Configuration of Web Services as Parametric Design

Annette ten Teije[1], Frank van Harmelen[2], and Bob Wielinga[3]

[1] Dept. of AI, Vrije Universiteit Amsterdam, annette@cs.vu.nl
[2] Dept. of AI, Vrije Universiteit Amsterdam
[3] Dept. of Social Science Informatics, SWI, University of Amsterdam

Abstract. The configuration of Web services is particularly hard given the heterogeneous, unreliable and open nature of the Web. Furthermore, such composite Web services are likely to be complex services, that will require adaptation for each specific use. Current approaches to Web service configuration are often based on pre/post-condition-style reasoning, resulting in a planning-style approach to service configuration, configuring a composite web service "from scratch" every time.
In this paper, we propose instead a knowledge-intensive brokering approach to the creation of composite Web services. In our approach, we describe a complex Web service as a fixed template, which must be configured for each specific use. Web service configuration can then be regarded as parametric design, in which the parameters of the fixed template have to be instantiated with appropriate component services. During the configuration process, we exploit detailed knowledge about the template and the components, to obtain the required composite web service.
We illustrate our proposal by applying it to a specific family of Web services, namely "heuristic classification services". We have implemented a prototype of our knowledge-intensive broker and describe its execution in a concrete scenario.

1 Introduction

Web services have raised much interest in various areas of Computer Science. In AI, the notion of *Semantic Web Services* has attracted much attention. According to [1]:

> "Semantic Web services build on Web service infrastructure to enable automatic discovery and invocation of existing services as well as *creation of new composite services* [...]".

In particular the configuration of Web services (the "creation of new composite services") has gained attention from AI researchers [2, 3]. This problem is particularly hard given the heterogeneous, unreliable and open nature of the Web. Furthermore, such composite Web services will be complex services, that will require adaptation for each specific use.

Current approaches to Web service configuration are often based on pre/post-condition-style reasoning. Given more or less semantic descriptions of elementary Web services, and the required functionality of the composite Web service, they aim to try to construct a "plan" of how to compose the elementary services in order to obtain

the required functionality. Techniques from the domain of planning are heavily being investigated for this purpose [4, 5].

This problem of creation of new composite web services is in principle equal to the old problem of generalised automatic programming. This problem is notoriously unsolved in general by any known techniques. There is no reason to belief that the Web service version of this problem will be any less resistant to a general solution.

In this paper, we propose instead a *knowledge intensive* approach to the creation of composite Web services. Following the general maxim of knowledge-based systems, problems that are in general hard (or even unsolvable) are perfectly solvable in the context of specialised knowledge for specific tasks and domains. In our approach, we describe a complex Web service as a fixed template, which must be configured for each specific use. Web service configuration can then be regarded as parametric design, in which the parameters of the fixed template have to be instantiated with appropriate component services. During the configuration process, we exploit detailed knowledge about the template and the components, to obtain the required composite web service.

Our approach is directly based on well-established work from Knowledge Engineering, and results obtained there in the 90's. Knowledge Engineering has extensively studied the notion of *reusable components* for knowledge-based systems, in particular reusable problem-solving methods: see [6, 7] for general reusability frameworks, and [8, 9] for example collections of reusable components. In essence, our contribution is nothing more than the insight that these results for the configuration of knowledge-based systems from reusable components can be directly brought to bear on the problem of configuring web-services. Indeed, we will propose to use exactly the same configuration method for web-services as has been used for the configuration of KBS components [10].

Whereas in other work the main metaphor is "Web service configuration = planning" (i.e. generalised reasoning based on only component specifications), our approach is based on the metaphor "Web service configuration = brokering" (i.e. reasoning with specialised knowledge in a narrow domain). A planner is assumed to be "domain free": it is supposed to work on any set of components, given simply their descriptions. A *broker* on the other hand (as in: a stock broker, a real-estate broker) exploits specific knowledge about the objects he is dealing with.

The idea of re-using preconfigured templates for Web service configuration also appears in other work: the notion of "generic procedures" in [11], the instantiation of predefined BPEL process models [12], and the coordination patterns from [13].

In the remainder of this paper, we describe how Web services advertise themselves as components to be used by a particular Web service broker, and how such a broker can be equipped with configuration knowledge on how to combine these web services.

In section 2 we describe our general parametric-design approach to Web service configuration. In sections 3 and 4 we illustrate our proposal by applying it to a specific family of Web services, namely "heuristic classification services". In section 5 we describe a specific implementation and execution of our approach.

2 Web Service Configuration as Parametric Design

In this section we will describe what parametric design is, why parametric design is a good basis for a Web service broker, and what descriptions of Web services are required to enable parametric-design reasoning by a broker, and finally we will describe a computational method for solving parametric design problems.

