

Entwurf und Koordination von kontextsensitiven Diensten für intelligente Heimumgebungen auf Basis von erweiterten Feature Modellen

Verteidigung der Masterarbeit



Sebastian Garn
sebastian.garn@posteo.de
TU-Berlin

Gliederung

- Prognose & Problem
- Anforderungen
- Lösungsansatz
 - Framework
 - Laufzeitkomponente
- Ergebnisse
- Ausblick & Zusammenfassung

Prognose

- Ablösung von Insellösungen
 - Nutzer sucht sich in Store o.ä. Dienste für intelligente Wohnung zusammen
 - Vergleichbar mit Entwicklung Smartphone-Markt
 - Ermöglicht Individualisierung; fördert Flexibilität
- Bedeutet: Entwicklung von Diensten für intelligente Heimumgebungen erfolgt verteilt

Problem




- Verteilte Entwicklung von Diensten: jeder Dienst in sich abgeschlossene Einheit
 - Dienste sind auf Erfüllung eigener Aufgabe bedacht
 - laufen jedoch gemeinsam in einer Heimumgebung
- > Wechselwirkungen zwischen Diensten können auftreten! Speziell: Zugriff auf gleiche Ressourcen, zukünftig **Kollisionen** genannt**

Problembeispiel

Lichtsteuerung vs. Klimasteuerung

- Lichtsteuerung
 - Führt Jalousien hoch um Stromkosten zu senken (Daylight-Harvesting)
- Klimasteuerung
 - Führt Jalousien herunter um Sonneneinstrahlung (und somit Temperatur) zu vermindern
- Beide Dienste haben gleiches Ziel (Energiekosten verringern), behindern sich jedoch gegenseitig

Lösungsmöglichkeiten

- Kein Dienst darf adaptieren 
 - nicht im Interesse des Nutzers
- Ein Dienst darf adaptieren 
 - Wer entscheidet?
 - Ist die Entscheidung Situationsgebunden?
- Beide Dienste dürfen *mit Einschränkungen* adaptieren 
 - Verwendung von Alternativverhalten

Gesucht

- Ein System, dass Dienste untereinander koordinieren kann
 - Überprüfung der Ressourcen, die durch die Dienste benötigt werden zwecks **Kollisionserkennung im Vorfeld**
 - Selbstständige **Konfliktlösung**
 - Finde Konsens
 - Im Zweifelsfall muss Nutzer eingreifen

Anforderungen

- Dienste müssen **Alternativverhalten** bereithalten
- **Komplexität** für Entwickler möglichst gering halten
- **Fairness** bei Wahl des Alternativverhaltens
- **Transparenz** und **Kontrolle** für Endanwender
- **Skalierbarkeit** des Systems
- **Robustheit** – stets definierter Zustand

Wie lösen es Andere?

- **Agentenbasierte Dienste** ([CCFN11])
 - Funktionalitäten werden auf einzelne, autonome Agenten abgewälzt
- **OSGI basierende Ansätze** ([OPE], [GPZ04], [RS08])
 - Funktionalitäten werden auf Services aufgeteilt
 - Übergeordnete Komponente entscheidet über Einbinden/Entfernen von Diensten
- **Modellbasierte Ansätze** ([HSSF06], [CGFP09])
 - bspw. Feature Modell mit zusätzlichen Annotationen

Mangelhaft

- Bisherige Lösungen erlauben zwar Dienste mit Alternativverhalten (abhängig von Kontext)
- **Aber:** mangelhafte Betrachtung von Kollisionen
 - Vorgefertigte Repositories mit Funktionalitäten
 - Für verteilte Entwicklung keine überschneidungsfreien Funktionalitäten gewährleistet
 - Einfache Prioritätslisten
 - Ein Dienst darf, der andere nicht
 - Regelbasiert
 - Ausführungsreihenfolge wird über Regeln festgelegt

