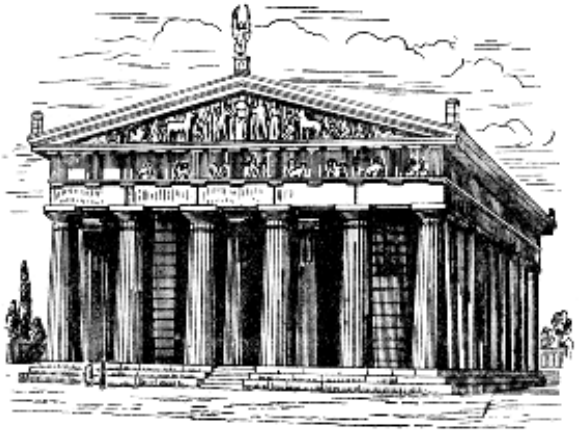
Softwarearchitektur

(Architektur: *αρχή* = Anfang, Ursprung + *tectum* = Haus, Dach)

2. Grundlagen der Softwarearchitektur



Vorlesung Wintersemester 2008 / 2009

Technische Universität München
Institut für Informatik
Lehrstuhl von Prof. Dr. Manfred Broy

Dr. Klaus Bergner, Prof. Dr. Manfred Broy, Dr. Marc Sihling

Inhalt

- Motivation und Darstellungsweise
  - Motivation und Darstellung
  - Komponenten und Instanzen
- Systemmodell für Architektur: Instanzen
  - Grundkonzepte und Strukturen
  - Zeit, Kommunikation und Interaktion
  - Strukturänderungen
  - Mobilität
- Systemmodell für Architektur: Beschreibungen
  - Definition und Bedeutung
  - Reflexivität
- Zusammenfassung

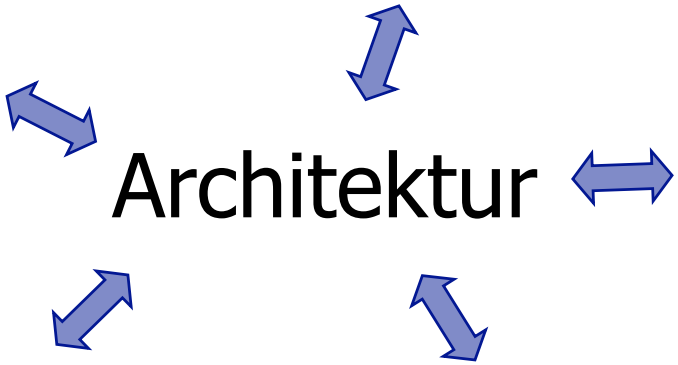
Inhalt

- Motivation und Darstellungsweise
  - Motivation und Darstellung
  - Komponenten und Instanzen
- Systemmodell für Architektur: Instanzen
  - Grundkonzepte und Strukturen
  - Zeit, Kommunikation und Interaktion
  - Strukturänderungen
  - Mobilität
- Systemmodell für Architektur: Beschreibungen
  - Definition und Bedeutung
  - Reflexivität
- Zusammenfassung

Wiederholung: Warum Softwarearchitektur

- Divide and Conquer (Teile und Herrsche):
  - Arbeitsteilung in der Projektorganisation
  - Komplexitätsreduktion durch Schnittstellenabstraktion und Kapselung
    - Austauschbarkeit, Portierbarkeit
  - Beherrschbarkeit von Qualität
  - Integration9
  - Wartung
- Wiederverwendung
  - Der Architektur (Referenzarchitektur)
  - Von Lösungsteilen (Design Patterns)
  - Von Komponenten
- Verteilung der Software auf der Hardware

Einflussfaktoren



Architektur

Unterschiedliche Techniken und Komponenten

- COBOL, Java, .NET, ODBMS, XML, HTML, ...

Unterschiedliche Anwendungsgebiete

- Telekommunikation, eCommerce, Engineering,...

Unterschiedliche Programmierparadigmen

- Objektorientiert, Prozedural, Deklarativ, ...

Unterschiedliche Arten von Systemen

- Informationssysteme, eingebettete Systeme, mobile Systeme, ...

Unterschiedliche Methoden und Werkzeuge

- OOA/OOD, RUP, UML, CASE-Werkzeuge, IDEs, KM-Werkzeuge, ...

Problematik bei der Modellierung von SWAs

- SW-Architektur ist ein relativ neues Gebiet
  - Erst seit relativ kurzer Zeit als eigene Disziplin erkannt
  - Grundlagen noch wenig wissenschaftlich aufgearbeitet
- Begriffe sind noch stark im Fluss
  - Oft gibt es noch keine angemessenen Abstraktionen
  - Konzepte stammen meist aus konkreten Systemen
  - Begriffe in unterschiedlichen Ansätzen unterschiedlich genutzt
  - Unterschiede werden betont (.NET versus Java)
- Darstellung erfolgt meist anhand von Beispielen
  - Irrelevante technische Details erschweren Verständnis
  - Übertragung auf andere Systeme schwierig
  - Darstellung kostet viel Zeit
- Zusammenhang zu Programmiersprachen oft ungenügend

Gesichtspunkte

- Daten - Kapselung
- Struktur: Gliederung des Systems
  - Horizontal: Aufspaltung eines Systems in Komponenten und ihre Verbindungen (Schnittstellen)
  - Vertikal: Weitere Untergliederungen der Komponenten
  - Es entsteht ein Struktur/Komponentenbaum (auch Struktur /Komponentenhierarchie genannt)
- Verhalten
  - Interaktion zwischen den Komponenten
  - Schnittstellenverhalten
- Verteilung

Darstellungsweise in der Vorlesung

- Definition eines abstrakten Systemmodells als Basis
  - Weglassen der technischen Details
  - Definition allgemeingültiger Begriffe und ihrer Zusammenhänge
  - Geeignet für sämtliche Anwendungsgebiete, Arten von Systemen, Programmierparadigma, etc.
- Bildung zusätzlicher Begriffs-Ebenen
  - Einführung zusätzlicher Konzepte auf Basis des Systemmodells
  - Angepasst an spezielle Anwendungsgebiete, Arten von Systemen, Programmierparadigma, etc.
- Veranschaulichung jeweils anhand konkreter Beispiele
  - Anwendung der Begriffe
  - Aufzeigen der technischen Umsetzung

Was ist ein Systemmodell?

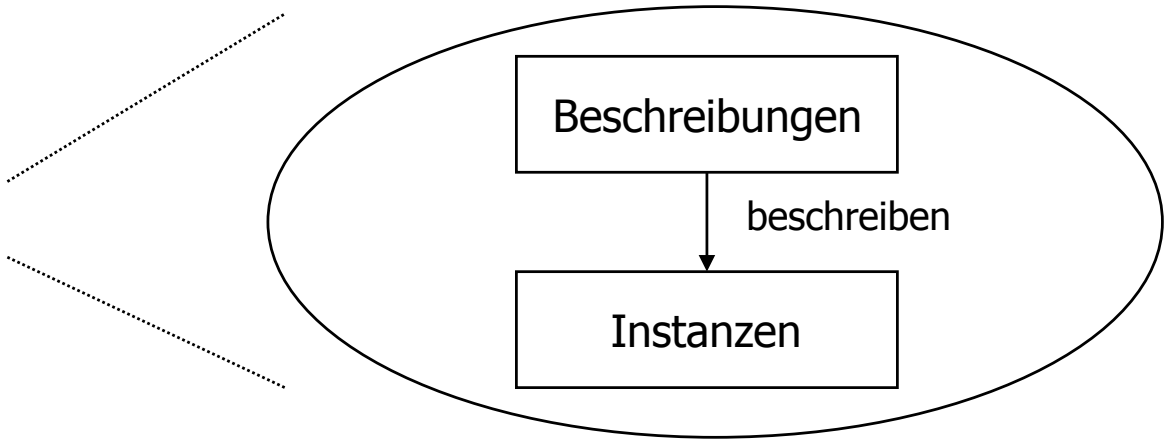
Dient der Modellierung von Systemen:

- Definiert Grundkonzepte und ihre Zusammenhänge
  - Konzentration auf relevante Strukturen und Konzepte
  - Gibt ein bestimmtes Grundverständnis vor
- Abstrahiert von unwichtigen Details
  - Technische Details eines bestimmten Systems
  - Syntaktische Details einer bestimmten Beschreibung
- Verwendet mathematische Definitionen
  - Eindeutige und unmissverständliche Darstellung
  - Effiziente, sehr kompakte Darstellung
  - Möglichst einfach verständlich ☺
- Veranschaulichung durch Bilder und Diagramme

Beschreibungen und Instanzen

- Architekten benötigen
  - Verständnis der Systemstrukturen und -Abläufe zur Laufzeit
  - Fähigkeit, diese Systeme auf unterschiedliche Art und Weise zu beschreiben
- Systemmodell enthält zwei Grundkonzepte
  - Komponenten-Beschreibungen repräsentieren Entwicklungssicht
  - Komponenten-Instanzen repräsentieren Laufzeitsicht

Mathematik
+ Bilder



Komponenten: Beschreibungen und Instanzen

Beschreibungen

- Komponenten mit deren Beschreibungen von Verhalten sind die Entwurfseinheiten zur Entwicklungszeit (Entwurfseinheiten).
- Entwurfseinheiten dienen der Strukturierung des Entwurfs.
- Entwurfseinheiten haben eine statische, gegebenenfalls hierarchische Struktur.
- Der Entwurf beschreibt die die Struktur und das Verhalten der Instanzen zur Ausführungszeit.