2.1 Parametric Design

Parametric Design is a method for designing objects which is a simplification of general configuration. As any design task, it takes as input the requirements to be met, and produces a design that satisfies these requirements. Parametric Design assumes that the objects-to-be-configured all have the same overall structure in the form of preconfigured templates. Variations on the configuration can only be obtained by choosing the values of given parameters within these templates.

The canonical example of Parametric Design is the design of elevators: every elevator has the same basic structure, namely a column, cable, cabin, counterweight, motor, etc, all in a fixed "template structure". Individual elevators differ only in the values for these parameters: the height of the column, the diameter of the cable, the capacity of the motor, etc. Elevator configuration can be reduced to simply choosing the right values for all these parameters [14].

In the case of web-service configuration, the "template" is a skeletal control structure, which determines how a number of component services will have to be composed. Each component service is then a possible value for one of the parameters within the overall template.

The advantages of Parametric Design in general are: (i) it is one of the easiest forms of configuration, (ii) it is well-studied in the literature [15], and (iii) computational methods are known and tractable (in section 2.2 we will describe one of these methods: propose-critique-modify). Advantages of parametric design for Web services configuration specifically are that the re-use of preconfigured templates avoids repeated multiple configurations of similar composite services for similar applications. These preconfigured templates are a way of "encoding" knowledge that can be used to obtain more sophisticated services than would be possible when configuring "from scratch" (in the sense of planning).

Parametric Design requires that the object-to-be-designed (in our case: a Web service) is described in terms of a fixed structure containing parameters with adjustable values. The following question must be answered before we can confidently apply Parametric Design to the problem of Web service configuration:
Question 1: can realistic classes of Web services be described in this way? This question will be tackled in section 3.

2.2 Propose-Critique-Modify

An existing reasoning method for parametric design is *Propose-Critique-Modify*, or PCM for short [15]. The PCM method consists of four steps:

The propose step generates an initial partial or complete configuration. It proposes an instance of the general template used for representing the family of services.

The verify step checks if the proposed configuration satisfies the required properties of the service. This checking can be done by both analytical pre/post-condition reasoning, and by running or simulating the service.

The critique step . If the verification step fails the critique step analyses the reasons for this failure: it indicates which parameters may have to be revised in order to repair these failures.

The modify step determines alternative values for the parameters identified as culprits by the critique step. After executing the modification step, the PCM method continues again with a verify step. This loop is repeated until all required properties of the service are satisfied.

This method for solving configuration problems has a number of important characteristics: (i) it tries to *incrementally* improve a configuration: when the current candidate configuration does not meet all requirements, it is not thrown away, but instead it is modified in incremental steps. (ii) each of the four steps exploit specific domain knowledge about the objects-to-be-configured (in our case Web services in general, and classification-services in particular). Such domain knowledge is used to propose a good initial configuration, to analyse potential causes of failure, to identify possible modifications, etc. (iii) it does not solve a configuration problem from scratch, but exploits the structure of a predefined template.

The propose-critique-modify method for Parametric Design requires specific types of configuration knowledge to drive the different steps of the configuration process
Question 2: can this PCM-knowledge be identified for realistic classes of Web services? This question will be tackled in section 4.

3 Classification: an Example Family of Web Services

In this section we will illustrate our proposal by applying it to a specific family of Web services, namely "heuristic classification services". This is a good example because:
(i) They are of general applicability and value on the Web. They are used for example, on e-commerce web-sites, to classify products into categories based on their features (price, size, performance, etc) ; in web-site personalisation, to classify pages based on occurences of keywords, date-of-writing, picture-intensity, etc ; or to classify message in streams such as email or news. based on keyword occurrences, sender, date, size etc.
(ii) Heuristic classification services are complex services that require configuration. They must be adjusted to the presence of noise in the dataset, the degree of reliability of the classification rules, the required degree of soundness and completeness of the final classification, etc. All these properties must be taken into account during service configuration.
(iii) Classification is well-studied in the AI literature, so a sufficient body of theory is available as the basis for a configuration theory [16, 17].

The common definition of classification can be found in [18]:

> "To classify something it to identify it as a member of 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 solutions space, whose elements are known in advance."

More formally,

$$\mathit{Classification} : \mathit{Observations} \times \mathit{Knowledge} \rightarrow \mathit{Classes}$$

where *Observations* is a set of $\langle$*feature*,*value*$\rangle$-pairs, and the *Knowledge* involved is a map of sets of $\langle$*feature*,*value*$\rangle$-pairs to *Classes*.