Lösungsansatz

- Auf zwei Ebenen:
 - Ein **Framework** zur **Konstruktion** von kontextsensitiven Diensten
 - Wichtig: einheitliche Struktur, auf die Dienste aufsetzen
 - Nötig, damit Assistenten „kommunizieren“
 - Eine (Laufzeit-) **Komponente**, die Dienste **koordiniert**

Lösungsansatz

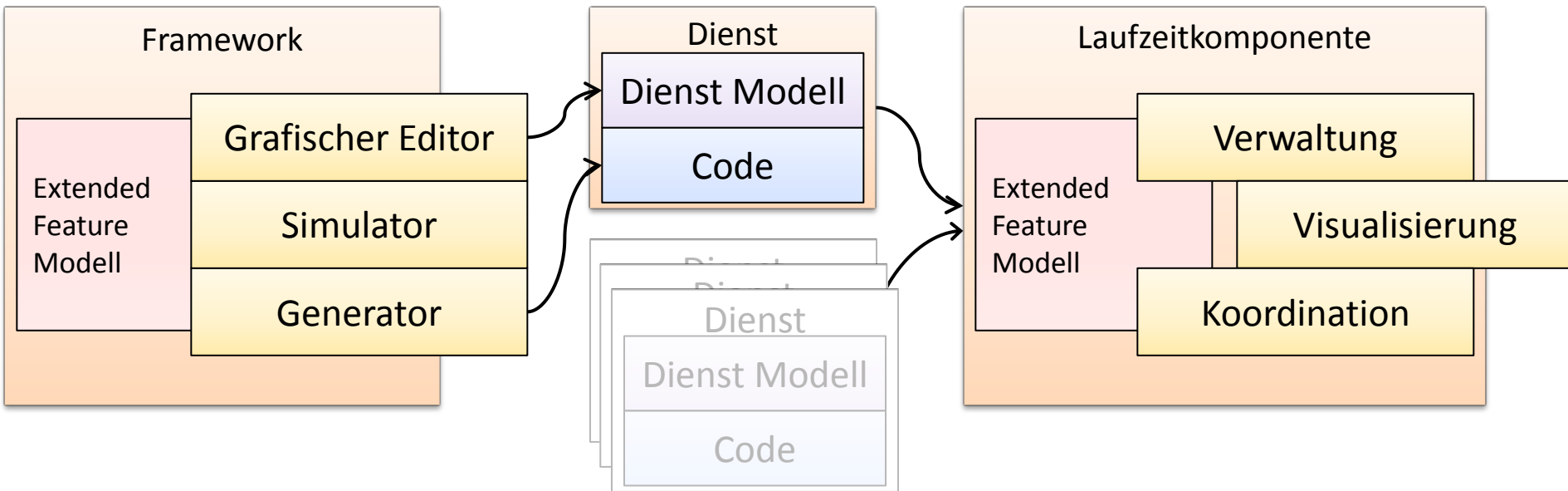


Abb. 1: Zusammenhang Framework & Laufzeitkomponente

Lösungsansatz

- Auf zwei Ebenen:
 - Ein **Framework** zur **Konstruktion** von kontextsensitiven Diensten
 - Wichtig: einheitliche Struktur, auf die Dienste aufsetzen
 - Nötig, damit Assistenten „kommunizieren“
 - Eine (Laufzeit-)Komponente, die Dienste koordiniert

Lösungsansatz

Framework: Konstruktion von Diensten

- Eclipse-Plugin zur grafischen Erstellung von Diensten
- „Grundlegende Struktur“ der Dienste bildet **Extended Feature Meta Modell**
 - Feature Modelle finden Anwendung im „Software Product Line Engineering“
 - Erzeugung von Software-Produktlinien
 - Wiederverwendung von bereits erzeugten Features
 - Jedes Produkt setzt sich aus einzelnen Features zusammen