Instanzen

- Instanzen existieren zur Laufzeit. (Ausführungseinheiten)
- Ausführungseinheiten dienen der Organisation der Ausführung.
- In der Regel sind die Menge der Instanzen und ihre Verbindungen dynamisch (ändern sich während der Ausführungszeit).
- Das Verhalten der Instanzen wird durch die Entwurfs -einheiten beschrieben.

Beschreibung und Instanz

- Es gibt eine reiche Vielfalt unterschiedlicher Konzepte von Komponenten und ihres Verhaltens:
  - Funktion
  - Klasse
  - Zustandsmaschine
  - …
- Auf der Menge der Verhalten sind Kompositions-Operationen definiert (aus gegebenen Verhaltensbeschreibungen lässt sich eine komplexere Verhaltensbeschreibung zusammensetzen).
- Es gibt eine reiche Vielfalt von Möglichkeiten, Verhalten zu beschreiben:
  - Programmiersprachen: z.B. Assembler, C, Java
  - Modellierungssprachen: z.B. UML
  - Spezifikationen: z.B. Z, Zustandsmaschinen, Logik OCL

Beschreibung und Instanz

Eine Instanz entsteht in der Regel zur Laufzeit aus einer Entwurfseinheit:

- Eine Instanz besitzt
  - eine Identität (einen Identifikator)
  - einen Zustand (genauer Zustandsraum)
  - ein Laufzeit-Verhalten
- Zustände umfassen
  - lokale („verkapselte“) Daten- und Kontrollzustände
  - Verbindungen zu anderen Instanzen
  - enthaltene andere Instanzen
- Das Laufzeit-Verhalten umfasst
  - die Kommunikation mit anderen Instanzen
  - die Änderung des Zustands über die Zeit hinweg
  - Lokalisierung auf Hardwarekomponenten (Verteilung, Deployment)

Systeme: Mengen von Beschreibungen und Instanzen

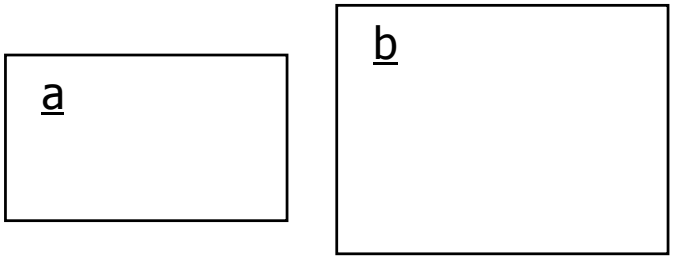
- Ein Systembeschreibung umfasst
  - eine Menge von Verhaltensbeschreibungen (Entwurfseinheiten)
  - eine Beschreibung, wie aus den Verhaltensbeschreibungen (Entwurfseinheiten) Instanzen gebildet werden und wie diese verbunden sind (wie das System sich aus Instanzen zusammensetzt)
- Zur Ausführungszeit (Laufzeit) besteht ein System zu jedem Zeitpunkt aus
  - einer Menge von Instanzen
  - der Struktur ihrer Verbindungen
- Der Systemzustand ist zu jedem Zeitpunkt gegeben durch
  - die Menge der existierenden Instanzen und ihren Datenzuständen
  - die Nachrichten, die über die Verbindungen und Schnittstellen laufen
  - die Struktur der Verbindungen

Inhalt

- Motivation und Darstellungsweise
  - Motivation und Darstellung
  - Komponenten und Instanzen
- Systemmodell für Architektur: Instanzen
  - Grundkonzepte und Strukturen
  - Zeit, Kommunikation und Interaktion
  - Strukturänderungen
  - Mobilität
- Systemmodell für Architektur: Beschreibungen
  - Definition und Bedeutung
  - Reflexivität
- Zusammenfassung

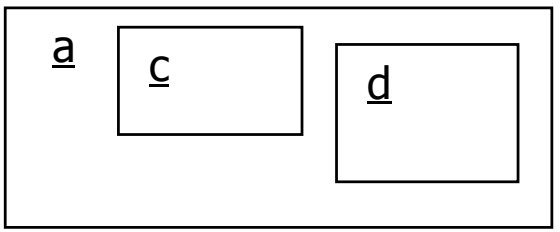
Komponenten-Instanzen

- Ein System besteht zur Laufzeit aus einer Menge von **Komponenten-Instanzen**.
- Definition:
  - COMPONENT : unendliche Menge aller nur möglichen („potentieller) Komponenteninstanzen (Identifikatoren und Verhalten)
- Beispiel:
  - a, b ∈ COMPONENT



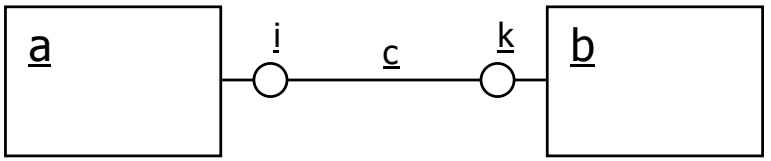
Komponentenhierarchie

- Komponenten(instanzen) können hierarchisch in andere Komponenten(instanzen) **zerlegt** bzw. aus anderen Komponenten(instanzen) **zusammengesetzt** werden.
- Definition:
  - parent : COMPONENT → COMPONENT
    Partielle Abbildung, ordnet einer Instanz ihre umschließende Eltern-Instanz zu. Instanzen auf oberster Ebene müssen keine Eltern-Instanz haben.
- Konsistenzbedingung:
  - parent ist azyklisch
- Beispiel:
  - parent(c) = parent(d) = a
  - parent(a) nicht definiert



Verbindungen und Schnittstellen

- **Komponenten**(instanzen) besitzen **Schnittstellen**(instanzen).
- Schnittstelleninstanzen sind untereinander **verbunden**.
- Definitionen:
  - INTERFACE : Menge aller Schnittstelleninstanzen
  - assigned : INTERFACE → COMPONENT
    Totale Abbildung, ordnet jeder Schnittstelle die Komponente zu, der sie zugeordnet ist. Freie Schnittstellen sind nicht erlaubt.
  - CONNECTION : Menge aller Verbindungsinstanzen
  - connects : CONNECTION → INTERFACE × INTERFACE
    Totale Abbildung, gibt für jede Verbindung an, welche Schnittstellen sie verbindet. Eine Schnittstelle kann viele Verbindungen haben.

