An Immune System-Inspired Approach for Composite Web Services Reuse

Rosanna Bova [1], Salima Hassas [1], Salima Benbernou [1]

Abstract. Recently, a growing number of Web Services have emerged at a fast rate. However, there are many situations where individual web services, alone, cannot satisfy user requirements. This has raised the need for service providers and application developers to develop value added services by combining existing web services into composite web services.

Web services composition is now a very active research field, and several solutions have been proposed. However, few existing solutions address the issue of service composition reuse and specialization, i.e, how applications can be built upon existing simple or composite web services by reuse, restriction, or extension.

In this paper, we introduce the concept of abstract composite web service, that can be specialized to particular concrete compositions, and that can be reused in the construction of larger or extended compositions. We propose an approach based on immune system, in which we combine structural and usage information in order to promote and create *stable* composite web services, in an affinity maturation process.

1 INTRODUCTION

A web service is a software system designed to support interoperable machine-to-machine interaction over a network, based on standard web protocols and XML encoding.

There has been a great deal of research in the area of web services in the past few years. A large part of these studies has been dedicated to web services composition. The original idea of "service composition" is not really new in software design. It can be traced back to the simple concept of an executable library in many programming languages, the so-called "code reuse." Applying this idea to software engineering, people have developed new applications using previously written software components. This is called "software composition" or "software reuse." Now the efforts are mainly dedicated to some parts of the problem, which, generally speaking, are the discovery of useful partner web services, their composition and execution.

In this paper, we are interested in studying how web services can be composed to provide more complicated features, by reusing abstract web service composites that can be specialized to particular concrete composites, or reused in the construction of larger or extended composites. The former type of reuse can be viewed as a descendent approach, while the latter is rather an ascendant approach, where small composites are used as building blocks in possibly larger composites.

This paper aims at defining a composition framework, and a composer system, to achieve these goals. Using our framework, some tasks are to be achieved by software engineers, and some may be automated by a web service composer system. Section 2 gives our problem formulation, illustrated by an example scenario in section 3. Section 4 presents and discusses our approach and the processes involved, along with the immune system metaphor. Section 5 situates it with related works, and section 6 concludes and presents the main perspectives of this work.

2 PROBLEM FORMULATION

Our goal is to define a web service composer system and framework. This web service composer system will both be able to provide (reuse) and execute a composite web service requested by a user or another system. As such, we assume that the composer system is able to obtain success feedback about the execution of each concrete composite web service. If the composite service is to be executed by an external system, then a feedback mechanism is needed to provide this feedback loop.

We consider that a composite service is a web service involving the calling of other web services (composite or elementary) during its execution. These referenced web services are typically structured in a sort of script or program using alternatives (conditions), loops, parallel and sequence operators. For example, a BPEL script defines such a composite web service. From the outside, a composite web service is like a normal executable web service, published and accessed using the same protocols.

In the context of our approach, however, we see a composite web service as a grey box, rather than a black box, in the sense that part of its internal structure is accessible to our system.

To be able to reason about the similarity or compatibility of functionalities across several web services, we assume that there exists a higher level ontology and description language to describe these functionalities, which goes beyond the mere interface (operations, inputs, outputs) of the web service. Defining such ontology is in itself a difficult problem, but is out of scope of this paper.

We also suppose that this ontology is associated with a repository or matching engine, taking in charge the matching of a selection of web services (composite or not) compatible with a given semantic description. Thus, as a simplification, we will consider that a user request for a web service is a web service semantic description (without considering possible additional constraints or user preferences). Our work is rather focused on how composite web services are specialized or reused, than on the matching process with a given semantic description.

We will also assume that our system does not create new compositions from scratch, but that compositions already exist, created externally by software designers or architects. Our goal is then to reuse these existing composite web services, and possibly adapt them, by substituting some web services that they reference by other compatible ones.

In order to do that, we introduce the concept of *abstract composite web service*, based on Melloul and Fox's web service high level patterns, in [1]. In an abstract composite, some concrete web services are replaced by web service semantic descriptions, i.e. high level descriptions of the functionalities fulfilled by this web service, using the above mentioned service ontology. The structure of the initial composite service is conserved. Only one web service may be described by an abstract description (in other words, semantic descriptions may not span over several web services "slots" in their hosting abstract web service).

The motivations behind this concept are two fold:

- First, it allows us to reuse composites by generalization and specialization, by adapting composite services to other contexts, and relieving some context specific constraints;
- Second, abstract composites will serve us as a link between structural information and usage information.

