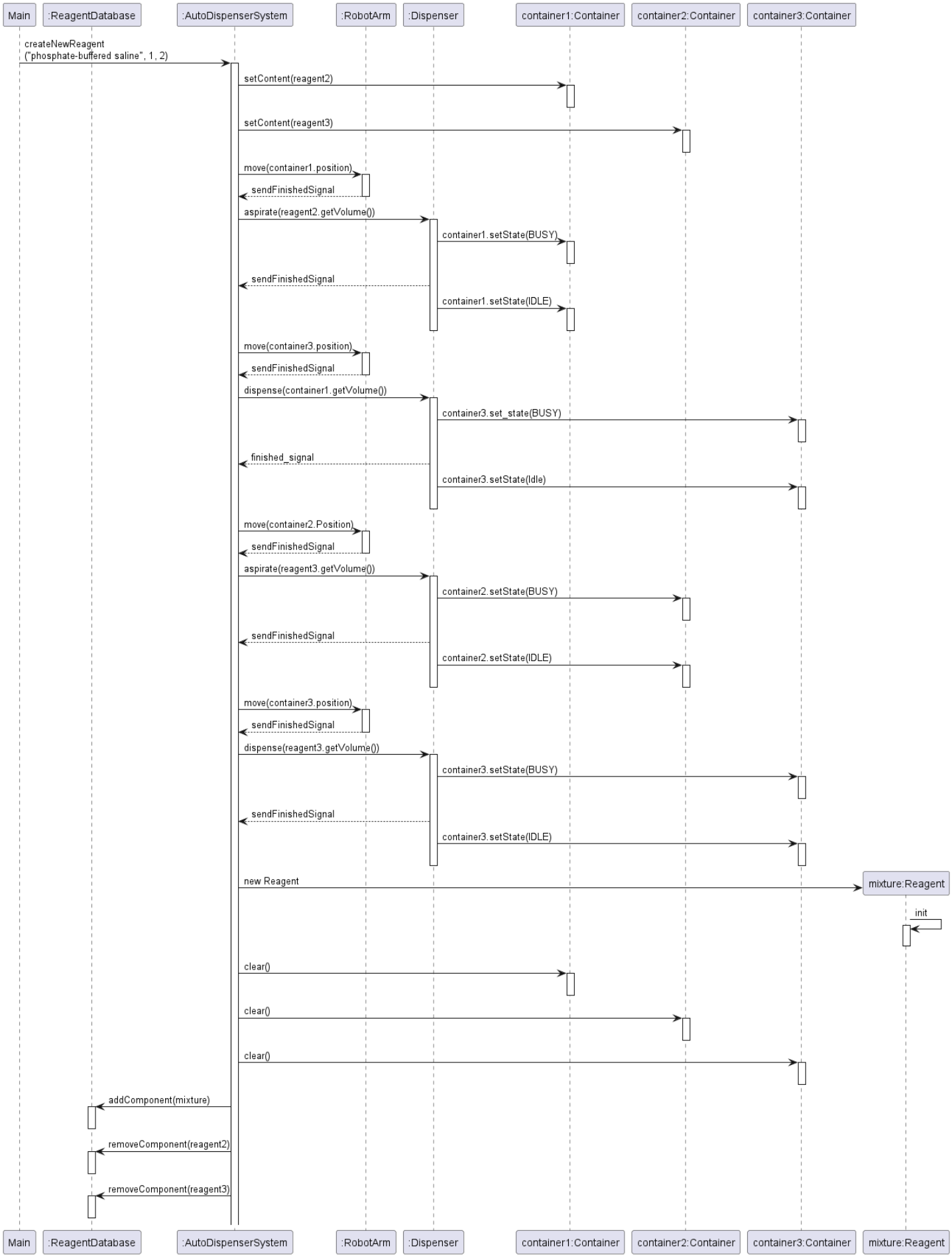


Figure 9. データベース登録のシーケンス図

3.2　新規試薬の作製

1. 初期状態ではロボットアームは position=0 にある。
2. そこから容器 1 まで移動し、分注機が容器 1 の内容物である reagent2 を吸引する。
3. 終わると、ロボットアームは容器 3 まで移動し、分注機が液量を全て吐出する。
4. つぎに、ロボットアームは容器 2 に移動し分注機がその内容物である reagent3 を吸引する。
5. 再びロボットアームは容器 3 に移動し、分注機が吐出する。
6. データベースを更新する。


