Quasar Best Practices

- Kontext, Begriffe
- Kompositionsmanager
- Schnittstellen, Fehler / Fassaden
- Beziehungen
- A-Komponenten, Datentypen

Buchempfehlung

Wer sich für Architektur (= Design im Großen) interessiert:
Moderne Softwarearchitektur ...

- bietet viele Details zu besprochenen Themen
- bietet Anleitungen für die Identifikation geeigneter Komponenten beim Entwurf
- beschreibt Mechanismen/Komponenten für Persistenz, Autorisierung etc. vor
- stellt eine (verständliche!) Sprache zur Beschreibung von Schnittstellen vor

Buch-Kapitel

Quasar –
Best Practices

- Begriffe
- Kompositions Manager
- Schnittstellen
- Fehler, Fassaden
- Beziehungen
- A-Komponenten
- Datentypen
- …
- ...
- ...
- ...
- ...

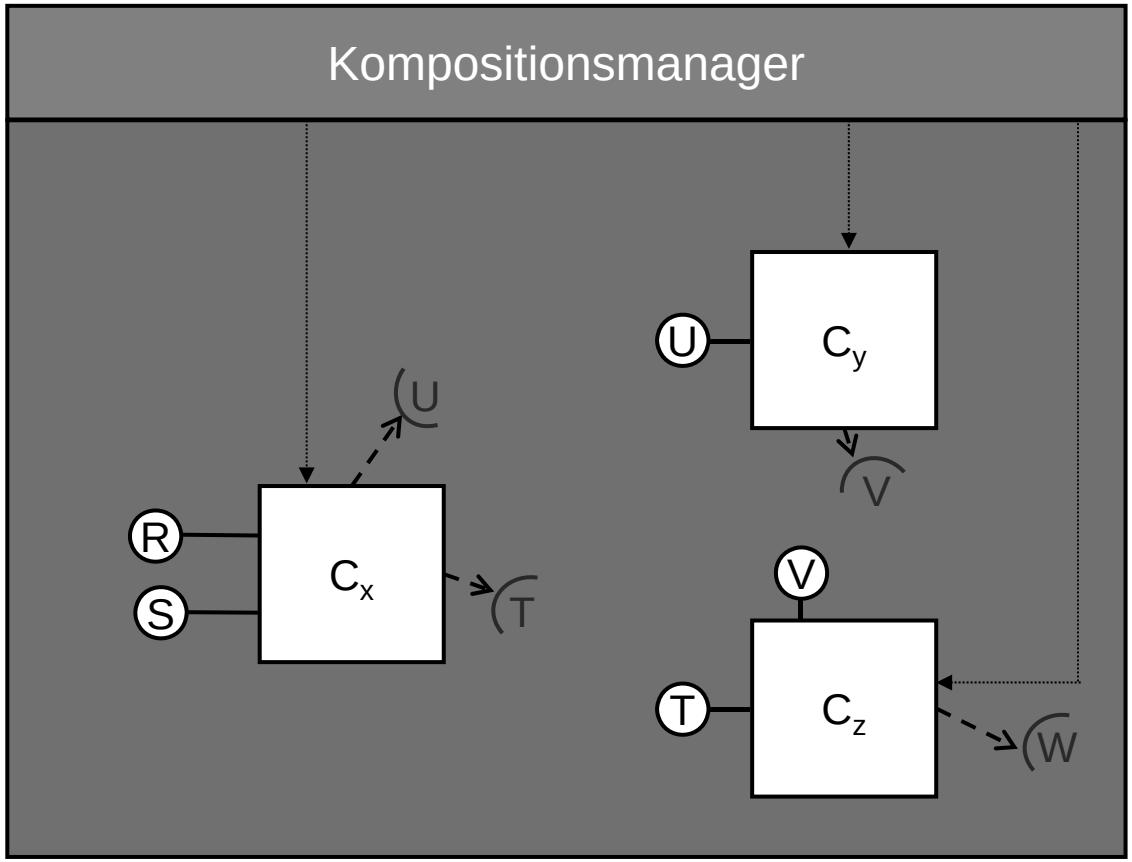
1. Softwarearchitektur als Herausforderung
2. Klassen und Schnittstellen
3. Komponenten und Schnittstellen
4. Softwarekategorien – wie findet man Komponenten?
5. Fehler und Ausnahmen – Rechte und Pflichten
6. Spezifikation von Schnittstellen
7. Softwarearchitekturen
8. Anwendungskomponenten und Anwendungskern
9. Pools, Persistenz und Transaktionen
10. Architektur von Informationssystemen

Komponenten

Eine **Komponente**

- *exportiert* eine oder mehrere *Schnittstellen*, die im Sinne eines Vertrags garantiert sind.
- *importiert Schnittstellen* (keine Komponenten!); sie ist erst lauffähig, wenn alle importierten Schnittstellen zur Verfügung stehen (dies ist Aufgabe der *Konfiguration*).
- *versteckt die Implementierung* und kann daher durch eine andere Komponente ersetzt werden, die dieselbe Schnittstelle exportiert.
- eignet sich als Einheit der Wiederverwendung, denn sie kennt nicht die Umgebung, in der sie läuft, sondern macht darüber nur minimale Annahmen.
- kann andere Komponenten enthalten (“Komposition”).
- ist neben der Schnittstelle die wesentliche Einheit des Entwurfs, der Implementierung und damit der Planung.