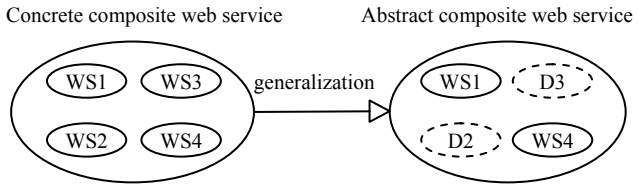

Figure 1. Abstract and concrete composite web service example

Figure 1 shows an abstract composite web service; without detailing structural information, constructed by substituting two web services in a concrete specialization, WS2 and WS3, by two abstract descriptions D2 and D3.

However, care must be taken when defining such abstract compositions, as stated by Melloul and Fow in [1]: a too abstract composite may be unusable and useless, as it would contain too little concrete information, and would be very difficult to specialize to a given context. On the contrary, a very specific abstract composite is difficult to reuse in different situations, obviously. There is a compromise between the two, allowing a sufficient but useful reuse, as in any component reusing problematic.

Another hypothesis in our work is that concrete composites contain only fixed (resolved at composition time) web service references: they may not for example include in their execution the process of searching web services, and calling them afterward.

Finally, our problem is to find the proper and most usable composite web service, that we will define as "stable" composite web services, with respect to a given user request, and evaluation criteria such as validity, pertinence, availability, and robustness, by utilizing the information about the structure and usage of composite web services and their components. The composite web service proposed by the composer system will either be existing concrete composites, or automatic specializations of abstract composites.

3 TRAVEL AGENCY EXAMPLE

We use the classical travel agency example, since it involves several kinds of web services and composite tasks using these web services, in an overall service: planning a journey. Let us define the following web services available, grouped by semantic categories:

- Travel ticket booking: *plane1* (cover flight from and to London), *plane2* (cover flights from and to Paris), plane3 (covers flight from and to Milan and Trento), and *eurostar*;
- Hotel booking: *paradise hotel* (in London and Paris), *coconut hotel* (in Milan and Trento);
- Car rental: *car1* (Milan, Trento), *car2* (all cited cities);

We assume that we already have a concrete composite web service named *TravelToLondonFromParis*, as shown in figure 2, which consists in requesting a reservation web service for the *eurostar* train from Paris to London, followed by the parallel booking of a *ParadiseHotel* chain in London, and the renting of a car with *car2*.

The main rounded box represents the composite with its global inputs (period) and outputs (train ticket, hotel booking and car rental). Now let us suppose that there exists an abstract composite web service named *TravelToLondon*, generalization of our previous concrete composite service, where the *eurostar* service is replaced by the semantic description *TravelTicket*, with an input parameter, *destination*, fixed to the value *London* (see figure 3), and the paradise hotel by the semantic description *HotelBooking*.

We introduce another abstract composite, named *DirectTravel*, which is an abstraction of *TravelToLondon* and also transitively an abstraction of *TravelToLondonFromParis*, is presented on figure 4. Here all involved web services are replaced by the abstract semantic descriptions *TravelTicket*, *HotelBooking* and *CarRental*, with the global inputs origin, destination, period, and the same outputs as the two previous composite web services.