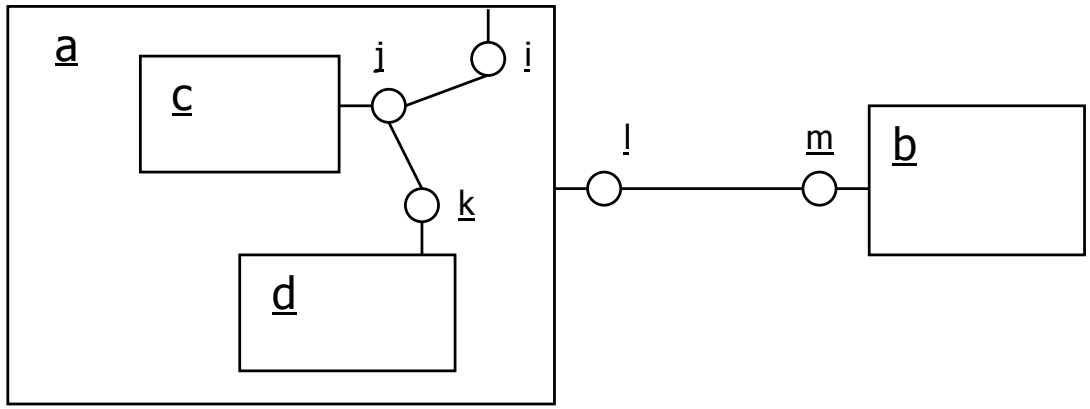


Systeme als Strukturen

- Komponenten, Schnittstellen und Verbindungen geben die Struktur eines Systems vor.
- Die Struktur ist einer der wichtigsten Aspekte bei der Betrachtung der Architektur eines Systems.
- Definitionen:
  - $\mathrm{INSTANCE} = \mathrm{COMPONENT} \cup \mathrm{INTERFACE} \cup \mathrm{CONNECTION}$
    Menge aller möglichen Instanzen
  - $\mathrm{alive} : \mathrm{INSTANCE} \rightarrow \mathrm{Boolean}$
    Totale Abbildung; gibt an, ob eine Instanz für die Systemstruktur des betrachteten Systems relevant/existent ist.
  - $\mathrm{STRUCTURE} = \mathrm{ASSIGNED} \times \mathrm{PARENT} \times \mathrm{CONNECTS} \times \mathrm{ALIVE}$
    Menge aller möglichen Systemstrukturen, wobei PARENT die Menge aller Funktionen parent ist, analog für $\mathrm{CONNECTS}$ und ALIVE.

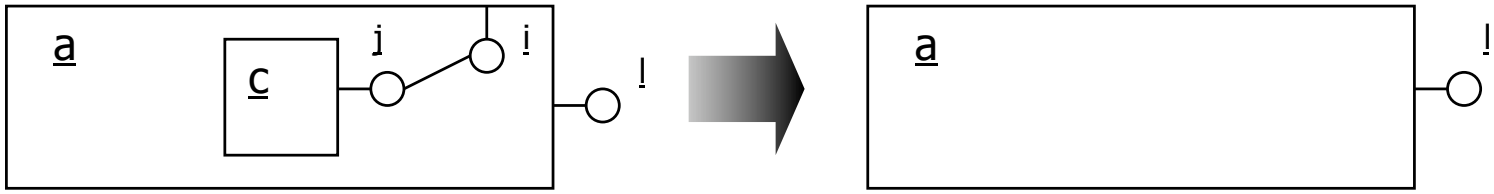
Systemstruktur - Beispiel

- a, b, c, d ∈ COMPONENT; i, j, k, l, m ∈ INTERFACE
- v, w, x ∈ CONNECTION
- ∀α ∈ INSTANCE : alive(α) ⇔ α ∈ {a, b, c, d, i, j, k, l, m, v, w, x}
- parent(c) = parent(d) = a
- assigned(i) = assigned(l) = a; assigned(m) = b
- assigned(j) = c; assigned(k) = d
- connects(v) = (i, j); connects(w) = (j,k); connects(x) = (l, m)



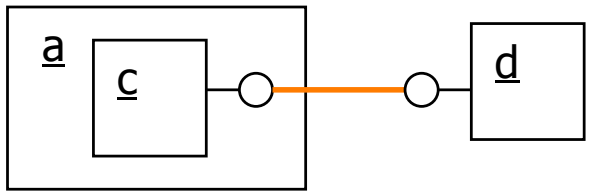
Schnittstellen und Implementierung

- Die Hierarchie und die Schnittstellen erlauben es, innere Strukturen auszublenden (Geheimnisprinzip).



white box *black box*

- Verbindungen dürfen dazu keine Komponentengrenzen kreuzen
  - $\text{connects}(v) = (i, j) \wedge \text{assigned}(i) = c \wedge \text{assigned}(j) = d \Rightarrow$
    $c = d \vee \text{parent}(c) = d \vee \text{parent}(d) = c \vee \text{parent}(c) = \text{parent}(d)$



Architekturprinzipien zur Komplexitätsreduzierung

- Geheimnisprinzip: Interne Strukturen und Abläufe werden vor der Außenwelt versteckt und sind nur über Schnittstellen zugreifbar.
  - Interne Strukturen sind für das Verständnis des Zusammenspiels der Komponente mit anderen Komponenten nicht relevant.
  - Interne Strukturen und Abläufe können geändert werden, ohne dass sich das Verhalten der Komponente nach außen ändert.
- Lose Kopplung – starke Bindung: Eine Komponente sollte möglichst viel Komplexität kapseln, indem sie
  - einen starken inneren Zusammenhang hat und
  - möglichst wenige, schmale Schnittstellen nach außen bereitstellt.

Inhalt

- Motivation und Darstellungsweise
  - Motivation und Darstellung
  - Komponenten und Instanzen
- Systemmodell für Architektur: Instanzen
  - Grundkonzepte und Strukturen
  - Zeit, Kommunikation und Interaktion
  - Strukturänderungen
  - Mobilität
- Systemmodell für Architektur: Beschreibungen
  - Definition und Bedeutung
  - Reflexivität
- Zusammenfassung

Verhaltensmodelle

- Grundsätzliche Ausführungskonzepte
  - Sequenzieller Kontrollfluss
  - Paralleler Kontrollfluss
- Grundsätzliche Interaktionskonzepte für Komponenten
  - gemeinsamer Speicher
    - gemeinsame Programmvariablen
  - gekapselter Speicher
    - Methoden/Prozeduraufruf
    - Nachrichtenaustausch

Verhaltensmodelle für Komponenten

- Zustandsmaschinen
  - Das Verhalten von Komponenten kann durch Zustandsmaschinen mit Ein-/Ausgabe modelliert werden
- Schnittstellenmodelle
  - Erfassung der Interaktion durch Daten-/Nachrichtenströme

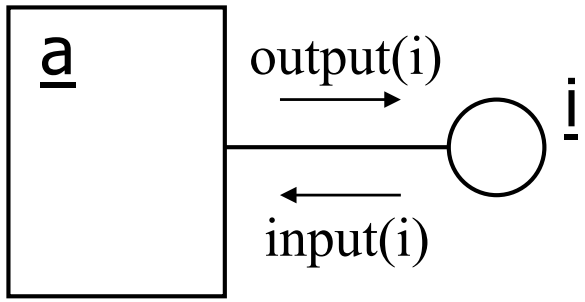
Verhalten und Zeitströme

- Bis jetzt konnten wir nur statische Strukturen beschreiben … Verhalten bedeutet aber, dass es Veränderungen über die Zeit hinweg gibt.
- Was ist Zeit? Zeit hat in dem Modell folgende Eigenschaften
  - Es gibt eine globale Zeit, die überall gleich schnell läuft.
  - Es gibt eine kleinste, atomare Zeiteinheit.
  - Annahme dabei: Wählen wir die Zeiteinheit klein genug, dann können wir alles mitbekommen (analog wie beim Sampling von Tonfrequenzen).
- Wir betrachten Zeitströme als Sequenzen von einzelnen Zeitschritten und definieren:
  - Sei X eine Menge von Signalen, Nachrichten, Ereignissen, Zuständen.
  - $T = \{t_0, t_1, t_2, t_3, ...\}$: Zeit - Menge aller betrachteten Zeitpunkte
  - $X^T = T \rightarrow X$: Zeitstrom der Elemente aus der Menge X
  - x_t : das Element aus X, das in X^T dem Zeitpunkt t zugeordnet ist

Kommunikation und Interaktion I

- Miteinander verbundene Komponenteninstanzen können kommunizieren, indem sie Nachrichten austauschen.
- Nachrichten fließen von Schnittstellen zu Komponenten und umgekehrt.
- Definitionen:
  - MESSAGE : Menge aller möglichen Nachrichten
  - $\varepsilon \in$ MESSAGE : leere Nachricht
  - input : INTERFACE → MESSAGE
  - output : INTERFACE → MESSAGE

Partielle Abbildungen; geben an, welche Nachrichten in eine bzw. aus einer Komponente fließen. Definition von Zeitströmen input^T und output^T wie eben.

