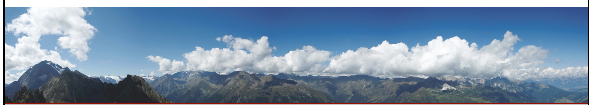
Artificial Intelligence

Problem-Solving Methods



Where are we?

#	Title
1	Introduction
2	Propositional Logic
3	Predicate Logic
4	Reasoning
5	Search Methods
6	CommonKADS
7	**Problem-Solving Methods**
8	Planning
9	Software Agents
10	Rule Learning
11	Inductive Logic Programming
12	Formal Concept Analysis
13	Neural Networks
14	Semantic Web and Services

Agenda

- Motivation
- Technical Solution
  - Development of Knowledge-based systems towards Problem-Solving Methods
  - Problem-Solving Methods
- Illustration by a Larger Example
- Extensions
- Summary
- References

MOTIVATION

Motivation

STI · INNSBRUCK

- In order to allow automation in the achievement of complex problems we should like a general solution with the following characteristics:

  Knowledge
  - Based on *reasoning* with *knowledge;*
  - Works with a *declarative*, rather than an algorithmic, representation of knowedge;

  Process
  - Represents knowledge on the problem-solving *process*, i.e., the dynamics of the solution;

  Reuse
  - Allows *reuse* of reasoning knowledge;
  - Abstracts from implementation and *domain* to increase reusability.

5

Motivating Example

STI · INNSBRUCK

- As a motivating example we consider the task (i.e., goal) of *parametric design*, which can be defined over:
  - **Design space** – the space that contains all possible designs;
  - **Requirements** – a finite set of which are assumed to be provided by the user;
  - **Constraints** – a finite set of which model additional conditions for a valid design;
  - **Preference** – a function over the design space which can be used to discriminate between different solutions.

6

Motivating Example

STI · INNSBRUCK

- *Domain knowledge* is implicit in the parametric design description in several places:
  - **Design space** – the concrete definition of the design space is domain specific knowledge
  - Requirements
  - **Constraints** – domain knowledge concerning regularities in the domain in which the design is constructed;
  - Preference

7

Motivating Example

STI · INNSBRUCK

- Formally, a design artifact is described by a set of attribute-value pairs.

  Let $A_1, \ldots, A_n$ be a fixed set of parameters with fixed ranges $R_1, \ldots, R_n$:
  - **Design space** is the cartesian product $R_1 \times \ldots \times R_n$;
  - **Requirements** – a relation R which defines a subset of this space;
  - **Constraints** – a relation C which defines a subset of this space;
  - **Preference** – a partial function P having all possible designs as its domain and preference values as its range.

8

Motivating Example

STI · INNSBRUCK

design space

Visually we represent these aspects as the *domain view*

constraints

requirements

preference

domain view (i.e., static knowledge role)

Motivating Example

STI · INNSBRUCK

- Several types of design are implicitly defined by these aspects of the problem:
  - **Possible design** – An element in design space ($\in R_1 \times ... \times R_n$);
  - **Desired design** – A design that fulfills all requirements ($\in R$);
  - **Valid design** – A design that fulfills all constraints ($\in C$);
  - **Solution** – A design which is *desired* and *valid* ($\in R \cap C$);
  - **Optimal solution** - A solution for which no other solution which has a higher preference value exists (any solution x such that $\forall y \in R_1 \times ... \times R_n .\ P(y) \leq P(x)$).

  There is also the possible extension:
  - **Acceptable solution** - Given a threshold t, an acceptable solution is any solution x such that $P(x) > t$.

Motivating Example

STI · INNSBRUCK

design space

Visually we represent these as *dynamic* data in a store:

possible design

constraints

requirements

valid design

desired design

solution

data store (i.e., dynamic knowledge role)

preference

domain view (i.e., static knowledge role)