Main
:ReagentDatabase
:AutoDispenserSystem
:RobotArm
:Dispenser
container1:Container
container2:Container
container3:Container
mixture:Reagent
createNewReagent
("phosphate-buffered saline", 1, 2)
setContent(reagent2)
setContent(reagent3)
move(container1.position)
sendFinishedSignal
aspirate(reagent2.getVolume())
container1.setState(BUSY)
sendFinishedSignal
container1.setState(IDLE)
move(container3.position)
sendFinishedSignal
dispense(container1.getVolume())
container3.set_state(BUSY)
finished_signal
container3.setState(Idle)
move(container2.Position)
sendFinishedSignal
aspirate(reagent3.getVolume())
container2.setState(BUSY)
sendFinishedSignal
container2.setState(IDLE)
move(container3.position)
sendFinishedSignal
dispense(reagent3.getVolume())
container3.setState(BUSY)
sendFinishedSignal
container3.setState(IDLE)
new Reagent
init
clear()
clear()
clear()
addComponent(mixture)
removeComponent(reagent2)
removeComponent(reagent3)

3.3 Java で実装したプログラムの抜粋とその説明

説明はコメントアウトの通り。

- Main.java の抜粋

```
public static void main(String args[]) {
        // 試薬をデータベースへ登録する
        ReagentDatabase database = new ReagentDatabase();

        ReagentComponent reagent1 = new RawMaterial("500 mM magnesium chloride", 30);
        // ... 続く

        database.addComponent(reagent1);
        // ... 続く

        Dispenser dispenser = new Dispenser();
        RobotArm robotArm = new RobotArm();

        AutoDispenserSystem autoDispenserSystem = new AutoDispenserSystem(dispenser, robotArm, database);
        // キーボード入力により一時停止、再開するためのイベントリスナーを開始
        autoDispenserSystem.startEventListener();

        // 新規試薬を作製する
        autoDispenserSystem.createNewReagent("phosphate-buffered saline", 1, 2);

        // ...
    }
```

- AutoDispenserSystem.java の抜粋

```
// 新規試薬の作製
public void createNewReagent(String newName, int index1, int index2) {
        // ...
        // 試薬を容器にセットし終わったところから
        // 容器1まで移動、内容物を吸引
        robotArm.move(container1.getPosition());
        dispenser.aspirate(reagent1.getVolume(), container1);

        // 容器3まで移動、内容物を吐出
        robotArm.move(container3.getPosition());
        dispenser.dispense(reagent1.getVolume(), container3);
        //  .... 続く
}

//  キーボード入力により一時停止、再開するためのイベントリスナーの開始
public void startEventListener() {
        new Thread(() -> {
            while (isRunning) {
                if (scanner.hasNextLine()) {
                    String input = scanner.nextLine().trim().toLowerCase();
                    handleInput(input);
                }
            }
        }).start();
    }

// キーボード入力が"s"のとき、ロボットアームあるいは分注機を一時停止、"r"の時、再開する。
private void handleInput(String input) {
    System.out.println("Received command: " + input);
    switch (input) {
        case "s":
            robotArm.stop();
            dispenser.stop();
            break;
        case "r":
            robotArm.resume();
            dispenser.resume();
            break;
        default:
            System.out.println("Unknown command. Available commands: s and r");
    }
}
```

