

WP4.1 & 4.2: Task & Method Adaptation

Annette ten Teije

Frank van Harmelen

Vrije Universiteit Amsterdam

Abstract

The central idea of IBROW is to reuse and execute knowledge intensive components that are available in libraries on the internet. A central question is then “Which components are optimal for a given problem?”. Consequently, the components must be selected from these libraries. In this document we describe a method for the selection and construction of knowledge intensive components from a particular library, namely the classification library developed elsewhere in the project (task 1.2).

An Intelligent Brokering Service for Knowledge-Component Reuse on the World-Wide Web

Identifier	D12a
Class	Deliverable
Version	Final
Version date	Jan 2003
Status	Final
Distribution	Public
Responsible partner	Vrije Universiteit Amsterdam

IBROW Consortium

This document is part of a research project funded by the IST Programme of the Commission of the European Communities as project number IST-1999-19005. The partners in this project are: the University of Amsterdam, coordinator (NL), the University of Karlsruhe (DE), The Open University (UK), The Spanish Council for Scientific Research-III A (ES), Stanford University (USA), Intelligent Software Components, S.A. (ES), and the Free University Amsterdam (NL).

¹University of Amsterdam

Department of Social Science Informatics (SWI)
Roetersstraat 15
NL-1018 WB Amsterdam, The Netherlands
Tel: +31 20 525 6796; Fax: +31 20 525 6896
Contact person: B.J. Wielinga
E-mail: Wielinga@swi.psy.uva.nl

³The Open University

Knowledge Media Institute
Walton Hall
MK7 6AA, Milton Keynes, United Kingdom
Tel: +44 1908 653506; Fax: +44 1908 653169
Contact person: E. Motta
E-mail: e.motta@open.ac.uk

⁵Stanford University

Stanford Medical Informatics
251 Campus Dr., Suite 215
94305-5479, Stanford, CA, USA
Tel: +1 650 723 6979; Fax: +1 650 725 7944
Contact person: M. Musen
E-mail: musen@smi.stanford.edu

⁷Vrije Universiteit Amsterdam

Faculty of Sciences
Division of Mathematics and Computer Science
De Boelelaan 1081a
1081 HV Amsterdam, the Netherlands
Tel: +31 (0)6 51850619; Fax: +31 20 872 2722
Contact person: D. Fensel
E-mail: dieter@cs.vu.nl

Industrial advisory committee:

DaimlerChrysler (D)
Deutsche Telekom (D)
Bolesian (NL)
Unilever (NL)
IBM (JP)
Artificial Intelligence Applications Institute (UK)

²University of Karlsruhe

Institute AIFB
D-76128 Karlsruhe
Germany
Tel: +49 721 608 3923; Fax: +49 721 608 6580
Contact person: R. Studer
E-mail: studer@aifb.uni-karlsruhe.de

⁴Spanish Council of Scientific Research (CSIC)

Artificial Intelligence Research Institute (III A)
Campus U.A.B.
08193 Bellaterra, Catalonia (Spain)
Tel: +34 935 809 570; Fax: +34 935 809 661
Contact person: Enric Plaza
E-mail: enric@iii.csic.es

⁶Intelligent Software Components, S.A.

iSOCO Sant, Ctro.Neg. Can Castanyer
Crta. de Rubí, 88 bj. A
08190 St.Cugat (Barcelona), Spain
Tel +34 93 544 2048; Fax +34 93 589 5669
Contact person: M. López-Sánchez
E-mail: maite@isoco.com

Revision information

Revision date	Version	Changes
Nov '02	First Draft	
Dec '02	Pre final	Added scenario
Jan '03	Final	Final editorial and fine-tuning of scenario after tests

Contents

1	Goal	2
2	General approach	4
3	Representation of classification tasks	5
3.1	Parameterised schema for classification	6
3.2	Instantiating the parameterised structure	9
3.2.1	parameter: observation-constraints	9
3.2.2	parameter: feature-scoring-mechanism	10
3.2.3	parameter: macro-scoring-mechanism	10
3.2.4	parameter: solution-admissibility-criterion	11
3.2.5	parameter: better-match-scores	13
3.3	Lessons learned	13
4	Configuration/adaptation knowledge for the classification library	14
4.1	Propose knowledge	14
4.2	Verification knowledge	17
4.3	Critique knowledge	17
4.4	Modify knowledge	18
4.4.1	Goals	18
4.4.2	Assumptions	20
4.5	Lessons learned	21
5	Scenario: conference papers classification	22
5.1	Solution space & feature space	22
5.2	Goals and assumptions	22
5.3	Configuration of task	23
5.3.1	Propose step	24
5.3.2	Verification step (1)	25
5.3.3	Critiquing step (1)	25
5.3.4	Modify step (1)	25
5.3.5	Verification step (2)	26
5.3.6	Critiquing step (2)	26
5.3.7	Modify step (2)	26
5.3.8	Verification step (3)	26
5.3.9	Critiquing step (3)	26
5.3.10	Modify step (3)	27
5.3.11	Verification step (4)	27
5.3.12	Critiquing step (4)	27
5.3.13	Modify step (4)	27
5.3.14	Verification step (5)	28
5.3.15	Critiquing step (5)	28
5.3.16	Modify step (5)	28
5.3.17	Verification step (6)	28
5.4	Discussion	28
6	Conclusions	30

1 Goal

The main idea of the IBROW project [8] is to reuse and execute knowledge intensive components. These components are available in libraries on the Internet. A central question is then "Which components are optimal for a given problem?". Consequently, the components must be selected from a library or even constructed. In the former case, components are selected from a predefined set, while in the latter case components are combined or newly defined parts are combined to construct a new component.

The overall architecture that is used for describing these knowledge intensive components is the Unified Problem-Solving Method Description Language (UPML [5, 6]¹). UPML provides us an architecture for components of a knowledge based system (task, problem-solving method, domain-model and ontology), and adapters (kind of connectors) between those components. See figure 1. There are two types of adapters: bridges and refiners. Bridges model the relationship between the components task, problem-solving method, domain-model, and ontology, where as the refiners model the relationship between one type of such a component. For instance, the relationship between different task components.

In this document we describe a method [12] for the selection and construction of knowledge intensive components from a particular library, namely a classification library taken from [10]². In terms of the UPML architecture these library components are task and problem-solving methods components for classification. Selection and construction of components is modelled in the UPML architecture as refiners. The structure of the library is such that the bridge between task and problem-solving method is trivial. This results in our focus on task-refiners. We will show that we can use the general approach of [12] for selection and adaptation of task-definitions. for an independently developed classification library [10].

Current state

In [12] we have given a proposal is given for configuring the competence (functionality) of problem solving methods. The main idea is to consider configuring the competence as a standard configuration task, namely parametric design. A standard method for parametric design is used as solving method, namely propose -critique-modify. Taking this approach, this means that (1) there is a need for describing the competence for a specific type of task (e.g. diagnosis, classification) in a parameterised scheme, and (2) that we have to come up with knowledge that can be used in the propose -critique-modify method. This propose-critique-modify knowledge is the brokering knowledge that is needed for adapting the competence of problem solving methods (task components).

We use the classification library from task 1.2, which means that we developed a parameterised scheme for classification based on this library. All the methods in the classification library had to be expressed as instances of this classification scheme. Furthermore, we have extended the library such that we can come up with some interesting propose-critique-modify knowledge, since this knowledge depends on the content of the library.

¹Validation of UPML and tool support for UPML have been done earlier in the project (Workpackage 2).