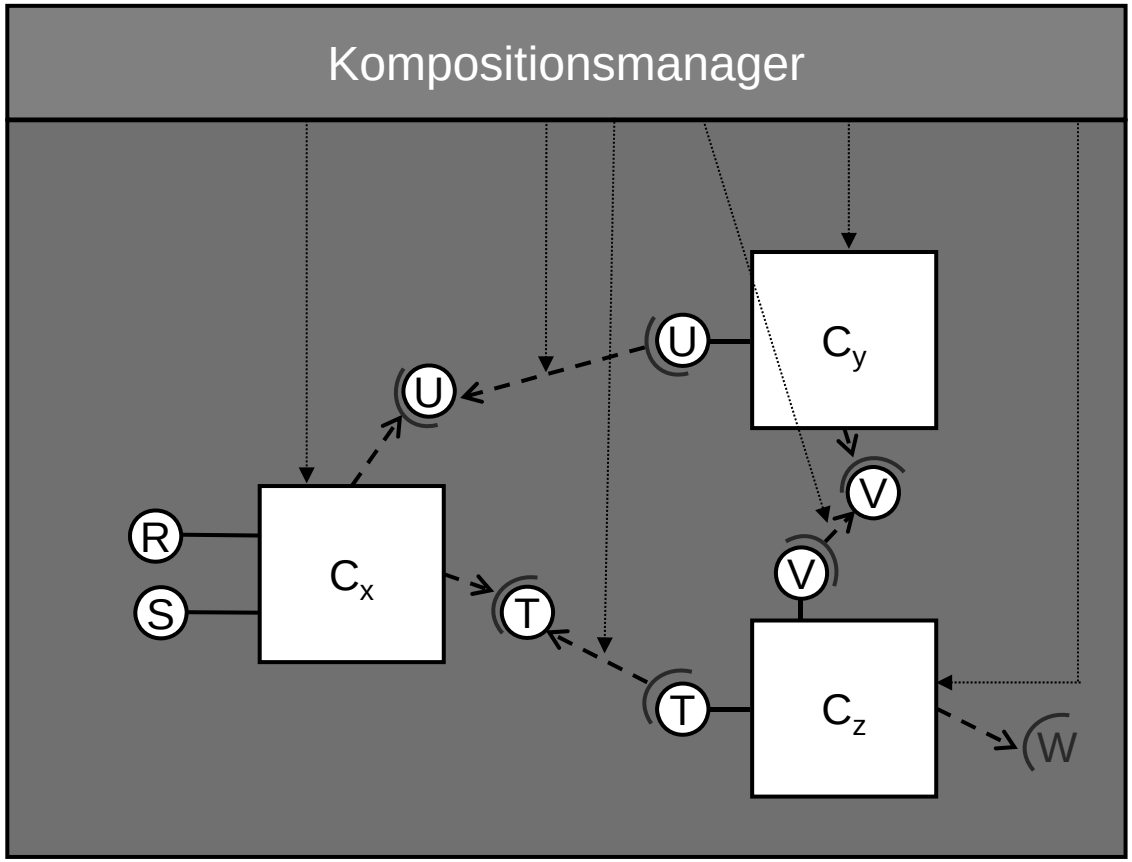
Veranschaulichung



1. Aufgabe des Kompositionsmanagers

- Komponenten / Klassen erzeugen

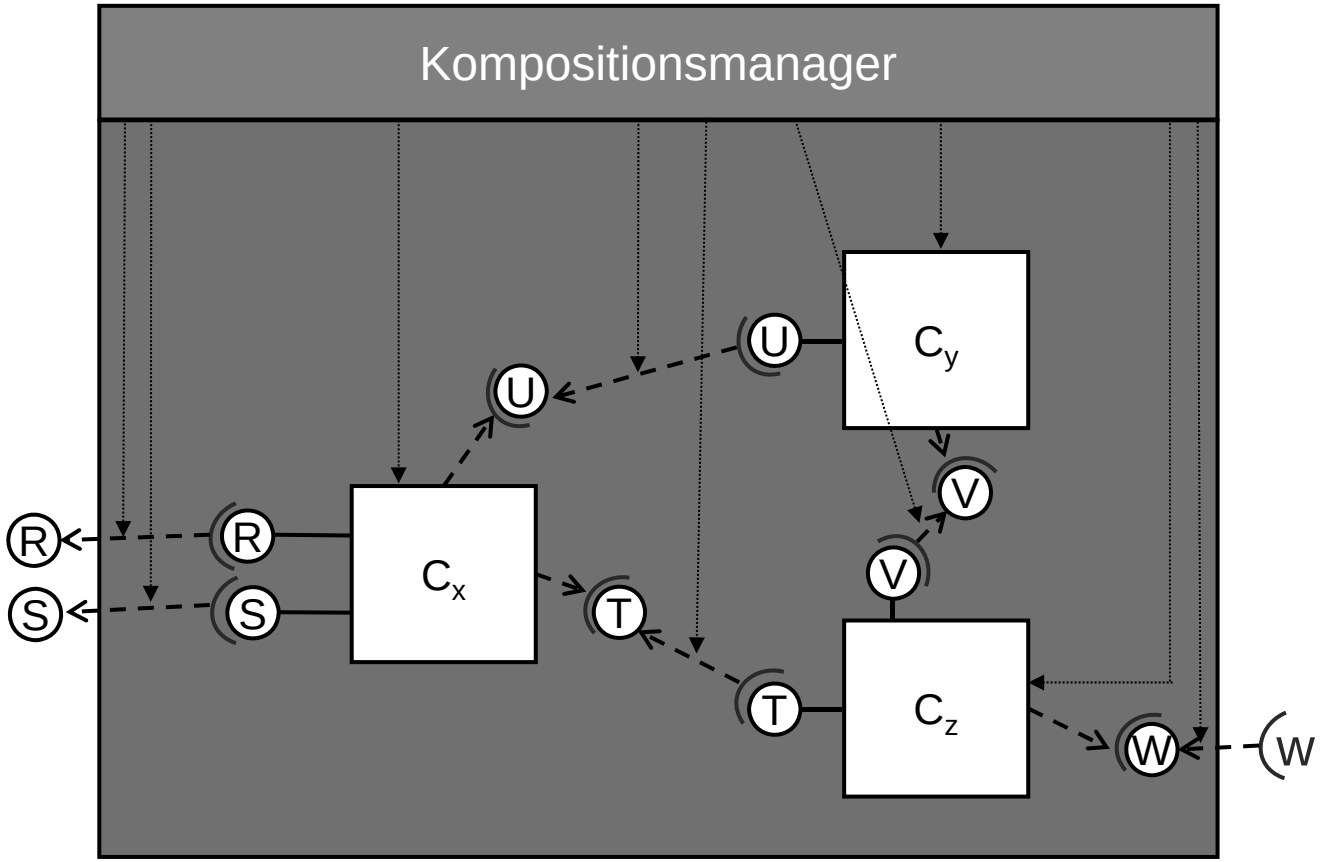
Veranschaulichung



2. Aufgabe des Kompositionsmanagers

- Komponenten / Klassen in der Komponente verbinden

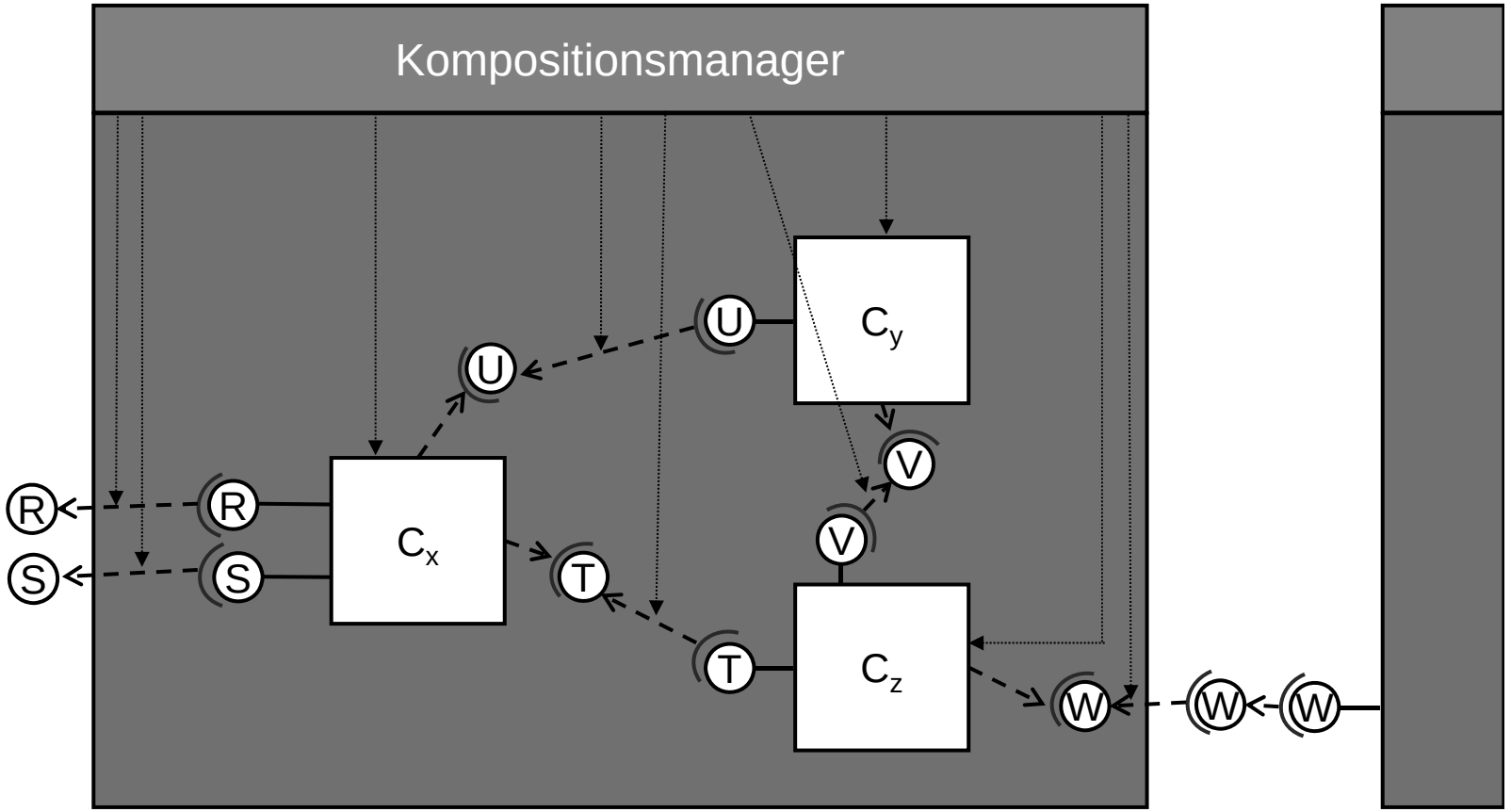
Veranschaulichung



3. Aufgabe des Kompositionsmanagers

- Komponenten / Klassen nach draußen durchreichen

Veranschaulichung



Aufgabe des umgebenden Kompositionsmanagers (Hauptprogramms)

- Komponenten / Klassen miteinander verbinden

Schnittstellen 1/2

- Als Typen von Parametern etc. *Schnittstellen* verwenden
  (keine implementierende Klassen)
- Immer die *allgemeinste* Schnittstelle auswählen
- Tests gegen Schnittstellen schreiben
  (sie gelten dann für alle Implementierungen)

