



Different Approaches to Semantic Web Service Composition

Thomas Weise, Steffen Bleul, Diana Comes and Kurt Geihs
Distributed Systems Group
University of Kassel, Germany
{weise,bleul,comes,geihs}@vs.uni-kassel.de

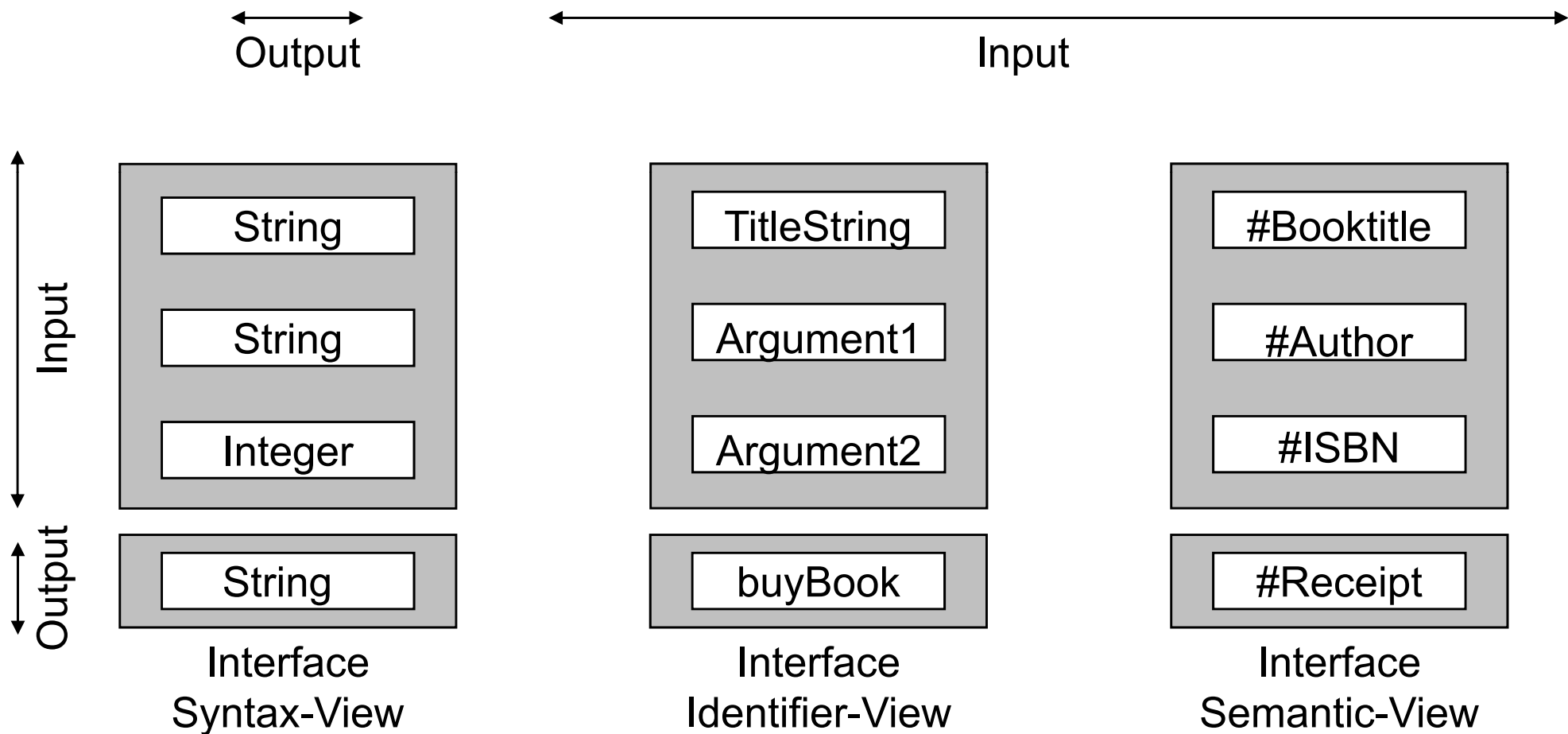


Contents

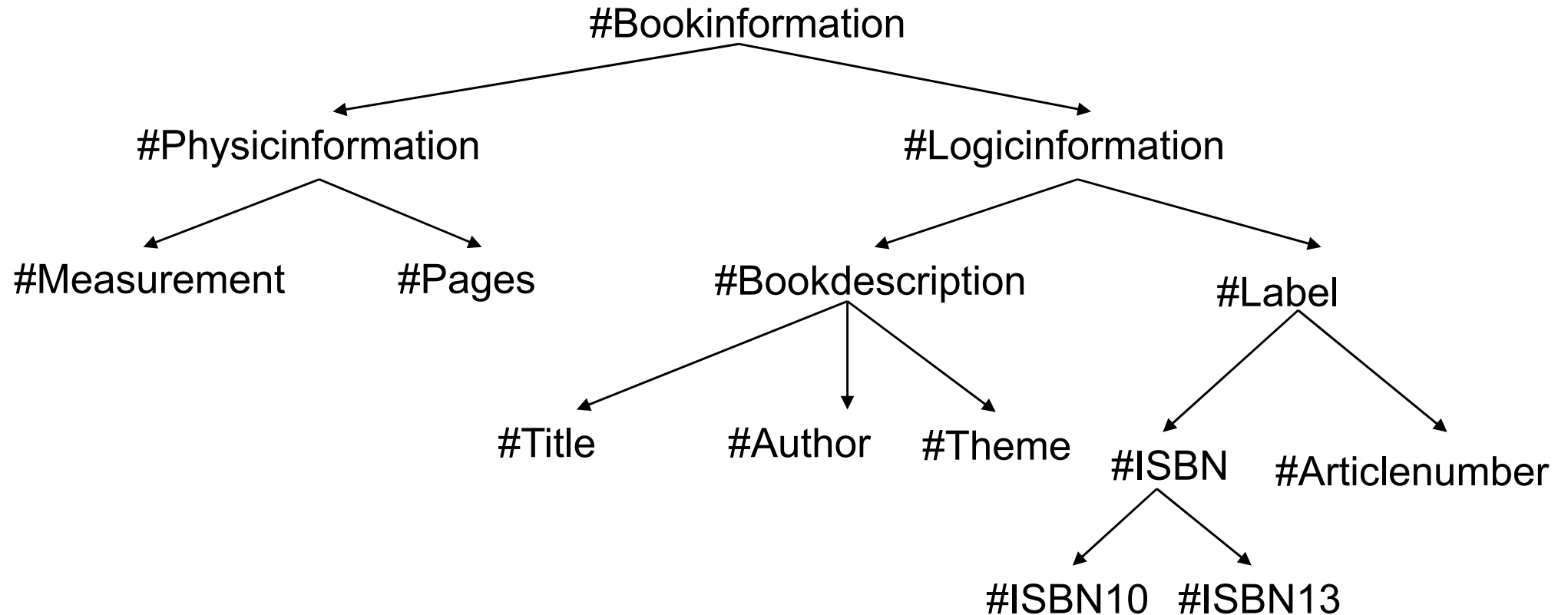
- Semantic Service Discovery and Composition
- The ADDO Composition System
- Semantic Service Composition Algorithms
 - IDDFS Algorithm
 - Greedy Algorithm
 - Genetic Algorithm
- Conclusion

Semantic Services

Java: `public String buyBook(String TitleString, String Argument1, Integer Argument2)`