optimal solution

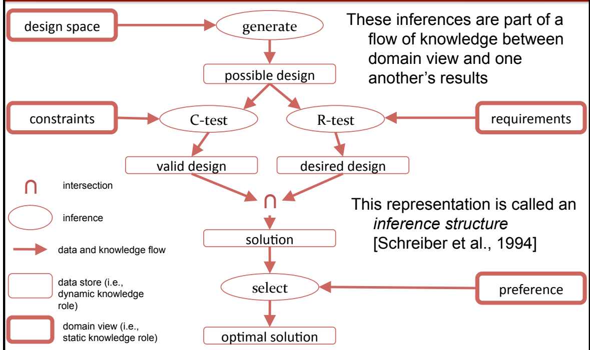
Motivating Example

STI · INNSBRUCK

- An inefficient naive approach to parametric design is called *generate & test*, which depends on the following inferences:
  - **generate** – requires knowledge that describes what constitutes a possible design (i.e., the design space);
  - **R-test** – requires knowledge that describes which possible designs are desired (i.e. the user requirements);
  - **C-test** – requires knowledge that describes which possible designs are valid (i.e., the domain constraints);
  - **select** – requires knowledge that evaluates solutions (i.e., knowledge that describes what constitutes a preferred design).

Motivating Example

STI · INNSBRUCK



These inferences are part of a flow of knowledge between domain view and one another's results

This representation is called an *inference structure* [Schreiber et al., 1994]

Motivating Example

STI · INNSBRUCK

- This naive solution can also be represented as:

  possible design := **generate**$_{all}$;
  valid design := **C-test**(possible design);
  desired design := **R-test**(possible design);
  solution := valid design ∩ desired design;
  optimal solution := **select**(solution)

- Using the definition of acceptable solution this can be made somewhat more efficient as:

  repeat
  possible design := **generate**$_{one}$;
  valid design := **C-test**(possible design);
  desired design := **R-test**(possible design);
  solution := valid design ∩ desired design;
  acceptable solution := **select**(solution)
  until Ø ≠ acceptable solution

Lessons from Example

STI · INNSBRUCK

- generate & test has the following characteristics:
  – it separates the different types of knowledge;
  – it is not efficient (all possible designs are generated);
  – It may not terminate if the design space is infinite.
- From the literature on expert systems [Stefik et al., 1983]:
  – "an important issue is the distribution of knowledge between the generator and the tester: putting as much knowledge as possible into the generator often leads to a more efficient search."
- A much more clever strategy is therefore to use these knowledge types to guide the generation of possible designs.
- However to do so requires strong assumptions about the domain knowledge.

STI · INNSBRUCK

TECHNICAL SOLUTIONS

STI · INNSBRUCK

DEVELOPMENT OF KNOWLEDGE-BASED SYSTEMS TOWARDS PROBLEM-SOLVING METHODS

Development

STI · INNSBRUCK

1. General Problem Solver
2. Knowledge-is-power hypothesis
3. Knowledge levels
   3a. Newell's 3 levels of knowledge
   3b. Brachman's 5 levels of knowledge
4. Problem Solving Methods

1. General Problem Solver

STI · INNSBRUCK

- The General Problem Solver (GPS) is a universal problem solving approach.
- GPS is the first approach that makes the distinction between *knowledge of problems domains* and *how to solve problems*
- GPS is domain and task independent approach.
- GPS does not put any restrictions both on the domain knowledge and on the task.
- Examples of GPS are: *automated theorem proving* and *generic search methods*

Automated theorem proving

STI · INNSBRUCK

- Automatic theorem provers are GPS for which every problem can be expressed as logical inference
- Automated theorem proving is about proving of mathematical theorems by a computer program

See Lecture 4

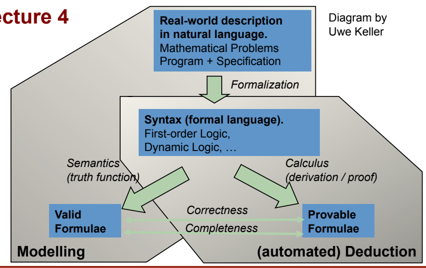


Diagram by
Uwe Keller

Generic Search Methods

STI · INNSBRUCK

- Generic Search Methods are GPS for which every problem can be expressed as search
- One particular example of a Generic Search Method is the **A*** algorithm.