Schnittstellen 2/2

- Komponenten sollten 2 Arten von Schnittstellen anbieten
  - Operative: möglichst einfach, effizient
  - Administrative: je nach Bedarf
- Schnittstellen enthalten 2 Arten von Methoden
  - *Abfragen*: lassen den Zustand einer Komponente unverändert
  - *Kommandos*: ändern den Zustand einer Komponente
- Zu jeder Schnittstelle sollte es mindestens 2 Implementierungen geben
  1. Für Tests, zum Erfahrungen sammeln, für die Entwicklung der benutzenden Komponenten
  2. Für den operativen Gebrauch

State Retrieval

Use Cases

Fehler und Ausnahmen

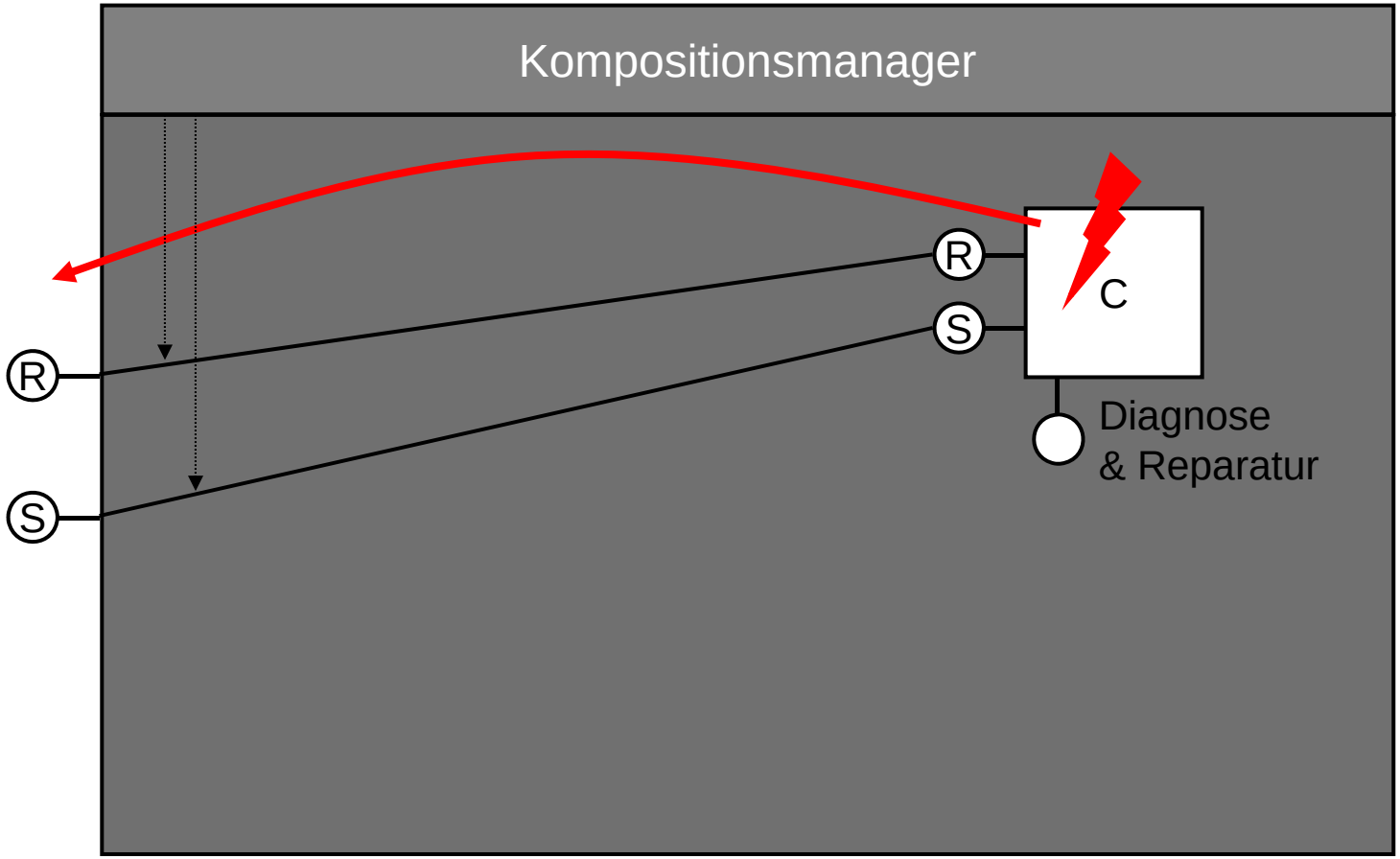
- Abnorme Ergebnisse / Situationen sollten als *ungeprüfte Ausnahmen* (*RuntimeException*) gemeldet werden
  → Annahme:
  Man kann selten vernünftig darauf reagieren
- Konsequenzen, falls das nicht so gehandhabt wird:
  - (in Java) *throws*-Deklarationen mit Implementierungsdetails werden über den ganzen Code verstreut, z.B.
    - → java.rmi.RemoteException
    - → java.sql.SQLException
    - → javax.xml.xpath.XPathFactoryConfigurationException

→ Ungeprüfte Ausnahmen sollten ausschließlich von *Sicherheitsfassaden*, *-filtern* behandelt werden