Semantic Service Discovery

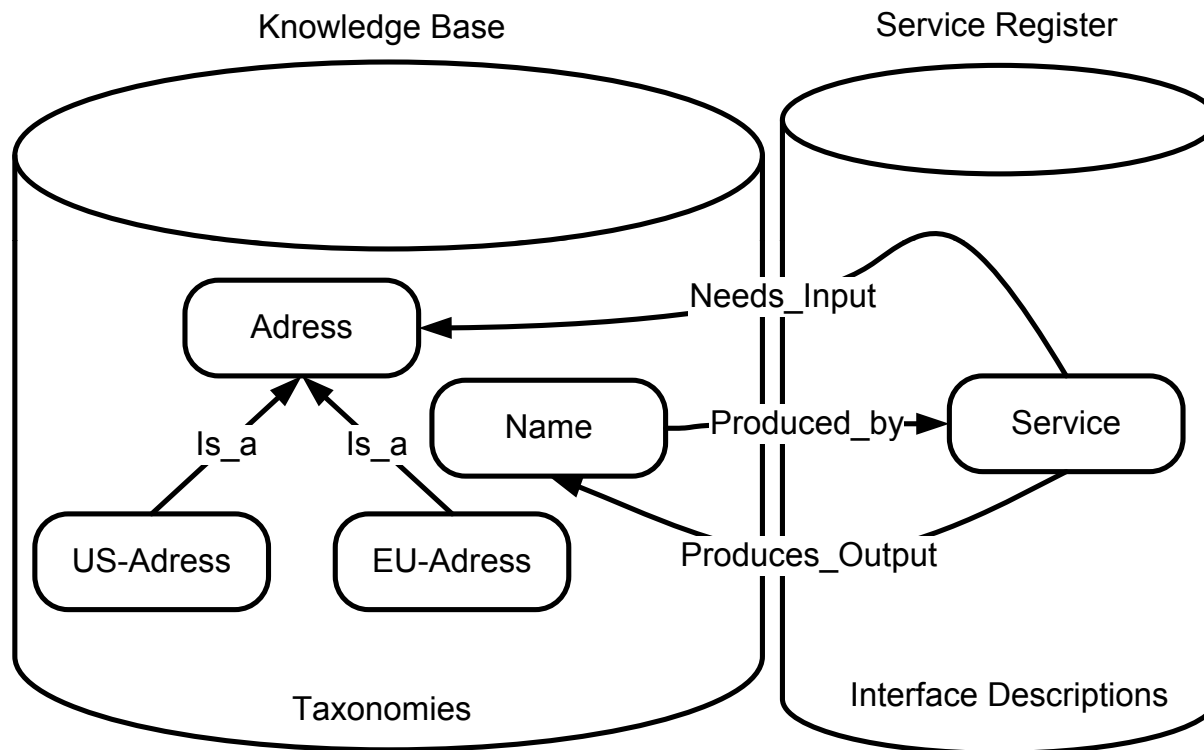


Predicate **subsumes(A,B)**

A, B – semantic concepts

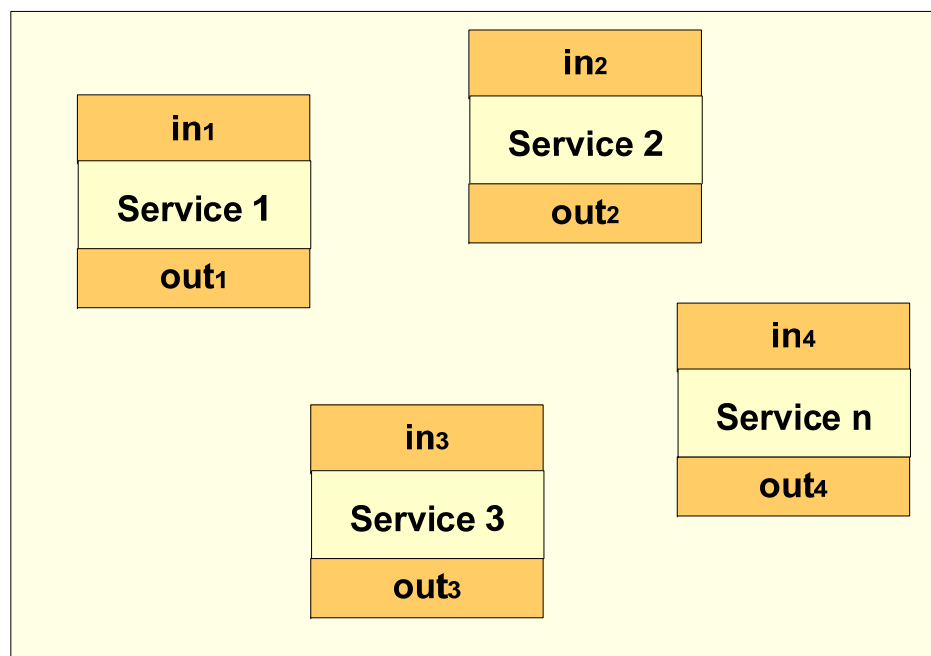
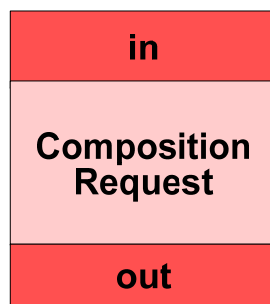
e.g.: subsumes(Bookdescription, Title)