- **A*** works for problems that can be represented as a state space i.e. a graph of states. Initial conditions of the problem are represented as *start state*, goal conditions are represented as *end state*
- **A*** is an informed search or heuristic search approach that uses the estimation function:

$$f(n)=g(n)+h(n)$$

  - g(n) the cost to get from the *star state* to current state *n*
  - h(n) estimated cost to get from current state *n* to *end state*
  - f(n) estimated total cost from *start state* through current state *n* to the *end state*

See Lecture 5

21

1. General Problem Solver (1)

STI · INNSBRUCK

- **However, GPS has a set of limitations**:
  - It works in theory but in practice works only on toy problems (e.g. Tower of Hanoi)
  - Could not solve real-world problems because search was easily lost in the combinatorial explosion of intermediate states

22

2. Knowledge-is-power hypothesis

STI · INNSBRUCK

Knowledge-is-power hypothesis, also called the **Knowledge Principle** was formulated by E.A. Feigenbaum in 1977:

"knowledge of the specific task domain in which the program is to do its problem solving was more important as a source of power for competent problem solving than the reasoning method employed" [Feigenbaum, 1977]

23

2. Knowledge-is-power hypothesis (1)

STI · INNSBRUCK

- The Knowledge-is-power hypothesis shifted the focus on how to build intelligent systems from ***inference*** to the ***knowledge base***.
- Problem solving is guided by experiential, qualitative, heuristic knowledge.
- The meaning of intelligence as knowledge is the common meaning in English world.
- The Central Intelligence Agency (CIA) defines intelligence as knowledge.
- The Knowledge-is-power hypothesis lead to the emergence of a new filed i.e. ***expert systems*** and a new profession i.e. ***knowledge engineer***

24

3. Knowledge levels

3a. Newell's 3 levels of knowledge
3b. Brachman's 5 levels of knowledge

3a. Newell's 3 levels of knowledge [Newell, 1981]

- In his work from 1981, Newell tried to answer questions such as:
  - How can knowledge be characterised?
  - What is the relation of this characterisation and the representation of knowledge?
  - What is characteristic about a system which holds knowledge?
- Newell distinguished 3 levels in the context of knowledge representation:
  - **Knowledge Level**
  - **Logical Level**
  - **Implementation Level**

3a. Newell's 3 levels of knowledge (1)

- **Knowledge Level**
  - The most abstract level of representing knowledge.
  - Concerns the total knowledge contained in the Knowledge Base
  - Example:
    The automated DB-Information system knows that a trip from Innsbruck to Vienna costs 120€

3a. Newell's 3 levels of knowledge (2)

- **Logical Level**
  - Encoding of knowledge in a formal language.
  - Example:
    Price(Innsbruck, Vienna, 120)

3a. Newell's 3 levels of knowledge (3)

- **Implementation Level**
  - The internal representation of the sentences.
  - Example:
    - *As a String "Price(Innsbruck, Vienna, 120)"*
    - *As a value in a matrix*

29

3b. Brachman's 5 Levels of Knowledge [Brachman, 1979]

- Brachman defines 5 levels for different types of representations.
- Levels interpret the transition from data to knowledge.
- Each level corresponds to an explicit set of primitives offered to the knowledge engineer.
- Ordering of knowledge levels from simple/abstract to complex/concrete:
  - **Implementational Level**
  - **Logical Level**
  - **Epistemological Level**
  - **Conceptual Level**
  - **Linguistic Level**

30

3b. Brachman's 5 Levels of Knowledge (1)

- **Implementational Level**
  - The primitives are pointers and memory cells.
  - Allows the construction of data structures with no a priori semantics

31

3b. Brachman's 5 Levels of Knowledge (2)

- **Logical Level**
  - The primitives are logical predicates, operators, and propositions.
  - An index is available to structure primitives.
  - A formal semantic is given to primitives in terms of relations among objects in the real world
  - No particular assumption is made however as to the nature of such relations

32

3b. Brachman's 5 Levels of Knowledge (3)

- **Epistemological Level**
  - The primitives are concept types and structuring relations.
  - Structuring relations provide structure in a network of conceptual types or units. (i.e. inheritance: conceptual units, conceptual sub-units)
  - The epistemological level links formal structure to conceptual units
  - It contains structural connections in our knowledge needed to justify conceptual inferences.