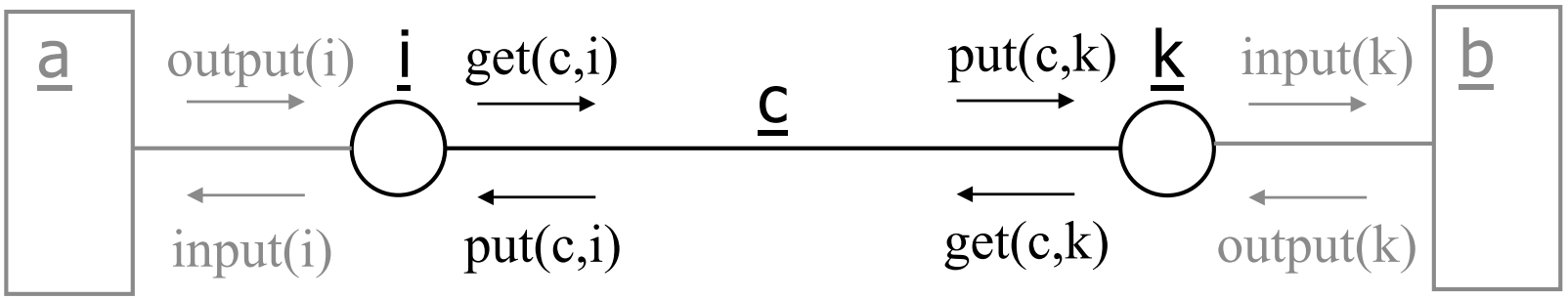


Kommunikation und Interaktion II

- Nachrichten fließen von Schnittstellen in Verbindungen und umgekehrt.
- Definitionen:
  - get : CONNECTION × INTERFACE → MESSAGE
  - put: CONNECTION × INTERFACE → MESSAGE

Partielle Abbildungen; geben an, welche Nachrichten in eine bzw. aus einer Verbindung fließen.

Definition von Zeitströmen get^T und put^T.



Verhalten von Verbindungsinstanzen I

- Das Verhalten einer Verbindungsinstanz lässt sich als Teilmenge aller Zeitströme der Ein- und Ausgaben beschreiben (genauer: aller Zeitströme der get- und put- Abbildungen, die diese Ein- und Ausgaben festlegen).
- Definitionen:
  - $\text{behavior} : \text{CONNECTION} \rightarrow \wp(\text{get}^T \times \text{put}^T)$
- Lokalität: Das Verhalten einer Verbindung c macht nur Aussagen über Nachrichten auf der Verbindung c (und nicht über Nachrichten auf anderen Verbindungen).
  - $(\text{gets}, \text{puts}) \in \text{behavior}(c) \wedge \text{connected}(c) = (i, k) \Rightarrow$
    $\forall t \in T: \quad \text{gets}_t \in \{ \{(c, i), (c, k)\} \rightarrow M \}$
    $\wedge \quad \text{puts}_t \in \{ \{(c, i), (c, k)\} \rightarrow M \}$

s für Zeit**s**trom

Verhalten von Verbindungsinstanzen II

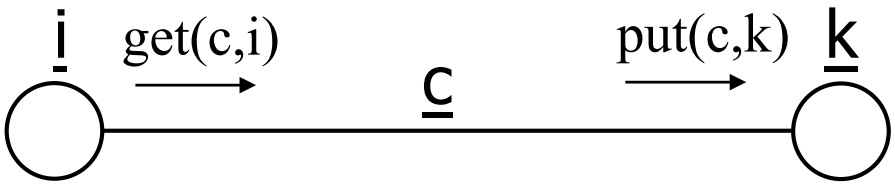
- Kausalität: Die Ausgaben einer Verbindung c zu einem bestimmten Zeitpunkt t können nur von den Eingaben bis zum Zeitpunkt t-1 abhängen (und insbesondere nicht von zukünftigen Eingaben).
  - $(gets1, puts1) \in behavior(c) \wedge (gets2, puts2) \in behavior(c) \wedge gets1{\downarrow}t\text{-}1 = gets2{\downarrow}t\text{-}1 \Rightarrow puts1_t = puts2_t$

    Notation: x↓t bezeichnet den Teil des Zeitstroms x bis einschließlich Zeitpunkt t
  - Die Formel bezeichnet den deterministischen Fall, bei dem die Eingabe bis zu einem Zeitpunkt die Ausgabe im nächsten Zeitschritt festlegt. Im nichtdeterministischen Fall (mehrere Ausgabeströme zu einem Eingabestrom) müssen die Mengen der möglichen zur Zeit t ausgegebenen Zeichen gleich sein.

Verhalten von Verbindungsinstanzen III

- Selbst unter Berücksichtigung von Lokalität und Kausalität gibt es noch viele Möglichkeiten, wie sich eine Verbindung verhalten kann:
  - Sie kann Nachrichten verlieren oder nicht.
  - Sie kann Nachrichten verfälschen / umwandeln oder nicht.
  - Sie kann selbständig Nachrichten erzeugen oder nicht.
  - Sie kann Nachrichten puffern oder nicht.
  - Sie kann die Reihenfolge von Nachrichten bewahren oder nicht.
  - Sie kann unterschiedliche Laufzeiten für Nachrichten bieten (ggf. abhängig von der Größe der Nachrichten).
  - Sie kann Nachrichten nur auf Anforderung ausliefern.
- Der Architekt muss die Eigenschaften von Verbindungen in seinem Laufzeitsystem kennen, um die Architektur einer Anwendung erarbeiten zu können.

Verhalten von Verbindungsinstanzen IV - Beispiele



- A) Sämtliche Nachrichten werden im nächsten Zeitschritt am anderen Ende ausgeliefert.
  - $get_t(c, i) = m \Rightarrow put_{t+1}(c, k) = m$
  - z.B. Objekte als Komponenten, Verweise als Verbindungen
- B) Nachrichten werden gepuffert und müssen vom Empfänger explizit angefordert werden (durch Senden einer speziellen Anforderungsnachricht).
  - über Definition einer Queue, Definition spezieller Nachrichten in der Menge M, Definition des Protokolls
  - z.B. Objekte als Komponenten, JMS-Kanäle als Verbindungen

Verhalten von Schnittstelleninstanzen I

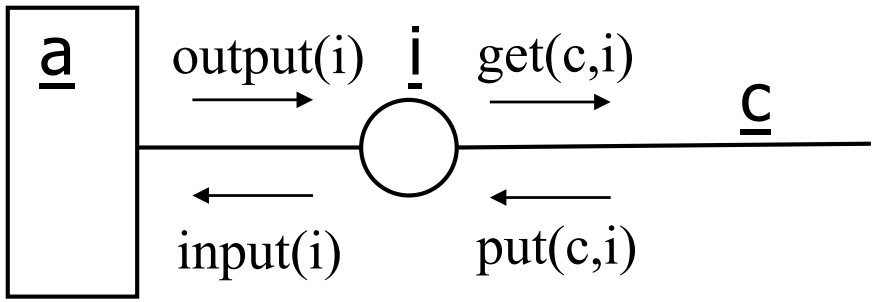
- Das Verhalten einer Schnittstelleninstanz lässt sich ähnlich wie das Verhalten einer Verbindung definieren und bezieht sich auf dieses.
- Definitionen: Schnittstellenverhalten
  - $\text{behavior} : \text{INTERFACE} \rightarrow \wp(\text{get}^T \times \text{put}^T \times \text{input}^T \times \text{output}^T)$
- Lokalität: Das Verhalten einer Schnittstelle i macht nur Aussagen über Nachrichten an der Schnittstelle und an den Enden der Verbindungen, die mit der Schnittstelle verbunden sind. Über alle anderen Nachrichten wird nichts ausgesagt (Unterspezifikation):
  - $(\text{gets}, \text{puts}, \text{inputs}, \text{outputs}) \in \text{behavior}(i) \Rightarrow \forall t \in T:$
    $\text{gets}_t, \text{puts}_t \in \{ \{(c,i)\} \rightarrow M \mid c \in \text{connections}(i)\} \wedge$
    $\text{inputs}_t, \text{outputs}_t \in \{ \{i\} \rightarrow M \}$

Menge der Verbindungen an i

Verhalten von Schnittstelleninstanzen II

- Kausalität wird ähnlich definiert wie bei Verbindungen.
- Wie bei Verbindungen gibt es viele Möglichkeiten, wie sich eine Schnittstelle verhalten kann, insbesondere:
  - Sie kann Nachrichten von mehreren Verbindungen mischen (nach einer bestimmten Strategie, beispielsweise fair).
  - Sie kann Nachrichten an Verbindungen schicken (an eine bestimmte Verbindung oder an alle).
- Das Verhalten einer Schnittstelleninstanz muss mit dem Verhalten der angrenzenden Verbindungsinstanzen zusammenpassen.
- Der Architekt muss die Eigenschaften von Schnittstellen in seinem Laufzeitsystem kennen, um die Architektur einer Anwendung erarbeiten zu können.