²This library is developed earlier in the project (task 1.2)

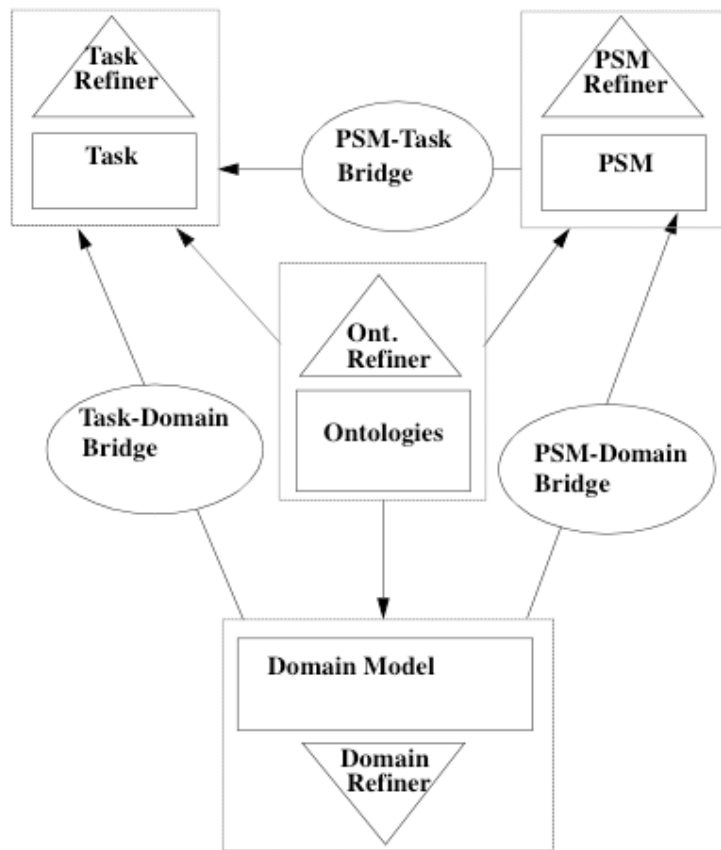


Figure 1: The UPML architecture for knowledge based system [6].

Currently, there is the Internet Reasoning Service (IRS) [4], which gives the user support for building a new application with the components in a particular library (e.g. the classification library). The main support is a browsing facility of possible components in the library, thus supporting the selection of components from a library. The work in this deliverable adds more automatic support for selection and adaptation of components, which can be implemented on top of the current IRS.

Contribution

Our contribution is that we show that we can use the general approach of [12] for classification task/method adaptations for the independently developed classification library [10] by giving (1) a general scheme for representing classification tasks, and (2) brokering knowledge for classification task adaptation.

2 General approach

In this section, we describe our from [12] which uses parametric design to construct problem-solving methods (PSMs). The goal of automated construction of the competence (functionality) of a method is to construct a method that produces acceptable solutions for a given problem under particular assumptions and desired goals. The question in configuration is “Which PSM competence is expected to be optimal?” and “What should be done if the PSM does not give the desired solution?”. The object of the construction is the method description, i.e. the functionality of the method. This task can be considered as a configuration task.

In order to automatically configure PSMs, we need a uniform representation of the objects to be configured (ie the PSMs). The central idea of our representation is that the functionality of a class of problem solving methods is captured in a single schematic formula. Some of the predicates and terms from such a formula are regarded as parameters that must be further instantiated to capture different members of the class of problem solving methods. Such a schematic formula defines the functionality of a whole class of PSMs, and different members of that class correspond to different definitions for the parameters occurring in the schematic formula. This approach has already been shown to be successful for diagnostic PSMs [12]. In this deliverable we will show that it is also applicable for classification PSMs.

Configuration is a special case of the more general task of design. In a configuration task the set of possible components is fixed and the possible connections between the components is fixed. Parametric design is a further simplification of a configuration task, because in parametric design the actual connections between the possible components are already fixed in a given structure. This reduces the configuration problem, because we only have to assign values to each parameter in its own range, and we do no longer have to configure the connections between the components.

The method that is used for parametric design is propose-critique-modify (PCM, [2, 1]). Characteristic of this method is that when a configuration is not a suitable configuration, the configuration process uses the test results for determining a new configuration instead of generating a new one from scratch. The PCM method consists of four steps:

The propose step gives a partial or a complete configuration. It has to propose an instance of the general schema that is used for representing a family of PSMs.

The verify step involves checking that the proposed configuration (PSM) satisfies the constraints. This includes a simulation-verification that consists of performing the task (e.g. diagnosis) followed by tests whether the goals are met.

The critique step is an analysis of why the verification failed, i.e. why the method is not an appropriate method.

The modify step is the determination of an appropriate modification of the method. This means an adaptation in the choices of the parameters.

To summarise this approach: for the automated configuration of PSMs of an arbitrary problem-type (in our case classification) there is need for a parameterised schema for describing a problem solver. Such a schema enables regarding configuration of problem solvers as a parametric design problem. The approach is knowledge intensive since in solving the parametric design problem exploitation of the knowledge of the problem type is required. To apply this approach, the following ingredients are necessary to enable the choice and revision of the parameter values:

1. a general parameterised schema for the representation of the PSMs to be configured
2. definitions of parameters in this general schema
3. propose-critique-modify knowledge that drives the configuration process

In this document we will apply this approach to the task-library of classification methods from WP1.2 [10]. First we have to come up with a structure of this library ((1) a general schema) and to identify the individual components ((2) definitions of parameters). We describe this in section 3. Second we have to look for relations between components, relations between task descriptions, relations between goals/assumptions and components etc. such we can exploit this knowledge as PCM-knowledge. This is knowledge that have to be used as brokering knowledge for classification tasks adaptation. This is described in section 4. All these elements are used together in an application scenario described in section 5.

3 Representation of classification tasks

The literature on Knowledge Engineering has identified a number of different problem types [3, 7], such as diagnosis, design, monitoring, and classification. We will focus on classification. Classification is described in a standard text book of knowledge based systems [11] as:

To classify something- an unknown object, phenomenon, pattern, measurement, or anything at all- is to identify it as a member of a known class. For example, a field biologist might use several features such as shape, size, color, and so on to identify a species of

plant or an animal. In this case, the sample plant or animal is the unknown thing and the species is a known class. Classification problems begin with data and identify classes as solutions. Knowledge is used to match elements of the data space to corresponding elements of the solution space. An essential characteristic of classification, as we use the term, is that it selects from a predefined set of solutions.

Our analysis of classification is based on the classification library developed earlier in the IBROW project [10]. In this library a classification task has a class as a solution (singleton solutions), and gives possible more than one solution (exhaustive search).

In this section we give a general description for the classification tasks. Besides this general scheme, we describe a number of possible instances of the parameters of this scheme.

3.1 Parameterised schema for classification

The classification task of the classification library has as input a set of $\langle \textit{feature}, \textit{value} \rangle$ pairs, and as output a set of solutions. Each solution is represented as a pair $\langle \textit{classname}, \textit{score} \rangle$.

We can describe the classification task as follows (see deliverable D1 for more details): First the observations have to be verified whether they are legal. These legal observations ($\langle \textit{feature}, \textit{value} \rangle$ pairs) have to be scored to every possible solution in the solution space using a mechanism for a feature scoring. These individual scorings of $\langle \textit{feature}, \textit{value} \rangle$ pairs have to be combined in a scoring for the whole set of observations per possible solution, based on a so called macro scoring mechanism. These scores are the basis for the solution admissibility criterion. This criterion determines whether a possible solution is indeed a real solution. A filter is applied on these solutions, which results in the solutions which are the solutions that satisfied the better match scores criterion.

Studying the classification library in OCML, we identified the following six parameters:

- solution-space,
- observation-constraints,
- feature-scoring-mechanism,
- macro-scoring-mechanism,
- solution-admissibility-criterion,
- better-match-scores.

Figure 2 depicts the general scheme for classification methods. The grey boxes are the parameters of the scheme. Concrete task descriptions correspond to concrete values of these boxes.

We discuss each of this parameter in turn, both informally and more formally, and at the end we give the relations between these parameters in a formula.