33

3b. Brachman's 5 Levels of Knowledge (4)

- **Conceptual Level**
  - The primitives are conceptual relations, primitive objects and actions.
  - The primitives have a definite cognitive interpretation, corresponding to language-independent concepts like elementary actions or thematic roles

34

3b. Brachman's 5 Levels of Knowledge (5)

- **Linguistic Level**
  - The primitives are words, and (lingustic) expressions.
  - The primitives are associated directly to nouns and verbs of a specific natural language
  - Arbitrary relations and nodes that exist in a domain

35

4. Problem Solving Methods

- Problem Solving Methods (PSM) abstract from details of the implementation of the reasoning process.
- Characteristics of PSM [Birmingham&Klinker, 1993] :
  - A PSM specifies which inference actions have to be carried out for solving a given task.
  - A PSM determines the sequence in which these actions have to be activated.
  - Knowledge roles determine which role the domain knowledge plays in each inference action.

36

STI · INNSBRUCK

PROBLEM-SOLVING METHODS

Technical Solution Overview

STI · INNSBRUCK

- What are *problem-solving methods (PSMs)*?
  - "Reasoning strategies that gain efficiency through *assumptions*."

[Fensel, 2000]

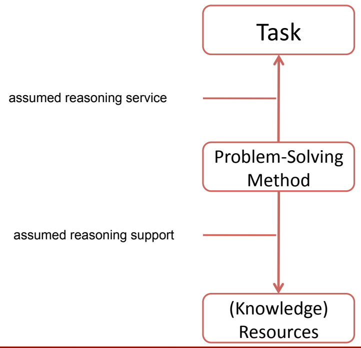
Technical Solution Overview

STI · INNSBRUCK

- Problem-solving methods achieve an efficient realization of functionality by making *assumptions*.
- *Assumptions* put restrictions on the context of the problem-solving method, such as the domain knowledge and the possible inputs of the method or the precise definition of the functionality.
- Assumptions play two roles:
  - they formulate requirements on reasoning support that is assumed by PSMs;
  - they put restrictions on the reasoning support that is provided by PSMs.
- In consequence, assumptions link PSMs with the domain knowledge they use and tasks they are applied to.

Technical Solution Overview

STI · INNSBRUCK



Example Revisited

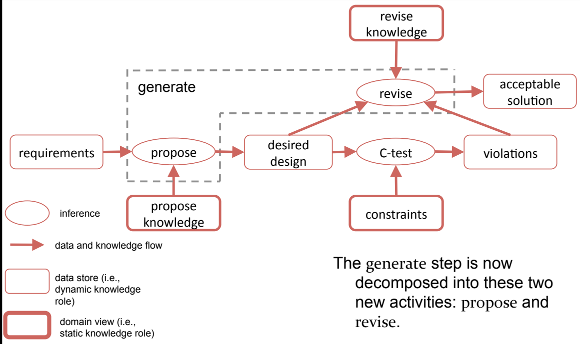
STI · INNSBRUCK

- We consider again the task of parametric design.
- A more efficient method is named *propose & revise* and depends on the following inferences:
  - **propose** – derives an initial design based on the requirements;
  - **C-test** – as before;
  - **revise** – tries to improve an incorrect design based on the feedback of the C-test step.
- In other words, instead of proposing a complete design which is then repaired, we can also incrementally develop a design and repair it at each step where a constraint violation occurs.

41

Example Revisited

STI · INNSBRUCK



The generate step is now decomposed into these two new activities: propose and revise.

42

Example Revisited

STI · INNSBRUCK

- A parameter which should receive a value in the next **propose** step is *nondeterministically* selected:
  - The selection process does not make further assumptions about knowledge that could guide this second selection step
  - The implicit assumption is that this selection does not a affect the performance of the problem solving process and the quality of its result
  - **These are very strong assumptions because to improve performance, heuristic methods are definitely needed**
  - At any time there is either precisely one applicable propose rule or one user input to derive the value of the selected parameter
  - A parameter should not depend on itself (no recursive derivation rules)

43

Example Revisited

STI · INNSBRUCK

- **revise** is decomposed into:
  - **select-violation** - nondeterministically selects a constraint violation from those detected by C-test; implicit assumption is that this selection does not influence the performance of the problem solving method and the quality of the result; **strong assumption again**
  - **derive-fixes** - computes the set of all possible fix combinations that could possibly resolve the selected constraint violation; each combination must be finite
  - **select-fix** - selects a fix combination, guided by a cost-function
  - **apply-fix** - applies a fix combination

