Ontology based Algorithm Modeling: obtaining adaptation for SOA environment

Simone Grassi
Trinity College Dublin
CS Department
DSG Group, D2, Dublin
+353-85-1571903
grassis@cs.tcd.ie

Stephen Barrett
Trinity College Dublin
CS Department
DSG Group, D2, Dublin
+353-1-896-2730
ebarrett@cs.tcd.ie

Francesco Sordillo
Università di Bologna
via Sacchi n.2
47023, Cesena (FC)
sordillo@csr.unito.it

ABSTRACT
Our work addresses the issue of software adaptation in Service Oriented Architecture (SOA) environments. We aim to support a wide range of adaptations using a new formulation of Web Service (WS) model based on client driven service adaptation via ontological description of algorithms. We describe how this approach can be applied to a SOA scenario involving heterogeneous systems, and report on experimental work that demonstrates how services can be transformed in practice, using a framework approach.

Categories and Subject Descriptors
D.2.10 [**Software Engineering**]: Design Tools and Techniques - *Software libraries, Computer-aided software engineering.*

General Terms
Algorithms, Design, Languages, Theory.

Keywords
Adaptable Software, Ontology, Software Synthesis, Semantic Web Services, Service Oriented Architectures.

1.INTRODUCTION
Software maintenance and adaptation is a challenging field. The presence of many interconnected interactions make modern systems difficult to maintain, despite continuous pressure to adapt [8][9]. Web solutions are now integrated and interacting and the resulting computations cross organization boundaries. Coordination technologies [3], the Semantic Web and WS composition techniques [4][5][6][7] allows service providers to interact in massive distributed systems. Those systems provide orchestrated Business Processes (BPs) for a wide range of services, from travel information, to e-government, document management and so forth. In this context of e-business the traditional development life cycle, involving redesign and redevelopment by software engineers, is too expensive and too slow, a more agile process is needed. The problems are amplified in the heterogeneous context of Internet based systems like SOA.

In this paper we argue for an alternative approach to WS construction that supports the dynamic adaptation of services on a specific client request basis. We propose an approach focused on the ontological expression of a services algorithms in a form open to client based modification.

We use an ontological approach to enable Domain Specific Model of WS algorithms, that enable the use of an open-adaptive approach [10], in contrast to close-adaptive approaches [10] which include a limited and predetermined range of possible adaptations. The use of ontologies enable the modeling of algorithms using a standard like OWL, enabling also the use of common tools for creating, managing and reasoning on an ontology.

In section 2 we introduce the structure of our solutions. In section 3 a case study is presented, which demonstrates an adaptable Web Service [1]. In section 4 we discuss related work both of the case study and the approach in general. Section 5 contains a brief about future works and section 6 contains the conclusions.

2.AN ONTOLOGICAL APPROACH TO WS CONSTRUCTION
The fundamental feature of our modeling approach is to provide a mechanism for client and servers to reason over algorithms so that change may be agreed. The Ontology Algorithm (OA) is structured as set of OWL ontological individuals, on top of an ontology made by two parts. The Logic Ontology (LO), describes the composing elements needed to construct the abstract algorithm, and the data Domain Ontology (DO), that describes the domain of interest. Together the LO and DO provide the building blocks of the described algorithms. We visually describe the OA like a tree, that we call algorithm tree.

This structure allows to model an algorithm as a set of individuals built on top of a common ontology, that describe a system independent specification. Then, extending the platform specific ontology, we move the specification from platform independent to platform specific. The general architecture of the approach is described in Figure 1, that illustrates the delivery of a request for adaptation, originated by a government, and received by an administrative region. Both the systems are assumed to host a BPs and WSs. The OA created by the government is based on a common shared ontology. The region use this as the basis of a concretization with system dependent extension. This extension allows the Local Adaptation Engine (LAE) to generate system specific code, that will constitute the new version of a WS, as

MW4SOC'07, November 26, 2007, Newport Beach, CA, USA

requested by the government. In section 3, we cover the case study, and we will illustrate how to use an aspect oriented style adaptation request and how to merge different versions of an algorithm reasoning on the OAs.