3.1 Template for classification services

We address question 1 above: can a realistic class of classification services be described in a single template? [19] does indeed present such a general template, on which the following structure from fig. 1 is based:

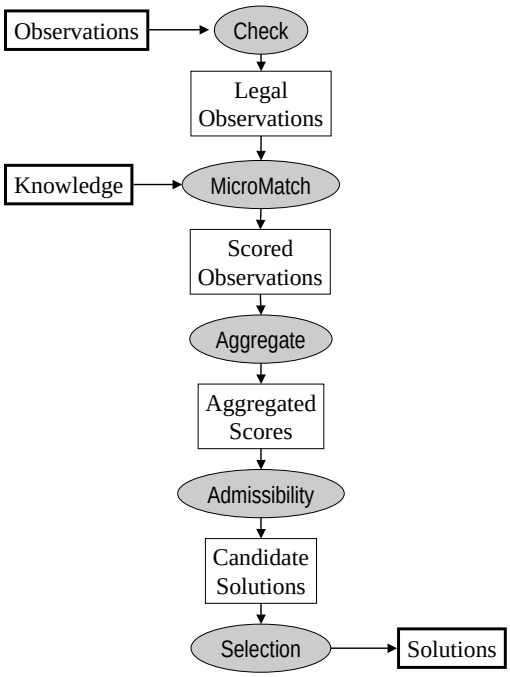

Fig. 1. Structure of classification services. The boxes with thick lines are input and output. The ovals are the parameters of the template for the family of classification services.

First the observations have to be verified whether they are legal (Check). Each of these legal observations ($\langle$*feature*,*value*$\rangle$-pairs) have to be scored on how they contribute

to every possible solution in the solution space (MicroMatch). These individual scores are then aggregated (Aggregate). These aggregated scores are the basis for determining the candidate solutions (Admissibility). A final step (Selection) then selects among these candidate solutions the best final solutions.

This structure constitutes the overall template for classification services. Each box from fig. 1 is one parameter to configure in this fixed template.

We will now show that such a template structure can also be easily captured in current Web service description languages, such as OWL-S [20]. Any OWL-S description is conceptually divided into three sub-parts for specifying what a service does (the *profile*, used for advertising), how the service works internally (the *process model*) and how to interoperate with the service via messages (the *grounding*). We use the schematic notation for OWL-S introduced in [21]. `I(.)` and `O(.)` denote input- and output-arguments. We have used the capitals-only abbreviated version of the identifiers from figure 1 for typesetting reasons.

```
*Profile: PrClassification(I(O), I(K), O(S))
            (hasProc = CPClassification)

*ProcessModel
  CompositeProcess: CPClassification: sequence
  {AtomicProcess: APCheck(I(O), O(LO))
   AtomicProcess: APMicroMatch(I(LO), I(K), O(SO))
   AtomicProcess: APAggregate(I(SO), O(AS))
   AtomicProcess: APAdmissibility(I(AS), O(CS))
   AtomicProcess: APSelection(I(CS), O(S))
  }
```

Different forms of classification are made up of different values for the five "AtomicProcess" components (and their groundings as specific pieces of code). These AtomicProcess components are the parameters within the predefined template of the OWL-S description.

3.2 Components for classification services

We now give some example values of the different parameters, to illustrate the search space of the service-configuration process (more examples can be found in [22]):

Example values of the Check parameter:
• single-value: each feature is required to have at most one value.
• required-value: each feature is required to have at least one value.
• legal-feature-value(P): This specifies that a given predicate P must be true for each $\langle$*feature*,*value*$\rangle$-pair.

Example values of the MicroMatch parameter:
• MicroMatch-IEUM: Each feature can have the status inconsistent, explained, unexplained or missing. A feature is inconsistent w.r.t. a class if its value does not satisfy the

feature condition of this class. A feature is explained w.r.t. a class if it is observed and satisfies the feature condition of this class. A feature is missing w.r.t. a class if it is not observed yet the class has a feature condition for this feature. A feature is unexplained w.r.t. a class if it is observed yet the class does not have a feature condition for this feature. MicroMatch-IEUM computes for each feature for each class whether the feature is inconsistent, explained, unexplained or missing.
• MicroMatch-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], with 0 for unknown values).

Example values of the Aggregate parameter:
• Aggregate-IEUM: Collect per class the set of features that are inconsistent, explained, unexplained or missing and represent these in a 4-tuple $\langle I, E, U, M\rangle$, where I denotes the set of inconsistent features, etc.
• Aggregate-#-IEUM: Count per class the number of features that are inconsistent, explained, unexplained or missing and represent these in a 4-tuple $\langle |I|, |E|, |U|, |M|\rangle$.

Example values of the Admissibility parameter:
(The following are all taken from [18]).
• weak-coverage: Each $\langle$*feature,value*$\rangle$ pair in the observations has to be consistent with the feature specifications of the solution. In other words, a class c_1 is a solution if its set I denoting the inconsistent features of c_1 is empty ($I = \emptyset$).
• weak-relevant: Each $\langle$*feature,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 ($|E| > 0$).
• strong-coverage: These are weak-coverage solutions with no unexplained features ($U = \emptyset$).
• explanative: These are weak-coverage solutions for which no feature specifications are missing ($M = \emptyset$).
• strong-explanative: These solutions satisfy both the requirements of strong-coverage and of explanative.