実装を利用してオブジェクトを実体化し，動作させた結果の報告

四種類の試薬をデータベースに登録し、新規試薬を作製、データベースへの登録を実行した。初期状態と最終状態のデータベースの内容を表示し、新規試薬の組成を表示した。また、ロボットアームの現在地、分注機のプランジャーの現在位置を System.out.println で表示した。ロボットアームが初期位置から容器 1 へ移動する途中、s をキーボード入力し一時停止、r で再開した。また、分注機が容器 2 の内容物を吸引する途中、s をキーボード入力し一時停止、r で再開した。

```
Initial reagents in the database:
500 mM magnesium chloride
100 mM sodium chloride
phosphate buffer
double distilled water
----------------------------
Starting dispensing process...
Moving to destination: 10
Current position: 0
Current position: 1
Current position: 2
Current position: 3
Current position: 4
Current position: 5
s
Received command: s
Stopped at position: 6
r
Received command: r
Resuming movement to destination: 10
Current position: 6
Current position: 7
Current position: 8
Current position: 9
Reached destination: 10
Sending reached signal
Reached destination: 10
Sending reached signal
```

```
----------------------------
Aspirating 50 mL of 100 mM sodium chloride from container 1
Aspirating volume: 50 mL from container: 10
Current plunger position: 0
Current plunger position: 1
Current plunger position: 2
Current plunger position: 3
Current plunger position: 4
Finished
Sending finished signal
----------------------------
Moving to destination: 30
Current position: 10
Current position: 11
Current position: 12
Current position: 13
Current position: 14
Current position: 15
Current position: 16
Current position: 17
Current position: 18
Current position: 19
Current position: 20
Current position: 21
Current position: 22
Current position: 23
Current position: 24
Current position: 25
Current position: 26
Current position: 27
Current position: 28
Current position: 29
Reached destination: 30
Sending reached signal
----------------------------
Dispensing 50 mL into container 3
Dispensing volume: 50 mL to container: 30
Current plunger position: 5
Current plunger position: 4
Current plunger position: 3
Current plunger position: 2
Current plunger position: 1
Finished
Sending finished signal
----------------------------
Moving to destination: 20
Current position: 30
Current position: 29
Current position: 28
Current position: 27
Current position: 26
Current position: 25
Current position: 24
Current position: 23
```

```
Current position: 22
Current position: 21
Reached destination: 20
Sending reached signal
----------------------------
Aspirating 100 mL of phosphate buffer from container 2
Aspirating volume: 100 mL from container: 20
Current plunger position: 0
Current plunger position: 1
Current plunger position: 2
Current plunger position: 3
Current plunger position: 4
s
Received command: s
Stopped at pTemp: 5
r
Received command: r
Resuming ASPIRATING
Current plunger position: 5
Current plunger position: 6
Current plunger position: 7
Current plunger position: 8
Current plunger position: 9
Finished
Sending finished signal
----------------------------
Moving to destination: 30
Current position: 20
Current position: 21
Current position: 22
Current position: 23
Current position: 24
Current position: 25
Current position: 26
Current position: 27
Current position: 28
Current position: 29
Reached destination: 30
Sending reached signal
----------------------------
Dispensing 100 mL into container 3
Dispensing volume: 100 mL to container: 30
Current plunger position: 10
Current plunger position: 9
Current plunger position: 8
Current plunger position: 7
Current plunger position: 6
Current plunger position: 5
Current plunger position: 4
Current plunger position: 3
Current plunger position: 2
Current plunger position: 1
Finished
Sending finished signal
----------------------------
```

```
Returning to home position
Moving to destination: 0
Current position: 30
Current position: 29
Current position: 28
Current position: 27
Current position: 26
Current position: 25
Current position: 24
Current position: 23
Current position: 22
Current position: 21
Current position: 20
Current position: 19
Current position: 18
Current position: 17
Current position: 16
Current position: 15
Current position: 14
Current position: 13
Current position: 12
Current position: 11
Current position: 10
Current position: 9
Current position: 8
Current position: 7
Current position: 6
Current position: 5
Current position: 4
Current position: 3
Current position: 2
Current position: 1
Reached destination: 0
Sending reached signal
------------------------------
------------------------------
New reagent phosphate-buffered saline created with volume 150
------------------------------
Updated components in the database:
500 mM magnesium chloride
double distilled water
phosphate-buffered saline
------------------------------
Reagent: phosphate-buffered saline (Total Volume: 150)
50 mL of 100 mM sodium chloride
100 mL of phosphate buffer
```

静的モデリング・動的モデリングに関する考察，及び，自分の考え

静的モデリングについて

- 本システムでは AutoDispenserSystem クラスが Container クラス、Dispenser クラス、RobotArm クラス、ReagentDatabase の集約から成る構造とした。ここで、Container クラスは溶液の実体とシステムの実装の境界の役割を持つ。また、Dispenser クラスが Container クラスに操作するという関連は Dispenser クラスが Container クラスの containerState を変化させることで表現した。後になって気づいたが、吸引し終わったときに content を null にしたり吐出し終わったときに content をセットしたりすればこの変数は必要なかった。
- 試薬同士を混ぜて新しい試薬を作製することから、デザインパターンのうちコンポジットパターンを使用した。それによって、組成が一種類しかない RawMaterial といくつかの試薬を混合した Reagent を同列に扱うことができた。

動的モデリングについて

- 本システムではロボットアームと分注機の状態をモデリングした。どちらも主に Idle, Working, Finished の 3 状態から成ると考えた。分注機では、Aspirating と Dispensing をまとめた Working とすることで、stop や resume メソッドの呼び出しにおける挙動を、ロボットアームと同一にすることができ、コードの再利用性が向上した。実装では、状態に応じた条件分岐により更新する update メソッドを用意したが、ステートマシン図では、その条件分岐の中身を記述し、それが update メソッドに書かれているということはわからない。update 関数によって条件分岐による挙動をまとめて実装できたものの、ステートマシン図だけからは読み取れないような実装にしてしまったという違和感がある。
- クラス図、ステートマシン図、シーケンス図を全て用意したあと、どこから実装するのがよいか迷った。また、実装しているときにモデリングの間違いに気づくことがありソフトウェア設計における手戻りコストの重要性を理解した。