3.CASE STUDY

We consider a tax environment where unpredictable changes are needed, due to changes in the legislation of the central government and in the specific decision made autonomously, region by region. In such a context, a capability to migrate service or behavior in this scenario could provide a major advantage.

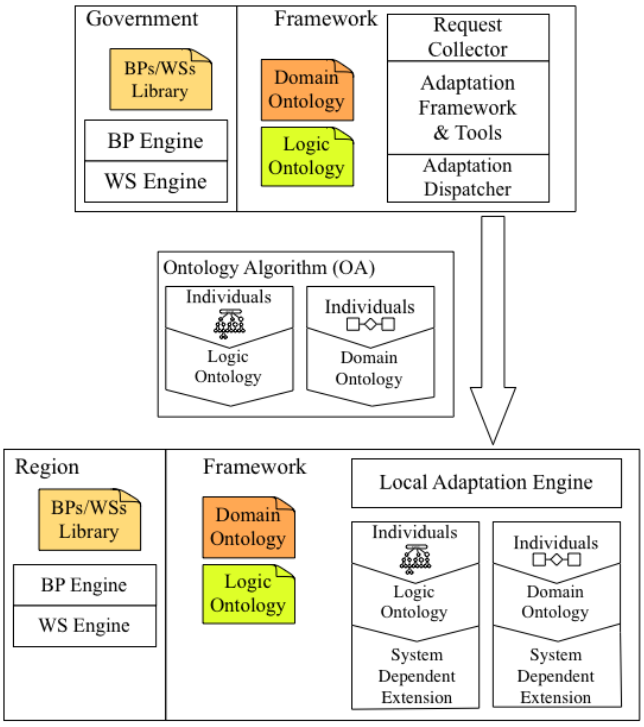

Figure 1: Request of adaptation, sending an OA

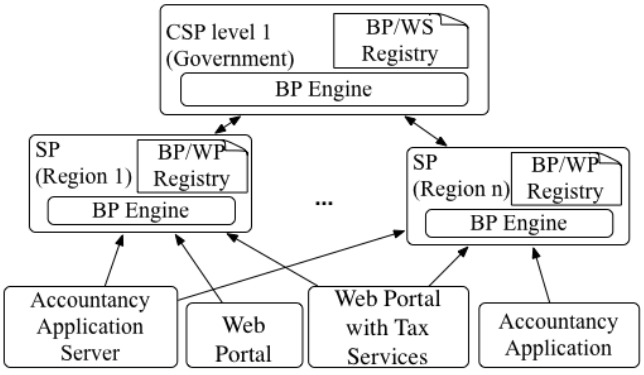

Figure 2. Case Study Scenario

In Figure 2 we consider a service providing tax calculations for different jurisdictions, that is required to adapt its core behavior not only to the context of a particular client, but also to longer period changes in government legislation in any jurisdiction. Regions, in turn, provide a set of services for public access. We separate the government level from a region level to let regions introduce adjustments to the central government rules. Accountancy software solutions and public web portal, can use services published by regions to require tax calculations. We now consider the dynamic adaptation of some provided services, and we propose a way to solve the issue using adaptable WS [1].

Maintaining a system that is capable of accurately functioning in a large set of taxation contexts is a challenging task, even though we might reasonably assume that most of the core calculations and workflow will be similar in each jurisdiction. The contemporary approach to this problem is to develop a framework or component oriented solution which can be specialized to a particular jurisdiction. Thus, by specializing or otherwise injecting bespoke behavior, a generic solution is tuned to the problem at hand. However, in the context of SOA, where pre-configured instantiations deployed as WSs are the fundamental unit of composition, it becomes difficult to tune such components to the needs of clients.