Example values of the Selection parameter:
• IEUM-size: Compute the minimal element under the lexicographic ordering on $\{|I|, -|E|, |U|, |M|\}$ using $<$. In other words: minimising inconsistent features, maximising explained features, and minimising unexplained and missing features (in order of importance).
• IE-size: As IEUM-size, but disregarding unexplained and missing features.
• Single-solution: Simply choose an arbitrary solution from the candidates.
• No-ranking: Return all candidate solutions, ie. there is no ranking at all, and all values are considered as "best" scores.

We have illustrated a number of instances of the parameters that can be used in the overall template for classification services. We now show that such parameter instances can be described in current Web service description languages (e.g. OWL-S). Below we give an example of a Check parameter.

```
*Profile: PrSingleValue(I(O), O(LO))
            (hasProc = APSingleValue)
            (serviceCategory = Check)
*ProcesModel:
  AtomicProcess: APSingleValue(I(O), O(LO))
*Grounding:
  GrSingleValue(APSingleValue--> single-value)
```

This describes an atomic service with a specific implementation (*grounding*). The service is registered to belong to the given `serviceCategory` `Check`. This allows the configuration process to discover that this specific component can be used to instantiate the AtomicProcess "APCheck" in the overall ProcessModel.

Summary: In summary, in this section we have shown that it is possible to develop a general structure (template) for a complex family of Web services (in our case for heuristic classification services), and that there exists a large variety of possible values for the individual components ("parameters") in this general template. This means that we can apply parametric design for constructing and adjusting classification Web services. We have also shown that both the the general structure and the individual components can be described in OWL-S.

4 PCM-Broker Knowledge

In the previous section, we have seen that indeed Web services can be represented in the form that is required for parametric design.

The question still remains if it is possible to identify the knowledge required for the propose-critique-modify method and each of its four steps (i.e. question 2 identified in section 2.2). We will now show that this is indeed the case, by giving parts of the PCM knowledge required to configure classification services. (Again, more examples can be found in [22]).

Example Propose knowledge for the Admissibility parameter:

- The following values for the Admissibility parameter are compatible with the value MicroMatch=MicroMatch-IEUM: weak-relevant, weak-coverage, strong-coverage and explanative.
- if many ⟨*feature,value*⟩ pairs are irrelevant, then do not use strong-coverage (because strong-coverage insists on an explanation for *all* observed features, including the irrelevant ones).

Example Propose knowledge for the Selection parameter:

- The following values for the Selection parameter are compatible with the value MicroMatch=MicroMatch-IEUM: IEUM-size, IE-size, no-ranking and single-solution.
- if not all observations are equally important, then do not use Selection=IEUM-size, since IEUM-size simply counts numbers of features in each category, given them all equal weight.

Example Critique knowledge for the Selection parameter:
• When the solution set is too small (e.g. empty) or too large (e.g. > 1), then adjust the Admissibility or the Selection parameter. *How* this adjustment should be done is part of the modify-knowledge for these parameters:

Example Modify knowledge for the Admissibility parameter:
• If the solution set has to increased (reduced) in size, then the value for the Admissibility parameter has to be moved down (up) in the following partial ordering:

$$\begin{array}{l} \text{weak-coverage} \prec \text{weak-relevant} \\ \text{weak-coverage} \prec \text{strong-coverage} \prec \text{strong-explanative} \\ \text{weak-coverage} \prec \text{explanative} \quad \prec \text{strong-explanative} \end{array}$$

• If the configuration Admissibility=explanative gives no solutions, then choose Admissibility=strong-coverage. (This amounts to shifting from a conjunctive reading of class-definitions in terms of $\langle$*feature*,*value*$\rangle$ pairs to a disjunctive reading).

Example Modify knowledge for the Selection parameter:
• If the solution set has to be increased (reduced), then the value for the Selection parameter has to be moved down (up) in the following ordering:

$$\text{no-ranking} \prec \text{IE-size} \prec \text{IEUM-size} \prec \text{single-solution}$$

Conclusion: These examples affirmatively answer our question 2 from section 2.2. A PCM-broker requires knowledge about component-services in order to perform its task, and it has turned out to be possible to identify such knowledge for a realistic class of classification Web services.