Semantic Service Discovery



- **Semantic Input Matching:**
Service needs Input A.
Question: Is input A or a specialisation of Input A available?
- **Semantic Output Matching:**
Service produces Output B.
Question: Is Output B the wanted Output or a specialisation of the wanted Output?
- **promising(A)** – all services that provides as output the concept A (or a specialization of A)

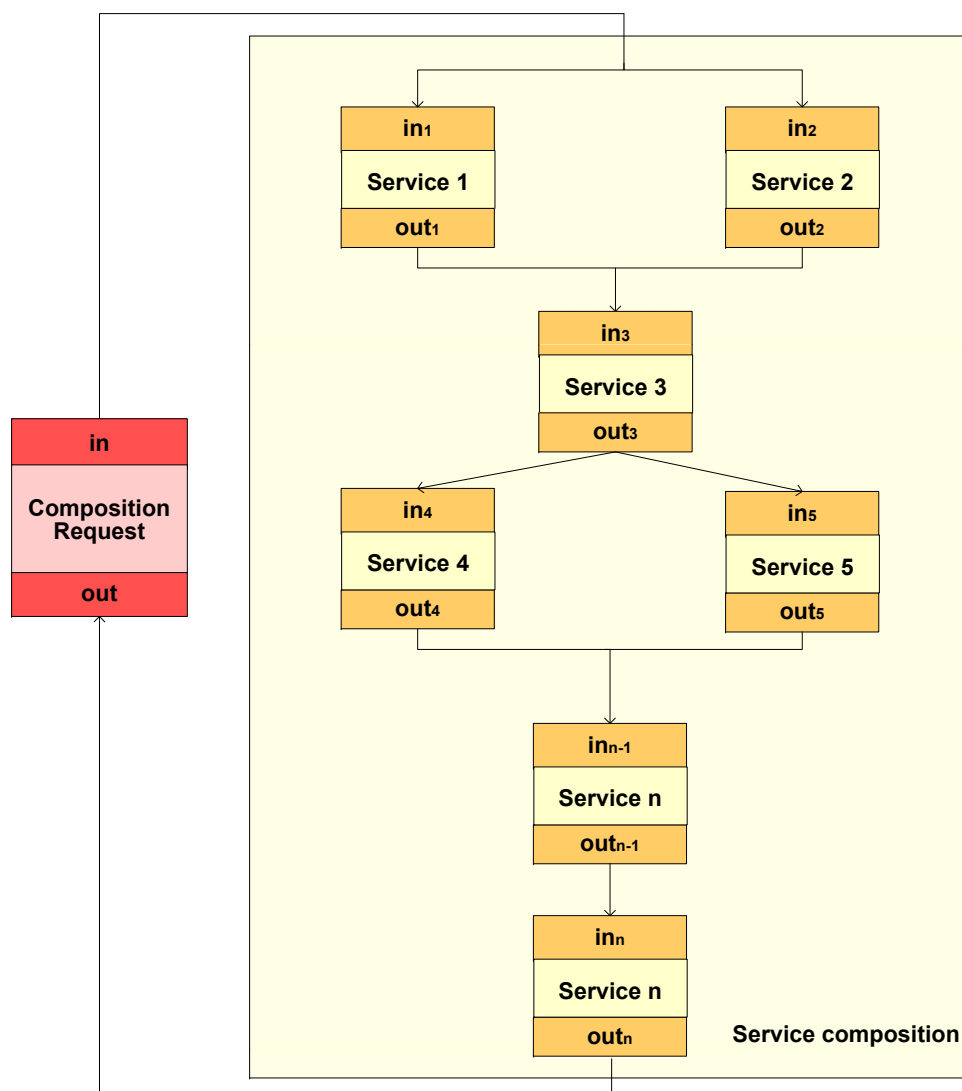
Semantic Composition System



- **Input**
 - Composition Request
 - Available input parameters
 - Wanted output parameters