Our approach involves the explicit model of WS algorithm in a form open to adjustment post deployment. We consider how the algorithm for the calculation of yearly tax liability of an individual can be described and adjusted so that its WS implementation can be changed in response to client need. A possible basic calculation scheme is one with layers. A no-tax zone up to 12k, 20% up to 24k, 30% up to 40k and 40% over 40k. Figure 2, describes a Central Service Provider (CSP), the government, who set general tax policy. A second level of Service Providers (SP), the regions, who can amend tax policy and a set of applications (ex: accountancy solutions and web portals for tax calculation services) who provide user level services that combine features provided by government and regions. Algorithm 1 (Figures 3 and 4) is used to calculate taxes to pay for a specific person. We use tree structure in the reminder of this paper to emphasize change as structural adjustment.

We consider an adaptation originated by a single region. In Algorithm 2, Figure 5, taxes are reduced by ***py*** for young people, aged under ***y***, for the first ***nla*** layers of taxation. This originates at a region so that is has some freedom in autonomously adapting taxation schemes. The peer differing elements in Algorithm 2 have gray background color, and more parameters are present (***y*** is the maximum age to obtain the reduction; ***yp*** is the age of the person; ***py*** is the percentage of reduction obtained, 0.9 stand for 10%; ***nla*** are the number of taxation layers affected by the reduction, starting from the lower one and up). The subtree in gray has been added to obtain a multiplication by a factor that introduce the discounted rate for young people.

The new Algorithm 2 is an update of Algorithm 1 and the LAE of the region that originated the modification can generate an implementation as before. It is a local adaptation specific to one region, and other regions and the government are aware of this change. At any stage the government itself may ask for another adaptation due to a new legislation modification.

We consider now an extension of this scenario where the government sends a further change regarding the taxation calculation. Further reduction is applied by the government, to all

people with at least one child. In Figure 6, Algorithm 3 is represented. The changes relative to Algorithm 1 are in gray background. There are also new parameters (***ch*** the number of child in charge; ***pc*** the percentage of tax reduction, 0.9 is 10%; ***nlc*** are the number of taxation layers affected by the reduction). The region that performed its own modification, now has the problem of merging its own changes with this new version of the original algorithm.

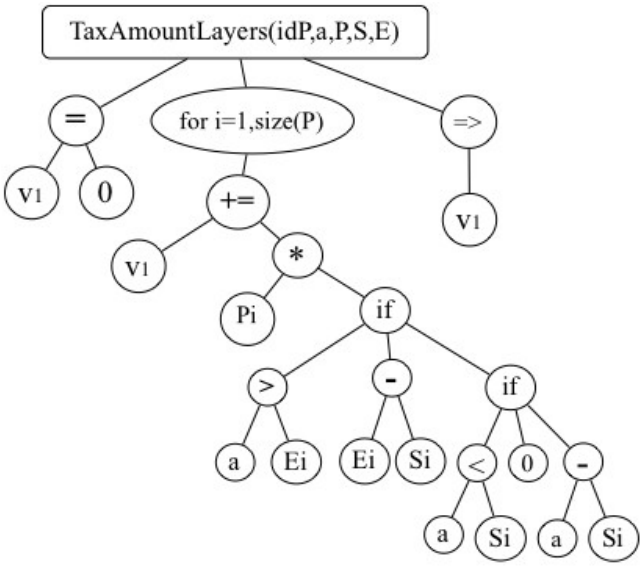

Figure 3: Algorithm 1

Legenda:

idP: personID **a**: total income
S: layers tarting values (array) **E**: layers ending values (array)
P: percentage to pay for each layer (array)

```
function TaxAmountLayers(idP,a,P,S,E) {
 v1 = 0;
 for (i=1 to sizeOf(P)) {
  v1 = v1 + P[i]*(  if (a > E[i])  then (E[i]-S[i])
                    elseif (a < S[i]) then 0
                        else (a - S[i]));  }}
```

Figure 4: Algorithm 1, pseudo-code

Having the original algorithm (Algorithm 1) and the two modified versions (Algorithm 2 and 3) we propose a solution based on merging. We automatically generate a new algorithm including both the modifications. To achieve this, we use rules, based on the semantic knowledge of the elements that compose the algorithm itself. We limit this to an easy example, to show the potential of the approach, to aid the human interaction with automatic or semi-automatic reasoning on adapting algorithms. We utilize a few definitions, described below, that can be applied to single elements of the algorithm, subtree or entire trees. Those rules are applied to one of those elements in relation to another element, subtree or full tree.