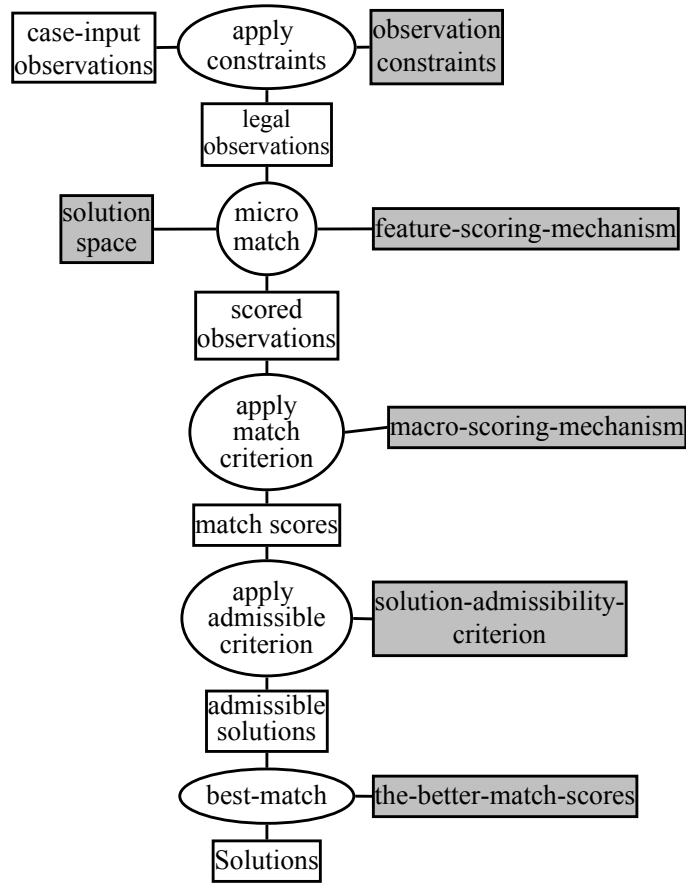


Figure 2: The classification scheme. The grey boxes are the parameters of the scheme. Concrete task descriptions correspond to concrete values of these parameters.

parameter: solution-space

The solution-space contains the class names where a object possibly belongs to. Whether an object belongs to a particular class depends on the feature conditions which are specified for each class.

Every possible solution is connected with a set of $\langle \textit{feature}, \textit{specification} \rangle$ pairs. The value of the feature has to satisfy a given condition. For instance for classifying apples, a possible solution is a granny, and one of its feature value pair is $\langle \textit{colour}, \textit{colour}=\textit{green} \rangle$.

More formally: a solution-space is a set of tuples $\langle \textit{solution}, \textit{feature-specifications} \rangle$, and feature-specifications is a set of tuples of $\langle \textit{feature}, \textit{specification} \rangle$. The specification is a predicate that has to be true for the feature.

parameter: observation-constraints

If the input (the observations) satisfies the observation-constraints then the observations are legal. *More formally:* observation-constraints is a predicate.

parameter: feature-scoring-mechanism

This is the scoring mechanism for each observation with respect to a possible solution. The scoring mechanism indicates a score for each $\langle \textit{feature}, \textit{value} \rangle$ pair of the input to every solution in the solution-space with respect to its feature conditions that are specified for that solution.

More formally: A function *micromatch* has as arguments the parameter *feature-scoring-mechanism*, and a triple $\langle \textit{feature}, \textit{value}, \textit{score} \rangle$ and gives a score for the feature with respect to the solution and observed value.

parameter: macro-scoring-mechanism

The macro-scoring-mechanism aggregate the microscores to a score for the whole candidate class. It assigns a score to each solution.

More formally: A function *macromatch* has as arguments the parameter *macro-scoring-mechanism*, and a set of $\langle \textit{feature}, \textit{value}, \textit{microscore}, \textit{sol}, \textit{f} \rangle$ or a particular *sol*, and gives the score for this solution.

parameter: solution-admissibility-criterion

The solution-admissibility-criterion specifies the requirements for a solution. These requirements are in terms of the results of the macro-scoring-mechanism.

More formally: A predicate *apply-admin* has as arguments the solution-admissibility-criterion and a tuple $\langle \textit{sol}, \textit{macroscore} \rangle$.

parameter: better-match-scores

This criterion specifies a selection of the solutions which are in some sense the best solutions.

More formally: A predicate selection has as arguments the better-match-scores, the candidate solutions and a set of solutions.

The general scheme for classification (figure 2) can be described more precisely by the following formula.

$$\begin{aligned}
& \text{classification}(\text{obs}, \text{sols}) \leftrightarrow \\
& \text{obs-constraints}(\text{obs}) \wedge \\
& \text{scored-obs} = \{ \langle f, v, \text{microscore}, \text{sol} \rangle | \\
& \quad (\langle \text{sol}, \text{fspec-set} \rangle \in \text{solSpace} \wedge \\
& \quad ((\langle f, v \rangle \in \text{obs} \wedge \langle f, s \rangle \in \text{fspec-set}) \vee \\
& \quad (\langle f, v \rangle \in \text{obs} \wedge \langle f, s \rangle \notin \text{fspec-set}) \vee \\
& \quad (\langle f, v \rangle \notin \text{obs} \wedge \langle f, s \rangle \in \text{fspec-set})) \wedge \\
& \quad \text{microscore} = \text{micromatch}(\text{feature-scoring-mechanism}, \langle f, v, s \rangle) \} \wedge \\
& \text{scored-cands} = \{ \langle \text{sol}, \text{macroscore} \rangle | \\
& \quad (\langle \text{sol}, \text{fspec-set} \rangle \in \text{SolSpace} \wedge \\
& \quad \text{scoredobssol} = \{ \langle f, v, \text{microscore}, \text{Sol} \rangle | \\
& \quad \quad \langle f, v, \text{microscore}, \text{Sol} \rangle \in \text{scoredobssol} \} \wedge \\
& \quad \text{macroscore} = \text{macromatch}(\text{macro-scoring-mechanism}, \text{scoredobssol}) \} \wedge \\
& \text{adm-sols} = \{ \langle \text{sol}, \text{macroscore} \rangle | \\
& \quad \text{apply-admin}(\text{solution-admissibility-criterion}, \langle \text{sol}, \text{macroscore} \rangle) \wedge \\
& \quad \langle \text{sol}, \text{macroscore} \rangle \in \text{scored-cands} \} \wedge \\
& \text{selection}(\text{better-match-score}, \text{adm-sols}, \text{sols})
\end{aligned}$$

There is one restriction on these parameters. It is required that the solution-admissibility-criterion respects the better -match-scores. That means if solution S_1 scores higher than S_2 on the better-match-scores, and S_2 passes the solution-admissibility-criterion then S_1 must also pass the solution-admissibility-criterion.

3.2 Instantiating the parameterised structure

In this section we give a number of concrete parameters for the classification schema discussed in the previous section. The complete library is described by these instances of parameters. We discuss every type of parameter in turn, and indicate which instances are in the current classification library and which could be added.

3.2.1 parameter: observation-constraints

Possible values for this parameter are:

OC1-single-value-constraint This says that each feature has at most one value.

OC1a-multi-value This says that it is possible to drop the single-value-constraint on a per-feature basis, resulting in a designated set of multi-valued features.

OC2-legal-feature-value This specifies a predicate that must be true for each feature/value-pair.

OC3-at-least-one-value This specifies that every feature has at least one value.

The library currently contains the parameter values *OC1-single-value-constraint* and *OC2-legal-feature-value*.

The chosen value for the parameter observation-constraints is either one or a combination of the above described observation-constraints. For instance *OC3-at-least-one-value* together with *OC1-single-value-constraint* means that each feature has exactly one value.

3.2.2 parameter: feature-scoring-mechanism

Possible values for this parameter are:

FS1-IEUM A feature can have the following status: inconsistent, explained, unexplained or missing. A feature is inconsistent with respect to a class if the feature is observed in a way that does not satisfied the feature condition of this class. A feature is explained with respect to a class if the feature is observed and satisfied the feature condition of this class. A feature is missing with respect to a class if the feature is not observed and the class has a feature condition for this feature. A feature is unexplained with respect to a class if a feature is observed and the class does not have a feature condition for this feature.

Figure 3 shows an example of this. In this example, the observations are $\{\langle a, o \rangle, \langle b, 2 \rangle, \langle d, 4 \rangle\}$. This makes a an inconsistent feature of class $c1$, b is a explained feature of class $c1$, c is a missing feature for class $c1$, and d is an unexplained feature for class c .

Notice that an observed feature is consistent, explained or unexplained for a class, and only not observed features can be missing (hence the name).

The feature-scoring-mechanism *FS1-IEUM* computes for each feature for each class whether the feature is inconsistent, explained, unexplained or missing.

FS2-closeness Compute for each feature per class how "close" the observed value is to the value prescribed for the class (e.g. giving a number in $[-1,1]$, 0 for unknown values).

The library only contains the parameter value *FS1-IEUM*.