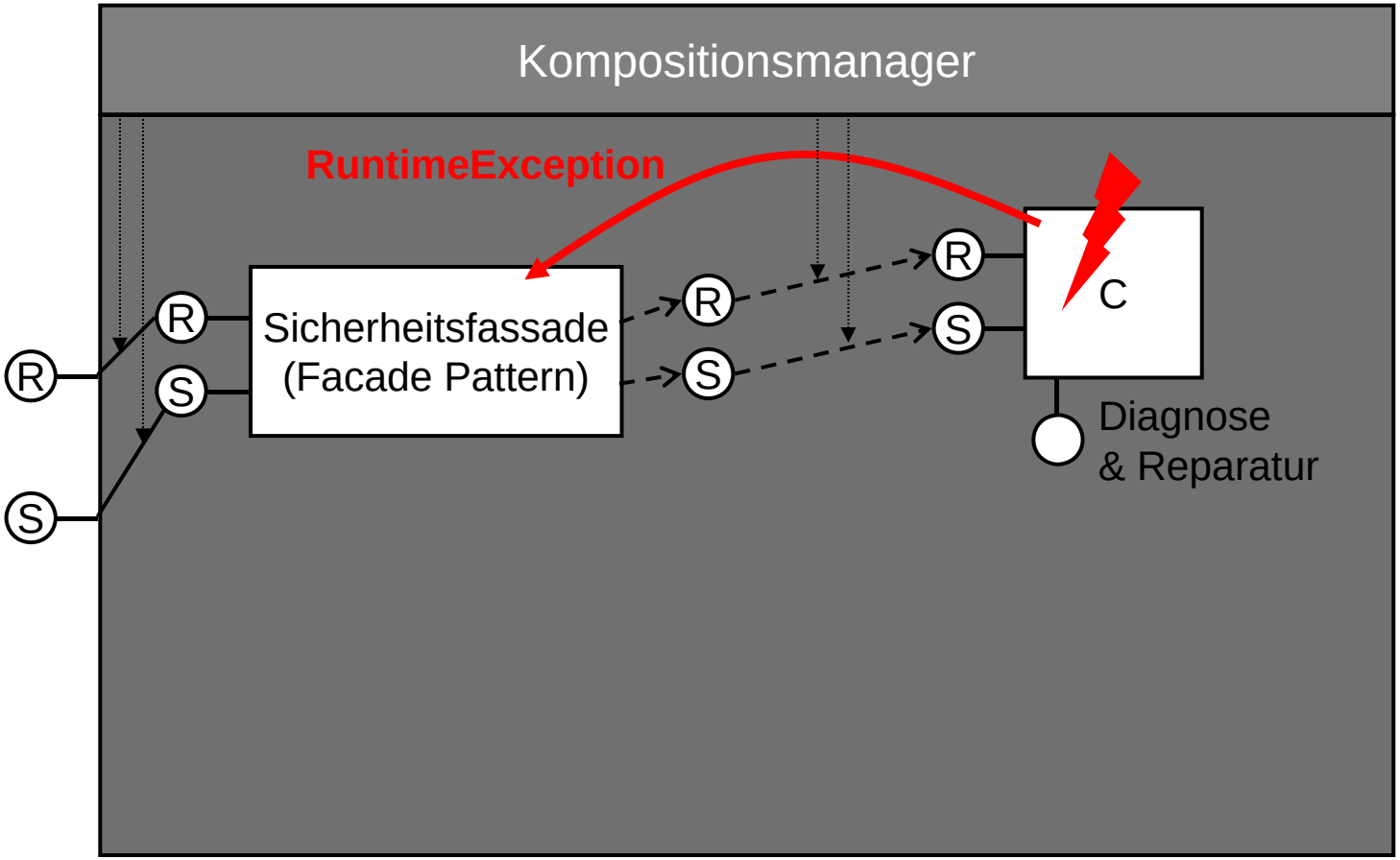
Fehler und Ausnahmen: Sicherheitsfassaden