Verhalten von Schnittstelleninstanzen III – Beispiel



- „Vielzweck-Schnittstelle“: Sei i ∈ INTERFACE
  - hinausgehende Nachrichten gehen an alle Verbindungen (Schnittstelle als Verteiler - broadcasting)

    $c \in connections(i) \wedge alive_t(i) \wedge alive_{t+1}(c) \Rightarrow get_{t+1}(c,i) = output_t(i)$

    z.B. verwendbar, um JMS-Topics zu modellieren
  - hineingehende Nachrichten kommen aus einer Verbindung (Schnittstelle als Mischer):

    $alive_t(i) \wedge input_t(i) = m \Rightarrow$

    $\exists\, c \in connections(i) : alive_{t-1}(c) \wedge put_{t-1}(c, i) = m$

Verhalten von Komponenteninstanzen

- Das Kommunikationsverhalten einer Komponente spezifiziert, wie sie Nachrichten empfängt und versendet.
- Definition:
  - $\text{behavior} : \text{COMPONENT} \rightarrow \wp(\text{input}^T \times \text{output}^T)$
- Das Verhalten einer Komponenteninstanz muss gewissen Regeln gehorchen:
  - Es muss lokal sein: Die Komponente regelt nur ihre eigene Kommunikation, nicht die anderer Komponenten.
  - Es muss kausal sein: Die Komponente kann nicht in die Zukunft schauen.

Kommunikationsverhalten von Komponenten

- Lokalität: Das Kommunikationsverhalten einer Komponente c macht nur Aussagen über input- und output-Nachrichten von Schnittstellen von c.
  - analog zu Lokalität bei Verbindungen und Schnittstellen
- Kausalität: Die Ausgaben einer Komponente c zu einem bestimmten Zeitpunkt t können nur von den Eingaben und dem Strukturverhalten von c bis zum Zeitpunkt t-1 abhängen.
  - $(inputs1, outputs1, structures1) \in behavior(c) \wedge$
    $(inputs2, outputs2, structures2) \in behavior(c) \wedge$
    $inputs1{\downarrow}t\text{-}1 = inputs2{\downarrow}t\text{-}1 \wedge$
    $structures1{\downarrow}t\text{-}1 = structures2{\downarrow}t\text{-}1 \Rightarrow$
    $outputs1_t = outputs2_t$

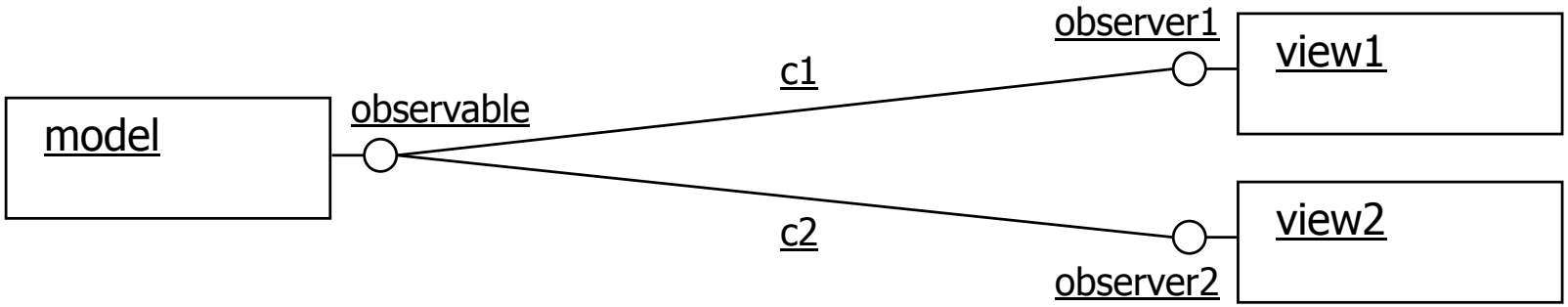
deterministischer Fall

Komposition von Instanzen und Verhalten

- Instanzen lassen sich in ein System einbauen, wenn ihr Verhalten zueinander kompatibel ist.
- Zulässiges Systemverhalten ergibt sich als Schnittmenge der zulässigen Verhalten der Komponenteninstanzen.

- Beispiel: Komponenteninstanzen a, b jeweils mit eigenem Kommunikationsverhalten
  - $\text{systembehavior} = \text{behavior}(a) \cap \text{behavior}(c)$

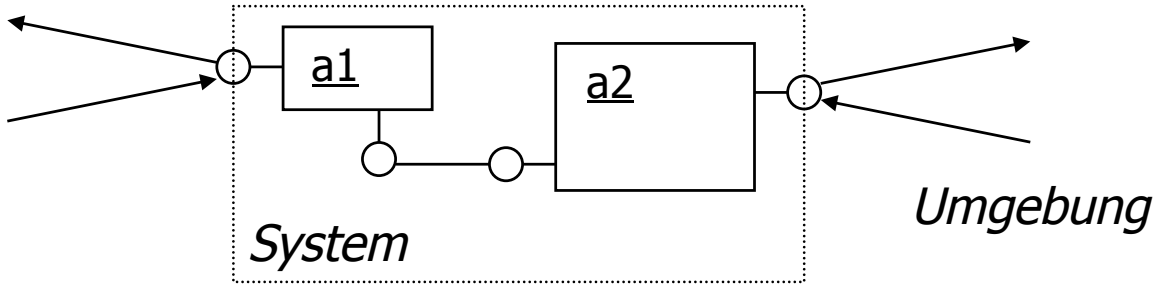
Systemverhalten am Beispiel des Observers



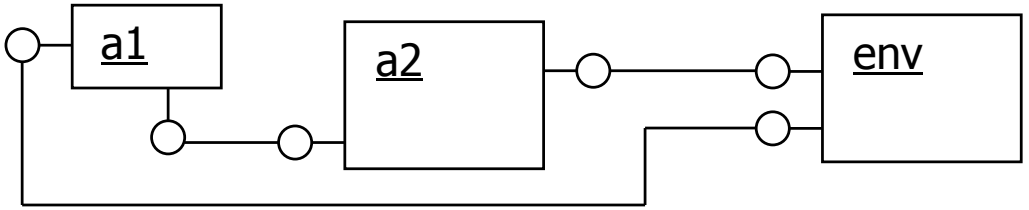
- Beispielablauf:
  - t_1: $output_{t1}$(observer1) = „changeTo(7)“
  - t_2: get_{t2}(c1, observer1) = „changeTo(7)“
  - t_3: put_{t3}(c1, observable) = „changeTo(7)“
  - t_4: $input_{t4}$(observable) = „changeTo(7)“
  - t_5: $output_{t5}$(observable) = „notify(danger)“
  - t_6: get_{t6}(c1, observable) = get_{t6}(c2,observable) = „notify(danger)“
  - t_7: put_{t7}(c1, observer1) = „notify(danger)“
  - t8: put_{t8}(c2, observer2) = „notify(danger)“

Kommunikation mit der Außenwelt

- In einem System kommunizieren typischerweise einige Komponenten mit der Umgebung.
- Darstellungsmöglichkeiten:
  - Umgebung kommuniziert über freie Schnittstellen:



  - Außenwelt wird als eigene Komponente repräsentiert:



Exkurs: Zustand

- In dem Systemmodell gibt es zwei Arten von Zustand.
- Daten-Zustand von Instanzen
  - Instanzen können Zustand halten (Zustände einnehmen).
  - Beispiel: Speicherung des Kontostands in einem Konto-Objekt.
  - Den Daten-Zustand modellieren wir nur implizit über das Kommunikationsverhalten: Die Ausgabe einer Instanz zu einem Zeitpunkt t kann grundsätzlich von allen früheren Eingaben abhängen.
  - Instanzen, deren Ausgaben jeweils nur von der letzten nichtleeren Eingabe abhängen, nennen wir zustandslos.
- Struktur-Zustand des Systems
  - Falls die Struktur dynamisch ist, kann das System sie zur Speicherung von Informationen nutzen.
  - Beispiel: Referenzen zwischen Kunden- und Konten-Objekten

