


Optimizing Abstract Data Types in Embedded Systems at Modeling Level

Anne Keller
Verteidigung Diplomarbeit
Mediensysteme
Bauhaus-Universität, Weimar

Erstgutachter: Prof. Dr. Benno Stein
Zweitgutachter: Prof. Dr. Bernd Fröhlich



- Einleitung
- ADT-Modell Transformationen
- Erweiterung von UML durch Metamodelle
- Embedded Systems Kostenfaktoren und ADT-Modell Transformationen
- 1 . Beispieltransformation - *Change Ordering*
 - Ergebnisse
- 2 . Beispieltransformation - *Split Container*
 - Ergebnisse
- Schlussfolgerungen und Ausblick

Embedded Systems - Einleitung

Embedded Systems

- Lebensdauer von **Batterien** ist begrenzt
 - Energieverbrauch verringern
- **Speicher** ist teuer
 - Speichernutzung verringern
- Kurze **Produkteinführungszeit**
 - Designzeit verkürzen

Problem

→ Mapping von Software auf Embedded Systems

- System hat **limitierte** Ressourcen
- Anwendungen sind **komplex**
- Input ist **dynamisch**



Lösungen für Data Mapping (Memory Allocation)

- Manuell
 - Embedded Systems Software-Designer
 - Ad hoc - nicht systematisch
 - Langsam
- Compiler
 - L0, Register-Level
 - Architektur-spezifisch (z.B. ARM Prozessor)
 - Nicht auf höherem Level der Speicher-Hierarchie, nur L0
 - Allgemeine Lösungen
- Real-time Operating System (RTOS)
 - Memory Allocation (z.B. Windows CE)
 - Allgemeine Lösungen
 - Betriebssystem verbraucht zusätzlich Ressourcen

ADT-Modell Transformationen

- Datennutzung des Systems wird modelliert
- Modellierung auf hoher Abstraktionsebene
 - Abstract Data Types - **ADT-Modelle**
- Modelle transformiert → für Embedded Systems optimiert

Resultat

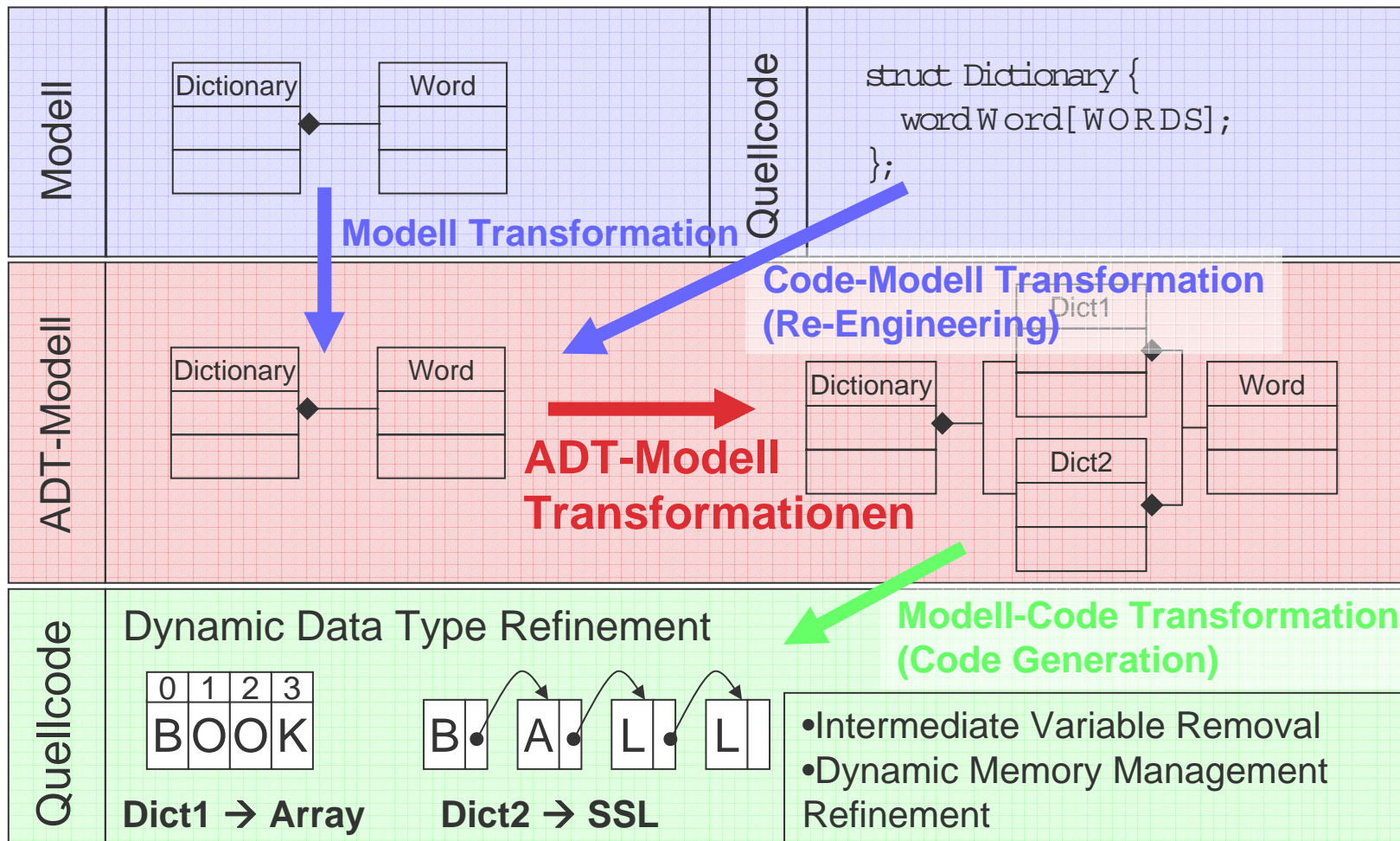
- **Systematische** Lösungen
- Optimierung für **spezielle** Domäne in Embedded Systems
 - Modellierung spezialisierten Wissens (Expertenwissens)
 - Identifikation zur Designzeit, Instantiierung zur Laufzeit
- Fokus auf **Struktur** der Daten und Verhalten
 - Abstraktion von C/C++ Implementation
- **Komplexität** von Hardware Plattformen **versteckt**
 - Früher Designabschnitt
 - Konzentration auf dynamischen Input und spezialisiertes Wissen