Quasar –
Best Practices

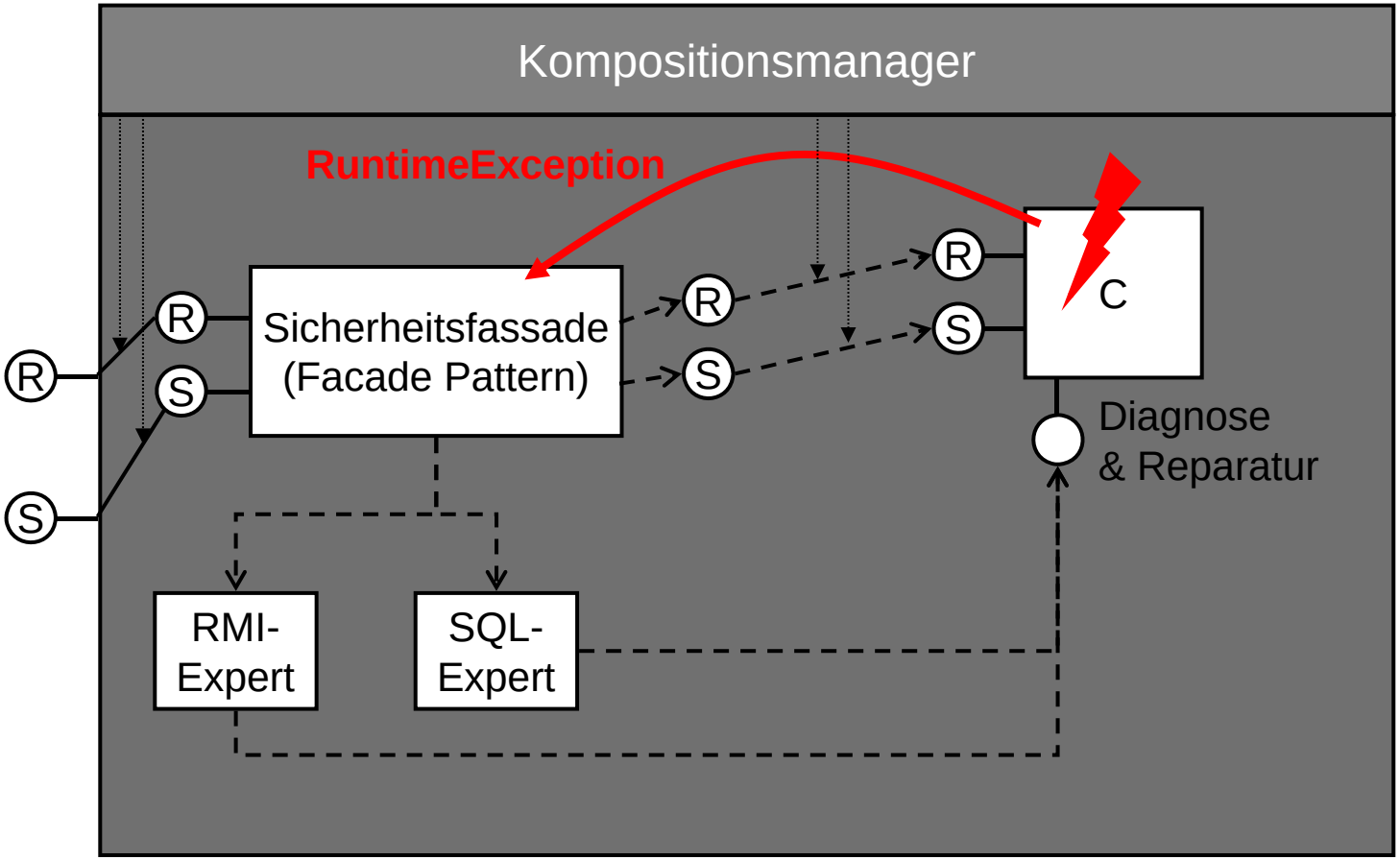
- Begriffe
- Kompositions-Manager
- Schnittstellen
- Fehler, Fassaden
- Beziehungen
- A-Komponenten
- Datentypen
- …
- ...
- ...
- ...
- ...



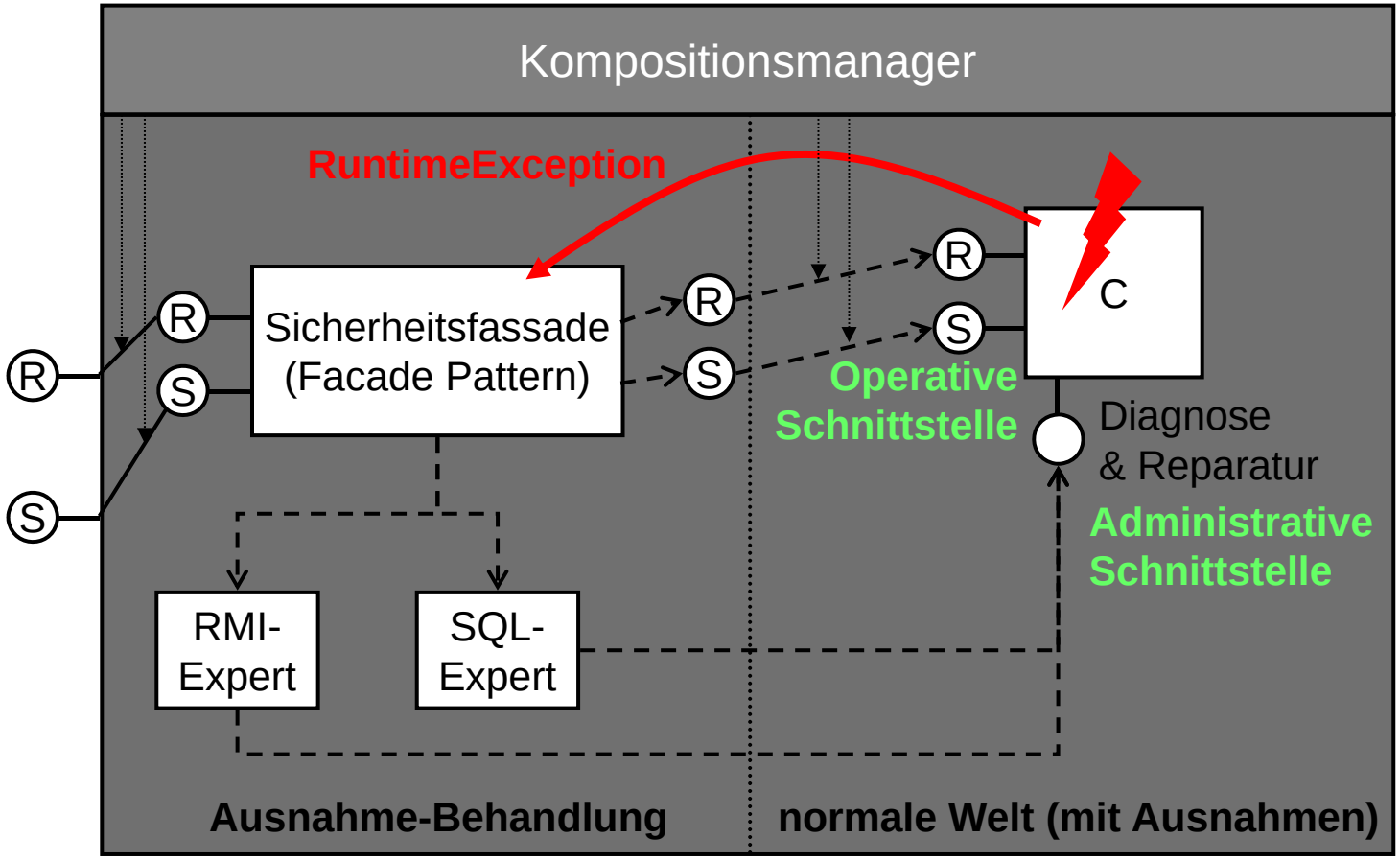
Fehler und Ausnahmen: Sicherheitsfassaden