3.2.3 parameter: macro-scoring-mechanism

Possible values for this parameter are:

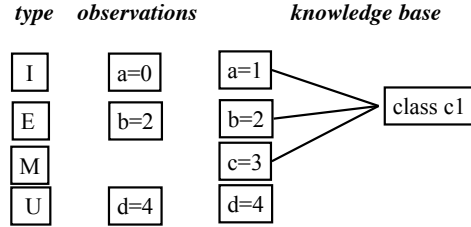


Figure 3: The types inconsistent (I), explained (E), unexplained (U) and missing (M). The observations are $\{\langle a, 0 \rangle, \langle b, 2 \rangle, \langle d, 4 \rangle\}$. a is an inconsistent feature of class $c1$, b is a explained feature of class $c1$, c is a missing feature for class $c1$, and d is an unexplained feature for class c .

MS1-IEUM Collect per class the features that are inconsistent, explained, unexplained or missing and represent these in a 4-tuple $\langle I, E, U, M \rangle$. I denotes the set of inconsistent features, E denotes the set of explained features, U denotes the set of unexplained features, and M denotes the set of missing features.

MS1a-IEUM Collect per class the number of features that are inconsistent, explained, unexplained or missing and represent these in a 4-tuple $\langle |I|, |E|, |U|, |M| \rangle$. Where I , E , U and M denote respectively the set of inconsistent features, explained features, unexplained features, and missing features.

MS2-numeric Some combination-function of all these feature-scores which are in a particular interval (using *FS2-closeness* for the feature scoring mechanism). A possibility is for instance to take the minimum, which is inspired by the conjunction operator in the Certainty Factor calculus. Another possibility would be to take the average.

MS3-cosine Compute per class the cosine between f/v-vector of class and observed f/v-vector, based on similar distance measures used in data-mining.

The current library only contains the value *MS-IEUM* for this parameter.

3.2.4 parameter: solution-admissibility-criterion

Possible values for this parameter are:

AC1-weak-relevant : Each $\langle \text{feature}, \text{value} \rangle$ pair in the observations has to be consistent with the feature specifications of the solution, and at least one feature is explained by the solution. A class c_1 is a solution if its set I denoting the inconsistent features of c_1 is empty ($I = \emptyset$), and its set E denoting the explained features is not empty ($|E| > 0$).

AC2-strong-coverage : A class c_1 is a solution if its set I denoting the inconsistent features of c_1 is empty ($I = \emptyset$), and its set U denoting the unexplained features of c_1 is empty ($U = \emptyset$).

AC3-weak-coverage : Each $\langle \text{feature}, \text{value} \rangle$ pair in the observations have to be consistent with the feature specifications of the solution. A class c_1 is a solution if its set I denoting the inconsistent features of c_1 is empty ($I = \emptyset$).

AC4-explanative : A solution satisfies the requirement of *AC3-weak-coverage* and all feature specifications of a solution have to be valid. A class c_1 is a solution if its set I denoting the inconsistent features of c_1 is empty ($I = \emptyset$), and its set M denoting the missing features is empty ($M = \emptyset$).

AC5-strong-explanative : A solution satisfies the requirement of *AC2-strong-coverage* and *AC4-explanative*. A class c_1 is a solution if its set I , U and M are empty ($I = U = M = \emptyset$).

AC($S_{consistent}$) variants These variants use a weaker requirement for $I = \emptyset$. We do not require that I is empty, but we require that at least certain features are not in I . Therefore we need to define a set of features $S_{consistent}$. All features in $S_{consistent}$ must be consistent, and features outside $S_{consistent}$ are allowed to be inconsistent. If $S_{consistent}$ contains all possible features then *AC-x* is equal to *AC-x*($S_{consistent}$), for instance *AC1-weak-relevant* and *AC1-weak-relevant*($S_{consistent}$). The requirement of $I = \emptyset$ becomes weaker, namely $I \cap S_{consistent} = \emptyset$.

AC(S_{known}) variants These variants use a weaker requirement for $U = \emptyset$. We do not require that U is empty, but we require that at least certain features are not unexplained. Therefore we need to define a set of features S_{known} . All features in S_{known} must be explained when they are part of the observations. Features outside S_{known} are allowed to be unexplained, when they are observed. If S_{known} equal to all possible features then *AC2-strong-coverage* is equal to *AC2-strong-coverage*(S_{known}), idem for *AC5-strong-explanative*. The requirement of $U = \emptyset$ becomes weaker, namely $U \cap S_{known} = \emptyset$.

AC($S_{present}$) variants These variants use a weaker requirement for $M = \emptyset$. We do not require that M is empty, but we require that at least certain features are not missing. Therefore we need to define a set of features $S_{present}$. All features in $S_{present}$ must be observed when they are part of the feature specifications of a solution. Features outside $S_{present}$ are allowed to be not observed. If $S_{present}$ equal to all possible features then *AC-4* and *AC-5* are equal to *AC-4*($S_{present}$) and *AC-5*($S_{present}$) respectively. The requirement of $M = \emptyset$ becomes weaker, namely $M \cap S_{present} = \emptyset$.

AC-variants($S_{consistent}, S_{known}, S_{present}$) There are a number of useful variants possible. Read the S arguments as optional: *AC-strong-coverage*($S_{consistent}, S_{known}$), *AC-explanative*($S_{consistent}, S_{present}$), and *AC-strong-explanative*($S_{consistent}, S_{known}, S_{present}$).

AC6-threshold-cosine : using a threshold value for cosine rule *MS3-cosine*.

AC7-threshold using a threshold value for *MS2-numeric* score.

The current library contains the parameter values *AC1-weak-relevant* and *AC2-strong-coverage*. The parameters *AC2-strong-coverage*, *AC3-weak-coverage*, *AC4-explanative*, and *AC5-strong-explanative* are more or less standard variations in the classification literature. The introduction of parameters like *S_{consistent}* are studied in [9].

3.2.5 parameter: better-match-scores

Possible values for this parameter are:

BM1-lexical-IEUM-size Lexicographic ordering on $\{|I|, -|E|, |U|, |M|\}$ using \langle

BM2-lexical-IEUM-subset Lexicographic ordering using \subset on $\{I, all\ features \setminus E, U, M\}$

BM2-E-min Ordering on $|E|$ using \langle . This can be applied on solutions that are scored in IEUM-tuples.

BM2-lexical-IMUE/IUME-size/subset Other orderings of the values in the IEUM-tuple are possible *and even more logical, considering the admissibility criterion).

BM3-domain-ordering Ignore the score and use a domain-specified ordering on classes (e.g. most-urgent, most-dangerous, most-surprising, etc)

BM4-highest Choose highest value.

BM5-top($\langle n \rangle$) Choose top n values.

BM6-difference(ϵ) Choose all values within ϵ of highest value.

BM7-single-sol Choose the first one of the list. This can be applied on solutions that are scored in IEUM-tuples.

BM8-no-ranking Choose all values that are given. There is no ranking at all. All values are considered as "best" scores. This can be applied on solutions that are scored in IEUM-tuples.

The library only contains the parameter value *BM1-IEUM-size*

3.3 Lessons learned

We have shown that it is possible to develop a general framework for classification tasks that capture the content of the independently developed classification library from [10]. This amounts to restating the classification library in a form such we are able to apply parametric design for constructing and adjusting classification tasks. Based on this framework it is possible to propose new parameter values for the library.

Our approach was a kind of reengineering the general scheme by studying the library. A more efficient way is to include this analysis during the developing of the library. It would be attractive

to annotate a library with its general scheme and its specific instances of parameters of this scheme, such that users of a library have access to the competence of the library (and possibility to extend the library with new instances).

The general framework is developed such that it fits to the classification library. However the framework can be improved by extending the scheme for more classification tasks. There are a number of possible extensions: (1) to deal with multi-class classification, (2) to deal with feature-interactions, (3) to deal with multi-valued features, (4) to add a distinction between contextual features and non-contextual features. The extensions (1) and (2) are significant extension of the library contents. The extensions (3) and (4) are conceptually not difficult.

4 Configuration/adaptation knowledge for the classification library

In this section we identify the brokering knowledge that can be used during construction and adaptation of a classification task. As mentioned in section 2 this brokering knowledge is required for the propose-critique-modify method. Relations between parameters, parameter instances, assumptions and desired goals are important ingredients for this brokering knowledge. We will discuss in turn the knowledge types: the propose knowledge, the verification knowledge, the critique knowledge and the modify knowledge.