Definition 1: An element, or a set of elements forming a subtree or a full algorithm is **strongly independent** from another element, set of elements or full algorithm if,

rule 1: It does not use values if not previously initialized

rule 2: access data from local data layer only if not modified by other elements positioned before

rule 3: access data only using an element that is a parameter and it has not been modified from other elements positioned before

Definition 2: An element, or a set of elements forming a subtree or a full algorithm is **weakly independent** from another element, set of elements or full algorithm if rule 1 applies, but rule 2 or 3 or both does not apply.

Definition 3: An element, or a set of elements forming a subtree or a full algorithm is **dependent** from another element, set of elements or full algorithm if rule 1 does not apply.

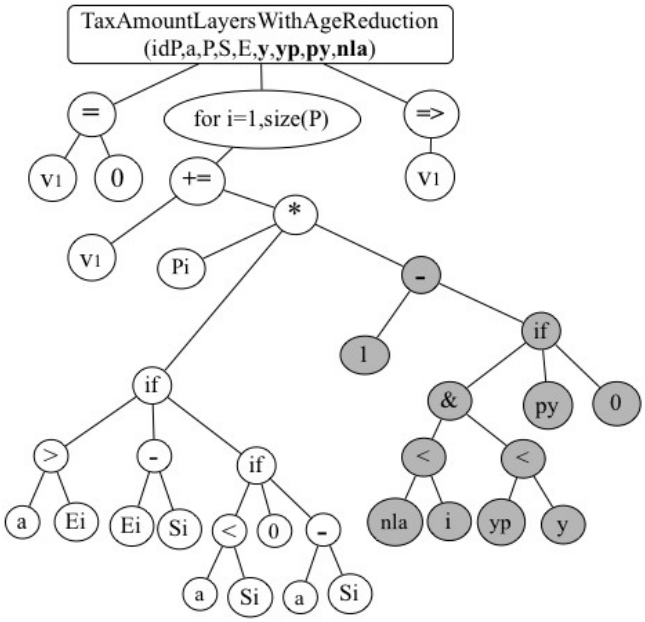

Figure 5: Algorithm 2, adaptation by a Region

Having a set of differences between Algorithm 1, 2 and 3, we can determinate what modified subtrees are strongly independent, weakly independent, or dependent. This information allows us to decide if it is possible to merge. In our case we propose the merge as in Figure 7, this is possible because the two subtree in gray, from Algorithm 2 and 3, are strongly independent between each other. We don't need a specific order because the '*' element, is commutative. The information about being commutative is included in the LO OWL. The decision about the merge can be done automatically, a visit of the algorithms find out the differences, and if one by one they can be merged we have one or more possible solutions. Semi-automatic solutions can be used adding a human interaction, if needed, at any stage of the adaptation. That can be used to select between different proposed adaptation or to adjust one of the proposed adaptations.