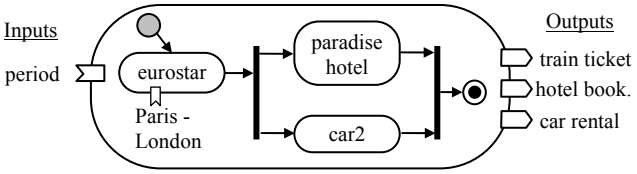

Figure 2. UML activity diagram of *TravelToLondonFromParis*

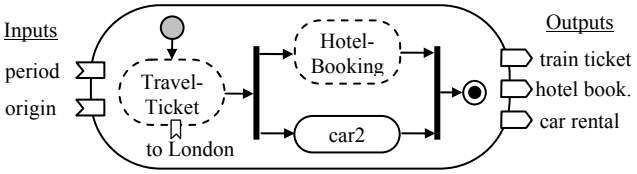

Figure 3. UML activity diagram of *TravelToLondon*

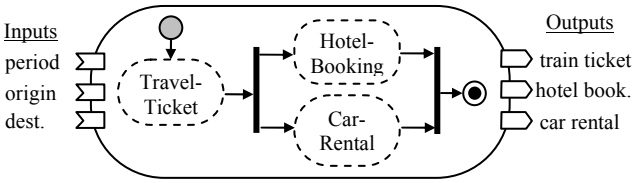

Figure 4. UML activity diagram of *DirectTravel*

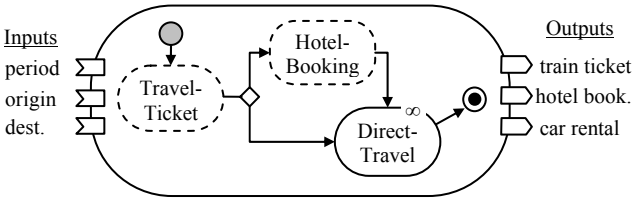

Figure 5. UML activity diagram of *IndirectTravel*

Finally, we consider that there exists a last abstract composite web service named *IndirectTravel*, which includes a reference to the former abstract composite *DirectTravel*, as shown on figure 5. A hotel booking may be necessary if the travel spans one night in the intermediate city, represented by the alternative. The ∞ sign on *DirectTravel* indicates that it refers to another abstract composite definition.

4 IMMUNE SYSTEM INSPIRED APPROACH

4.1 A simplified view of immune systems

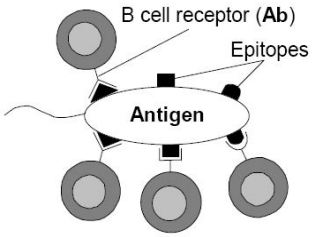

Figure 6. Pattern recognition of an antigen by B-cells

One of the roles of an immune system [2] is to protect our body from attacks of invasive foreign substances. Such a foreign substance is called a pathogen, and is recognized by the immune system as an antigen. The mechanisms used by the immune system for this purpose are:

- The Pattern Recognition of foreign antigen in the immune system, that is carried out by receptors on the surface of antibodies released from the immune cells (lymphocytes: B-cells and T-cells). The binding of an antigen to the different antibodies requires that portions of the two structures have complementary shapes that can closely approach each other. The area on an antigen where it has contact with an antibody is called an epitope. The corresponding area on an antibody is called a paratope. The strength of the binding between an antigen and the different antibodies is dependent on the affinity between them. The higher the affinity, the stronger the binding.
- The Immune Response, constituted by two kinds of response. A primary response is provoked when the immune system encounters an antigen for the first time. A number of antibodies will be produced by the immune system in response to the infection, which will help to eliminate the antigen from the body. However, after a period of days the level of antibody begins to degrade, until the time when the antigen is encountered again. The secondary immune response is said to be specific to the antigen that first initiated the immune response, and involves the process of affinity maturation (see below).
- The Clonal Selection. When antibodies of a B-cell bind with an antigen, the B-cell becomes activated and begins to proliferate. New B-cell clones are produced that are an exact copy of the parent B-cell, but then undergo somatic hypermutation and produce antibodies that are specific to the invading antigen.
- The Affinity Maturation process that guarantees that the immune system becomes increasingly better at the task of recognising patterns. After the primary immune response, when the immune system first encounters a foreign substance and the substance has been removed, a certain quantity of B-cells remains and acts as an immunological memory. This is to allow the immune system to launch a faster and stronger attack against the infecting agent, called the secondary immune response. This second, faster response is attributed to memory cells remaining in the immune system, so that when the antigen, or similar antigen, is encountered, a new immunity does not need to be built up, it is already there. This means that the body is ready to better combat any re-infection.

4.2 Immune system metaphor for our web service composition reuse problem

Although immune systems have greatly inspired our approach, our ambition is not to define an exact correspondence between the concepts, processes and mechanisms of our model, and those of the immune system. We try to use this metaphor as much as possible, especially when it helps understanding the rational of the model, however some specificities of our problem and solution still don't have a counterpart in immune systems, and vice versa.

In our model, different composite web services are proposed to the user in order to answer to a composition request, which represent the antigen aggression.

In an immune system, when neutralizing antigens, the immune cells specialize in attacks of this or similar antigens, and become *memory cells* through a process of *affinity maturation*. Thus memory cells are specialized cells for one category of antigens.

We imagine, in the same manner that an existing selection of concrete composite web services exists, specialized to answer to a category of request. However, as detailed in section 4.5.2, the sole semantic compatibility with the request, managed by the semantic matching mechanism, is not enough to ensure that a given concrete composite will be successful and stable. We claim that there need to be some sort of affinity value, derived from usage feedback information.

In the immune system, the process of *affinity maturation* [3] strongly depends on the processes of specialization and *clonal selection* of the immune cells, which are based on the affinity value between immune cells and antigens. The higher the affinity value is, the more adequate the immune cells answer is. The affinity value is based on the degree of complementarity between immune cells and antigens.