Lösungsansatz

Framework: Konstruktion von Diensten

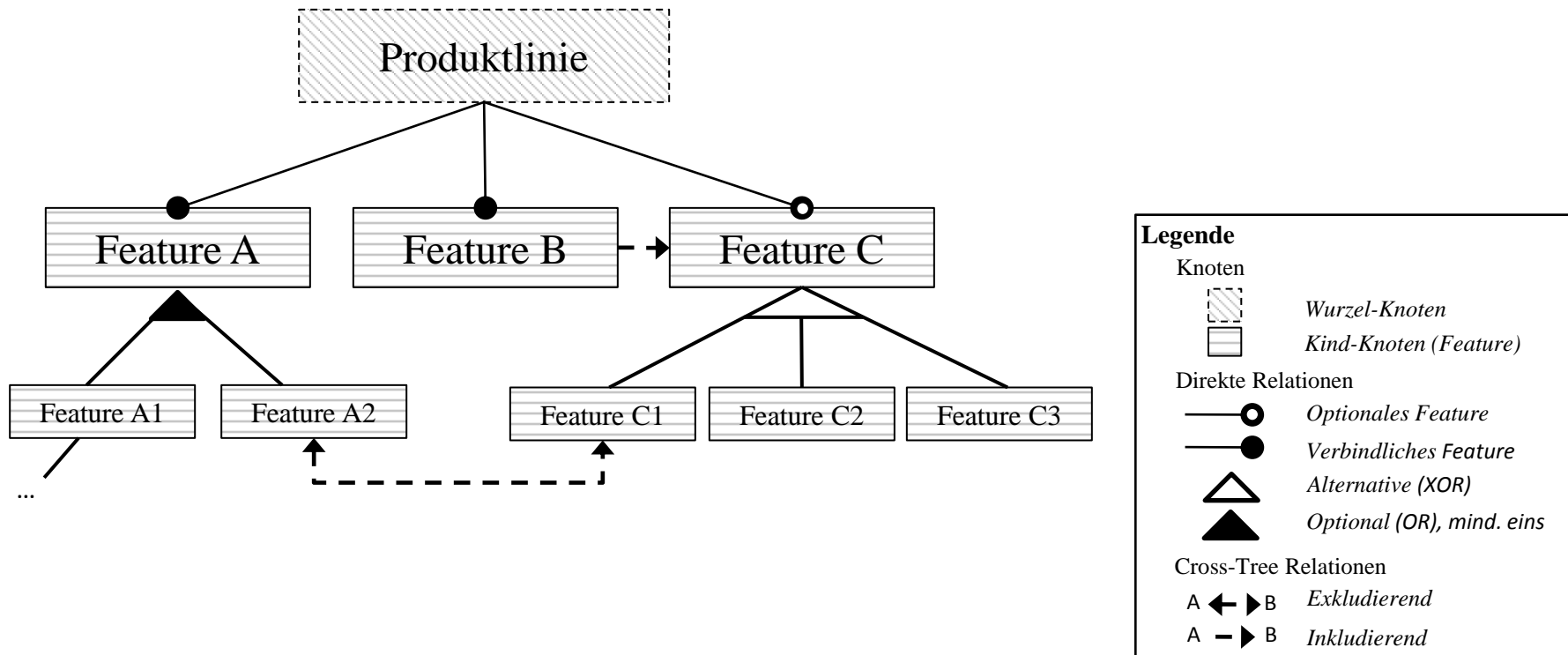


Abb. 2: Beispiel Feature Modell

Lösungsansatz

Framework: Konstruktion von Diensten

- Erweiterung einfacher Feature Modelle um
Zusätzliche Meta-Informationen und
Konstrukte

- + Leicht verständlich
- + Design und Umsetzung
nahe beieinander
- + Ermöglicht Modellierung von
Alternativverhalten