Fragestellung der Diplomarbeit

- Anwendbarkeit von ADT-Modell Transformationen zeigen
 - Bekannt Transformationen implementieren und validieren
 - Neue Transformationen entwickeln, beschreiben

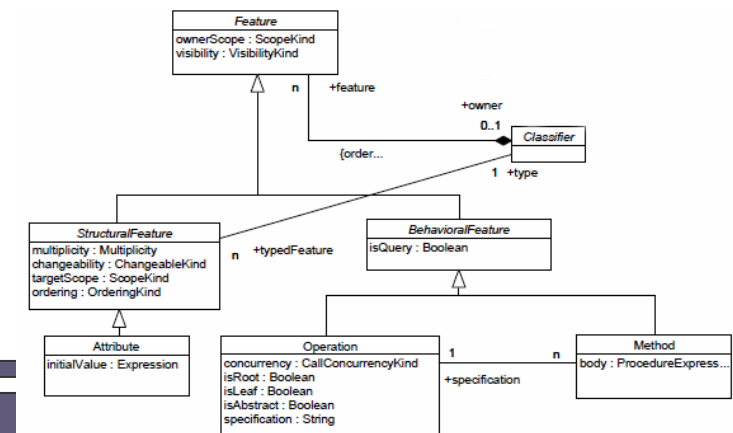
→ **Transformationskatalog**
- Transformationen beschreiben
 - schrittweise Transformationen mit Vor-Bedingungen und Nach-Bedingungen (Refactoring)
 - Mögliche Formalisierung zeigen
 - Mögliche Automatisierung erkunden

ADT-Modell Transformationen und mehr - ein Arbeitsfluss



Erweiterung von UML durch Metamodelle

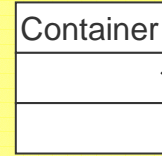
- Profile
 - Anpassung von UML an spezielle Domäne oder speziellen Zweck (z.B. Real-time UML) → *Stereotyped Packages*
 - Erweiterungstechniken: Stereotypes, Tagged Definitions, Constraints
 - **Spezifiziert Teilmenge von UML**
 - Keine Änderungen an existierendem Metamodell, nur Erweiterung
→ Light weight approach!
- Metamodelle
 - **Definiert die Modellierungssprache**
 - Änderungen in Terminologie (Plattform-spezifisch)
 - Änderungen semantischer Informationen
 - Änderungen an UML Symbolen
→ Änderungen am Metamodell
→ Heavy weight approach!



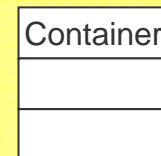
Informationen auf verschiedenen Abstraktionsebenen

Meta-Ebene

Metamodell Beschreibung

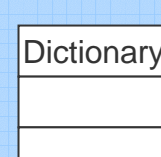
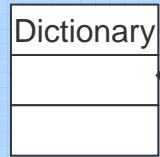


- Transformationsschritte
- 1) Container kreieren
 - 2) Attribute kopieren
 - 3) ...



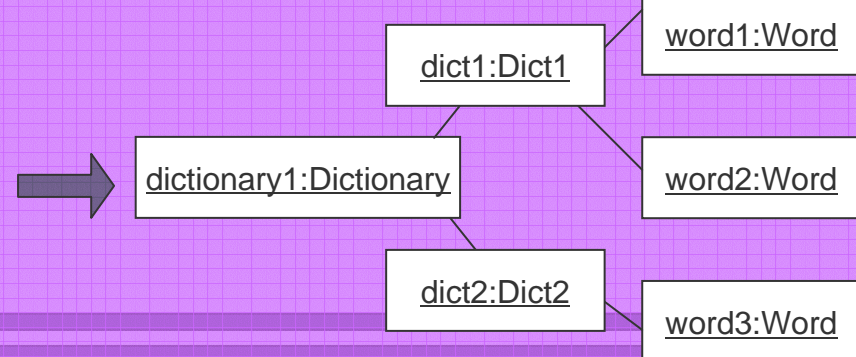
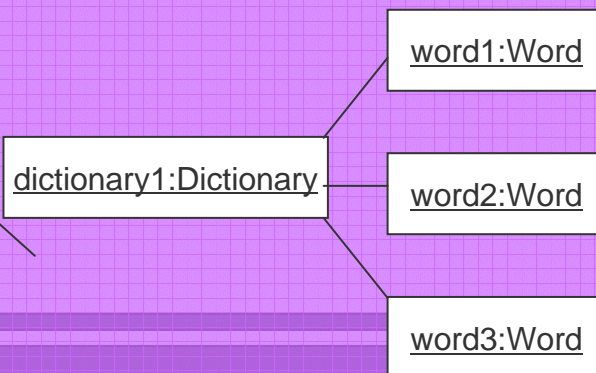
Anwendungs-Ebene

Anwendungs-spezifisches Wissen
→ *Splitting Criterion*
z. B. Buchstabenhäufigkeit