In our model, we define some usage information representing how and how many times web services or concrete composites are used with respect to a given context, represented in turn by a chain of ancestor abstract composites. Then, we define the process of maturation as a process of electing composite web services as stable composite web services.

Table 1. Correspondence with immune system

Immune System	WS Composer System
Antigen	User request
Pattern recognition	Semantic matching with existing WS + user choice
Affinity	Semantic compatibility + Relative and global affinity values
Affinity maturation	Stable composite WS election
Memory cells	Stable composite WS
B cells	Concrete or virtual composite WS

Table 1 summarizes these correspondences. Some of the terms used here are detailed in section 4.5 .

4.3 Model and motivations

For achieving our goal, our idea is that structural information should be combined with usage information extracted from the composite web services, in order to promote and possibly publish

stable and relevant composite web services. Stable composite web services will represent potential building blocks reusable in new compositions corresponding to a category of requests, in the same manner that the memory cells in an immune system react to a certain category of antigens.

In particular, the structural information comprises: the composite definition itself, i.e. the structural constructs linking the referenced internal web services together (that we will refer to as the children of this composite), organizing them in a well defined process; plus the generalization relations between composites. The exact structural information available depends on the workflow or process description language used to describe composite web services, however we can always extract the composition dependencies between a composite and its children.

The usage information is represented by a metric in our model: the *relative affinity value*, associated to a relation between each child component of a composite web service and its various abstractions. In the example of section 3, we have a relative affinity relation between: *eurostar* and *TravelToLondon*, *eurostar* and *DirectTravel*; *ParadiseHotel* and *TravelToLondon*, *ParadiseHotel* and *DirectTravel*; *car2* and *DirectTravel*.

We assume that we have a local repository that contains information for a set of existing composite or simple web services. Formally a web services, composite or simple, is defined as ws = {I, O}, where I is its set of inputs, O its set of outputs.

Along with this information stored for each composite or simple web service, we add in the repository some meta-information about the structure – the generalization relationships and the composition dependency relationships – and about the usage – the relative affinity (valued) relationships.

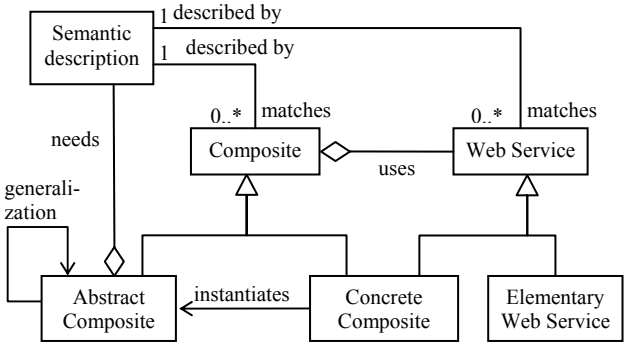

Figure 7. Composite Web Services meta-information model

Figure 7 shows a UML class diagram describing our model. For clarity reasons the relative affinity relation is not represented. Also, in theory abstract composites may also contain (use) other abstracts composites (not visible here). It includes notably:

- Concrete web service, which is both a composite and a real web service. It uses web services, its children, that may in turn be other concrete composites or elementary web services as well;
- Abstract composite, which is a composite but also contains semantic descriptions.

The important relations (UML associations) are:

- The composition dependency relation, here the *uses* association;
- The generalization relation, represented by both the *instantiates* and *generalization* directed associations.