44

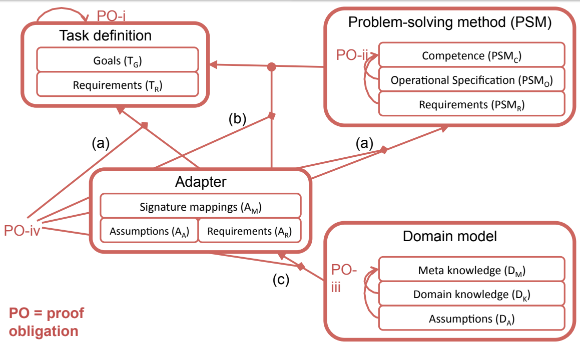
Example Revisited

- **Test** – propose & revise does not require an explicit R-test, the method assumes that:
  - propose derives only desired designs;
  - revise delivers designs that are desired or that requirement violations that it does not x must be accepted.
- **Selection** – does not contain such a process concerning user preferences:
  - It assumes that the propose step and the revise step deliver acceptable (or optimal) solutions or that the functionality of the task is reduced to finding an arbitrary solution.

Description of Problem-Solving Methods

- The main elements of a specification in the PSM framework are:
  - **the task** – specifies the *goals* that should be solved in order to solve a given problem. A second part of a task specification is the definition of *requirements* on domain knowledge;
  - **the problem-solving method** – describes an reasoning steps to perform a task as an *operational specification* separately from a description of the *competence,* and a description of the *requirements* on domain knowledge;
  - **the domain model** – usually ontology-based description in three parts, a meta-level characterisation of properties, the domain knowledge, and (external) assumptions of the domain model;
  - **the adapter** – maps the different terminologies of the task definition, problem-solving method and domain model. Moreover, gives further requirements and assumptions needed to relate the competence of the PSM with the functionality of the task.

Description of Problem-Solving Methods



PO = proof obligation

Description of Problem-Solving Methods

- Several proof obligations follow the conceptual model of such a specification of a knowledge-based system:
  - **PO-i**: the consistency of the task definition to ensure that a model exists, otherwise one could define an unsolvable problem;
  - **PO-ii**: that the operational specification of the PSM describes a terminating process which has the competence as specified;
  - **PO-iii**: the internal consistency of the domain knowledge and domain model, also that the assumptions on domain knowledge implies its meta-level characterisation;
  - **PO-iv**: relationships between the specification elements –
    a) the requirements of the adapter imply the knowledge requirements of the PSM and task,
    b) the adapter's additional requirements on domain knowledge and assumption guarantee that the competence of the PSM is strong enough for task,
    c) The requirements of the adapter are implied by the meta knowledge of the domain model.

PSM Framework Example

STI · INNSBRUCK

- To illustrate the description of PSMs we consider search
- Local search can be refined into other versions using adapters, specifically:
  - **Hill-climbing**: a local search algorithm which stops when it has found a *local optimum*, based on recursively considering the *successors* to a start object, selecting better at each stage;
  - **Set-Minimizer**: finds a *minimal* but still *correct subset* of a given set, with respect to hill-climbing –
    - generic 'object' becomes a set,
    - 'successor' relationship is hard-wired,
    - preference is implicit;
  - **Abductive Diagnosis**: receives a set of *observations* as input and delivers a *complete* (explains all input data) and *parsimonious* (no subset of hypotheses explains all observations) *explanation*.

49

PSM Framework Example

STI · INNSBRUCK

- Local search can be represented in the following operational specification:

operational specification local search
  output := local search(input);
  local search(X)
    begin
      currents := $select_1(X)$;
      output := recursion(currents);
    end
  recursion(X)
    begin
      successors := generate(X);
      new := $select_2$(X, successors);
        if X = new
          then output := $select_3(X)$;
          else recursion(new);
    end

$select_1(x) \subseteq x$

$x \in generate(y) \leftrightarrow x \in input \land \exists z.(z \in y \land successor(z, x))$

$select_2(y, y') \subseteq y \cup y'$

$\neg \exists x, z . (z \in (y \cup y') \setminus select_2(y, y') \land x < z)$

$y \cup y' \neq \varnothing \rightarrow \exists x.(x \in select_2(y, y'))$

$select_3(y) \subseteq y$

$\neg \exists x, z . (z \in (y \setminus select_3(y)) \land x \in select_3(y) \land x < z)$