4.1 Propose knowledge

The propose step proposes a configuration, i.e. an instance of the general scheme that we use for representing classification tasks. We have the following propose knowledge for the parameters of the classification scheme:

propose knowledge: observation-constraints

if operating under time-pressure
then allow unknown values (do not use *OC3-at-least-one-value*)

if domain-knowledge is available on often missing features
then do not use *OC3-at-least-one-value*

if domain-knowledge is available on multi-valuedness
then use *OC1a-multi-value*

if domain-knowledge is available on value-ranges
then use *OC2-legal-feature-value*

propose knowledge: feature-scoring-mechanism, macro-scoring-mechanism

if values are difficult to observe/noisy

then use *FS2-closeness* and *MS2-numeric*

only use *MC3-cosine* if each f/v pair has a numeric value

if total score should be some "average" of the score on each feature

then use *MS2-numeric* or *MS2-cosine*

propose knowledge: solution-admissibility-criterion

by default $S_{consistent} = S_{known} = S_{present} =$ all features

if unreliable observations
then move these outside $S_{consistent}$

if using *MS1-IEUM*
then use one of *AC1-weak-relevant*, *AC2-strong-coverage*, *AC3-weak-coverage*, *AC4-explanative*,
AC5-strong-explanative

if using *MS3-cosine*
then use *AC6-threshold-cosine*

if using *MS2-numeric*
then use *AC7-threshold*

if many irrelevant observations
then use *AC3-weak-coverage*, *AC4-explanative* or *AC1-weak-relevant*

if missing observations
then use *AC2-strong-coverage* and use domain knowledge to determine $S_{consistent}$, S_{known} ,
 $S_{present}$

if unreliable domain knowledge (with overspecified class definitives)
then do not use *AC2-strong-coverage*, *AC4-explanative* or *AC5-strong-explanative*, and shrink
 $S_{consistent}$

if unreliable domain knowledge (with underspecified class definitions)
then use *AC4-explanative* or *AC5-strong-explanative*

if unreliable domain knowledge (with incorrectly specified class def's)
then behave as with unreliable observations

if expensive observations
then move these out of $S_{present}$

if correctness of solution is important
then use *AC5-strong-explanative*

if completeness of solution is important
then use *AC1-weak-relevant*

propose knowledge: better-match-scores

if not all observations are equally important

then don't use *BM1-lexical-IEUM-size*

if domain ordering knowledge is available

then use *BM3-domain-ordering*

if using *MS1-IEUM*

then use one of *BM1-lexical-IEUM-size*, *BM2-lexical-IEUM-subset*, *BM2-E-min*, *BM7-single-sol*, *BM8-no-ranking*

if using *MS3-cosine*

then use one of *BM4-highest*, *BM5-top($\langle n \rangle$)*, *BM6-difference(ϵ)*

if using *MS2-numeric*

then use one of *BM4-highest*, *BM5-top($\langle n \rangle$)*, *BM6-difference(ϵ)*

if there is no requirement on the completeness of the solutions

then use *BM7-single-sol*

4.2 Verification knowledge

There are two different types of verification knowledge. One type is whether the chosen parameters fit together. We have already captured this type of knowledge in the propose phase. For instance: “if using *MS3-cosine* then use *AC6-threshold-cosine*”. The other type of verification knowledge is whether the goals and assumptions are satisfied with the proposed task. This verification can be realised by executing the proposed task, and then check whether the results are as desired. If this is the case the configuration process is finished. Otherwise the verification phase indicates which goals and or assumptions are violated.

4.3 Critique knowledge

In the critique phase, we have to indicate which parameter can help to improve the task such that the goals and assumption are possibly satisfied.

Goals:

- number of solutions (> 1 , $= 1$, < 1 , $< n$): solution-admissibility-criterion, better-match-scores
- soundness of solutions (no wrong solution): solution-admissibility-criterion, better-match-scores
- completeness of solutions (no missing solutions): solution-admissibility-criterion, better-match-scores

Assumptions:

- availability of number of values per feature ($= 1, > 1, < 1$): observation-constraints
- availability of data: feature-scoring-mechanism, macro-scoring-mechanism, solution-admissibility-criterion
- amount of noise in the data/knowledge: observation-constraints, feature-scoring-mechanism, macro-scoring-mechanism, solution-admissibility-criterion
- observation costs: solution-admissibility-criterion

Although there are alternatives for critiquing a parameter with respect to a particular goal or assumption, we have no knowledge about how to choose between these alternatives. For instance, when to choose parameter solution-admissibility-criterion or better-match-scores for configuring a possibly more appropriate task description. A critique step is based on the verification results: the violation of goals and assumptions.

4.4 Modify knowledge

In the modify phase, we have to indicate how a particular parameter can be adapted such that the goals and/or assumption become satisfied.

In this section we give some examples of modification knowledge of parameters. This knowledge can be used in two ways: first as knowledge that possibly improves the method, and second as knowledge that indicates which changes do not have any positive effect.

4.4.1 Goals

parameter: solution-admissibility-criterion A number of examples of concrete values for the solution-admissibility-criterion are given in section 3.2. Now we will formulate a theorem that we can use during modifying if the verification phase turns out that there are too many or too few (none) solutions. First we have to define an ordering on the defined solution-admissibility-criterion. (everywhere, read $S \subseteq S'$).

Definition 1 (Ordering on solution-admissibility-criterion)

$$\begin{array}{lll}
 AC5\text{-strong-explanative} & \prec & AC4\text{-explanative} & \prec & AC3\text{-weak-coverage} \\
 AC5\text{-strong-explanative} & \prec & AC2\text{-strong-coverage} & \prec & AC3\text{-weak-coverage} \\
 AC1\text{-weak-relevant} & & & & \prec & AC3\text{-weak-coverage}
 \end{array}$$

$AC3\text{-weak-coverage}(S'_{consistent})$	$\prec AC3\text{-weak-coverage}(S_{consistent})$
$AC2\text{-strong-coverage}(S'_{known})$	$\prec AC2\text{-strong-coverage}(S_{known})$
$AC4\text{-explanative}(S'_{present})$	$\prec AC4\text{-explanative}(S_{present})$
$AC2\text{-strong-coverage}(S'_{consistent}, S'_{known})$	$\prec AC2\text{-strong-coverage}(S'_{consistent}, S_{known})$
$AC2\text{-strong-coverage}(S'_{consistent}, S'_{known})$	$\prec AC2\text{-strong-coverage}(S_{consistent}, S'_{known})$
$AC4\text{-explanative}(S'_{consistent}, S'_{present})$	$\prec AC4\text{-explanative}((S'_{consistent}, S_{present}))$
$AC4\text{-explanative}(S'_{consistent}, S'_{present})$	$\prec AC4\text{-explanative}((S_{consistent}, S'_{present}))$
$AC5\text{-strong-explanative}(S'_c, S'_p, S'_k)$	$\prec AC5\text{-strong-explanative}(S_c, S'_p, S'_k)$
$AC5\text{-strong-explanative}(S'_c, S'_p, S'_k)$	$\prec AC5\text{-strong-explanative}(S'_c, S_p, S'_k)$
$AC5\text{-strong-explanative}(S'_c, S'_p, S'_k)$	$\prec AC5\text{-strong-explanative}(S'_c, S'_p, S_k)$

Theorem 1 (A higher solution-admissibility-criterion gives more solutions) given two solution admissibility criteria s_1 and s_2 , with $s_1 \prec s_2$, and all other parameters are equal, then the set of solutions using s_1 will be contained in the set of solutions using s_2 .

If we want to reduce the number of solutions (classes), then we must replace the current definition for the solution-admissibility-criterion by a definition which is lower in the ordering. For instance we can replace *AC3-weak-coverage* by *AC2-strong-coverage* or *AC5-strong-explanative*. The above theorem can be used in the modify phase if a goal related to the number of solutions ($> 1, = 1, < 1, < n$) is violated. We can use the same knowledge for improving completeness or soundness. For instance if we want to improve the completeness, we have to choose a solution-admissibility-criterion that is higher in the ordering. If the current solution-admissibility-criterion is *AC2-strong-coverage*(S'_{known}) then we can reduce the set S'_{known} . Improving the soundness works in the opposite direction.

However, under some conditions, some changes have no effect at all [9].

