RWTH RHEINISCH-WESTFÄLISCHE TECHNISCHE HOCHSCHULE AACHEN

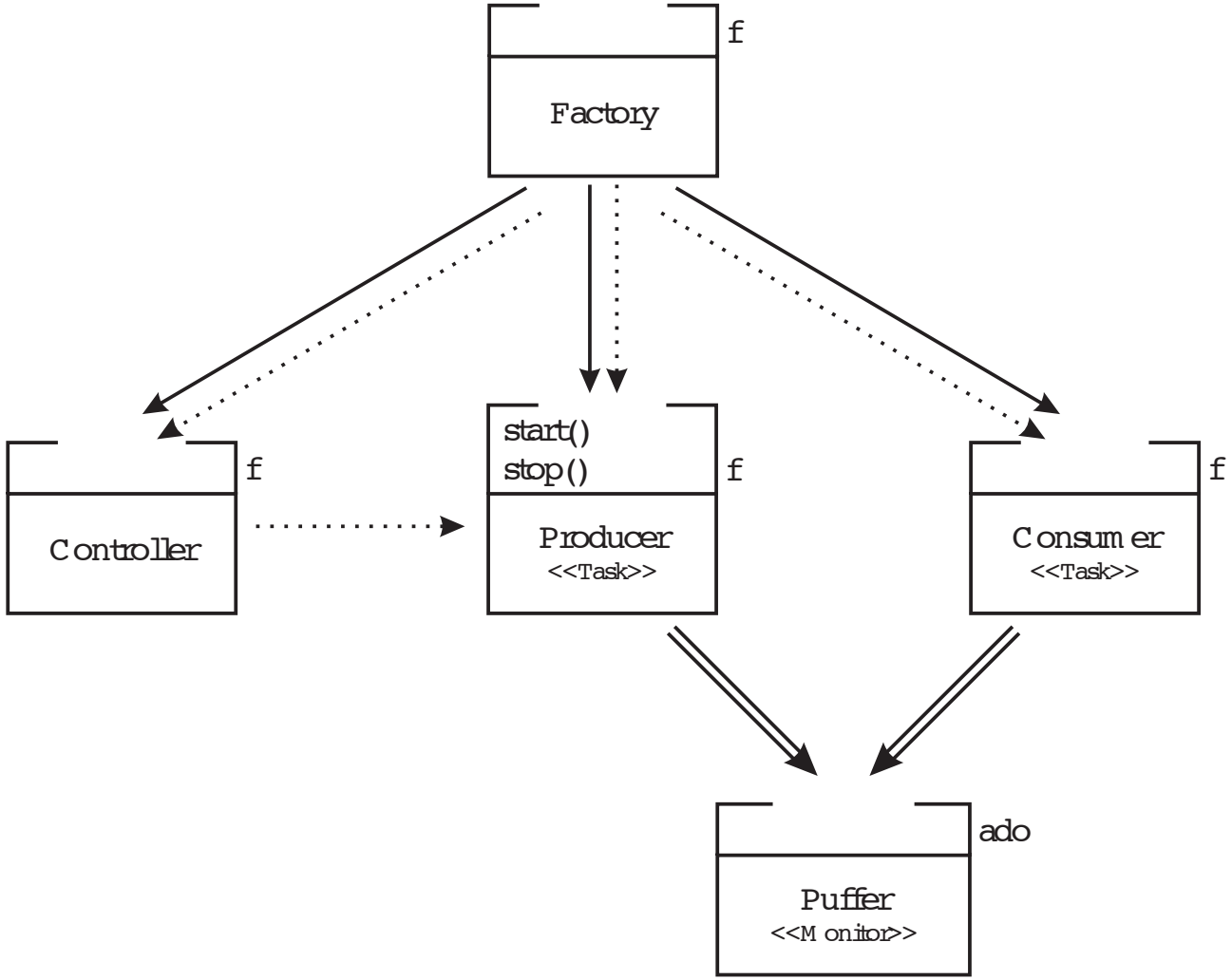
9. Übungsblatt

Umsetzung nach Java
Algebraische Spezifikation

6. Juli 2006

Dipl.-Inform. Christian Fuß

Architekturübertragung nach Java



Factory

```
/*
 * functional module Factory is
 *   local import from Controller using all
 *   local import from Producer
 *   local import from Consumer
 */
public class Factory {

    private static Producer p = new Producer();
    private static Consumer c = new Consumer();

    • • •

    public static void main(String[] args) {
        Controller.startupFactory();
        try {
            Thread.sleep(30000);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
        Controller.shutdownFactory();
    }
}
```

Controller

```
/*
 * functional module Controller is
 *   local import from Producer using start, stop
 *   local import from Consumer using start, stop
 */
static class Controller {
    public static void startupFactory() {
        p.start();
        c.start();
    }

    public static void shutdownFactory() {
        p.stop();
        c.stop();
    }
}
```

Producer

```
/*
 * functional module Producer is
 *   general import from Item using all
 *   general import from Buffer using write
 */
static class Producer extends Thread
{
    public void run() {
        while (true) {
            produce();
        }
    }

    private void produce() {
        try {
            Thread.sleep((long) (Math.random() * 500));
            Item item = new Item();
            System.out.println("P: Produced item with timestamp "
                    + item.getTimestamp());
            Buffer.write(item);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }
}
```

Buffer 1

```
/*
 * abstract data object module Buffer is Monitor
 *   general import from Item
 */
public class Buffer {

    private static final int SIZE = 3;
    private static Item[] storage = new Item[SIZE];