$y \neq \varnothing \rightarrow \exists x.(x \in select_3(y))$

endoperational spec

50

PSM Framework Example

STI · INNSBRUCK

- Adapters between refinement levels can be represented as follows:

PSM refinement adapter set-minimizer -> abduction-method
  correct(x) = complete(x);
  input = {h | h is hypothesis};
  $H \subseteq H' \rightarrow explain(H) \subseteq explain(H')$
endPSM refinement adapter
PSM refinement adapter hill-climbing -> set-minimizer
  correct(input);
  $select_1(x) = \{x\}$;
  $successor(x, y) \leftrightarrow \exists z . (z \in x \land y = x \setminus \{z\})$;
  $x < y \leftrightarrow correct(y) \land y \subset x$
endPSM refinement adapter
PSM refinement adapter local-search -> hill-climbing
  $|select_1(x)| = 1$;
  $\neg \exists z . (z \in y' \land y < z) \rightarrow select_2(\{y\}, y') = \{y\}$;
  $|select_2(\{y\}, y')| = 1$
endPSM refinement adapter

51

STI · INNSBRUCK

ILLUSTRATION BY A LARGER EXAMPLE

52

Sisyphus-I

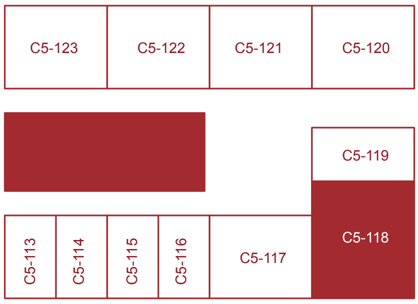
STI · INNSBRUCK

- The Sisyphus-I office allocation problem was the first of a series of test applications for approaches to building knowledge-based systems.
- The aim is to automate the problem-solving behaviour of ‘Siggi’, a hypothetical domain expert .
- Specifically Siggi is required to make assignments of the staff of a research group ‘YQT’ to offices.
- The 4-page problem statement describes the office layout and relevant data about the 15 group members.
- A ‘protocol’ is provided, describing the steps Siggi takes.

53

Sisyphus-I Office Layout

STI · INNSBRUCK



54

Sisyphus-I Protocol

STI · INNSBRUCK

1) Thomas in C5-117	a) The head needs a central office, so that he is close to all members of the group. This should be a large office. b) This assignment is defined first, as the location of the office of the head restricts the possibilities of the subsequent assignments.
2) Monika and Ulrika in C5-119	a) The secretaries’ office should be located close to the office of the head. Both secretaries should work together in one large office. b) This assignment is executed as soon as possible, as its possible choices are extremely constrained.
3) Eva in C5-116	a) The manager must have maximum access to the head and the secretariat. At the same time she should have a centrally located office. A small office will do. b) This is the earliest point where this decision can be taken.
4) Joachim in C5-115 5) Hans in C5-114 6) Katharina in C5-113	a) The heads of large projects should be close to the head and the secretariat. b) There is no reason for the sequence of assignments of Joachim, Hans and Katharina.
…	

55

Sisyphus-I YQT Members Data

STI · INNSBRUCK

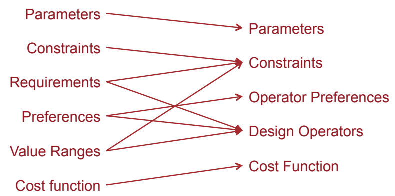
- From the protocol can be drawn data such as:

Name	Role	Project	Smoker	Hacker	Works with
Werner	Researcher	RESPECT	No	Yes	Angi, Marc
Mark	Researcher	KRITON	No	Yes	Angi, Werner
Andy	Researcher	TUTOR	Yes	No	
Harry	Researcher	BABYLON	No	Yes	Jurgen, Thomas
Thomas	Researcher	EULISP	No	No	Jurgen, Harry
Ulrike	Secretary		No	No	Thomas, Monika, Eva
Eva	Manager		No	No	Thomas, Ulrike, Monika
Kathatina	Researcher	MLT	Yes	Yes	
Jurgen	Researcher	EULISP	No	Yes	Thomas, Harry
…					