Inhalt

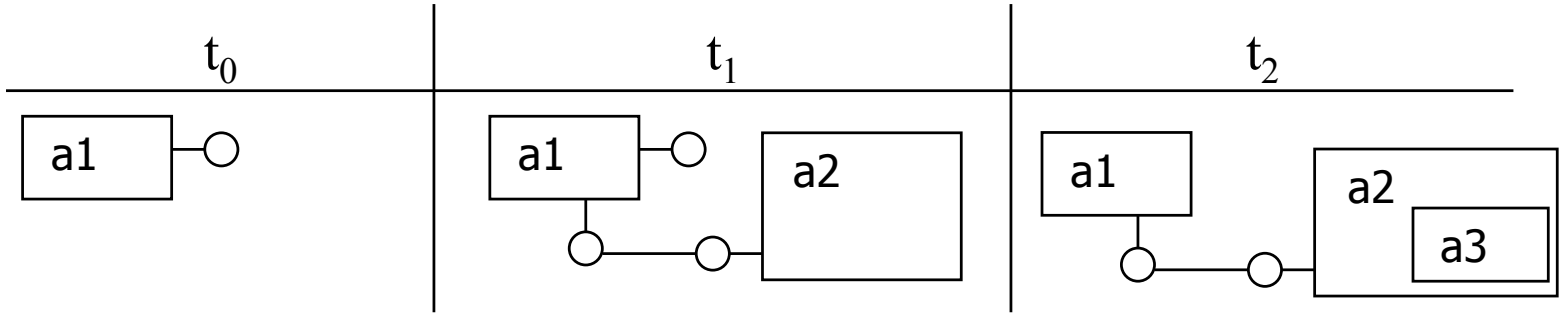
- Motivation und Darstellungsweise
  - Motivation und Darstellung
  - Komponenten und Instanzen
- Systemmodell für Architektur: Instanzen
  - Grundkonzepte und Strukturen
  - Zeit, Kommunikation und Interaktion
  - Strukturänderungen
  - Mobilität
- Systemmodell für Architektur: Beschreibungen
  - Definition und Bedeutung
  - Reflexivität
- Zusammenfassung

Strukturänderungen

- Bis jetzt konnten wir nur statische Systemstrukturen beschreiben.
- Die Struktur vieler Systeme ist aber dynamisch:
  - Komponenten und Schnittstellen kommen hinzu oder fallen weg
  - Verbindungen werden gezogen oder gelöscht
  - Systemstrukturen ändern sich
- Die Strukturänderungen können dabei unterschiedliche Ursachen haben:
  - Das System selbst kann seine Struktur programmgesteuert verändern.
  - Instanzen des Systems können ausfallen und so die Struktur verändern.

Modellierung der Strukturänderungen

- Definitionen:
  - $\mathrm{STRUCTURE}^T = \mathrm{ALIVE}^T \times \mathrm{ASSIGNED}^T \times \mathrm{PARENT}^T \times \mathrm{CONNECTS}^T$
- Ein Element aus $\mathrm{STRUCTURE}^T$ beschreibt einen möglichen Systemablauf aus Struktursicht.
- Beispiel:



- Das Strukturverhalten eines Systems ist darstellbar als die Menge all seiner möglichen Systemabläufe.
  - $\mathrm{systembehavior} \in \wp(\mathrm{STRUCTURE}^T)$

Konsistenzbedingungen für die Struktur

- Nicht sämtliche möglichen Abläufe sind erlaubt. Eine Vielzahl von Konsistenzbedingungen regelt, welche Abläufe zulässig sind:
  - Tote kehren nicht wieder:
    $\forall\ i \in \mathrm{INSTANCE} : \neg\exists\ r, s, t \in T: r < s < t \wedge \mathrm{alive}_r(i) \wedge \neg\mathrm{alive}_s(i) \wedge \mathrm{alive}_t(i)$
  - Interfaces wechseln ihre Komponente nicht:
    $\mathrm{assigned}_t(i) = c \Rightarrow \mathrm{assigned}_{t+1}(i) = c$
  - Interfaces leben nicht ohne ihre Komponente:
    $\mathrm{assigned}(i) = c \Rightarrow \forall\ t \in T:\ \mathrm{alive}_t(i) \Rightarrow \mathrm{alive}_t(c)$
- Weitere grundlegende Konsistenzbedingungen:
  - Verbindungen verbinden immer die selben Schnittstellen.
  - Am Anfang der Zeit leben nur endlich viele Instanzen.
  - Von einem Zeitschritt zum nächsten gibt es nur endlich viele Änderungen an der Struktur.

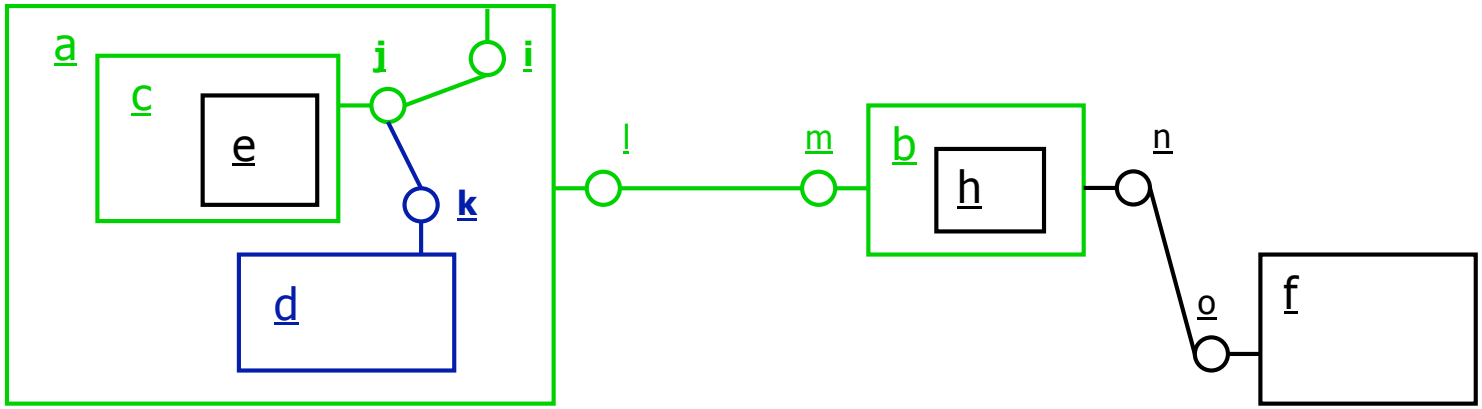
Struktur-Änderungsverhalten von Komponenten

- Für die meisten Strukturänderungen sind Komponenten verantwortlich.
- Wir erweitern deshalb die Definition des Verhaltens einer Komponenteninstanz um das Strukturänderungs -Verhalten.
- Definition:
  - $\text{behavior} : \text{COMPONENT} \rightarrow \wp(\overbrace{\text{INPUT}^T \times \text{OUTPUT}^T}^{\text{Kommunikations-verhalten}} \times \overbrace{\text{STRUCTURE}^T}^{\text{Strukturänderungs-verhalten}})$
- Das Strukturänderungsverhalten von Komponenten kann unterschiedlich definiert sein - dabei muss es aber immer lokal und kausal sein.

Strukturänderungsverhalten von Komponenten I

- **Lokalität:** Das Strukturänderungsverhalten einer Komponente c macht nur Aussagen über Instanzen, auf die die Komponente zugreifen kann.
- Hier gibt es unterschiedliche Möglichkeiten. Zugreifbar sind im Normalfall mindestens:
  - die Komponente c selbst
  - die Schnittstellen der Komponente c
  - (bestimmte) Verbindungen an Schnittstellen von c
  - (bestimmte) Schnittstellen und Komponenten, die unmittelbar über hinausgehende Verbindungen mit c verbunden sind
- Zusätzlich können zugreifbar sein:
  - die Kind-Komponenten von c und Verbindungen zwischen ihnen

Strukturänderungsverhalten von Komponenten II



- Die zugreifbaren Instanzen zu einem Zeitpunkt t können jeweils durch eine Formel beschrieben werden:
  - $reachable_t(c) =$
    1. $\{c\} \cup$
    2. $\{i \mid assigned_t(i) = c\} \cup$
    3. $\{con \mid connected_t(con) = (i,j) \wedge assigned_t(i) = c\} \cup$

       ...