This leaves open the question on how this knowledge is best represented in the broker. However, it should be clear that OWL-S is *not* the language in which this knowledge is expected to be stated. OWL-S is only used (1) to specify the general schema for the overal service to be configured (in the form of a `CompositeProcess`, and (2) to specify the seperate atomic services that can be used to fill out this general schema (in the form of a `AtomicProcess`). The implementation behind our example scenario in section 5 uses an ad-hoc Prolog representation for the brokering knowledge, but more principled representations can be found in the Knowledge Engineering literature

5 An Example Scenario

In section 3 we have already argued that classification services are used in many Web service scenario's (e-commerce, personalisation, alerting-services, etc.). To test our proposed brokering approach to Web service composition, we have chosen to configure the services needed to support Programme Chairs of major scientific conferences. Such services are available on commercial websites[4]. Currenlty, it is up to the programme chair to configure the services offered by such sites. Ideally, such web-services should

[4] e.g. `http://www.conferencereview.com/`

be configured in a (semi-)automatic scenario, which is what we will investigate in this section.

All scientific conferences are in essence similar, yet no two are exactly the same. Papers are always received, classified into areas, and allocated to reviewers, but the details of this process vary greatly: how many areas are available, how are they characterised, are papers allowed to fall under multiple areas, etc. This makes classification of conference papers a good example case for a parametric-design broker: a generally valid template, but with so much variation that a non-trivial configuration process is required.

In our experiment, we have emulated the paper-classification process for the ECAI 2002 conference. There were 605 submissions to ECAI 2002, each characterised by a set of author-supplied keywords, i.e. each keyword is a ⟨*feature,value*⟩-pair with value either 0 (keyword absent) or 1 (present). In total, 1990 keywords were given by authors. These had to be mapped onto 88 classes ("topic areas"): 15 broad classes which were further subdivided into 73 more specific classes. Of the 650 papers, 189 were classified by hand by the Programme Chair. These classifications can be considered as a golden standard.

Requirement 1: The classification service must classify each paper in at least one of the 15 major categories (since these reflected the structure of the programme committee).

Requirement 2: The service must reproduce the Chair's solution on the 189 handclassified papers.

Important characteristics of this domain are that:

Characteristic 1: the feature-values are often noisy (authors choose remarkably bad keywords to characterise their paper), and

Characteristic 2: it is hard to determine in advance what the required classification mechanism should be. Requiring all keywords of a paper to belong to a solution class might be too strict, resulting in many unclassified papers, and violating requirement 1. But requiring only a single keyword to appear might well be too liberal, causing violation of requirement 2. Again, these characteristics ensure that this domain is indeed suited for a dynamic configuration of the classification process.

We now discuss the iterative service-configuration process performed by our PCM broker. The scenario is summarised in the table below:

Iteration	Answers	Golden Standard	Modification
1	**0 (0%)**	0 (0%)	Admissibility
2	**93 (15%)**	16 (8%)	Admissibility
3	595 (98%)	**81 (45%)**	Selection
4	595 (98%)	**103 (54%)**	Selection
5	595 (98%)	**145 (76%)**	Selection
6	595 (98%)	**169 (89%)**	

• **Propose$_1$:** the broker generates an initial configuration. Based on the domain characteristics described above, a simple Check-parameter checks whether all features have at most one 0/1-value. The default-choice MicroMatch=MicroMatch-IEUM is taken, with the corresponding value Aggregate=Aggregate-IEUM. Of the values for the Admissibility parameter that are compatible with the chosen MicroMatch method, the broker initially

takes the most conservative choice: Admissibility=explanative. The given requirements and domain characteristics do not strongly favour any particular value for the Selection parameter, so the broker chooses the default-value Selection=single-solution.
• **Verify**$_1$**:** The broker now determines that this initial choice is not very successful (see the first entry in the table): no papers are assigned to any class, violating both requirement 1 and 2.
• **Critique**$_1$**:** The broker now determines which parameter has to be adjusted. Increasing the number of solutions can be realised by adapting the Selection-criterion and by adapting the Admissibility-criterion.
• **Modify**$_1$**:** Since the number of solutions has to increase, it is attractive to adopt Admissibility=strong-coverage (switching from a conjunctive to a disjunctive reading of the class definitions in terms of keywords).
• **Verify**$_2$**:** This does indeed improve the results of the classification (2nd iteration in the table above), but not enough. Both requirements are still strongly violated.
• **Critique**$_2$**:** Again the broker decides to adjust the Admissibility-criterion.
• **Modify**$_2$**:** The next value that is one step weaker than the current choice is Admissibility=weak-coverage.
• **Verify**$_3$**:** Now requirement 1 is all but fulfilled, but requirement 2 still fails (iteration 3 in the table).
• **Critique**$_3$**:** An option to remove this failure is again to increase the set of solutions. Since the value Admissibility=weak-coverage cannot be reasonably weakened anymore, the broker decides to adapt the Selection-parameter.
• **Modify**$_3$**:** The next option down from the current value is Selection=IEUM-size.
• **Verify**$_4$**:** Although increasing, the Golden Standard is not yet achieved (only 45%).
• **Critique**$_4$**:** By the same reasoning as in *Critique*$_3$, the broker decides to further adjust the Selection-parameter.