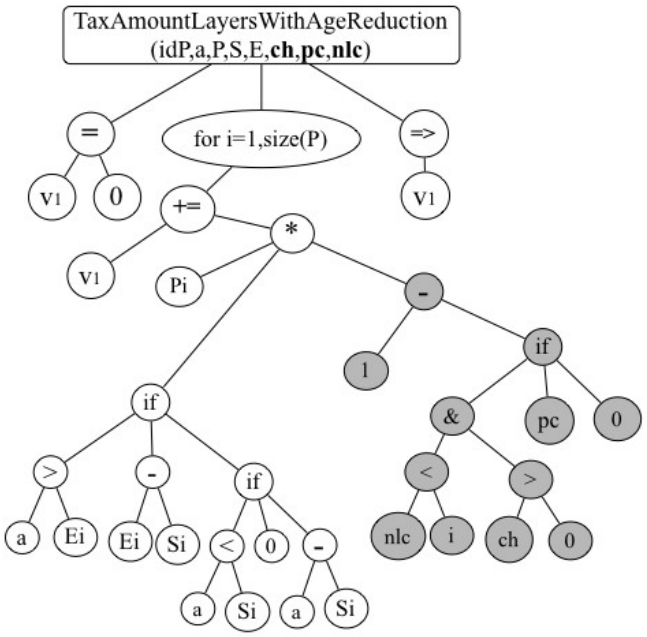

Figure 6: Algorithm 3, adaptation by the government

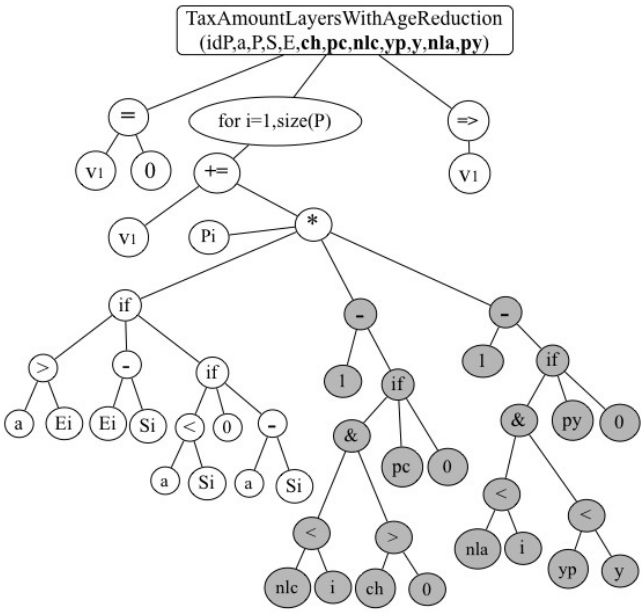

Figure 7: Merging algorithm 2 and 3

We end up with the description of an adaptation requests and how it can be specified using an aspect oriented style approach, and can be then accepted and deployed by a client using an additional OA specification on the original OA. The scenario is again about a government, sending an adaptation, to the regions. The new adaptation is specified as a description of an aspect to be weaved into the original algorithm. In Figure 8 the usual tree representation of the ontology is presented, the request specifies a modification for all the read accesses for the element **a** to become accesses to the element **a*pc** if the condition **a<S[nlc]** is true.

In Figure 8 the **foreach** element specifies that its first element a, should be modified when the conditions specified in the second element subtree are true. In this case when the elements < and **read-access** are both true. We say both, because of the father node **&**, that join them in logic and. The third element specify the action to adopt, in this case we have to **substitute** element **a** with the subtree **a*pc**. For brevity we don't show reasons of all the ontology elements, but the semantic informations are described with OWL.

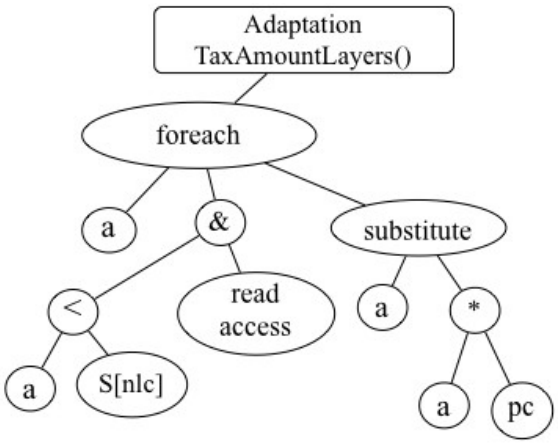

Figure 8: Algorithm adaptation by the Government

The benefit we aim to obtain is the decoupling of algorithms from the implementation and deployment details. Moreover to change the level of abstraction of an algorithm specification importing and extending ontologies instead of a more traditional rewriting of algorithm specification. That enables the use of abstract oriented style specification but in the same way the use of a domain specific language approach. Finally we enable the use of standard technologies as ontology to reason on algorithms. That can be used to evaluate the adoption of an algorithm, to merge different versions of algorithm. Finally we consider an interesting path for further changes to weave requests of adaptation in an existing algorithm at an abstract level.