Strukturänderungsverhalten von Komponenten III

- Lokalität als Formel: Das Strukturänderungsverhalten einer Komponente c macht nur Aussagen über Instanzen, auf die die Komponente zugreifen kann.
  - $(\text{inputs}, \text{outputs}, \text{structures}) \in \text{behavior}(c) \wedge$
    $\text{structures} = (\text{alives}, \text{assigned}, \text{parent}, \text{connects}) \Rightarrow \forall t \in T:$
    $\text{alives}_t \in$
    $\{ x \mid x \in \text{COMPONENT} \cap \text{reachable}_t(c) \} \rightarrow M \} \wedge$
    $\text{assigned}_t \in$
    $\{ x \mid x \in \text{INTERFACE} \cap \text{reachable}_t(c) \} \rightarrow \text{COMPONENT} \} \wedge$
    $\text{parent}_t \in$
    $\{ x \mid x \in \text{COMPONENT} \cap \text{reachable}_t(c) \} \rightarrow \text{COMPONENT} \} \wedge$
    $\text{connects}_t \in$
    $\{ x \mid x \in \text{CONNECTION} \cap \text{reachable}_t(c) \} \rightarrow$
    $\text{INTERFACE} \times \text{INTERFACE} \}$

Zwischenstand I

- Bis jetzt haben wir ein grundlegendes Vokabular für Architektur definiert.
  - Instanzen von Komponenten, Schnittstellen und Verbindungen
  - Strukturen aus Instanzen sowie Strukturänderungen
  - Kommunikation zwischen Komponenten über Schnittstellen und Verbindungen
  - Verhalten einzelner Instanzen sowie Komposition von Instanzen
- These: Das grundlegende Systemmodell ist allgemein genug, um über die Architektur aller möglichen Software-Systeme reden zu können.
- Es definiert also allgemeine Eigenschaften des Laufzeitsystems einer „Architekturmaschine“.

Zwischenstand II

- Auf Basis des grundsätzlichen Laufzeitsystems lassen sich eine Reihe von zusätzlichen Definitionen aufsetzen, die spezielle Architekturen beschreiben, zum Beispiel:
  - Nachrichtenlaufzeiten von Verbindungen
  - Schnittstellen, die als Nachrichtenverteiler dienen
  - Komponenten, die andere Komponenten erzeugen können
- Derartige zusätzliche Annahmen müssen explizit gemacht werden, wenn man über eine Architektur spricht.
- Die bisher gezeigten Konzepte sollen insbesondere dazu dienen, den Blick dafür zu schärfen, auf welche Aspekte man achten muss.

Beispiele

- Komponenten sind große Anwendungen, die sich über http -Verbindungen XML-Dokumente schicken.
  - Übermittlung von Nachrichten nur mit (ungewisser) Verzögerung
  - Verbindungen können ausfallen
  - Nachrichten werden bei der Übertragung nicht verfälscht
  - Komponentenstruktur ändert sich nicht (oder nur selten)
  - ...
- Komponenten sind Objekte innerhalb einer Anwendung, die über Referenzen miteinander verbunden sind und Methoden aufeinander aufrufen.
  - Übermittlung von Nachrichten im Wesentlichen verzögerungslos
  - Verbindungen können nicht ausfallen
  - Nachrichten gehen nicht verloren
  - Nachrichten sind Aufrufe (Request/Reply)
  - Komponentenstruktur ändert sich häufig
  - ...

Inhalt

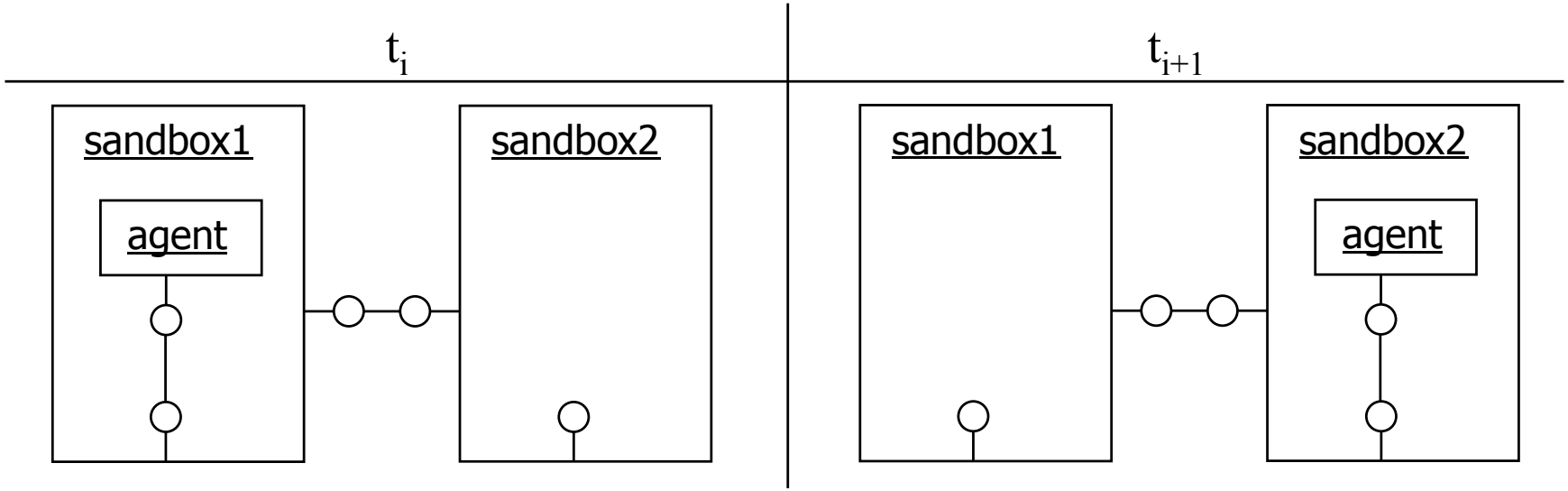
- Motivation und Darstellungsweise
  - Motivation und Darstellung
  - Komponenten und Instanzen
- Systemmodell für Architektur: Instanzen
  - Grundkonzepte und Strukturen
  - Zeit, Kommunikation und Interaktion
  - Strukturänderungen
  - Mobilität
- Systemmodell für Architektur: Beschreibungen
  - Definition und Bedeutung
  - Reflexivität
- Zusammenfassung

Mobilität

- Naive Definition: Eine Komponente ist mobil, wenn sie innerhalb eines Systems ihren Ort wechseln kann.
- Beispiele:
  - Ein mobiler Software-Agent, der innerhalb eines Netzwerks von einem Rechner zum nächsten wandern kann.
  - Ein Handy (Cell Phone) mitsamt seiner Software, das innerhalb des GSM-Netzes seine Funkzelle wechselt.
- Mobilität lässt sich auf mehrere Arten definieren.
  - Implizit: Wir fassen Orte als (ggf. spezielle) Komponenten auf. Mobilität lässt sich dann als zeitliche Änderung der Abbildung parent begreifen.
  - Explizit: Wir führen Orte als explizites Konzept neben den anderen Basiskonzepten ein.

Mobilität – implizite Variante

- Orte sind Komponenten
- Mobilität ist die zeitliche Änderung der Abbildung parent.



- Einfaches Modell, nur für Spezialfälle geeignet:
  - Modellierung spezieller Eigenschaften von Komponenten-Orten (beispielsweise räumliche Koordinaten) unhandlich
  - keine Darstellung überlappender Orte möglich

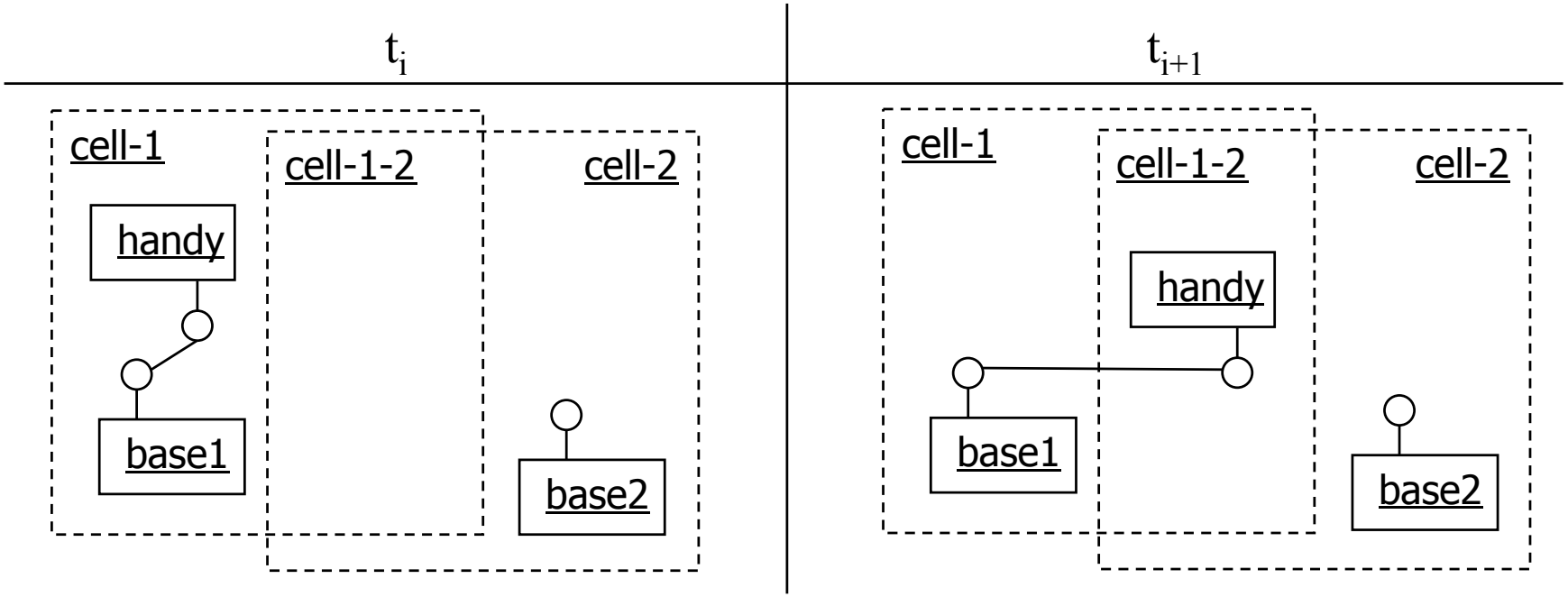
Mobilität – explizite Variante

- Es gibt ein spezielles Konzept für Orte.
- Grundlegende Definitionen:
  - LOCATION : Menge aller Orte