Objekt-Ebene

Behavior Preservation



ADT-Modell Transformationen und Embedded Systems Kostenfaktoren

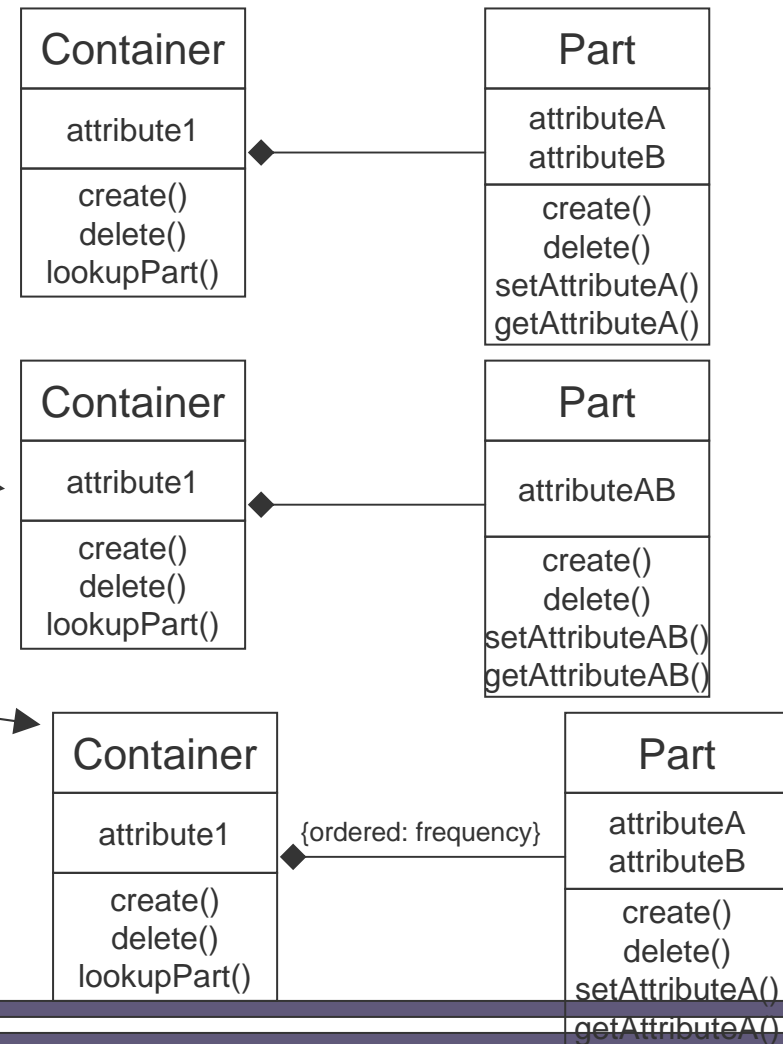
- Kostfaktoren:
 - Speichergebrauch
 - Datenzugriffe

➔ Verschiedene allgemeine Operationen

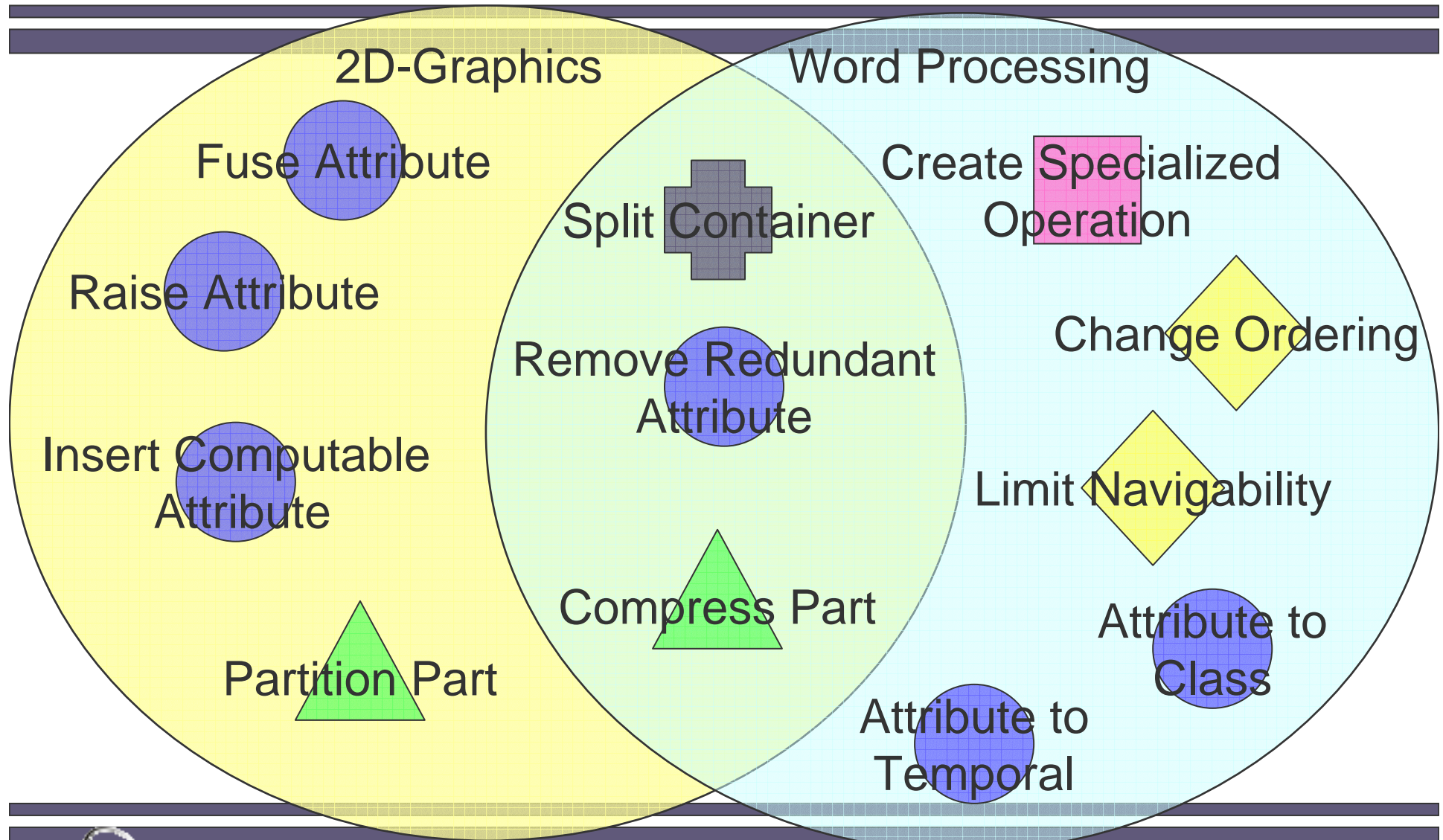
- Speichergebrauch verringern
 - Kompression *Fuse Attribute*
 - Löschen
 - Lebenszeit anpassen