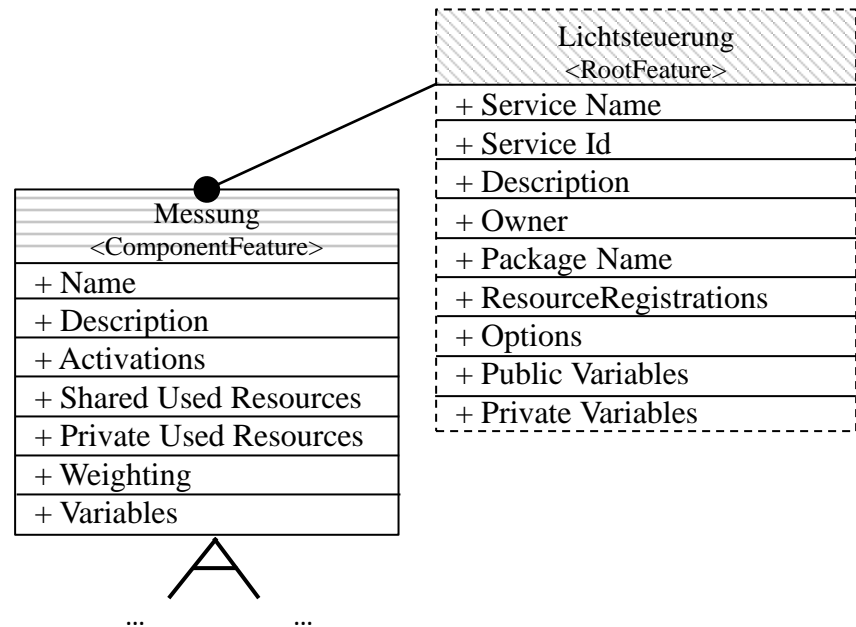


Abb. 3: Konzept Extended Feature Modell

Lösungsansatz

Framework: Überprüfung von Diensten

- Entwicklung von kontextsensitiven Diensten bereits ohne Alternativverhalten fehleranfällig
- Zusätzliche Komplexität von Diensten durch Alternativverhalten
 - **Simulator** erlaubt Überprüfung von Dienst Modellen auf Fehler über **Wertebereiche**
 - Fehlende Zustände
 - Modellinterne Konflikte
 - Fehler in Operationen