4.4 Application to the Travel Agency example

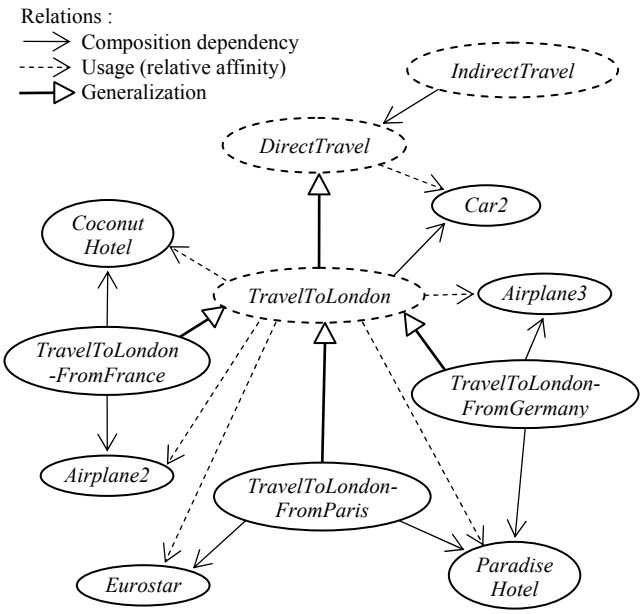

Figure 8. Travel agency example: usage and structure meta information

Figure 8 represents in a same diagram the composition dependency relations, the generalization relations, and the relative affinity valued relations. For sake of clarity, only the first level of relative affinity relations are shown on here; however these relations are by construction repeated from any abstract composite to its direct ancestor, following the generalization link, plus other ones. Thus, very abstract composites like *DirectTravel* will have a lot of potential children with which they have a relative affinity.

It is important to note on this diagram that relations of different nature are represented, which should not be confused: generalization links, which in the opposite direction represent specialization (or instantiation when leading to a concrete composite), are not of the same nature as composition dependency links. Composition dependency means that a given composite *uses* another composite or simple web service: this is neither an instantiation nor a specialization. Relative affinity links can be seen as a potential composition in a specialized form of the abstract composite, augmented with an affinity value.

With respect to section 3, we have added two new composites, which will serve us later on, *TravelToLondonFromFrance* and *TravelToLondonFromGermany*.

4.5 Process

Figure 9 represents our system as a global process. In practice, however, some parts of this process are distributed among several agents, the cells of our immune system. The distributed part includes the automatic specialization, the relative affinity update, and the affinity maturation steps.

As stated in section 2, we consider that a user request is equivalent to a semantic description, and that there exist a matching mechanism selecting compatible candidates among the composites and simple web services indexed in our repository.

After that, the composer system performs automatic specialization, which consists of specializing compatible (candidate) abstract composites, using relative affinity as guidance, into potential new concrete composites. This task does not create composites from scratch, but explores new instantiation possibilities.

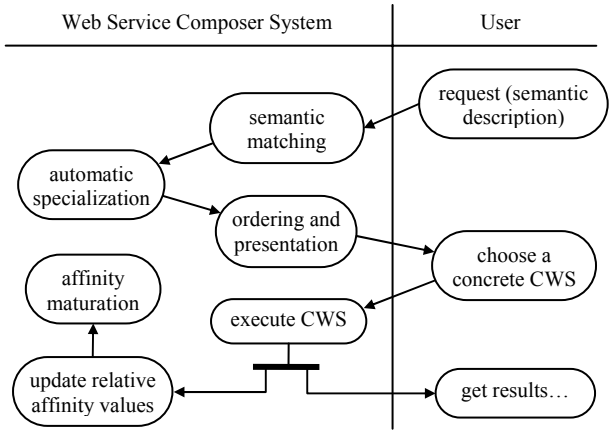

Figure 9. Global process from user request to composite execution

Then, the system reorders all composition propositions, including other candidate concrete composites obtained in the semantic matching step, using a global affinity value for each composite, and presents them to the user. The latter chooses one proposition: this choice in itself brings some exploration in the system, since a user is not forced to pick up the best composites. Then, the composite is executed by the system, and the relative affinity values are updated. The affinity maturation will possibly elect the new concrete composites, if used, as stable (i.e. memory cell), and associate it with a semantic description, so that it becomes selectable by the semantic matching engine. It is then usable in subsequent automatic specializations.

4.5.1 Relative affinity

Definition 1 Relative affinity. *A value of relative affinity is always associated to a relation between a concrete web service (composite or not), in execution, and the abstract composite of the composite that calls it. This value is updated every time that this concrete web service is executed as a child of one of this abstract's specialization. The relative affinity function is equal to:*

$$aff_r(c,a) = \frac{freq_{succ}}{freq_{usage}} \quad (1)$$

where $freq_{succ}$ is the function measuring the number of times that this concrete web service (c) has been used with success and $freq_{usage}$ is the total number of time that this concrete web service has been used.

Figure 10 illustrates the updating mechanism of the relative affinity values between concrete web services (*WS1*, *WS2*, *WS3*, *WS4* and *Y*) and the generalization of the parent composite that calls them. The execution of composite *X* includes the ordered execution of *WS1*, *WS2* and the composite *Y*. Once the concrete *WS1* has been executed, the relative affinity relation between A_X (the abstract composite of *X*) and *WS1* is updated, or generated if it did not exist, with an associated relative affinity value equal to the fraction between the number of successes and the total number of utilizations of *WS1* as a child of any specialization of A_X. The same happens between the abstract composite A_X and the concrete *WS2*.