4.IMPLEMENTATION

In our tests we implemented a LAE written in Java, that accept OWL and generates php5 code for the Symfony MVC [22] framework, see Figure 9. To move an OA to working code, i) the LO is translated into valid php5 code, ii) the DO uses the API of the creole library to generate the proper code to access the data layer, iii) the created algorithm is encapsulated as a method in a class and properly positioned as a new file in the framework tree. Having the publication of a WS as an objective, we then provided the creation of the proper configuration for the Apache server, in order to publish the WS, using the Pear SOAP libraries. In such a way we generate automatically a WS to be published using php SOAP libraries and the Apache web server.

The code of the LAE has two main hierarchy of classes. The first does the high level creation of the generated code. It cares about collecting the code of the basic elements (generated by the second

hierarchy of classes), then about positioning the code of the algorithm in the proper method and class, and finally to position the file into the proper path. The second hierarchy is a mapping of the building blocks described by the ontology, and it store the capability to generate the proper code of the covered element. Some of the elements may have a complex behavior, like the elements used to describe an adaptation in an aspect oriented style. In that case, the corresponding class or structure of classes, describes the code that is able to weave the adaptation to the original algorithm. To start the generation of an algorithm or an adaptation, the LAE is able to browse the OA and find the root of the algorithm tree, following then the structure of the OA, visiting the algorithm tree, thanks to the semantic information associated with the individuals that constitute the algorithm tree.

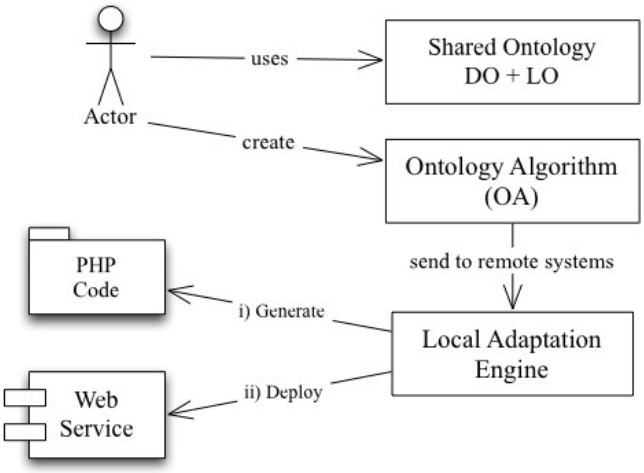

Figure 9: Implementation workflow

The powerful aspect of this approach is the capability to have a very flexible algorithm specification, because the ontology can be extended in order to store more semantic information, more abstract operations, or to specify an adaptation as a piece of an algorithmic tree instead of going to modify an existing algorithm. This flexibility, that can also be used to create an ad-hoc domain specific language, or a subset of it, has a complexity involved, and it is managed by the LAE. The core architecture of the LAE is generic and can be completely recycled to build the LAE for another system (like Ruby on Rails or the Java Framework, instead of php5-Symfony).