Konzept: Extended Feature Modell

Demo 1: Framework

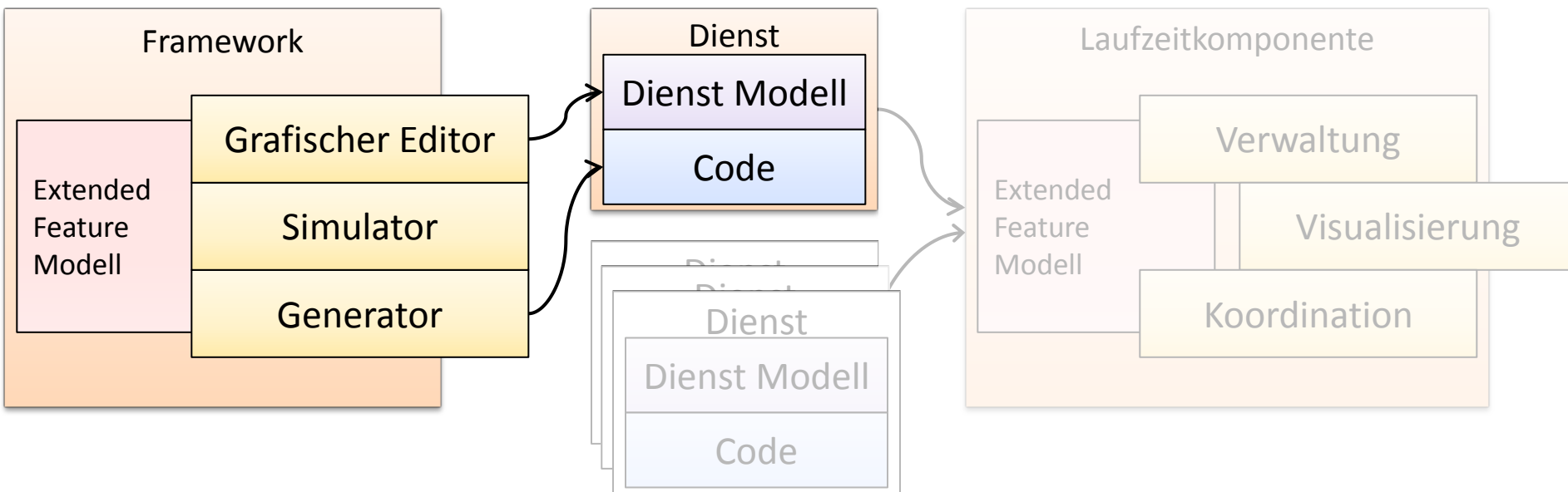


Abb. 4: Zusammenhang Framework & Laufzeitkomponente

Anforderungen - Rückblick

- Dienste müssen **Alternativverhalten** bereithalten
- **Komplexität** für Entwickler möglichst gering halten
- *Fairness* bei Wahl des Alternativverhaltens
- *Transparenz* und *Kontrolle* für Endanwender
- *Skalierbarkeit* des Systems
- *Robustheit* – stets definierter Zustand

Anforderungen - Rückblick

- Dienste müssen *Alternativverhalten* bereithalten
- *Komplexität* für Entwickler möglichst gering halten
- ***Fairness*** bei Wahl des Alternativverhaltens
- ***Transparenz*** und ***Kontrolle*** für Endanwender
- ***Skalierbarkeit*** des Systems
- ***Robustheit*** – stets definierter Zustand

Lösungsansatz

- Auf zwei Ebenen:
 - Ein **Framework** zur **Konstruktion** von kontextsensitiven Diensten
 - Wichtig: einheitliche Struktur, auf die Dienste aufsetzen
 - Nötig, damit Assistenten „kommunizieren“
 - Eine (Laufzeit-) **Komponente**, die Dienste **koordiniert**