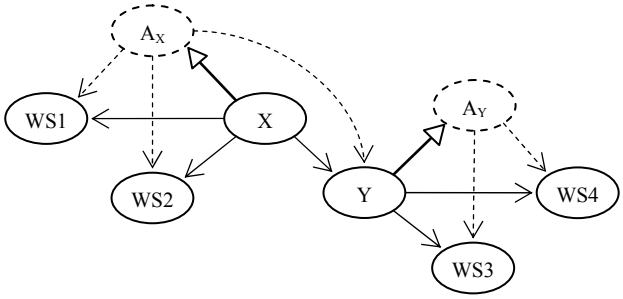

Figure 10. Relative affinity update example

Since execution of the composite *Y* includes the execution of *WS3* and *WS4*, a relation between A_Y and the two concrete web services *WS3* and *WS4* are generated or their relative affinity value updated. Then, when the execution of the composite *Y* terminates, a relation between the abstract composite A_X and the composite *Y* is generated with a numeric value of relative affinity.

4.5.2 Affinity maturation

During the automatic specialization step, new virtual concrete composite may be created by specializing abstract composites compatible with a user request, with different children than those present in existing concrete composites. As long as such a virtual composite is not considered stable by the system, it does not has a proper existence outside the current user session, and is not identified nor associated to a semantic description in the web service repository of the composer system.

The only trace of its existence is represented by the global affinity value calculated by the system, also used to order the propositions before presenting them to the user. This global affinity value only depends on the relative affinity values of the various children, and possibly grandchildren, etc., involved in the virtual composite.

Definition 2 Global Affinity. *The global affinity value represents a weighted average of the various relative affinity values, with respect to the whole ancestor chain of the concrete composite. Its value is given by the following function:*

$$aff_g(C) = \frac{\sum_{i=1}^{n}\sum_{j=1}^{m} \alpha_{ij}\, aff_r(c_i, a_i^j)}{n} \quad (2)$$

where C is the considered concrete composite, viewed here as a set of n children concrete web services ($c_1 \ldots c_i \ldots c_n$); m represents the number of ancestor abstract composites of C; a_i^j are the semantic descriptions of the children concrete web services to instantiate on the j^{st} ancestor, and corresponding to the concrete child c_i; α_{ij} are weight values, so that the sum of these weight values referring to the same concrete child web service is equal to one; and $aff_r(c_i, a_{ij})$ is the relative affinity of c_i with respect to a_i^j.

Definition 3 Virtual composite. *A virtual composite is a composite created by the composer system as a result of the automatic specialization step, in response to a user request. This composite is temporary to the session, and does not have an actual existence in the composer system web service repository.*

Definition 4 Stable composite. *A stable composite is a former virtual composite which has been used successfully at least one time with a global affinity value exceeding a predefined threshold (parameter of the composer system), that we refer to as the affinity maturation threshold.*

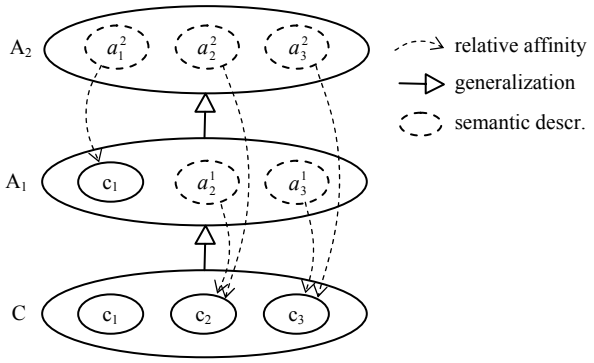

Figure 11. Example of global affinity value calculation

In the example of figure 11, the global affinity value is:

$$aff_g(C) = (\alpha_{11} aff_r(c_1, a_1^1) + \alpha_{12} aff_r(c_1, a_1^2) + \alpha_{21} aff_r(c_2, a_2^1) + \alpha_{22} aff_r(c_2, a_2^2) + \alpha_{31} aff_r(c_3, a_3^1) + \alpha_{32} aff_r(c_3, a_3^2)) / n$$

where $\alpha_{12} = \alpha_{21}+\alpha_{22}= \alpha_{31}+\alpha_{32} = 1$.

Once a concrete composite is considered stable, it is stored and indexed in the web service repository, and associated to a semantic description. As a real concrete composite, it can be matched directly to user requests without needing the specialization step, and also becomes eligible as a child concrete web service, inside other composite web services.