- Datenzugriffe verringern
 - Sortieren *Change Ordering*
 - Partitionieren
 - Löschen

➔ Lösungen wirken nicht nur auf einen Kostenfaktor → Trade-off



Transformationskatalog

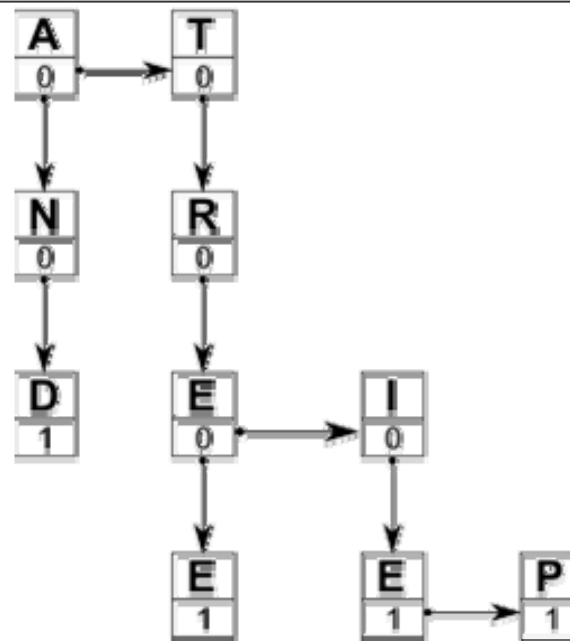


Evaluation der Transformationen - Rechtschreibprüfer

- Rechtschreibprüfer mit grundlegender Funktionalität
- Zwei Phasen
 - Phase 1 - Wörterbuch erstellen (57.046 Wörter)
 - Phase 2 - Rechtschreibung eines Textes prüfen
- Trie Abstract Data Type
 - Implementation mit zwei verschiedenen konkreten Datenstrukturen → Liste, Array
 - Stellen zwei Extreme im Suchraum der konkreten Datenstrukturen dar (DDTR - Dynamic Data Type Refinement, Atienza, et. al.)

List-structured Trie

- viele Datenzugriffe
- wenig Speichergebrauch

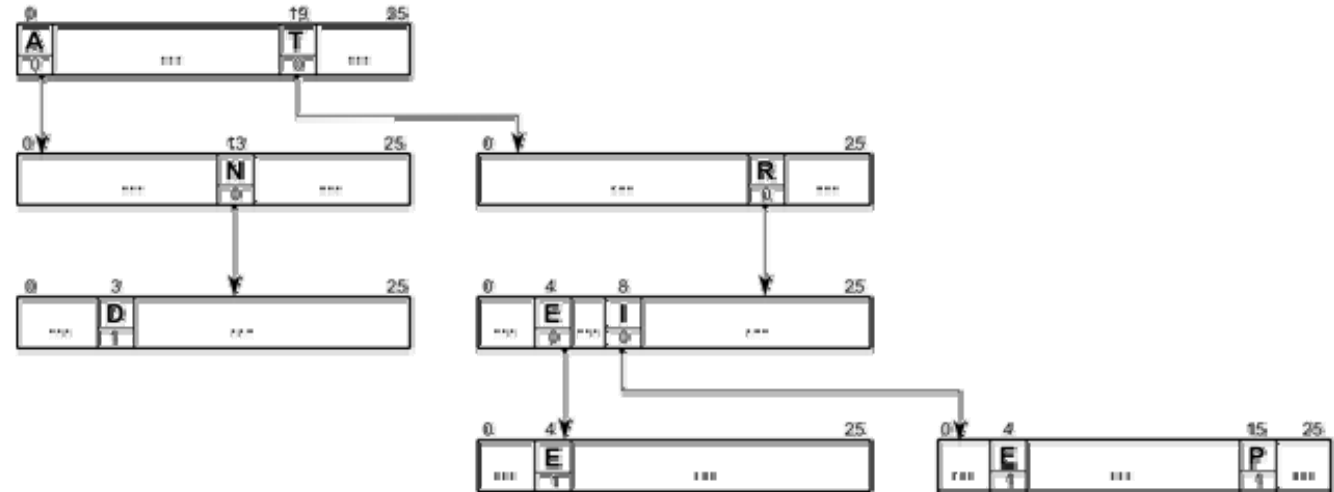


Evaluation der Transformationen - Rechtschreibprüfer

- Rechtschreibprüfer mit grundlegender Funktionalität
- Zwei Phasen
 - Phase 1 - Wörterbuch erstellen (57.046 Wörter)
 - Phase 2 - Rechtschreibung eines Textes prüfen
- Trie Abstract Data Type
 - Implementation mit zwei verschiedenen konkreten Datenstrukturen → Liste, Array
 - Stellen zwei Extreme im Suchraum der konkreten Datenstrukturen dar (DDTR - Dynamic Data Type Refinement, Atienza, et. al.)