After a repeated series of six of such cycles, the broker finally arrives at a web-configuration that satisfies requirements 1 and 2 to a sufficient degree.

A snapshot of part of the broker's searchspace for this scenario is displayed in figure 2: at some point in the brokering process, a particular service configuration consists of a certain set of components, say $\langle c_1, c_2, c_3, c_4, c_5, c_6 \rangle$, with each of the c_i being a value for the corresponding parameter in the template from figure 1. At that point, the verify step detects this configuration fails to satisfy requirement 1. An alternative path in this search space would be to notice that the other requirement is not satisfied. If all requirements had been satisfied, that would have lead to the current configuration $\langle c_1, c_2, c_3, c_4, c_5, c_6 \rangle$ as a terminal node in the search space. After noticing the failure to comply with requirement 1, a subsequent critique step determines the parameters that may be the culprit for this failure. Again, it is a matter of search strategy to decide which culprit to choose. Each of these choices leads to a subsequent modify step to repair the identified culprit. In our scenario, the broker decides to identify the Admissibility-criterion as the culprit. The modify step then has three options to adjust this Admissibility-criterion, and the broker chooses to select Admissibility=strong-coverage. This results in a new configuration, $\langle c_1, c_2, c_3,$ strong-coverage$, c_5, c_6 \rangle$, which is then again subject to a verify step in the next iteration in this search space.

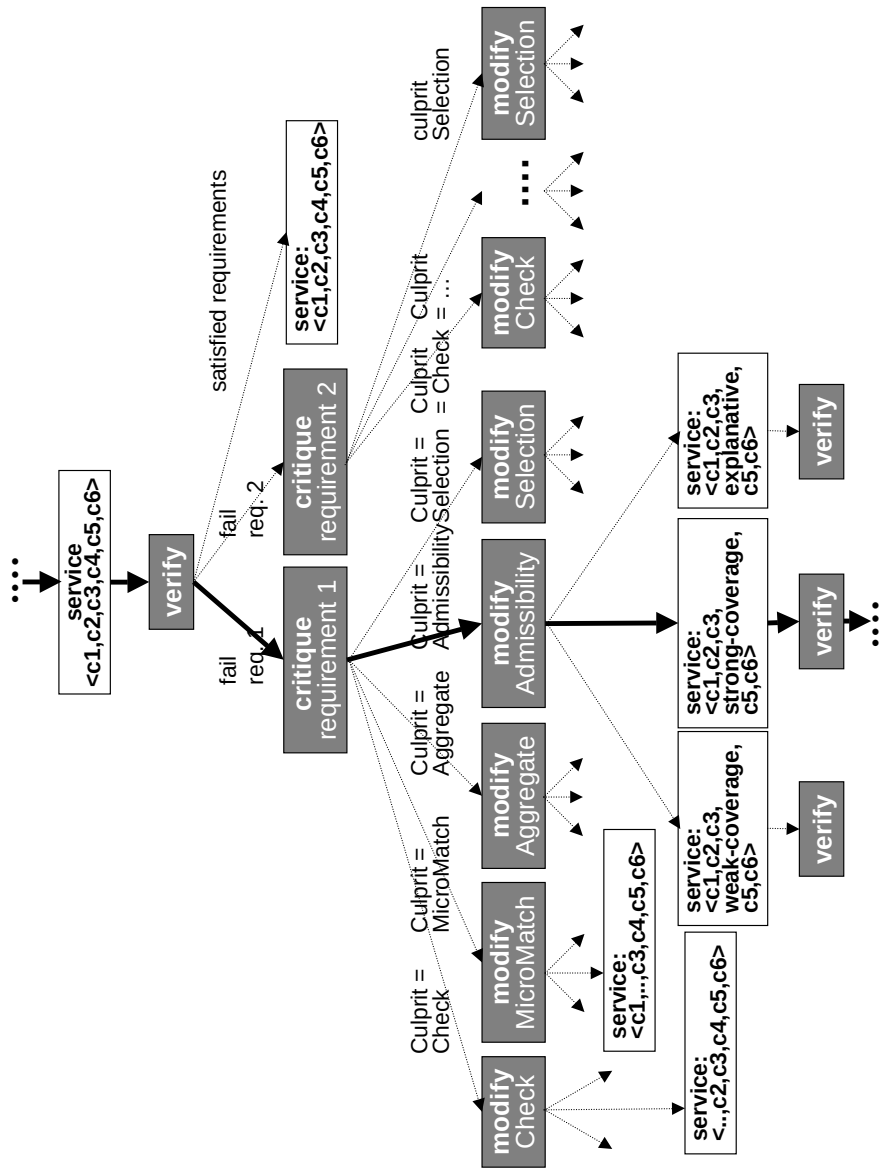

Fig. 2. Part of the search space of broker in the example scenario. Tuples $\langle c_1, c_2, c_3, c_4, c_5, c_6 \rangle$ in white boxes represent service configurations (ie *states* in the search space), while grey boxes represent steps of the propose-critique-modify method (ie *transitions* in the search space). The thick arrows indicate the path in the search space chosen by the broker in the example scenario, while thin arrows indicate possible alternatives in this search space.