This stable composite election process is inspired by the immune system's affinity maturation process. Affinity is stimulated by cross-usage, and different existing composites may contribute to the maturation of the same new stable concrete composite.

One of its goals is to ensure some sort of long term memory to the system, since stable composites may be kept for an arbitrary long time in the system, even if their global affinity value falls below the affinity maturation threshold for some period.

4.5.3 Example

To illustrate specialization and affinity maturation, let us suppose that a user wishes to travel in London from France, in the context of the scenario presented in section 3, and augmented in section 4.4, figure 8.

We assume that the user request is compatible with *TravelToLondonFromFrance*. However, we also assume that *TravelToLondonFromParis* and *TravelToLondonFromGermany* have been used largely with success, so that the relative affinity of *ParadiseHotel* with respect to the abstract composite *TravelToLondon* is higher than that of *CoconutHotel*. Although the context is slightly different, *ParadiseHotel* is still applicable to the semantic description *HotelBooking* included in *TravelToLondon.* Thus, the composer system will propose a virtual composite with *ParadiseHotel* instead of *CoconutHotel* to the user, with a higher rank.

Now, let us suppose that the user chooses this virtual composite web service, and that the execution is successful. If the global affinity is greater than the affinity maturation threshold, a new stable concrete composite is created and referenced in the repository, and can now be used directly.

4.6 Exploration / exploitation ratio

Apart from the obvious exploration due to the fact that the user chooses among different composite propositions from the system, there is another interesting form of exploration in this process, related to the automatic specialization step.

During this phase, the composer system tries to instantiate new concrete composite from existing abstract ones, by substituting some children web services. This exploration is guided by two factors: (i) the existence of abstract composites and (ii) the relative affinity values of potential children web services, with respect to these abstract composites. Abstract composites are supposed to be designed by system administrators or programmers, and influences the way the system will reacts.

However the creation and enforcement of new composites, for a given user request, is also highly influenced by the cross-usage of various composites related to these abstract composites, and involving other potential children web services in different contexts. This influence is not limited to the specific usage of the concrete candidate composites directly compatible with the user request.

As a consequence, the ratio between exploration and (usage) exploitation is mainly determined by the density and structure of the generalization and composition graphs formed by abstract composites, by the initial distribution of the concrete composites and the elementary web services in the system, and above all by the variability of the requests and choices of users.

4.7 Discussion

The semantic level and matching mechanism are not covered in this paper. However the language and ontology used for the semantic descriptions is indeed very important for our approach to work properly: the language used should allow designers to define abstract descriptions, for abstract composites, that remain compatible with more specific descriptions. Additionally, it should allow our system to specialize an abstract description, when an automatic specialization occurs. This specialized description will probably add some constraints related to the parameters fixed in the concrete composite, and also related to the newly associated children web services.

Another issue is the relative importance of the children of a given composite. We currently consider them evenly, with the same weight, in the calculation of the global affinity value. Assuming that if any of them fails, the whole composite fails (note that it is not always the case, especially if the composite includes some form of redundancy to increase its robustness), one might consider that the weakest affinity score should be considered. Alternatively, a more complex affinity value calculation, partly based on the workflow structure of the composite, could be investigated.

The affinity functions used in our approach does not define any absolute confidence value associated to each web service or composite web service, independently of any context. Even the global affinity function is always relative to a context, represented by the chain of ancestor abstract composites and the user request. This design choice might be considered somehow restrictive, and one may consider that the relative affinity values should be combined with an absolute confidence value for each web service, so that usage feedback may be shared more widely across usage contexts. Our motivation is that the success of a web service is often very context sensitive: a single parameter change can condition the success or failure of a request.

5 RELATED WORKS

This approach is inspired by the work by Melloul and Fox in [1]. In particular, the abstract composite concept in our model is close to their high level composition patterns. Our contribution mainly adds the automatic specialization process, the affinity relations, and the affinity maturation process.

Although we do not use class inheritance explicitly, our generalization/specialization relations suggest an object-oriented inheritance model. In this direction different works can be found in the area of workflow class inheritance. For example, in [4] Bussler presents a framework to analyze the requirements for supporting such workflow class inheritance. Different perspectives of inheritance are discussed and a workflow class definition language is proposed. In [5], Kappel and Lang present a workflow class specification, consisting in a set of object classes and rules. Subclasses and inheritance are supported, at least partially.