56

Sisyphus-I PSM-Based Approach

- The approach used rests on mapping from task concepts to method concepts as shown [Motta, 1999] :



Sisyphus-I Values Ranges

- From the protocol can be drawn value ranges, based on given justifications in the protocol:

Type of YQT Member	Value Range	Justification
Head of group	All large, central offices	"The head needs a central office… This should be a large office"
Secretary	All large offices	"Secretaries should work together in one large office"
Manager	A centrally-located office	"should have a centrally located office"
Head-of-project	A single office	Siggi allocates them in single offices
Researcher	Any office	Siggi does not indicate any kind of constraints

Sisyphus-I Requirements and Constraints

- From the protocol can be drawn the following requirements and constraints:

Requirements	Constraints
R1. Head of group in large central office	C1. Do not exceed room size
R2. The secretaries' office has to be close to the office of the head	C2. Smokers cannot share with non-smokers
R3. Manager, head of group, and heads of projects do not share	
R4. Secretaries share the same room	
R5. Manager goes into central office	

Sisyphus-I Preferences

- From the protocol can be drawn the following preferences:

Preference	Comment
P1. Head as close as possible to secretaries	The requirement specifies that they should be close. However, it makes sense to also add a preference so that solutions where the distance between the head and secretariat is minimised are given a higher ranking.
P2. Manager as close as possible to head and secretaries	Siggi talks about having maximum access to the head and the secretariat. This is modelled as a preference, stating that models which minimise the distance between the manager, the head and the secretaries are "better".
P3. Heads of large projects as close as possible to head and secretaries	Siggi actually talks about "heads of projects being close". However his solution does not satisfy his own requirement. Therefore this is modelled as a preference rather than a requirement.
P4. Members of same project should not share	Siggi states that members of the same project should not share. Again his solution does not satisfy this, so it is modelled as a preference.

Sisyphus-I Cost Function

- Cost function produces a 4-dimensional vector, $<n_1, n_2, n_3, n_4>$, where:
  - n_1 measures the distance between the room of the head of group and that of the secretaries;
  - n_2 measures the distance between the manager's room and the rooms of the head of group and the secretaries;
  - n_3 measures the distance between the heads of projects and the head of group and secretaries;
  - n_4 provides a measure of the 'project synergy' afforded by a solution.

61

Sisyphus-I Cost Function (cntd.)

- A design model in the Sisyphus-I domain d1, with cost function $<n_{11}, n_{12}, n_{13}, n_{14}>$ is cheaper than a design model d2, with cost $<n_{21}, n_{22}, n_{23}, n_{24}>$, iff one or more of the following conditions are satisfied:
  - $n_{11} < n_{21}$
  - $n_{11} = n_{21}$ and $n_{12} < n_{22}$
  - $n_{11} = n_{21}$ and $n_{12} = n_{22}$ and $n_{13} < n_{23}$
  - $n_{11} = n_{21}$ and $n_{12} = n_{22}$ and $n_{13} = n_{23}$ and $n_{14} < n_{24}$

62

Sisyphus-I PSM-based Solutions

- Solving by Gen-design-psm (Generic Model for Parametric Design), which enhances simple depth-first search via:
  - Focus selection – a DSR strategy analyses
    - **Value ranges** associated with each parameter,
    - The most constrained parameter in the **current design model**;
  - Operator selection – operator chosen according to given ordering.
- Solving by HC-design (Hill climbing) – same competence as Gen-design-psm, but much less efficient (measured at 10%, as compared to 78%).
- Solving by A*-design, based on heuristic function using estimated cost function – better competence (optimal solution), but comparable efficiency to HC-design.

63

EXTENSIONS

64

Application to Web Service Models

STI · INNSBRUCK

- A standard language was developed for the description of PSMs called the Unified Problem-Solving Method Language (UPML) [Fensel et al, 2002]
- UPML was applied to the modelling of Web Services in the Web Service Modelling Framework (WSMF) [Fensel & Bussler, 2002]
- The WSMF approach was encoded in the Web Service Modeling Ontology (WSMO) – wherein ontology-based models are builts for goals (~tasks), services (~methods) and these are linked by mediators (~adapters) [Fensel et al., 2007]
- WSMO was encoded in a family of ontology languages, WSML, and an open-source implementation was carried out in WSMX.