Lösungsansatz

Laufzeitkomponente: Koordination von Diensten

- Installation, Konfiguration, Koordination und ggf. Rollback von Diensten

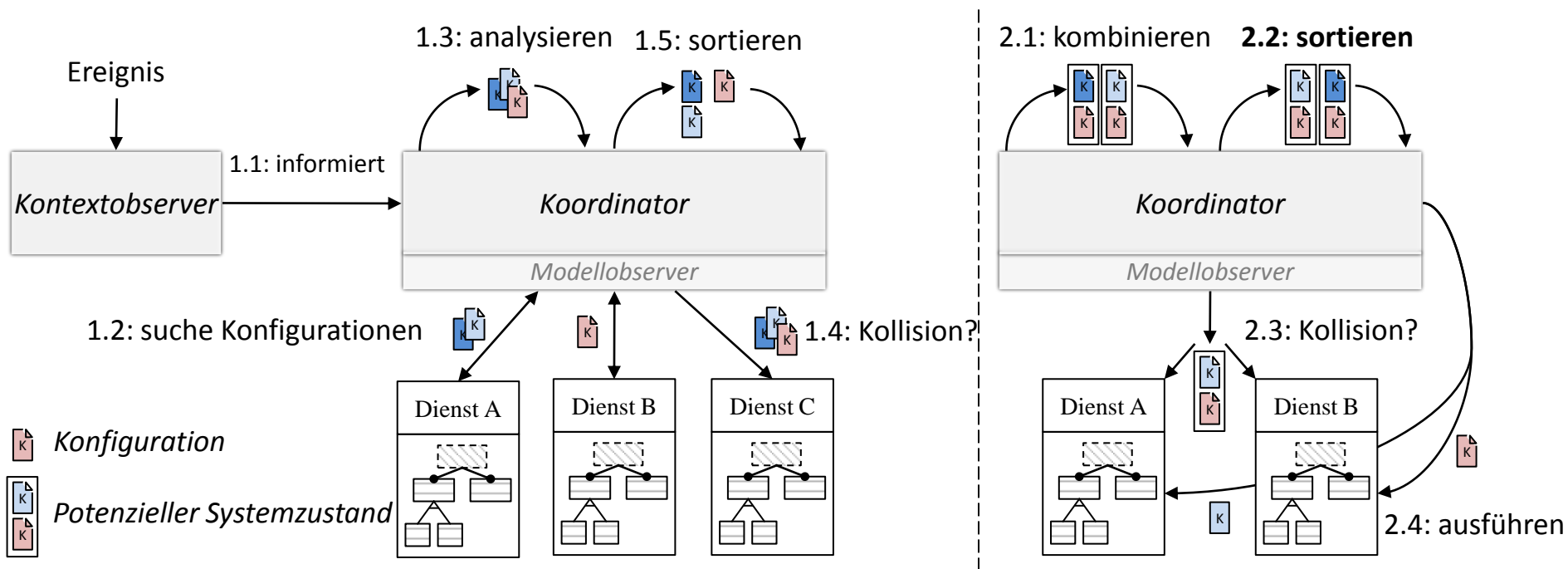


Abb. 5: Vereinfachter Koordinierungsalgorithmus

Lösungsansatz

Laufzeitkomponente: Fairness zwischen Diensten

- Dienste dürfen nicht auf Kosten anderer Dienste adaptieren

CFD	S1	S2	S3
C1	100	100	100
C2	90	40	80
C3	10	20	60

Tab 1: Beispiel CFD-Berechnung

- Definition: CFD - Context Fulfillment Degree
Wie gut ist eine vorgegebene Konfiguration eines Dienstes im Vergleich zu seiner besten Konfiguration

Lösungsansatz

Berechnung des Fairness-Faktors

$$F(K_i) = \text{sumCFD}(K_i) - \sum_{j=1}^{|S|} \text{penalty}(K_{i_j})$$

Wobei

K_i *Eine Konfigurationskonstellation*

$\text{penalty}(K_{i_j})$ *Ist Differenz zwischen Durchschnitts-CFD und CFD des Dienstes, sollte dessen CFD unterhalb des Durchschnitts liegen; sonst 0*