    loc
  - located : COMPONENT → LOCATION
    Totale Abbildung; ordnet jeder Komponente den Ort zu, an dem sie sich befindet.
  - sublocation : LOCATION → $\wp$(LOCATION)
    Totale Abbildung, definiert die Enthaltensein-Struktur der Orte.
- Modell ist komplexer, bietet aber höhere Mächtigkeit:
  - benötigt zusätzliche Konsistenzbedingungen z.B. für Azyklizität von sublocation, für Änderungen der Ortsstruktur, für Zusammenhang zwischen parent, sublocation, located etc.)
  - erlaubt natürliche Modellierung der Eigenschaften von Orten
  - erlaubt überlappende Orte – Komponente kann sich so an mehreren Orten gleichzeitig befinden

Mobilität – Beispiel für explizite Variante

- Redefinition (auch located ist veränderlich): $\text{STRUCTURE}^T = \text{alive}^T \times \text{assigned}^T \times \text{parent}^T \times \text{connects}^T \times \text{located}^T$
- cell-1, cell-2, cell-1-2 ∈ LOCATION; handy ∈ COMPONENT
- sublocation(cell-1-2) = {cell-1, cell-2}
- $\text{located}_{ti}(\text{handy}) = \text{cell-1}$; $\text{located}_{ti+1}(\text{handy}) = \text{cell-1-2}$



Inhalt

- Motivation und Darstellungsweise
  - Motivation und Darstellung
  - Komponenten und Instanzen
- Systemmodell für Architektur: Instanzen
  - Grundkonzepte und Strukturen
  - Zeit, Kommunikation und Interaktion
  - Strukturänderungen
  - Mobilität
- Systemmodell für Architektur: Beschreibungen
  - Definition und Bedeutung
  - Reflexivität
- Zusammenfassung

Beschreibungen - Motivation

- Der Architekt arbeitet aber meist nicht mit einem laufenden System (es sei denn, er erstellt einen Prototyp), sondern mit Beschreibungen des Systems.
- Hier stellen sich insbesondere folgende Fragen:
  - Was sind Beschreibungen?
  - Was sagen Beschreibungen über die Instanzen aus?
  - Welche Beziehungen gibt es zwischen den Beschreibungen?

Beschreibungen – Definition und Beispiele

- Definitionen:
  - DESCRIPTION : Menge aller möglichen Beschreibungen
  - describedBy : DESCRIPTION → $\wp$(INSTANCE)
    Jede Beschreibung kann viele Instanzen beschreiben.
- Beispiele für Beschreibungen:
  - Spezifikationen (*.doc-Dateien)
  - UML-Diagramme (*.mdl-Dateien)
  - Schnittstellendefinitionen (*.idl-Dateien)
  - Quellcode (*.java-Dateien)
  - Binaries (*.class-Dateien)
- Zwischen Beschreibungen gibt es viele mögliche Beziehungen (import, derived, etc.)

Bedeutung von Beschreibungen I

- Beschreibungen entsprechen Prädikaten; sie machen Vorgaben für Struktur und Verhalten der Instanzen.
- Beispiel: IDL-Datei Counter.idl

```
interface Counter {
    long getSum();
    void setSum(long x);
    long increment();
}
```

Counter.idl ist eine Beschreibung:
$\text{CounterIDL} \in \text{DESCRIPTION}$

- Standardumsetzung in Prädikate:
  - Die Beschreibung beschreibt nur (bestimmte) Schnittstellen:
    $i \in \text{describedBy}(\text{CounterIDL}) \Rightarrow i \in \text{INTERFACE}$

Bedeutung von Beschreibungen II

- Über Schnittstellen, die der Beschreibung gehorchen, laufen nur bestimmte Nachrichten:

  $i \in \text{describedBy}(\text{CounterIDL}) \wedge$

  $(\text{gets}, \text{puts}, \text{inputs}, \text{outputs}) \in \text{behavior}(i) \Rightarrow \forall\, t \in T:$

  $\quad \text{outputs}_t(c,i) \in \{\, \text{getSum}, \text{setSum}(n), \text{increment} \mid n \in \text{Nat} \,\} \wedge$

  $\quad \text{inputs}_t(c,i) \in \text{Nat}$

- Weiterhin kann noch die Semantik des entfernten Methodenaufrufs vorgegeben werden: Nach Senden von getSum muss zunächst eine Antwort erfolgen, bevor eine weitere Ausgabe möglich ist.
- Weitere Möglichkeiten: Exception-Handling spezifizieren, Anreicherung mit Vor- und Nachbedingungen, etc. etc.

Reflexivität

- Ein reflexives System ist ein System, bei dem Instanzen zur Laufzeit auf (bestimmte) Informationen aus den Beschreibungen zugreifen können.
- Typische Modellierung: Die Beschreibungen sind zur Laufzeit als Instanzen repräsentiert (z.B. Meta-Klassen).
  - Erfordert keine Erweiterung des Verhaltensbegriffs.
- Komplexe Modellierung: Instanzen greifen zur Laufzeit auf Beschreibungen zu.
  - Erfordert Erweiterung des Verhaltensbegriffs

$$\text{behavior} \in \wp(\text{STRUCTURE}^T \times \text{GET}^T \times \text{PUT}^T \times \text{INPUT}^T \times \text{OUTPUT}^T \times \text{DESCRIPTIONS}^T))$$

Inhalt

- Motivation und Darstellungsweise
  - Motivation und Darstellung
  - Komponenten und Instanzen
- Systemmodell für Architektur: Instanzen
  - Grundkonzepte und Strukturen
  - Zeit, Kommunikation und Interaktion
  - Strukturänderungen
  - Mobilität
- Systemmodell für Architektur: Beschreibungen
  - Definition und Bedeutung
  - Reflexivität
- Zusammenfassung

Zusammenfassung

- Systemmodelle erlauben eine abstrakte Darstellung der wesentlichen Aspekte von Systemen.
- Komponenten, Schnittstellen und Verbindungen sind die Grundbausteine jeder Architektur.
- Der Architekt muss die jeweiligen Eigenschaften dieser Grundbausteine in seinem System kennen, um die Architektur einer Anwendung erarbeiten zu können.
- Für das Verhalten eines Systems sind sowohl Struktur als auch Kommunikation relevant.
- Für den Architekten sind sowohl Instanzen als auch Beschreibungen relevant.

Literaturhinweise

- [BR95] M. Broy: *Mathematical System Models as a Basis of Software Engineering*, J. van Leeuwen (ed.), Computer Science Today, Springer -Verlag 1995.
- [BHH+] R. Breu, U. Hinkel, C. Hofmann, C. Klein, B. Paech, B. Rumpe, V. Thurner: *Towards a Formalization of the Unified Modeling Language*, In: Proceedings of the ECOOP´97, Springer-Verlag, LNCS, 1997.
- [BRS+] K. Bergner, A. Rausch, M. Sihling, A. Vilbig, M. Broy: *A Formal Model for Componentware*, Gary T. Leavens, Murali Sitaraman (eds.), Foundations of Component-Based Systems, Cambridge University Press, 2000.
- [BR] M. Broy, B. Rumpe: Modulare hierarchische Modellierung als Grundlage der Software- und Systementwicklung. Informatik-Spektrum Band 30, Heft 1, Februar 2007, Springer-Verlag