65

STI · INNSBRUCK

SUMMARY

66

Summary

STI · INNSBRUCK

- Problem-solving methods offer a means to structure and reuse elements of knowledge-based systems by abstracting from their domain.
- Efficiency is achieved by introducing assumptions that either restrict the size of the problem or that postulate strong requirements on the available domain knowledge.
- Adapters link tasks and PSMs to domains, allowing reuse of both, and express refinements between PSMs, allowing libraries to be built of these.

67

STI · INNSBRUCK

REFERENCES

68

References

STI · INNSBRUCK

- Mandatory reading:
  - [Fensel, 2000] "Problem-Solving Methods: Understanding, Description, Development and Reuse", D. Fensel, Springer LNAI 1791, 2000
- Further reading:
  - [Motta, 1999] "Reusable Components for Knowledge Modelling: Case Studies in Parametric Design", E. Motta, IOS Press, 1999
  - [Schreiber et al., 1994] "CommonKADS: A Comprehensive Methodology for KBS Development", A. Th. Schreiber, B. Wielinga and J. M. Akkermans, W. Van de Vedle, and R. De Hoog, *IEEE Expert*, 9(6):28–37, 1994
  - [Stefik et al., 1983] "Basic Concepts for Building Expert Systems", M. Stefik, J. Aitkins, R. Balzer, J. Benoit, L. Birnbaum, F. Hayes-Roth, and E. Sacerdoti in *Building Expert Systems*, Addison-Wesley, 1983

69

References

STI · INNSBRUCK

- [Smith & Lowry, 1990] "Algorithm Theories and Design Tactics", D. R. Smith and M. R. Lowry in *Science of Computer Programming*, 14:305–321Addison-Wesley, 1990
- [Fensel et al, 2002] "The Unified Problem-solving Method Development Language UPML", D. Fensel, E. Motta, R. Benjamins, M. Crubezy, S. Decker, M. Gaspari, R. Groenboom, W. Grosso, F. van Harmelen, M. Musen, E. Plaza, G. Schreiber, R. Studer, B. Wielinga, Knowledge and Information Systems, 5 (1), 83–131, 2002
- [Fensel & Bussler, 2002] "The Web Service Modeling Framework (WSMF)", D. Fensel and C. Bussler, Electronic Commerce Research and Applications, 1(2), 2002
- [Fensel et al., 2007] "Enabling Semantic Web Services:The Web Service Modeling Ontology (WSMO)", D. Fensel, H. Lausen, A. Polleres, J. de Bruijn, M. Stollberg, D. Roman, and J. Domingue, Springer, 2007

70

References

STI · INNSBRUCK

- [Brachman, 1979] "On the Epistemological Status of Semantic Networks" R. J. Brachman. In: N.V. Findler (ed.): Associative Networks: Representation and Use of Knowledge by Computers. New York: Academic Press, 1979, 3-50.
- [Birmingham&Klinker, 1993] "Knowledge Acquisition Tools with Explicit Problem-Solving Methods", W. Birmingham and G. Klinker. The Knowledge Engineering Review 8, 1 (1993), 5-25
- [Feigenbaum, 1977] "The Art of Artificial Intelligence: Themes and Case Studies of Knowledge Engineering," E.A. Feigenbaum. *Proceedings of the International Joint Conference on Artificial Intelligence, Cambridge, MA, 1977*
- [Newell, 1981] "The Knowledge Level" Newell, AI Magazine 2 (2), 1981, p. 1-20

71

References

STI · INNSBRUCK

- Wikipedia and other links:
  - http://en.wikipedia.org/wiki/Problem_solving
  - http://www.wsmo.org/
  - http://www.wsmo.org/wsml/
  - http://www.wsmx.org/
  - http://www.wsmo.org/papers/publications/wsmf.paper.pdf

72

Next Lecture

#	Title
1	Introduction
2	Propositional Logic
3	Predicate Logic
4	Reasoning
5	Search Methods
6	CommonKADS
7	Problem-Solving Methods
8	**Planning**
9	Software Agents
10	Rule Learning
11	Inductive Logic Programming
12	Formal Concept Analysis
13	Neural Networks
14	Semantic Web and Services

73

Questions?

74