Theorem 2 (No effect)

$$\forall S \in \text{solutions}(\dots, AC2\text{-strong-coverage}, \dots) \wedge \\ \{f : f \in \text{Obs}\} \subseteq \{f : \langle f, S \rangle \in \text{SolutionSpace}\} \rightarrow \\ \text{solutions}(\dots, AC2\text{-strong-coverage}, \dots) = \text{solutions}(\dots, AC3\text{-weak-coverage}, \dots)$$

$\text{solutions}(\dots, AC2\text{-strong-coverage}, \dots)$ and $\text{solutions}(\dots, AC3\text{-weak-coverage}, \dots)$ represent the solutions of the classification problem using the task definition except for the solution-admissibility-criterion. In this case a change from weak-classification to strong-classification makes no sense, or vice versa.

Other modify knowledge is:

modify knowledge: solution-admissibility-criterion

if *AC4-explanative* gives no solutions

then shift from conjunctive reading to disjunctive reading of the ontology and use *AC2-strong-coverage*

parameter: better-match-scores We have some guidelines for reducing the number of solutions by adapting the better-match-scores. We formulate a theorem that can be used in the modify

step for the better-match-scores, when there are too many or too few (none) solutions. First we have to define an ordering on the defined better-match-scores.

Definition 2 (better-match-scores)

$$\begin{array}{l}
 BM7-single-sol \quad \prec \quad BM1-lexical-IEUM-size \\
 BM1-lexical-IEUM-size \quad \prec \quad BM2-lexical-IEUM-subset \quad \prec \quad BM8-no-ranking \\
 BM1-lexical-IEUM-size \quad \prec \quad BM2-E-min \quad \prec \quad BM8-no-ranking
 \end{array}$$

Theorem 3 (A higher better-match-scores gives more solutions) given two better-match-scores bm_1 and bm_2 , with $bm_1 \prec bm_2$, and all other parameters are equal, then the set of solutions using bm_1 will be contained in the set of solutions using bm_2 .

If we want to reduce the number of solutions (classes), then we must replace the current definition for the better-match-scores by a definition which is lower in the ordering. For instance we can replace *BM2-lexical-IEUM-subset* by *BM1-lexical-IEUM-size*. The above theorem can be used in the modify phase if a goal related to the number of solutions ($> 1, = 1, < 1, < n$) is violated.

Other modify knowledge that can be used for reducing the number of number of solutions is:
modify knowledge: better-match-scores

if *BM2-lexical-IEUM-subset* is used, try *BM1-lexical-IEUM-size*

if top-n is used, try top-1

if epsilon-distance metric is use, decrease the epsilon-distance

To increase the number of solutions the adaptations are vica versa.

4.4.2 Assumptions

In this section we describe the modify knowledge for the adaptation of some parameters which are concerned with a violated assumption in the problem solving environment:

modify knowledge: availability of feature values

if the availability of number of values per feature ($= 1, > 1, < 1$) increases
then move from *OC1-single-value-constraint* to *OC1a-multi-value* or *OC3-at-least-one-value*.

modify knowledge: availability of data

if the availability of data is low
then move from *FS1-IEUM* to *FS2-closeness*

if the availability of data is low
then move from *MS1-IEUM* to *MS2-numeric*

modify knowledge: missing data

if there is missing data
then decrease the set $S_{present}$

modify knowledge: noisy data

if there is more noise in the data
then drop *OC2-legal-feature-value*

if there is more noise in the data
then move from FS1-IEUM to *FS2-closeness*

if there is more noise in the data
then decrease $S_{consistent}$ in the solution-admissibility-criterion parameter

modify knowledge: observation costs

if the cost of observation is higher
then decrease $S_{present}$ in the solution-admissibility-criterion parameter

modify knowledge: unreliable domain knowledge

if the domain knowledge is more unreliable
then decrease $S_{present}$ in the solution-admissibility-criterion parameter

4.5 Lessons learned

In general, identifying useful brokering knowledge is not an easy task, even in a specific case like brokering knowledge for classification tasks. To be more concrete: identifying goals and assumptions is a hard task, and identifying relations between parameters and these goals and assumptions too. However, we have shown in the previous section that it is doable.

During gathering the brokering knowledge we noticed many dependencies between the parameter choices. This is reflected in the propose knowledge where the choice of one parameter limited the choice of others.

At this moment there is a lack of control knowledge in the critique and modify knowledge. In the critique step we have no knowledge about how to decide which parameters to adjust, and in the modify step we have no knowledge about which option to choose for the adjustment of a particular parameter.

The last open issue is the question "where would the knowledge live that one need for intelligent support for task-configuration?" . Should the brokering knowledge be part of the library, as annotations to the components or should the brokering knowledge be part of the broker.

5 Scenario: conference papers classification

This section describes a scenario for the task adaptation process based on the classification scheme (section 3), the instances of the parameters (section 3.2), and the brokering knowledge (section 4).

The scenario concerns the topic classification of conference papers for the reviewing process. The classification is based on keywords given by the author. Automatic classification of conference papers is useful, because of the large volume of papers that have to be classified. Given the fact that authors are surprisingly bad in giving keywords to their paper, an adaptation of the task definition will give advantages. For instances if no solution can be found, the task can be changed such that some solutions are found.

We can use the adaptation of tasks in two ways in the context of the conference paper classification. First we can configure a classification task for the application of classifying conference papers. We use testcases for tuning the right task for this application. Another use of the adaptation of classification tasks is to configure for each paper the most appropriate task. In this section we take the first viewpoint. We use the general goals and assumptions for classifying conference papers, and then during the verification we adapt the task depending on the solutions of a number of test cases.

5.1 Solution space & feature space

The classification of conference papers (ECAI 2002) is based on knowledge about relations between keywords, area and superarea. The possible features are a pre-defined set of keywords. For instance, abduction, belief revision, design, KBS. Each keyword is a feature value of 0 or 1. Consequently there is no distinction between unknown keyword or an absent keyword. Both are represented as 0. The solution classes are the superarea's.

Available data & knowledge

There are 605 submissions for ECAI 2002. Each such submission contains a title, abstract, keywords, and the paper itself. In total there are 1990 keywords used by the authors. The pre-defined list of keywords contains 88 items, there are 15 major categories and 73 minor categories. Furthermore, we have the classification for 189 papers from the program chair. We consider these classifications as the golden standard.

5.2 Goals and assumptions

The classifying problem of conference papers for the reviewing process has a number of goals and assumptions.

G1 at least 1 solution per item (paper)

- I = number of papers classified by the method
- II = number of papers from I agreeing with the Golden Standard
- III = number of papers from I disagreeing with the Golden Standard
- IV = I as % of all papers (605)
- V = II as % of Golden Standard (189)

	I	II	III	IV	V
submitted, all papers	605			100%	-
PC manual, Golden standard	189	189	-	31.29%	100%
No author keys	10			1.66%	-

task definition	I	II	III	IV	V
<i>task</i> ₁	0	0	0	0	0
<i>task</i> ₂	93	16	4	15.4%	80%
<i>task</i> ₃	595	81	99	98.5%	45%
<i>task</i> ₄	595	103	77	98.5%	54.5%
<i>task</i> ₅	595	145	35	98.5%	76.7%
<i>task</i> ₆	595	169	11	98.5%	89.4%

Table 1: These results are obtained from test runs done by our project partners from the University of Amsterdam (SWI)

G2 the golden standard answer must appear in each solution per item

A1 each keyword is a feature value of 0 or 1

A2 there are irrelevant observations (observations which are no keyword from the pre-defined keyword set)

5.3 Configuration of task

The configuration of the classification task is based on the goals and assumptions. In this section we use the above mentioned goals and assumptions and hold these again our propose, critique, modify knowledge. We start with a propose step, which results in a proposed task/method. Then we continue with the verification step. When assumptions or goals are violated then we adapt the task and see whether this task is more appropriate. Below we will discuss the search space of classification tasks and how we traverse through this space (see figure 4).

In table 1 the results of test runs of different classifiers used in this scenario are given. These test runs are done by our project partners from SWI of the University of Amsterdam.

5.3.1 Propose step

In this step we propose a value for each parameter of the classification scheme based on the propose knowledge that we gave earlier.