- **Output**
 - Service Composition (Sequence of services)

Valid Service Composition



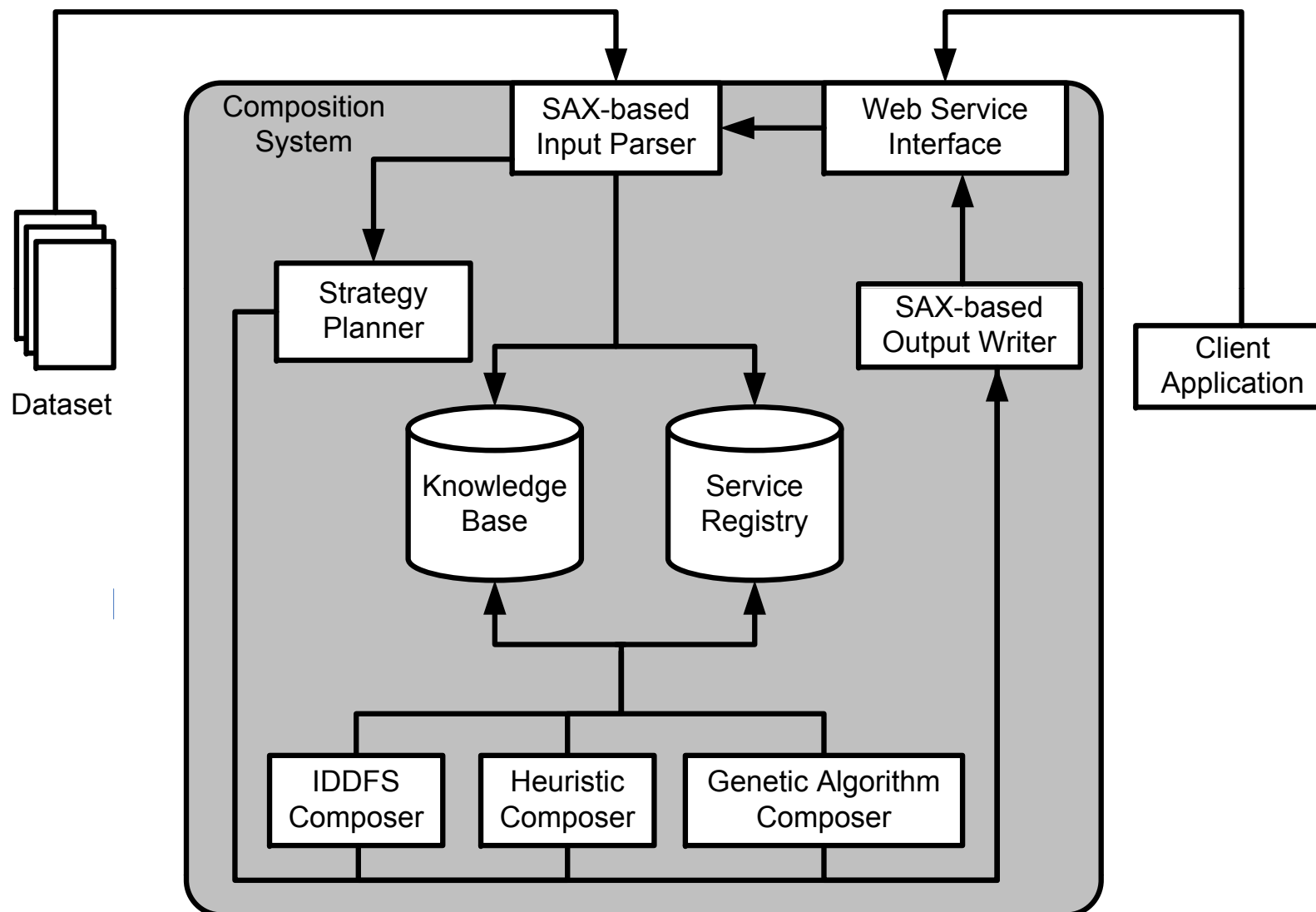
- Request.in
→ trigger service1
- Request.in, service1.out
→ trigger service2

Etc.