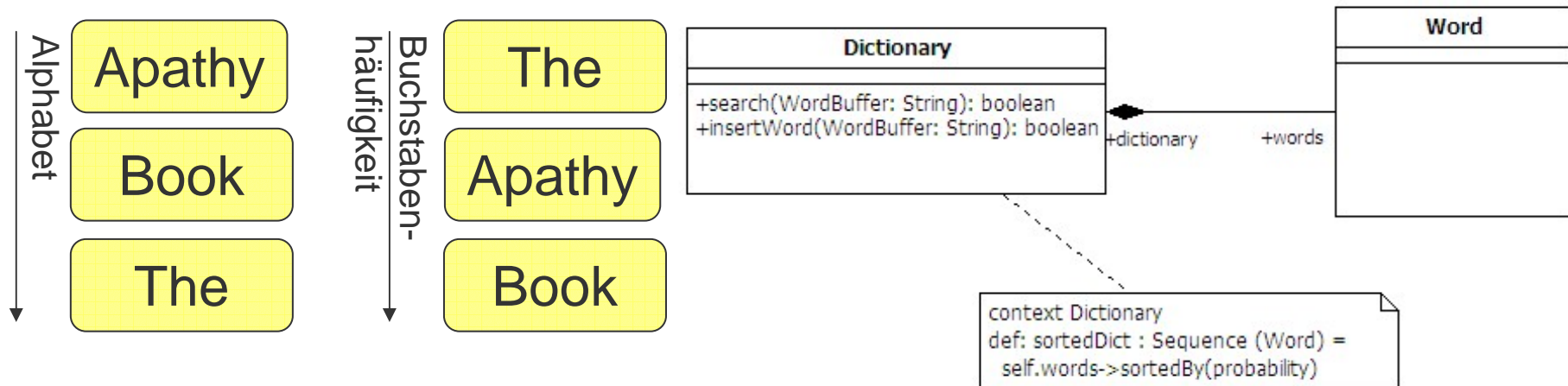
Array-structured Trie

- wenige Datenzugriffe
- hoher Speichergebrauch



Beispiel : Transformation *Change Ordering*

Verwertung dynamischen Inputs und spezifischen Wissens zu Design-Zeit



Motivation