In [6], Papazoglou and Yang describe a system called TOWE that implements a set of classes providing the basic mechanisms for workflow execution. Other workflow classes can then be developed by inheriting the functionality of basic workflow classes.

Our generalization relation, however, is defined by the substitution, in the composite definition, of one or more children concrete web services by semantic descriptions. This relation is derived from the composites structure: it is not a purely additional, higher level classification of existing web service, which would rather correspond to the semantic description level in our case.

Our approach differs from other related works about web services composition in that it focuses on reusing existing abstract composites that can be specialized into particular concrete composites, or reused in the construction of larger or extended composites.

Finally, we distinguish our work from automatic Web service composition such as in the work by McIlraith et al. [7] on semantic web service composition, and the work by Petrie et al. [8] on web service planning, where the goal is to produce a composition plan. Rather, we start with existing (high-level) plans, and focus on their different possible reuse, by exploiting the combination of cross-usage, the affinity relations, and structural meta-information, the composition and generalization relations.

6 CONCLUSION AND FUTURE WORKS

In this paper we have proposed an approach to deal with composite web service reuse and automatic specialization by children component substitution, inspired by the human body immune system. The shape correspondence between the antigen epitopes and the antibody paratopes is represented in our system by a relative affinity function, measuring the degree of success of the use of a concrete child web service, within the context of a concrete composite, with respect to a more general abstract composite web service.

The associated affinity maturation process allows the emergence of new stable concrete composites, resulting from automatic specializations and the accumulated cross-usage information. These stable concrete composites are then identified and semantically described, as any existing web service in our system.

This process, as well as the relative affinity calculation, is of course guided by the definition of meaningful abstract composites, which gives to the composer system administrators a degree of control on the affinity propagation and on the potential automatic specializations proposed by the system.

We are currently defining a prototype in order to validate the feasibility of this approach on simple scenarios (work in progress). A perspective is to extend the model to better account for the internal structure of web service composite in the relative affinity function (for example, differentiate redundant and mandatory children).

A second perspective is to consider the specificity of the request in our global affinity evaluation: if a request is very specific, it is reasonable to think that the relative affinity values related to the most specific abstract composites are more important than the relative affinity values related to the most abstract ones. On the contrary, a very vague request will not care too much about the former ones, but more about the latter.

A long term perspective is to leverage the distributed nature of the immune system model, and its natural tolerance to heterogeneity. Instead of having one global composer system, this approach can scale to a network of interconnected web service composition domains managed by local composer systems. Such composer systems may publish stable or abstract composites to each other, with respect to some diffusion policy. This diffusion would correspond to the spreading and cloning of memory cells into our blood.

REFERENCES

[1] L. Melloul and A. Fox 'Reusable Functional Composition Patterns for Web Services', in proceedings of the *IEEE International Conference on Web Services (ICWS)*, San Diego, CA, USA, 498-506 (2004).

[2] L. N. de Castro and F. J. Von Zuben, 'Artificial Immune Systems: Part I, Basic Theory and Applications', RT DCA Technical Report 1-98. (1999).

[3] Berek, C. and M. Ziegner, 'The Maturation of the Immune Response', *Immunology Today*, **14** (8), 400-402, (1993).

[4] C. Bussler, Workflow class inheritance and dynamic workflow class binding, In Proceedings of the *Workshop of Software Architectures for Business Process Management at the 11th Conference on Advanced Information System engineering,* Heidelberg, Germany, 1999.

[5] G. Kappel, P. Lang, S. Rausch-Schott, and W. Retschitzegger, 'Workflow Management Based on Objects, Rules and Roles', *IEEE Data Engineering Bulletin*, **18**(1), 11 – 18, (1995).

[6] M. P. Papazoglou and J.Yang, 'Design Methodology for Web Services and Business Processes', in Proceedings of the 3rd *VLDB-TES Workshop*, Hong Kong, 2002. Also in LNCS, **2444**, Springer, (2002).

[7] S. A. McIlraith, T. C. Son, and H. Zeng, 'Semantic Web Services', *IEEE Intelligent Systems, Special Issue on the Semantic Web*, **16** (2), 46-53, (2001).

[8] C. Petrie, M. Genesereth, H. Bjornsson, R. Chirkova, M. Ekstrom, H. Gomi, T. Hinrichs, R. Hoskins, M. Kassoff, D. Kato, K. Kawazoe, J. U. Min, and W. Mohsin, 'Adding AI to Web Services', *Agent Mediated Knowledge Management*, LNAI, **2926**, Springer, 322- 338, (2004).