$|S|$ *Menge aller Dienste*

Demo 2: Laufzeitkomponente

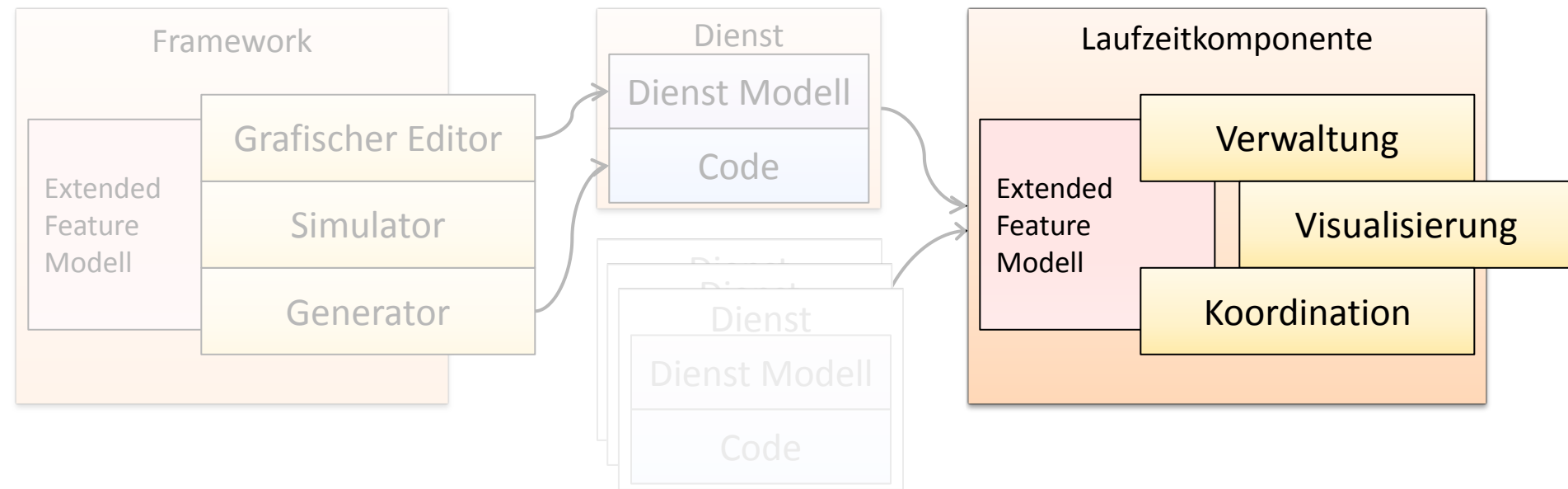


Abb. 6: Zusammenhang Framework & Laufzeitkomponente

Ergebnisse

- **Berechnung Fairness-Faktor nicht optimal**
 - Je mehr Dienste in ihr Optimum adaptieren können, desto größer ist Wahrscheinlichkeit, dass andere Dienste benachteiligt werden
- Auswirkung kann auch positiv gesehen werden
 - System berücksichtigt Fairness, aber in Grenzen

Ergebnisse

- **Bestrafung von Variabilität**
 - Durch Berechnung werden Dienste bevorzugt, die wenig Alternativverhalten anbieten
 - Dienste nach dem Alles oder Nichts-Prinzip
 - Zusätzlich entsteht subjektiver Eindruck seitens Nutzer, dass solche Dienste besser funktionieren

Ausblick

- Referenzieren anderer Modelle aus Dienst Modell heraus
 - Bspw. Abstract Resource Requirements-Modell
 - Lautsprecher benötigen Ressource Akkustik
- Benachrichtigung über freigegebene Ressourcen

Zusammenfassung

- Ansatz zur Vorhersage und Vermeidung von Ressourcenkollisionen vorgestellt
- Framework erlaubt Erzeugung von kontextsensitiven Diensten mit Alternativverhalten und ihre Simulation
 - Modellierung der Dienstlogik basierend auf Extended Feature Modellen
- Laufzeitkomponente erlaubt Koordination und Steuerung der Dienste
 - Fairness in Grenzen gewährleistet

Quellen

- [CCFN11] Davide Cavone, Berardina De Carolis, Stefano Ferilli, and Nicole Novielli, An agentbased approach for adapting the behavior of a smart home environment, WOA, 2011, pp. 105–111.
- [Ope] OpenHAB, Abruf zuletzt am 04.05.14, 21:51 uhr,
Url: <http://www.openhab.org/features-architecture.html>
- [GPZ04] T. Gu, H.K. Pung, and D.Q. Zhang, Toward an osgi-based infrastructure for contextaware applications, Pervasive Computing, IEEE 3 (2004), no. 4, 66–74.
- [HSSF06] Svein Hallsteinsen, Erlend Stav, Arnor Solberg, and Jacqueline Floch, Using product line techniques to build adaptive systems, Proceedings of the 10th International on Software Product Line Conference (Washington, DC, USA), SPLC '06, IEEE Computer Society, 2006, pp. 141–150.

Quellen

- [CGFP09] Carlos Cetina, Pau Giner, Joan Fons, and Vicente Pelechano, Using feature models for developing self-configuring smart homes, Proceedings of the 2009 Fifth International Conference on Autonomic and Autonomous Systems (Washington, DC, USA), ICAS '09, IEEE Computer Society, 2009, pp. 179–188.
- [RS08] Daniel Retkowitz and Mark Stegelmann, Dynamic adaptability for smart environments., DAIS (René Meier and Sotirios Terzis, eds.), Lecture Notes in ComputerScience, vol. 5053, 2008, pp. 154–167.

Simulator