- observation-constraints: None of the propose knowledge is applicable. For instance there is no time pressure, no multi-valued features. Except there is knowledge available on legal feature values, namely the feature is 0 or 1. Because of the absence of strong propose knowledge we use the simplest option of taking *OC1-single-value-constraint* and *OC2-legal-feature-value* (namely 0 or 1) as observation constraints.
- feature-scoring-mechanism, macro-scoring-mechanism: The observations do not have real values. This means that *FS2-closeness* is no candidate for the feature-scoring-mechanism, and *MS2-numeric* is no candidate for the macro-scoring-mechanism. We have no knowledge that the score is an average of features therefore *MS2-numeric* and *MS3-cosine* are no options. The option that is left for the feature-scoring-mechanism is the *FS1-IEUM*, and there are two options for the macro-scoring-mechanism namely *MS1-IEUM* and *MS1a-IEUM*. Arbitrary is chosen for *MS1-IEUM*.
- solution-admissibility-criterion: Because of the choice of *FS1-IEUM* and *MS1-IEUM*, the options *AC6-threshold-cosine* and *AC7-threshold* are ruled out. Based on the assumption that there are irrelevant observations three possibilities left: *AC1-weak-relevant*, *AC3-weak-coverage* and *AC4-explanative*. Since the propose knowledge concerning unreliable domain knowledge, unreliable observations, whether there are missing observations, expensive observations, importance of correctness and completeness is not applicable, we have to make choice between the three alternatives. We choose the strongest solution-admissibility-criterion among the alternatives, namely *AC4-explanative*.
- better-match-score: Since we do not know whether all observations are equally important, we cannot rule out *BM1-lexical-IEUM-size*. Because there is no domain ordering, possibility of *BM3-domain-ordering* is canceled. The choice of *MS1-IEUM* has the consequence that only the choices *BM1-lexical-IEUM-size*, *BM2-E-min*, *BM2-lexical-IEUM-subset*, *BM7-single-sol* and *BM8-no-ranking* are possible. No knowledge is available about the required completeness. This result in the choice of *BM7-single-sol*.

The proposed task definition is a task that pays attention to the goals and assumption of the specific application of classifying conference papers for the review process. The task definition is a proposal, in the sense that it may be violated some assumptions or goals. Based on the proposed task an improvement of the task will be given in the modify phase.

The current proposed task definition is $task_1$:

observation-constraints	<i>OC1-single-value-constraint, OC2-legal-feature-value</i>
feature-scoring-mechanism	<i>FS1-IEUM</i>
macro-scoring-mechanism	<i>MS1-IEUM</i>
solution-admissibility-criterion	<i>AC4-explanative</i>
better-match-score	<i>BM7-single-sol</i>

See figure 4, we start at the top with this proposed task. In the figure are short names used, ie. OC1 means *OC1-single-value-constraint*.

5.3.2 Verification step (1)

In the verification step we have to verify whether the combination of parameters is ok. This is already done in our propose step, because there we exploit the knowledge about unuseless task configuration. The check whether the solutions satisfy the goals and assumptions should be done by execution of the task and comparing the results to the goals.

Classifying the 605 papers with the proposed method gives no solutions for all these papers. This is in conflict with G1 that at least 1 solution per item (paper) should be given. Also the second goal (G2) is violated. In this scenario the critique step is based on the conflict with G1. See figure 4, the branch of G1, G2 and the work-out branch of G1.

5.3.3 Critiquing step (1)

The question here is which parameter have to be adapt based on the verification results. Increasing the number of solutions can be realised by adapting the better-match-score (BM) and by adapting the solution-admissibility-criterion (AC). See the two branches AC and BM in the figure 4. We work out the AC-branch.

5.3.4 Modify step (1)

The question here is how to adapt a particular parameter with respect to its violated goal. Assuming that we have to increase the number of solutions with the solution-admissibility-criterion our modify knowledge says that in case there are no solutions found with *AC4-explanative*, a switch from conjunctive to disjunctive reading of the ontology (classification knowledge) could be an improvement which leads to using *AC2-strong-coverage*

Other possibilities would be moving to *AC3-weak-coverage* that is higher in ordering or to allow that some features are absent by introducing a set of $S_{present}$, or to allow that some features are inconsistent by introducing a set of $S_{consisent}$. Again see the branches in the tree (fig. 4).

The current task definition is *task₂*:

observation-constraints	<i>OC1-single-value-constraint, OC2-legal-feature-value</i>
feature-scoring-mechanism	<i>FS1-IEUM</i>
macro-scoring-mechanism	<i>MS1-IEUM</i>
solution-admissibility-criterion	<i>AC2-strong-coverage</i>
better-match-score	<i>BM7-single-sol</i>

5.3.5 Verification step (2)

The results of using the adapted task is much better. Indeed the number of papers with a solution increases from 0 to 93. However, still not all papers have a solution, 93 is smaller than 595. Again we do have a conflict with goal G1 at least 1 solution per item.

5.3.6 Critiquing step (2)

Again we relax the solution-admissibility-criterion.

5.3.7 Modify step (2)

Based on our modify knowledge we will adapt the solution-admissibility-criterion which is lower in the ordering as AC2-strong-coverage. This adaption could increase the number of solutions, therefore we take AC1-weak-relevant.

The current task definition is *task*₃:

observation-constraints	<i>OC1-single-value-constraint, OC2-legal-feature-value</i>
feature-scoring-mechanism	<i>FS1-IEUM</i>
macro-scoring-mechanism	<i>MS1-IEUM</i>
solution-admissibility-criterion	<i>AC1-weak-relevant</i>
better-match-score	<i>BM7-single-sol</i>

5.3.8 Verification step (3)

Again, we do a rerun with the adapted task. The number of classified papers increases from 93 to 595. This means that goal G1 is achieved. Given the results there is a discrepancy with goal G2. The golden standard is not always contained in answers. The golden standard is contained in 45% of the cases (45% <<100%). We will continue with adaptation of the task.

5.3.9 Critiquing step (3)

An attempt to remove the discrepancy with goal G2, is again to increase the set of solutions. Since *AC1-weak-relevant* cannot be reasonably weakened anymore, we adapt the better-match-score.

5.3.10 Modify step (3)

For increasing the set of solutions by adapting the better-match-score our modify knowledge leads to use a better-match-score that is higher in the ordering. In our case the closest higher better-match-score in the ordering is *BM1-lexical-IEUM-size*.

The current task definition is *task*₄:

observation-constraints	<i>OC1-single-value-constraint, OC2-legal-feature-value</i>
feature-scoring-mechanism	<i>FS1-IEUM</i>
macro-scoring-mechanism	<i>MS1-IEUM</i>
solution-admissibility-criterion	<i>AC1-weak-relevant</i>
better-match-score	<i>BM1-lexical-IEUM-size</i>

5.3.11 Verification step (4)

Again, we do a rerun with the adapted task. Indeed, 45% of Golden Standard up to 54.5%. However there is still a discrepancy with goal G2. The golden standard is not always contained in answers (45% << 54.5%). We will continue with adaptation of the task, such that the set of answers increase.

5.3.12 Critiquing step (4)

As in critiquing step (3), an attempt to remove the discrepancy with goal G2, is again to increase the set of solutions. Since *AC1-weak-relevant* cannot be reasonably weakened anymore, we adapt the better-match-score.

5.3.13 Modify step (4)

For increasing the set of solutions by adapting the better-match-score our modify knowledge leads to using a better-match-score that is higher in ordering then *BM2-E-min* and *BM8-no-ranking* are candidates for the better-match-score. We choose the candidate that is closest to the current better-match-score, namely *BM2-E-min*. The current task definition is *task*₅:

observation-constraints	<i>OC1-single-value-constraint, OC2-legal-feature-value</i>
feature-scoring-mechanism	<i>FS1-IEUM</i>
macro-scoring-mechanism	<i>MS1-IEUM</i>
solution-admissibility-criterion	<i>AC1-weak-relevant</i>
better-match-score	<i>BM2-E-min</i>

5.3.14 Verification step (5)

Again, we do a rerun with the adapted task. Indeed, 54.5% of Golden Standard up to 76.7%. The goal G2 is not satisfied (76.6 % <<100%). We will continue with adaptation of the task, such that the set of answers increase.

5.3.15 Critiquing step (5)

As in critiquing step (3) and (4), an attempt to remove the discrepancy with goal G2, is again to increase the set of solutions. Since *AC1-weak-relevant* cannot be reasonably weakened anymore, we adapt the better-match-score.