5.RELATED WORKS

In the field of adaptive software, there are two main approaches to obtain adaptations. Closed-adaptive systems, are self-contained and not able to support the addition of new behaviors, on the other side there are open-adaptive systems, where new application behaviors and adaptation plans can be introduced [10]. What is still under study is the development of a comprehensive adaptation methodology suitable for a wide range of adaptation scenario. Adaptations can include the replacement, cancellation or addition of components and connections, the change of configuration parameters, and all changes must include a test and validation mechanism, avoiding inconsistent and unsafe adaptations. In the field of Semantic Web, some promising work, approaches the problem of dynamic composition using a detailed semantic representation and matchmaking algorithms to generate a BP specification from advertised web services. Adaptations are left to a matchmaking algorithm that relies on the existence of enough basic components to compose a BP [4][5][6]. In [7][6], compositionality is decided automatically, identifying attributes and separating them into syntax attributes, such as input and output parameters, static semantic and dynamic semantic attributes, covering respectively the domain of interest and the business logic. Some parameters are then used to assign a compositionality level, useful in deciding what is composable and what is not. In [11] a base theory is provided to support compositionality for Semantic WS, adding temporal properties like assumption and commitment that are used to validate not only initial and final state but also intermediate states. In [12] it is argued that more dynamic and behavioral aspect needs to be included in the exported semantic of the WS. An outstanding issue is the hosting of glue code, specially when it cannot be hosted by who is doing the composition (mainly for privacy or performance reasons). A modeling and transformation approach to express the distribution pattern of WS composition is presented by [13]. We see it as a well infrastructure to be used for the dynamic deployment of adaptable WS. An Aspect Oriented approach is used to align internal service specification to a standardized external specification in [2], that is a valuable approach to cover the issue of a service adaptation in some circumstances. A range of adaptations are obtained via protocol composition [8], treating protocols as building blocks that can be composed and adapted. In [14] and [15] the template mechanism of the CARE toolset is enhanced providing adaptation tool at different levels.

A number of component composition techniques uses different parameters to define a component and them properties. The same definition of composition can be discussed and a set of research is reported in [16], that are mainly done for a specific purpose. One of the definition is “the process of constructing applications by interconnecting software components through their plugs” [17]. But in any of those case is very important the role of the glue code, needed to adapt the interfaces and the behavior of components, and to compose or connect components to obtain a combined behavior [19]. In [18] a parameter adaptation method is used, to map predefined configuration of subcomponents. In such a way some adaptations are obtained, but the approach looks not adequate for a wide and specially unpredictable set of adaptation, specially involving unpredictable behaviors.

Formal methods needs to be mentioned as well, a lot of effort was done by different researches, starting from many years ago. The basic idea, that we subscribe, is to obtain a formal specification of components but also objects, parameters and theories, to be able to formally decide if a composition is allowed or not [18][19][20][21]. We conclude mentioning Evolutionary Programming, where a separation from architectural part and algorithmic part is the base to avoid the system to become unstable due to an attempt of adaptation [10]. This issue is very important in our work as well, where the adaptation must be executed automatically, deciding the proper policy both in case of successful or unsuccessful adaptation.system (like Ruby on Rails or the Java Framework, instead of php5-Symfony).

6.CONCLUSIONS

Our motivation is to isolate that which is most volatile (in terms of requirements for different clients) in WS behavior in a form capable of client adjustment. We illustrated a novel approach aiming to work as a modeling technique for algorithms, with the use of ontology as abstraction tool to separate the different aspects involved

in making changes in heterogeneous systems, thus providing a base to obtain automatic or semi-automatic adaptation of software algorithms. In this way we can model pieces of software, containing specific volatile algorithms, without trying to model the whole architectures of systems, but obtaining a wide range of adaptability in the modeled sections. The base is the use of ontological description in formal model style. We contend that the SOA paradigm can benefit from the WS formulation obtained with the described specification, in order to specify and distribute dynamic adaptation, in an heterogeneous environment.

7.REFERENCES

[1] Grassi, S.; Barrett, S., "Dynamic Architecture Adaptation in WS Environment," Autonomic and Autonomous Systems, 2006. ICAS '06. 2006 International Conference on, vol., no.pp. 26- 26, 19-21 July 2006.

[2] w. Kongdenfha, R. Saint-Paul, B. Benatallah, F. Casati, “An aspect-oriented framework for service adaptation”, Service oriented computing, ICSOC 2006, Eds. A. Dan, W. Lamersdorf Springer, Germany, 2006 pp 15-26.

[3] Malone, T. W. and Crowston, K. 1994. The interdisciplinary study of coordination. ACM Comput. Surv. 26, 1 (Mar. 1994), 87-119.

[4] B. Medjahed, A. Bouguettaya, A.K. Elmagarmid, “Composing Web services on the Semantic Web”, The VLDB Journal, 2003.