Service Composition:

Provides all concepts in Request.out (or more specific concepts)

ADDO Composition System



IDDFS Composition Algorithm

Iterative Deepening Depth-First Search

1. Repeat the ***search of the wanted concepts*** in the *maxdepth* (number of services) limit by increasing *maxdepth*

2. **Search of the wanted concepts**

For each of the ***wanted concepts***, search the ***promising services***

- Update the set of *wanted* concepts
- Add promising service to the composition
- Check if all wanted concepts were provided
- Recursive call by increasing the *depth* limit (if *depth* < *maxdepth*)

IDDFS Composition Example

Composition Request	Service (s1)	Service (s2)	Service (s3)
In: #Booktitle #Author	In: #ISBN	In: #Booktitle #Author #Year	In: #Author #Booktitle
Out: #Bill	Out: #Bill	Out: #ISBN13	Out: #Year

IDDFS Step 0	IDDFS Step 1	IDDFS Step 2	IDDFS Step 3
In: #Booktitle #Author	In: #Booktitle #Author	In: #Booktitle #Author	In: #Booktitle #Author
Eliminated: { }	Eliminated: #Bill	Eliminated: #Bill, #ISBN13	Eliminated: #Bill, #ISBN13, #Year
Wanted: #Bill	Wanted: #ISBN	Wanted: #Year	Wanted: { }
Composition: { }	Composition: {s1}	Composition: {s2, s1}	Composition: {s3, s2, s1}

Greedy Composition Algorithm

1. Consider a **set of composition candidates** (initially all promising services)
2. **Sort the set of compositions** descending using the comparator function
3. **Extract the last composition** from the set (the best) to expand next
 - Check if it is a valid composition and if so, return it
 - For each wanted concepts of this composition add all promising services to it
4. **Repeat** Steps 2,3 while the set of compositions is not empty

Comparator function

between 2 composition candidates

- Compare the **number of wanted parameters**
 - If we found a valid composition, it wins
 - If we found two valid compositions, the one with fewer services wins
 - Otherwise, compare the **number of eliminated parameters**
 - The composition with more eliminated parameters wins
 - If the same number of eliminated parameters, compare the **number of wanted parameters**
 - The composition with less wanted parameters wins



Greedy Composition Example

Composition Request
In: #Booktitle #Author
Out: #Bill

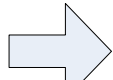
Service (s1)
In: #ISBN
Out: #Bill

Service (s2)
In: #Booktitle #Author #Description
Out: #ISBN

Service (s3)
In: #Description #ISBN
Out: #Bill

Service (s4)
In: #Booktitle #ISBN
Out: #Description

Step 1	
Composition 1	Composition 2
In: #Booktitle #Author	In: #Booktitle #Author
Eliminated: #Bill	Eliminated: #Bill
Wanted: #ISBN	Wanted: #Description, #ISBN
Composition 1: {s1}	Composition 2: {s3}

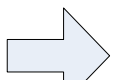
Comparator function


$|e1| = |e2| = 1$
 $|w1| < |w2|$

Sort desc

Expand comp1

Step 2	
Composition 2	Composition 1
In: #Booktitle #Author	In: #Booktitle #Author
Eliminated: #Bill	Eliminated: #Bill, #ISBN
Wanted: #Description, #ISBN	Wanted: #Description
Composition 2: {s3}	Composition 1: {s2, s1}