Fehler und Ausnahmen: Sicherheitsfassaden

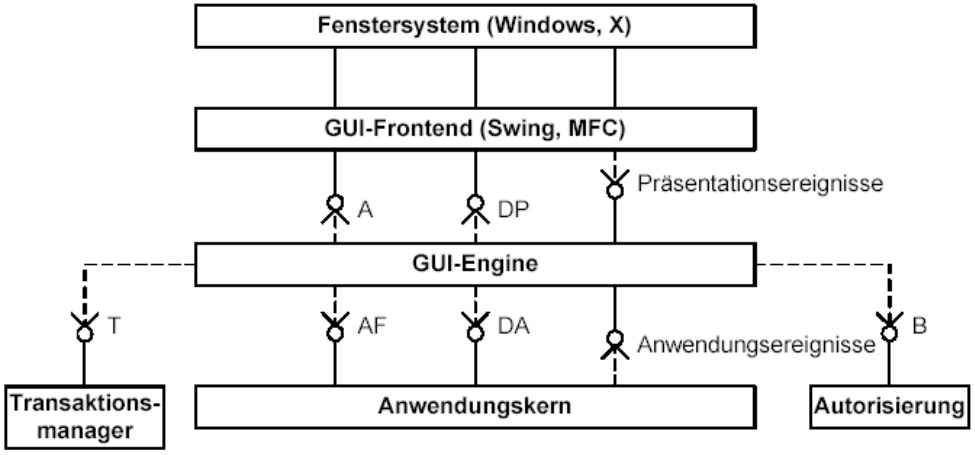


Fehler und Ausnahmen: Sicherheitsfassaden



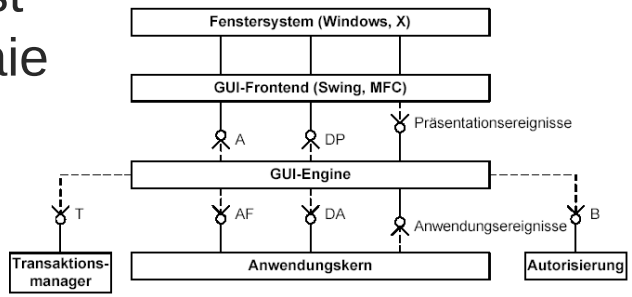
- Beziehungen zeigen sich *nicht* auf Klassenebene (*public* / *private*), sondern
- Beziehungen entsprechen *Import*-Deklarationen (*package*-Ebene)

Zyklische Beziehungen (= Abhängigkeiten) zwischen Komponenten sind *unbedingt* zu vermeiden



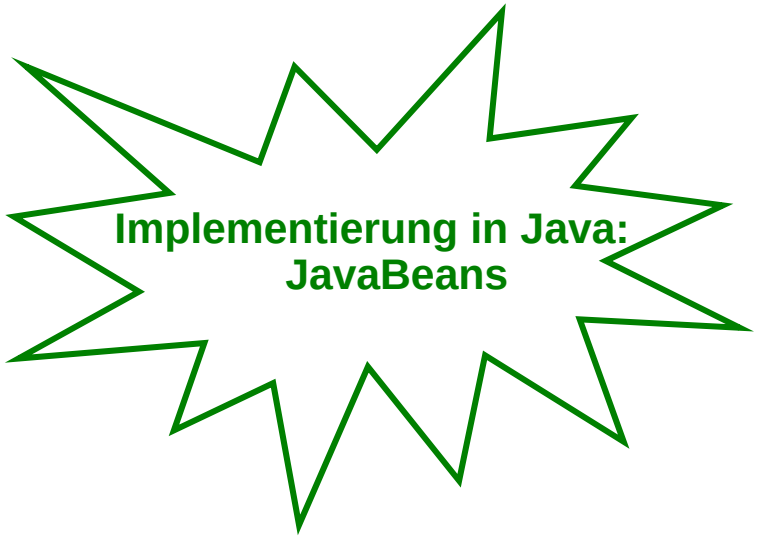
Implementierung des Anwendungskerns

- Komponenten des Anwendungskerns (= A-Komponenten) implementieren
  - Anwendungs-Entitätstypen
    - → haben einen fachlichen Schlüssel
    - → haben einen Lebenszyklus: man kann Exemplare davon anlegen, ändern und löschen
    - → prüfen auf Gleichheit: gleicher Schlüssel
    - → Beispiele: Mitarbeiter, Bestellung, Konto
  - Anwendungs-Datentypen
    - → sind Standard-Datentypen *der Software*
    - → enthalten niemals Referenzen auf Entitätstypen (nur umgekehrt)
    - → prüfen auf Gleichheit: Übereinstimmung aller Felder
    - → Beispiele: Datum, Adresse, Euro, Aufzählungstypen, Datentypen aus java.lang, java.util etc.
- Ziel:
  Jede Methode des Anwendungskerns ist so beschaffen, dass sie auch der DV-Laie versteht



Implementierung von Datentypen

- Arten von Funktionen
  - Prüfungen: statische Methoden, *nicht* im Konstruktor
  - Fachliche Funktionen z.B. Arithmetik für Zeit
  - Transformationen: extern → intern per Konstruktor; intern → extern per toString etc.
  - Standardmethoden: equals, compare etc.
- *Datentypen sollten völlig technikfrei sein*, weil sie oft für andere Komponenten exportiert werden