- Wörter werden nicht der Reihenfolge im Wörterbuch folgend abgefragt

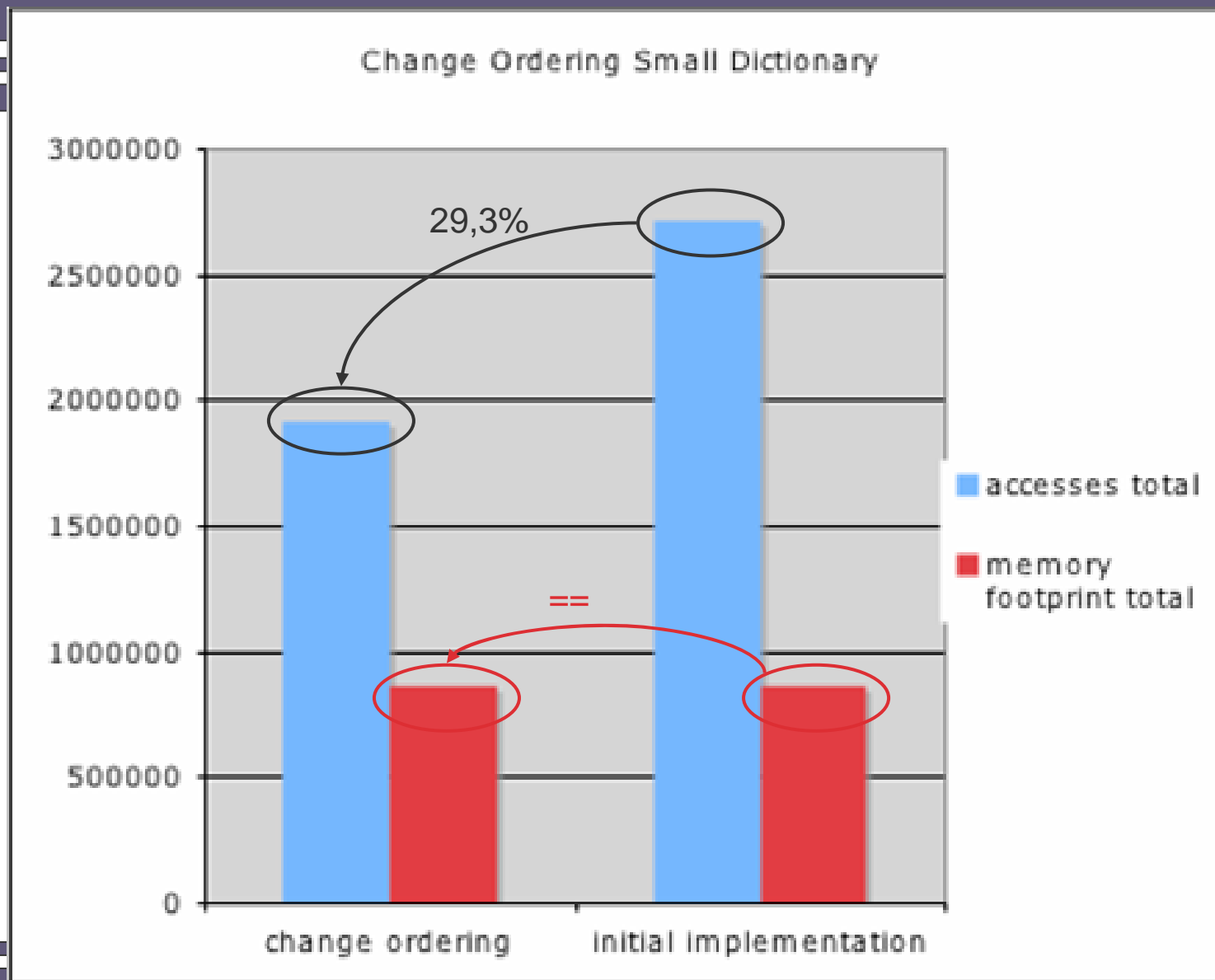
Lösung

- Ändere Reihenfolge → Sortierung
 - Alphabetisch → Buchstabenhäufigkeit in Englisch

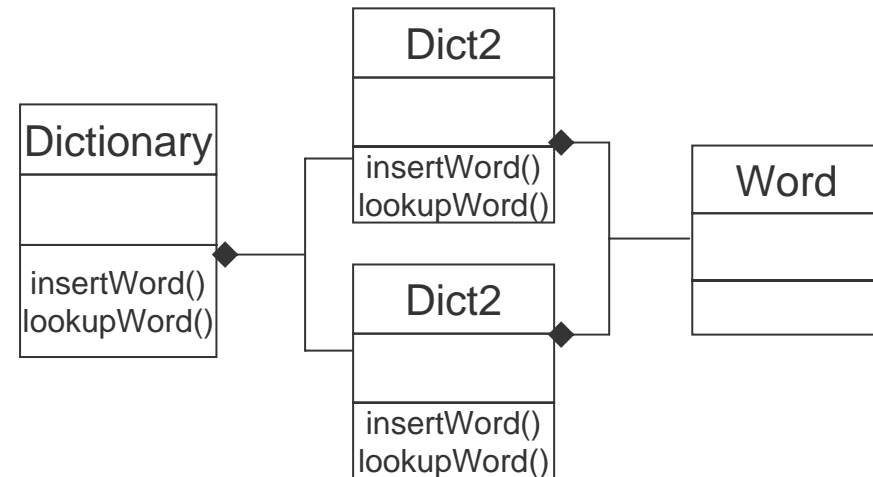
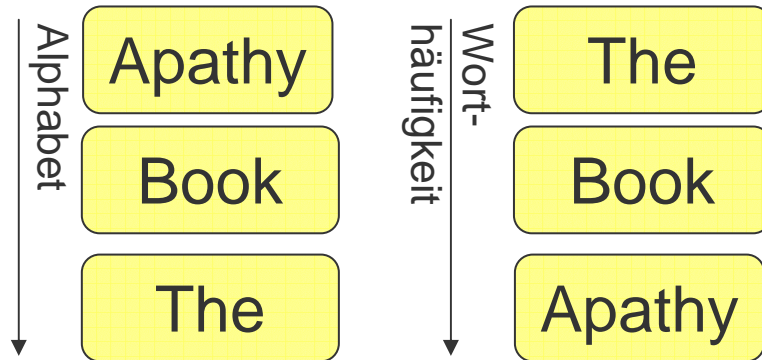
Resultat