Comparator function


$|e1| > |e2|$

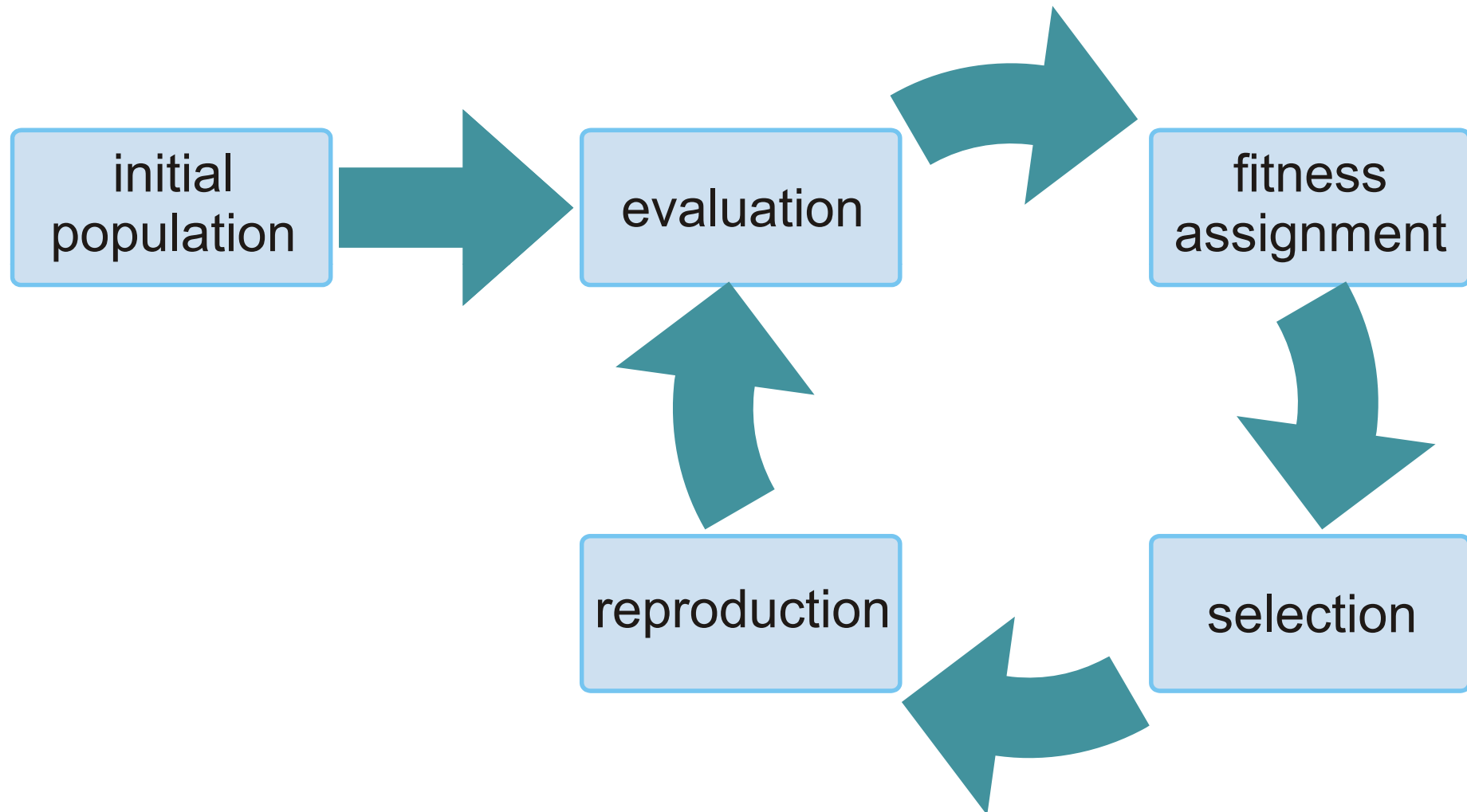
Sort desc

Expand comp 1

Step 3	
Composition 2	Composition 1
In: #Booktitle #Author	In: #Booktitle #Author
Eliminated: #Bill	Eliminated: #Bill, #ISBN, #Description
Wanted: #Description, #ISBN	Wanted: { }
Composition 2: {s3}	Composition 1: {s4, s2, s1}

$|e|$ - number of eliminated parameters in the composition
 $|w|$ - number of wanted parameters in the composition