[5] L. Li, I. Horrocks, “A software framework for match-making based on semantic web technology”, WWW ‘03: Proceedings of the 12th international conference on World Wide Web, 2003.

[6] B. Medjahed, “Semantic Web Enabled Composition of Web Services”, Doctor of Philosophy in Computer Science and Applications, Falls Church, Virginia, USA, 2004.

[7] Medjahed, B.; Bouguettaya, A. “A multilevel composability model for semantic Web services”; Knowledge and Data Engineering, IEEE Transactions on Volume 17, Issue 7, July 2005 Page(s):954 - 968.

[8] Nirmit Desai, Amit K. Chopra, Munindar P. Singh, “Business Process Adaptations via Protocols,” scc, pp. 103-110, IEEE International Conference on Services Computing (SCC’06), 2006.

[9] Jae-yoon Jung, Wonchang Hur, Suk-Ho Kang, Hoontae Kim, "Business Process Choreography for B2B Collaboration," IEEE Internet Computing, vol. 08, no. 1, pp. 37-45, Jan/Feb, 2004.

[10] P. Oreizy, M. Gorlick, R. N. Taylor, D. Heimbigner, G. Johnson, N. Medvidovic, A. Quilici, D. Rosenblum, A. Wolf. “An Architecture-Based Approach to Self-Adaptive Software”, IEEE Intelligent Systems, vol. 14, no. 3, pages 54-62. May/June 1999.

[11] Solanki, M., Cau, A., and Zedan, H. 2004. Augmenting semantic web service descriptions with compositional specification. In Proceedings of the 13th international Conference on World Wide Web (New York, NY, USA, May 17 - 20, 2004). WWW '04. ACM Press, New York, NY, 544-552.

[12] D. Berardi, G. De Giacomo, D. Calvanese, "Automatic Composition of Process-based Web Services: a Challenge", Web Service Semantics: Towards Dynamic Business Integration, workshop at 14th International World Wide Web Conference (WWW 2005).

[13] Barrett, R., Patcas, L. M., Pahl, C., and Murphy, J. 2006. Model driven distribution pattern design for dynamic web service compositions. In Proceedings of the 6th international Conference on Web Engineering (Palo Alto, California, USA, July 11 - 14, 2006). ICWE ‘06. ACM Press, New York, NY, 129-136.

[14] Hemer, D. & Lindsay, P. (2002), Supporting component-based reuse in CARE, in M. Oudshoorn, ed., `Proceedings of the Twenty-Fifth Australasian Computer Science Conference', Vol. 4 of Conferences in Research and Practice in Information Technology, Australian Computer Society Inc., pp. 95--104.

[15] D. Hemer, “A formal approach to component adaptation and composition”, ACSC2005, Newcastle, Australia, 2005.

[16] P.K. McKinley, et al., "A Taxonomy of Compositional Adaptation," tech. report MSU-CSE-04-17, Dept. Computer Science and Engineering, Michigan State Univ., 2004.

[17] O. Nierstrasz and L. Dami. Component-Oriented Software Technology. In Oscar Nierstrasz and Dennis Tsichritzis, editors, Object-Oriented Software Composition, pages 3–28. Prentice Hall, 1995.

[18] Steffen Göbel, “Encapsulation of structural adaptation by composite components”, Proceedings of the 1st ACM SIGSOFT workshop on Self-managed systems, 2004.

[19] Lumpe, M., Achermann, F., and Nierstrasz, O. 2000. A formal language for composition. In Foundations of Component-Based Systems, G. T. Leavens and M. Sitaraman, Eds. Cambridge University Press, New York, NY, 69-90.

[20] Joseph R. Kiniry, “Semantic Component Composition”, Computing Science Department, University of Nijmegen, Toernooiveld 1, 6525 ED Nijmegen, The Netherlands, 2002.

[21] M. Lumpe, F. Achermann and O. Nierstrasz, "A Formal Language for Composition, " Foundations of Component Based Systems, Gary Leavens and Murali Sitaraman (Eds.), pp. 69-90, Cambridge University Press, 2000.

[22] Symfony project, www.symfony-project.com