5.3.16 Modify step (5)

For increasing the set of solutions by adapting the better-match-score our modify knowledge leads to using a better-match-score that is higher in ordering. This results in using *BM8-no-ranking* instead of *BM2-E-min*.

The current task definition is *task*₆:

observation-constraints	<i>OC1-single-value-constraint, OC2-legal-feature-value</i>
feature-scoring-mechanism	<i>FS1-IEUM</i>
macro-scoring-mechanism	<i>MS1-IEUM</i>
solution-admissibility-criterion	<i>AC1-weak-relevant</i>
better-match-score	<i>BM8-no-ranking</i>

5.3.17 Verification step (6)

Again, we do a rerun with the adapted task. Indeed, 76.7% of Golden Standard up to 89.4%. This is considered as satisfying goal G2.

5.4 Discussion

With respect to the scenario above we see that there is not very strong propose knowledge concerning the observation-constraints. The search strategy of the PCM method is for the observation-constraints to take the most simple choices if strong knowledge is not available. When the choice is based on an ordering then the closest to the current value is chosen. In other cases the choice of a parameter instance is an arbitrary one if more options are left. As we already mentioned in section 3.3 we have too little knowledge for making a choice for the adjustment of a parameter when there are more possibilities left. For instances better-match-score or solution-admissibility-criterion. The

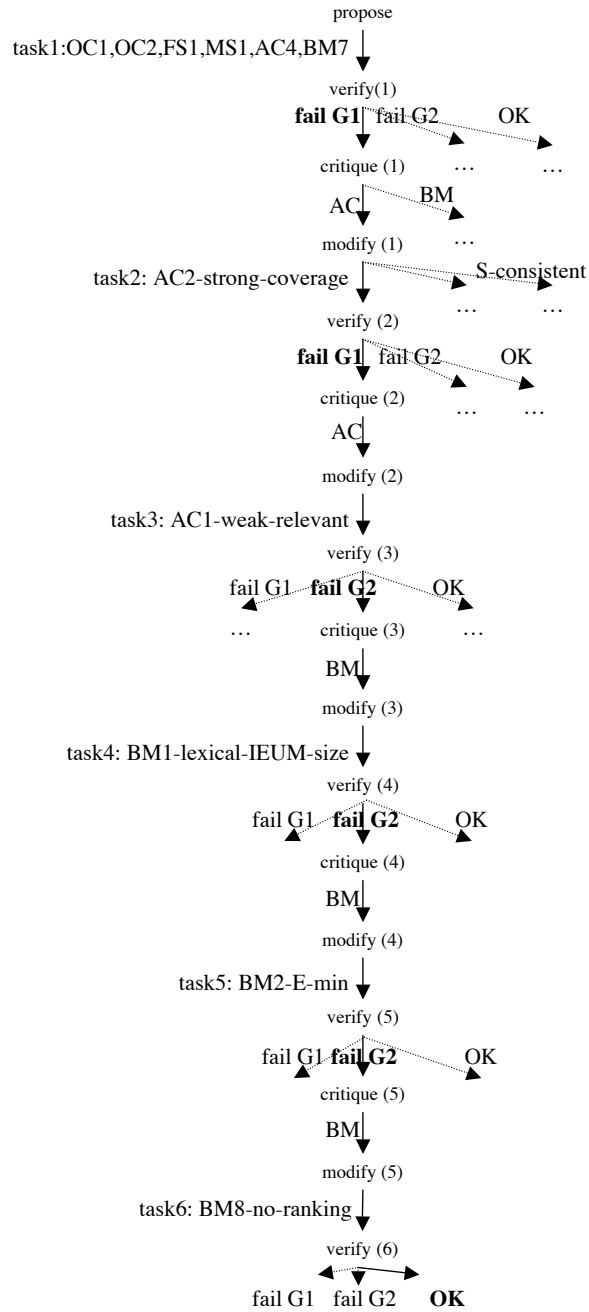


Figure 4: A part of the search space. The trace of the scenario is indicated with the solid arrows. "Task $i = x$ " means that task i is task $i - 1$ adapted with x .

same hold for the modify knowledge, so which modification to choose? In terms of figure 4 this is "how to choose between the branches of a critique step and modify step?"

6 Conclusions

In this deliverable, we have proposed an architecture for brokers. The central idea is that a broker can be seen as performing a parametric design task.

This is a significant reduction in complexity of the broker's task, for two reasons:

First, a broker no longer performs a completely open design task, but rather the task of the broker is limited to choosing parameters within a fixed parameterised structure. This requires that the "problem-solving task to be configured" can be described in terms of such a parameterised structure. For the purposes of our scenario, where a classification task must be configured, we have shown that classification tasks can indeed be represented as the required parameterised structure.

Secondly, viewing a broker as performing a parametric design task also gives a basis for a reasoning model of the broker: propose-critique-modify is a well-understood method for performing parametric design tasks, and can be exploited as the basis for our broker. To this end, the required knowledge for the propose-critique-modify method must be made available to the broker. In our case, we have shown that for the purposes of configuring classification tasks, this knowledge can be made sufficiently precise to be useable in an automated method-broker.

We have shown the feasibility of this approach to broker-construction by describing a specific broker that configures and adapts a classification method to be used in a realistic problem-solving domain, namely the classification of papers submitted to a large AI conference into the subareas needed for the assignment of papers to PC members. In a number of iterations, our broker is able to increase the quality of the classification by successive reconfigurations and adaptations.

We feel confident that this approach to broker-construction (viewing a broker as performing a parametric-design task, and using propose-critique-modify as the broker's method) is applicable in more than just our single example scenario. In earlier work, we have shown that the same approach could be used to successfully configure diagnostic reasoners.

Our experience in both these cases is that the main difficulty in realising the proposed approach lies in the identification of the knowledge for the critique and revise steps: if certain goals are not achieved by the current task-definition, which knowledge must be exploited to identify and repair the current-task-definition in order to improve its performance. Although we have now successfully met this challenge in two separate domains (diagnostic and classification reasoning), only further experiments can tell us if our proposal is indeed generally applicable across a wide variety of reasoning tasks.

References

- [1] D. Brown and B. Chandrasekaran. Design problem solving: knowledge structures and control strategies. *Research notes in Artificial Intelligence*, 1989.
- [2] B. Chandrasekaran. Design problem solving: A task analysis. *AI magazine*, 11:59–71, 1990.
- [3] W. Clancey. Heuristic classification. *Artificial Intelligence*, 27:289–350, 1985.
- [4] M. Crubézy, W. Lu, E. Motta, and M. Musen. Configuring online problem-solving resources with the internet reasoning service. Technical Report SMI-2002-0931, Stanford Medical Informatics, 2002.
- [5] D. Fensel, R. Benjamins, E. Motta, and B. Wielinga. Upml: A framework for knowledge system reuse. In T. Dean, editor, *Proceedings of the International Joint Conference on AI (IJCAI-99)*, Stockholm, Sweden, 1999.
- [6] D. Fensel, E. Motta, F. van Harmelen, V. R. Benjamins, M. Crubezy, S. Decker, M. Gaspari, R. Groenboom, W. Grosso, M. Musen, E. Plaza, G. Schreiber, R. Studer, and B. Wielinga. The unified problem-solving method development language upml. *Knowledge and Information Systems (KAIS)*, 5(1), 2003.
- [7] F. Hayes-Roth, D. Waterman, and D. Lenat. *Building expert systems*. Addison Wesley, New York, 1983.
- [8] IBROW. An intelligent brokering service for knowledge-component reuse on the world-wide-web, ist-1999-19005. Technical report, 1998.
- [9] M. Jansen. An overview of classification criteria for knowledge based systems, 2001. unpublished.
- [10] E. Motta and W. Lu. A library of components for classification problem solving. Technical report, Deliverable D1, IBROW project IST-1999-19005, 2000.
- [11] M. Stefik. *Introduction to knowledge systems*. ISBN: 1-55860-166-X. Morgan Kaufmann Publishers, 1995.
- [12] A. ten Teije, F. van Harmelen, G. Schreiber, and B. Wielinga. Construction of problem-solving methods as parametric design. *International Journal of Human-Computer Studies, Special issue on problem-solving methods*, 49(4), 1998.