- Durchschnittliche Anzahl der Datenzugriffe verringert

Ergebnisse - *Change Ordering*



Beispiel: Transformation *Split Container*



Motivation

- Wörter werden mit unterschiedlicher Häufigkeit in einer Sprache genutzt

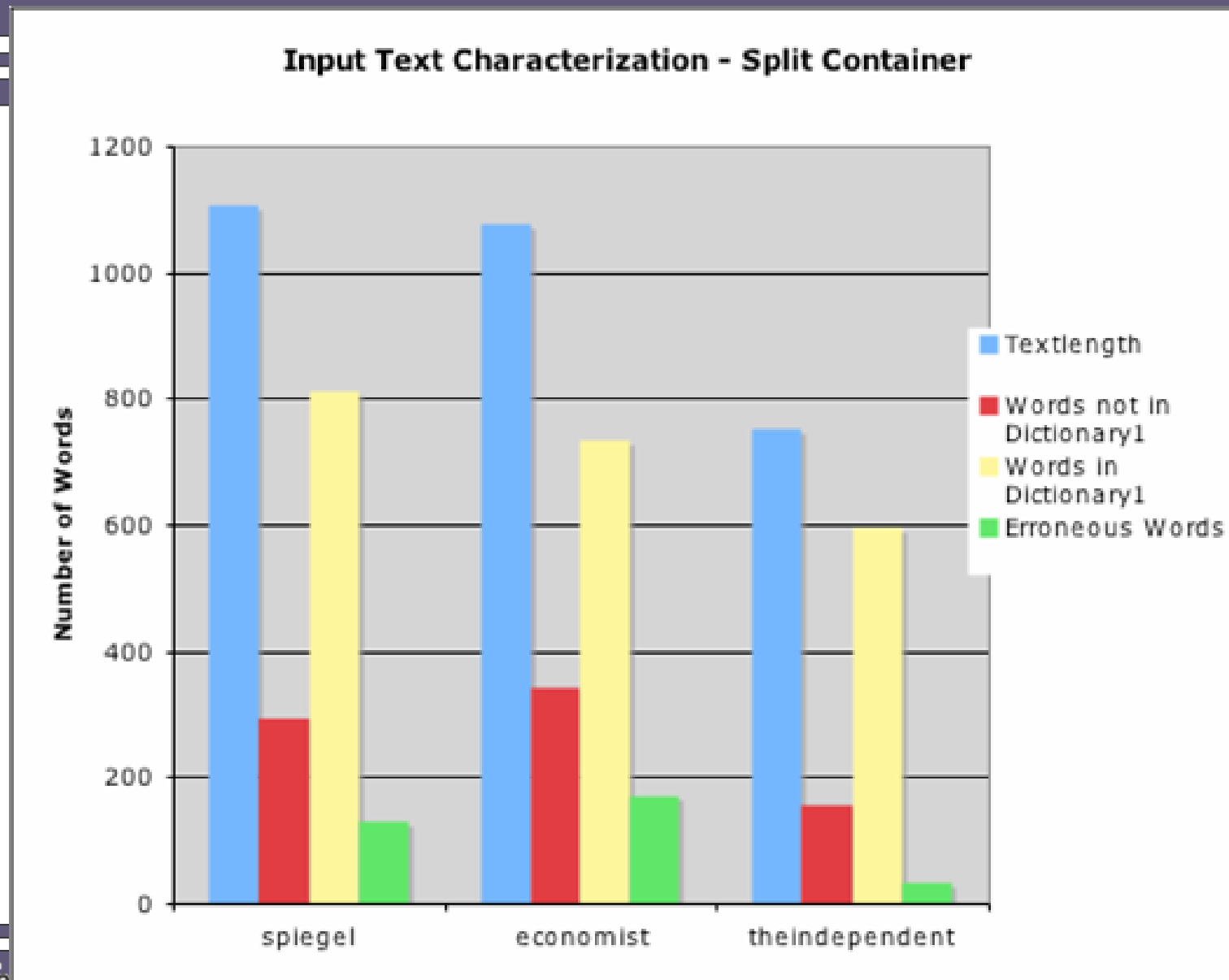
Lösung

- Verteile Wörter in häufig abgefragten Container und in weniger häufig abgefragten Container
- Teilungskriterium (*Splitting Criterion*) → Worthäufigkeit

Resultat

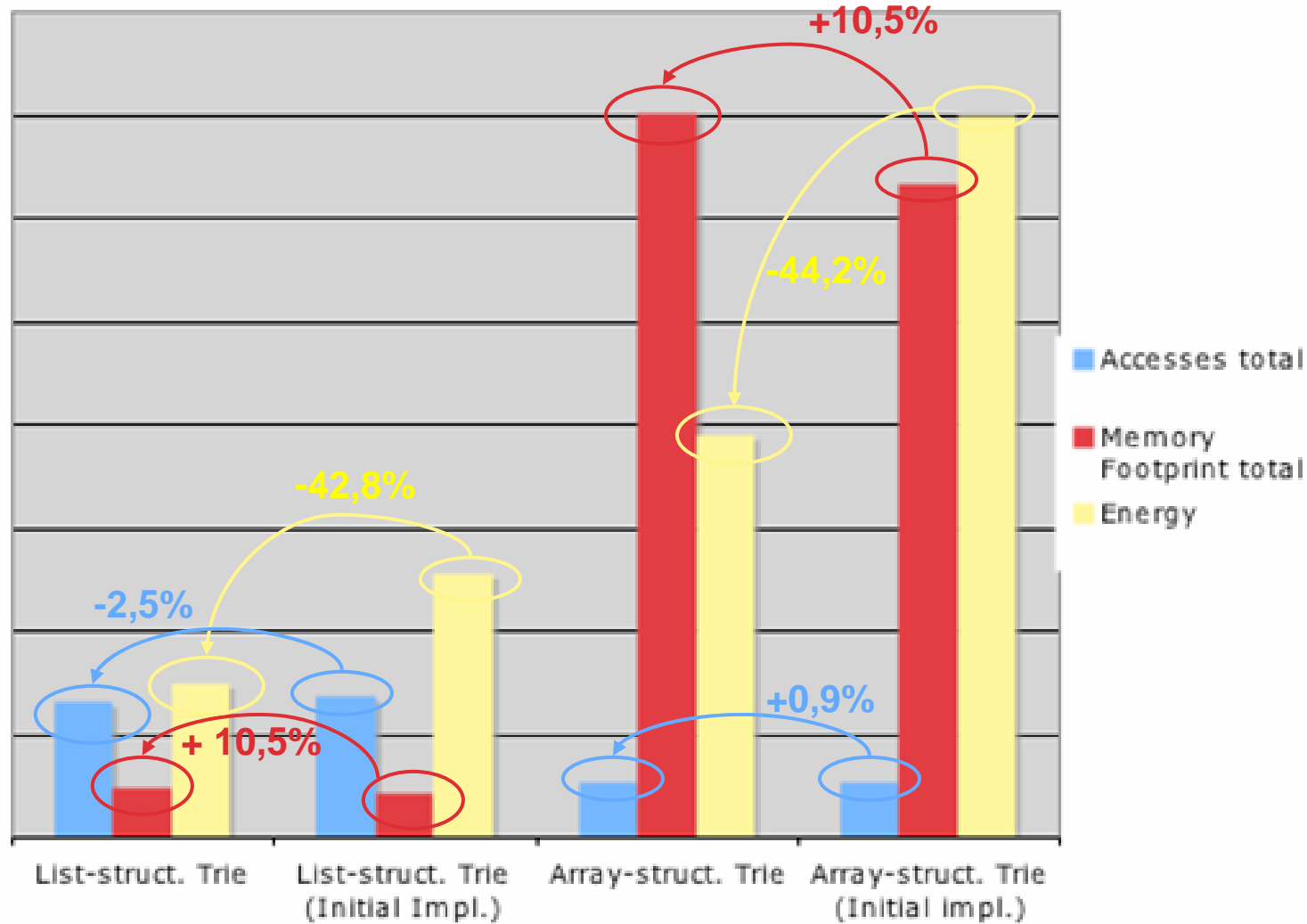
- kleiner, oft abgefragter Container (3.219 Wörter) in kleinen Speicher
- großer, weniger oft abgefragter Container (53.827 Wörter) in größeren Speicher