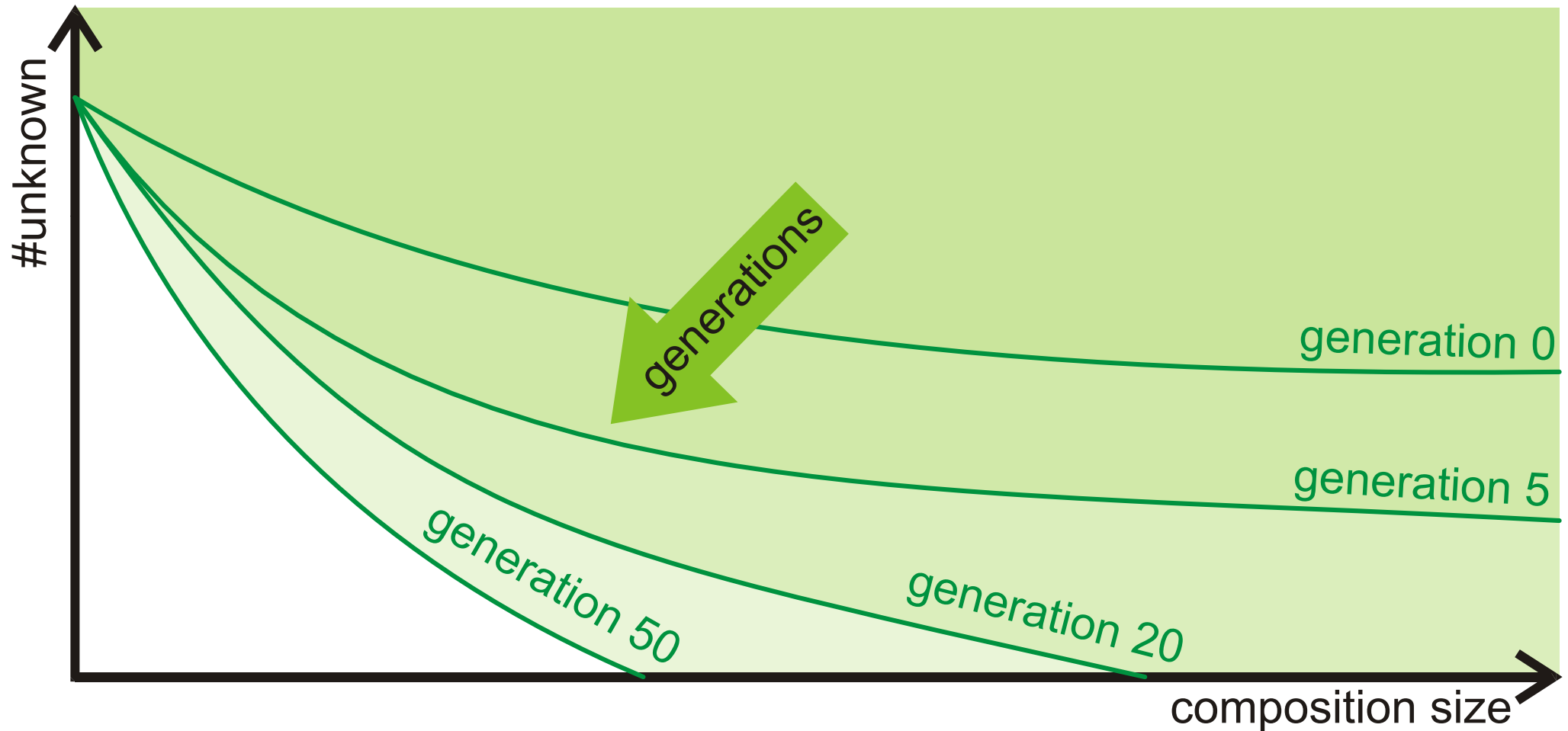
Genetic Composition Algorithm



Genetic Composition Algorithm

- **Mutate** operation for the composition candidate
 - Shrinking (delete the first service - *mutate1*)
 - Growing (add a promising service of a wanted parameter – *mutate2*)
- Create **initial population (mutate2)**
- **Comparator function:**
 - Compositions which are complete
 - Small compositions
 - Compositions that resolve many unknown parameters
 - Compositions that provide many parameters

Genetic Composition Example



Conclusion

- **IDDF (uninformed) Algorithm**
 - Good performance for small service repositories and short compositions
 - optimal if the problem requires an exhaustive search
- **HIDDF (informed) Algorithm**
 - The most efficient
 - Good solutions for all knowledge base- and service registry sizes
- **Genetic Algorithm**
 - Slower than greedy
 - Good solutions for all knowledge base- and service registry sizes



Thanks for your attention.

Questions?

DFG-Project ADDO & DGPF Framework

www.vs.uni-kassel.de/ADD0/
www.sourceforge.net/projects/dgpf