The entire scenario above is implemented in SWI-Prolog[5] The PCM-broker uses the template for classification services from figure 1, and a library of service-components much larger than those described in section 3, together with brokering knowledge as described in section 4.

The scenario from this section illustrates that indeed: (i) the broker configures a Web service by iteratively adjusting a fixed template of the service, and (ii) the broker uses extensive knowledge of the constituent services used to fill the parts of this template. This substantiates our metaphor in section 1 that our configuration process is "knowledge-intensive brokering", and not "generalised planning".

6 Limitations

As stated in the introduction, our approach to web-service configuration is based on earlier work on configuring reusable components of knowledge-based systems. Consequently, our proposal suffers from a number of limitations causes by mismatches between the old setting and the new one. We will now discuss some of these limitations.

The most obvious problem with our approach is the amount of high quality knowledge that the broker must be equipped with. This concerns both the general template (fig. 1) and the knowledge required to drive the propose-critique-modify steps (section 4). On the one hand, this meta-knowledge makes our approach to web-service configuration more computationally feasible then the generic planning approach, on the other hand the costs of acquiring this knowledge may well be prohibitive in a web-service scenario.

A second problem concerns that fact that candidate configurations are tested by actually executing them (the "verify"-step). In application domains where the service execution has irreversible effects in the real world, such multiple trials of a web-service would not be allowed (think for example what this would do to credit-card payments!). In such domains, the verification step must be done entirely through reasoning in the broker. Our previous experience in writing brokers ([10]) indicates that sufficiently strong verification knowledge will be very hard to obtain.

A final and more subtle problem concerns the fact that the current broker knowledge refers to individual web-service components by their name (see the examples in section 4). This is reasonable in a library-setting (as in the origins of our work in [10]), where the broker can be assumed to know which components are available. However, this is unrealistic in an open-world web-service scenario, where the broker cannot be assumed to know beforehand all the component services it has available for configuration. Ideally, new component services should be able to register themselves with the broker, declaring their type and properties, enabling the broker to include them in any informed choice it makes. This requires two changes over our current meta-knowledge: Firstly, the components must explicitly state their functional properties when they register themselves. Although principle in possible, current web-service languages like OWL-S do not provide any agreed-upon formalism for stating such functional properties [21]. Secondly, the broker must then use these properties to derive relations between

[5] `http://www.swi-prolog.org/`

components, such as the partial orderings in section 4, instead of having been given these relations explicitly, as is the case now. Of course, deriving such relations from the properties of the individual component-services would be a very hard reasoning task (and is currently done by the knowledge engineers that were building the broker (= us)).

7 Conclusion

In this paper, we have proposed an architecture for Web service brokers. The central idea is that a broker performs a parametric design task. This significantly reduces the complexity of the broker's task, for two reasons:

First, a broker no longer performs a completely open design task (as in more mainstream planning-style approaches to Web service configuration). Instead, the task of the broker is limited to choosing parameters within a fixed structure. This requires that the "Web service to be configured" can be described in terms of such a parameterised structure. For the case of classification Web services, we have shown that these can indeed be represented in this way, using current Web service description languages as OWL-S.

Secondly, viewing brokering as parametric design gives a reasoning model for the broker: propose-critique-modify (PCM) is a well-understood method for parametric design tasks, and can be exploited as the basis for the broker. PCM brokering offers the possibility of dynamically adapting the Web service on the basis of an assessment of the results of executing an earlier configuration. To this end, the required knowledge for the PCM method must be made available to the broker.

We have shown that for configuring heuristic classification tasks, this knowledge can be made sufficiently precise to be useable in an automated broker.

We have shown the feasibility of our approach by describing a specific broker that configures and adapts a classification service to be used for a realistic task, namely the classification of papers submitted to a large AI conference. In a number of iterations, our broker is able to increase the quality of the classification by successive reconfigurations.

We feel confident that this approach to Web service configuration is applicable in more than just our single example scenario (see for example our own work on diagnostic reasoners [10] and work by others on general task models [6]).

Our experience in realising the example scenario is that the main difficulty with the proposed approach lies in the identification of the knowledge for the critique and revise steps: if certain requirements are not met by the current service-configuration, which knowledge must be exploited to identify and repair the current configuration 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 if our proposal is indeed generally applicable across a wide variety of Web services.