    private static int count = 0;
    private static int head = 0;
    private static int tail = 0;

    public static void write(Item item) throws InterruptedException {
        synchronized (storage) {
            if (isFull()) {
                System.out.println("B: Buffer full, cannot write. Need to wait!");
                storage.wait();
            }
            storage[tail] = item;
            tail = (tail + 1) % SIZE;
            count++;
            System.out.println("B: Buffer count " + count);
            storage.notify();
        }
    }
```

Buffer 2

```
    public static Item read() throws InterruptedException {
        synchronized (storage) {
            if (isEmpty()) {
                System.out.println("B: Buffer empty, cannot read. Need to wait!");
                storage.wait();
            }
            Item item = storage[head];
            head = (head + 1) % SIZE;
            count--;
            System.out.println("B: Buffer count " + count);
            storage.notify();
            return item;
        }
    }

    public static boolean isEmpty() {
        return count == 0;
    }

    public static boolean isFull() {
        return count == SIZE;
    }
}
```

Algebraische Spezifikation

spec ITEM_QUEUE_TYPE

-- imports of other specs are necessary, e.g. of ITEM

sorts ITEM_QUEUE_TYPE

operations

-- export interface:

NEW: SIZE → ITEM_QUEUE_TYPE

ENQUEUE: ITEM x ITEM_QUEUE_TYPE → ITEM_QUEUE_TYPE

DEQUEUE: ITEM_QUEUE_TYPE → ITEM x ITEM_QUEUE_TYPE

IS_EMPTY: ITEM_QUEUE_TYPE → BOOLEAN

IS_FULL: ITEM_QUEUE_TYPE → BOOLEAN

Algebraische Spezifikation

preconditions

pre NEW(SZ) = SZ > 0

pre ENQUEUE(IT, QU) = not IS_FULL(QU)

pre DEQUEUE(QU) = not IS_EMPTY(QU)

equations

for all IT: ITEM, QU: ITEM_QUEUE_TYPE:

IS_EMPTY(NEW())

not IS_FULL(NEW())

not IS_EMPTY(ENQUEUE(IT, QU))

not IS_FULL(DEQUEUE(QU))

DEQUEUE(ENQUEUE(IT, QU)) = (IT, QU)

end spec ITEM_QUEUE_TYPE