Wörterbuch nach der Transformation



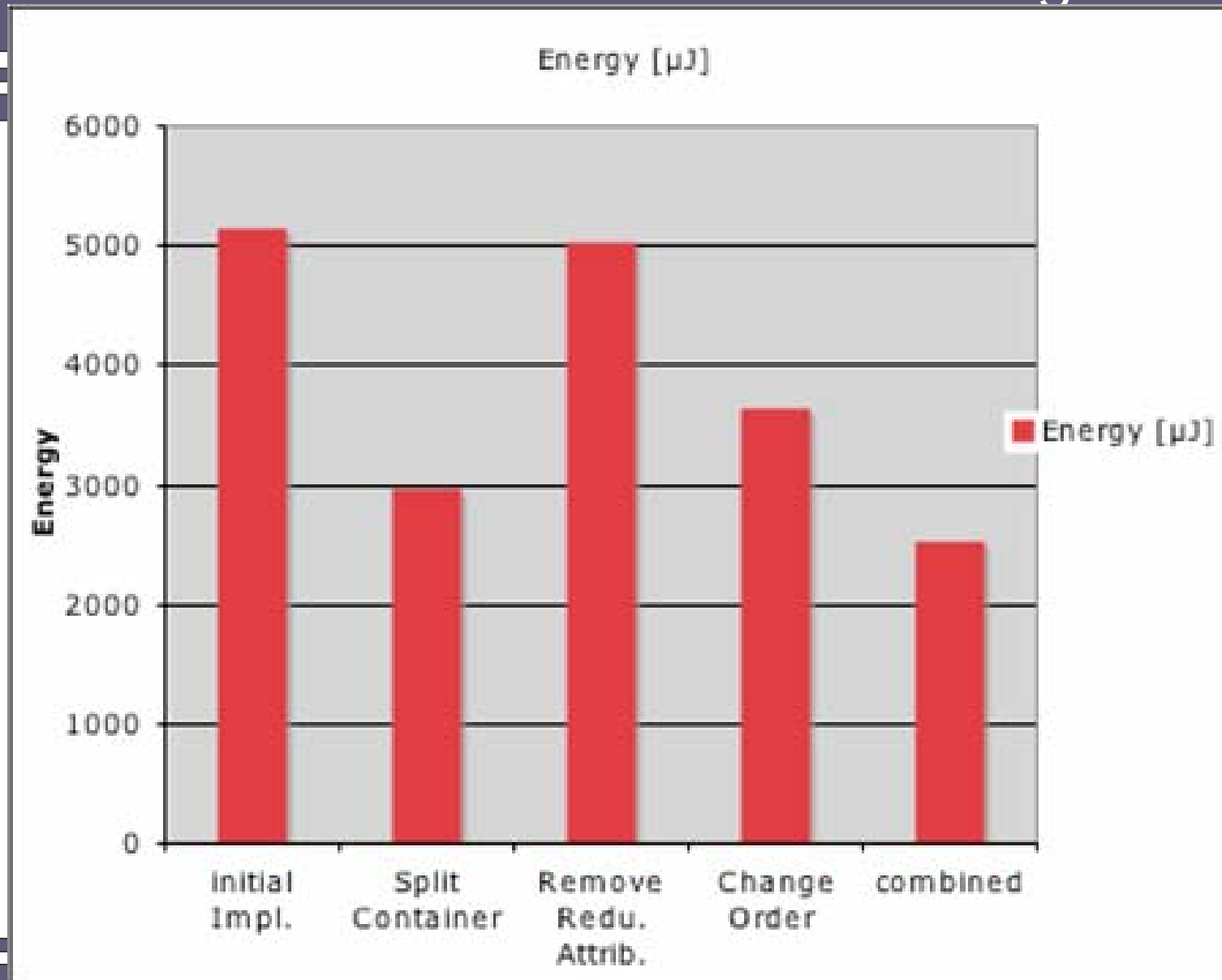
Ergebnisse - *Split Container*

Split Container Implementations



Kleines
Wörterbuch in
kleinen,
energie-
effizienten
Speicher!

Weitere Ergebnisse



- Anwendbarkeit von ADT-Modell Transformationen zur Kostenreduktion
 - Auf Basis von semantischen Informationen (Anwendungs-spezifische Informationen)
 - Auf Basis von dynamischen Informationen
- Energieverbrauch bis 70% reduziert
- Transformationskatalog erweitert
- Trade-off für Kostenfaktoren ermittelt
- Möglichkeit der Automatisierung
 - Transformationen können automatisiert werden (durch Formalisierung der Transformationsregeln)
 - Automatische Transformationserkennung ist schwierig

- Vollständige Beschreibung des ADT-Metamodells → Basis sämtlicher Automatisierungen
- Automatisierung der Transformationen (z.B. mit *Transformation Browser*)
- Code-Generierung

Vielen Dank für die Aufmerksamkeit!

Fragen?