Acknowledgements: This research was partially supported by the European Commission through the IBROW project (IST-1999-19005). Enrico Motta designed and implemented the library of classification components. Anjo Anjewierden implemented the

feature extraction module applied to ECAI manuscripts. The PCM implementation was based on an unpublished design of Guus Schreiber. Machiel Jansen provided useful insights in the nature of classification. Marta Sabou advised us on the use of OWL-S. We also thank three anonymous reviewers for their insightful suggestions on how to improve this paper.

References

1. M. Kiefer. Message to swsl-committee@daml.org, May 14, 2003.
2. Sirin, E., Hendler, J., Parsia, B.: Semi-automatic composition of web services using semantic descriptions. In: Web Services: Modeling, Architecture and Infrastructure workshop in conjunction with ICEIS2003. (2003)
3. Narayanan, S., Mcllraith, S.: Simulation, verification and automated composition of web services. In: Proc. of the Eleventh International World Wide Web Conference, Honolulu (2002)
4. Wu, D., Sirin, E., Parsia, B., Hendler, J., Nau, D.: Automatic web services composition using SHOP2. In: Proceedings of Planning for Web Services Workshop in ICAPS 2003. (2003)
5. Sheshagiri, M., desJardins, M., Finin, T.: A planner for composing service described in daml-s. In: Proceedings Workshop on Planning for Web Services, International Conference on Automated Planning and Scheduling, Trento (2003)
6. Schreiber, G., Akkermans, H., Anjewierden, A., de Hoog, R., Shadbolt, N., de Velde, W.V., Wielinga., B.: Knowledge Engineering and Management: The CommonKADS Methodology. ISBN 0262193000. MIT Press (2000)
7. Chandrasekaran, B.: Generic tasks as building blocks for knowledge-based systems: The diagnosis and routine design examples. In: The Knowledge Engineering Review. ? (1988) 183–210
8. Benjamins, V.R.: Problem Solving Methods for Diagnosis. PhD thesis, University of Amsterdam, Amsterdam, The Netherlands (1993)
9. Valente, A., Benjamins, R., de Barros, L.N.: A library of system-derived problem-solving methods for planning. International Journal of Human Computer Studies **48** (1998) 417–447
10. ten Teije, A., van Harmelen, F., Schreiber, G., Wielinga, B.: Construction of problem-solving methods as parametric design. International Journal of Human-Computer Studies, Special issue on problem-solving methods **49** (1998)
11. McIlraith, S., Son, T.: Adapting golog for composition of semantic web services. In: Proc. of the International Conference on the Principles of Knowledge Representation and Reasoning (KRR'02). (2002) 482–496
12. Mandell, D., McIlraith, S.: Adapting bpel4ws for the semantic web: The bottom-up approach to web service interoperation. In D. Fensel, C.S., Mylopolis, J., eds.: Proceedings of the International Semantic Web Conference (ISWC). Volume 2870 of LNCS., Springer Verlag (2003) 227–241
13. van Splunter, S., Sabou, M., Brazier, F., Richards, D.: Configuring web service, using structurings and techniques from agent configuration. In: Proceedings of the 2003 IEEE/WIC International Conference on Web Intelligence (WI 2003), Halifax, Canada (2003)
14. Schreiber, A., Birmingham, W.: The sisyphus-vt initiative. International Journal of Human-Computer Studies, Special issue on VT **44 (3/4)** (1996) 275–280
15. Brown, D., Chandrasekaran, B.: Design problem solving: knowledge structures and control strategies. Research notes in Artificial Intelligence (1989)
16. Clancey, W.: Heuristic classification. Artificial Intelligence **27** (1985) 289–350

17. Jansen, M.: Formal explorations of knowledge intensive tasks. PhD thesis, University of Amsterdam (SWI) (2003)
18. Stefik, M.: Introduction to knowledge systems. ISBN: 1-55860-166-X. Morgan Kaufmann Publishers (1995)
19. Motta, E., Lu, W.: A library of components for classification problem solving. In: Pacific Rim Knowledge Acquisition Workshop, Sydney, Australia (2000)
20. Ankolekar, A., Burstein, M., Hobbs, J., Lassila, O., Martin, D., McDermott, D., McIlraith, S., Narayanan, S., Paolucci, M., Payne, T., Sycara, K.: Daml-s: Semantic markup for web services. In Horrocks, I., Hendler, J., eds.: Proceedings of the International Semantic Web Conference (ISWC). Volume 2342 of LNCS., Sardinia, Springer (2002) 348–363
21. Sabou, M., Richards, D., van Splunter, S.: An experience report on using daml-s. In: Workshop on E-Services and the Semantic Web (ESSW '03), The Twelfth International World Wide Web Conference., Budapest, Hungary, (2003)
22. ten Teije, A., van Harmelen, F.: Ibrow deliverable wp4.1 & 4.2: Task & method adaptation (2003)