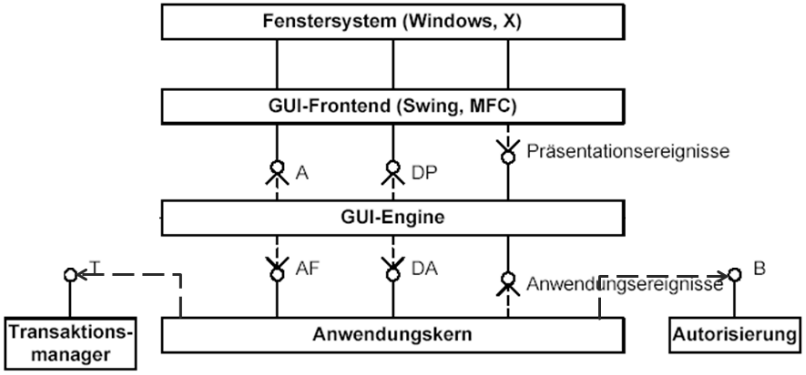
Betrachtet wird ein abstrakter Datentyp *Date*:
Der ADT *Date* repräsentiert ein bestimmtes Datum im Gregorianischen Kalender und bietet verschiedene Operationen auf Kalenderdaten an.

Die jeweiligen Operationen sind wie folgt definiert:

- Date: int × int × int → Date
  Wenn es sich um ein gültiges Datum handelt, liefert dieser Konstruktor ein entsprechendes *Date*-Objekt als Ergebnis; sonst wird ein *Date*-Objekt geliefert, für das getDay(), getMonth() und getYear() jeweils -1 als Ergebnis liefern
- equals: Date × Date → boolean
  Liefert genau dann *true*, wenn die beiden Kalenderdaten gleich sind
- getWeekday: Date → int
  Liefert den Wochentag zu einem Datum;
  der Montag entspricht der 1, der Dienstag der 2 usw.
- subtract: Date × Date → int
  Subtrahiert das 2. Datum vom 1. Datum; liefert die Differenz der beiden Datumsangaben in Tagen als ganze Zahl
  Voraussetzung: das 1. Datum ist größer als das 2. Datum, sonst ist das Ergebnis -1
- isValid: int × int × int → boolean
  Liefert genau dann *true*, wenn die angegebenen Zahlen in der Reihenfolge Tag, Monat, Jahr ein gültiges Datum repräsentieren
- getDay: Date → int
  getMonth: Date → int
  getYear: Date → int
  Liefern den Tag, den Monat bzw. das Jahr des angegebenen Datums als *int*-Wert

Implementierung von Datentypen

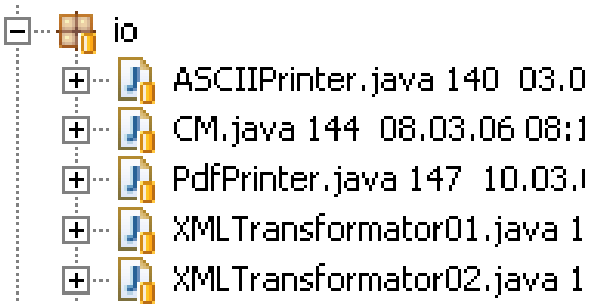
- Transformationsmethoden nicht in die Datentypen aufnehmen, sondern mittels Interface realisieren; Beispiel:
  - public interface Transformator {
    public Object toExtern(boolean b);
    public boolean toBoolean(Object externalBoolean);
    public Object toExtern(*SomeType* internalObject);
    public *SomeType* to*SomeType*(Object externalObject);
    ...
    }
- Der *Transformator* wird für alle elementaren und selbstdefinierten Datentypen erweitert
- Für jede gewünschte externe Darstellung (z.B. SQL, XML, RTF) wird dieses Interface implementiert



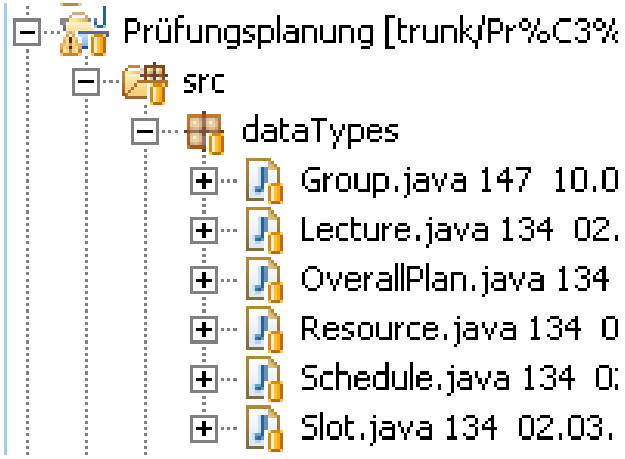
Transformator-Beispiel

```
public interface InputTransformator {

    String toString( Object externalString );

    OverallPlan toOverallPlan( Object externalOverallPlan );

    Group toGroup( Object externalGroup );

    Lecture toLecture( Object externalLecture );

    Resource toResource( Object externalResource );

    Schedule toSchedule( Object externalSchedule );

    Slot toSlot( Object externalSlot );
```

Implementierungen



Qu
Be
Fassaden
Beziehungen
A-Komponenten
Datentypen
...
...
...
...
...




```
public interface OutputTransformator {

    Object toExtern( String s );

    Object toExtern( OverallPlan op );

    Object toExtern( Group g );

    Object toExtern( Lecture l );

    Object toExtern( Resource r );

    Object toExtern( Schedule s );

    Object toExtern( Slot s );
```