Dynamic Web Service Composition: Challenges and Techniques

S. Prasath Sivasubramanian
Dept. of Computer Science, Avvaiyar Women’s Govt. College,
Karaikal, Puducherry, India

E. Ilavarasan
Dept. of Computer Science and Engineering,
Pondicherry Engineering College, Puducherry,
India

G. Vadivelou
Dept. of Computer Science, Kanchi Mamunivar
Centre for PG Studies, Puducherry,
India

Abstract -- ***One of the great challenges to be faced in order to enable the success of future Web-based applications is to find effective ways to handle with the interoperability demands. In this context, Service-Oriented Architectures and Web Services technology are being considered as the most affordable solution to promote interoperability, by applying strategies like Service Composition. Nevertheless, most composition approaches applied nowadays in real world contexts lack dynamism. In fact, there is not yet a consensus regarding what would really be a dynamic composition. In this paper we propose some criteria to identify the levels of dynamism and automatization in service compositions. Furthermore, taking into account a model driven approach, we propose a strategy where different techniques can be used to make compositions more dynamic and automatic. This strategy is then exemplified and discussed considering an e-Government composition scenario.***

Keywords -- ***Model Driven Architecture, Semantics, Service Selection and Web Services Composition.***

I. INTRODUCTION

The term interoperability can be defined as ’the ability of two or more systems or components to exchange information and to use the information that has been exchanged’ [8]. In fact, interoperability plays a key role in the new world of networked applications, specially in the e-Business and e-Government domains. In this context, much has been discussed in the literature regarding SOA (Service Oriented Architectures) and the so-called dynamic (or automatic, adaptive and even autonomic) Service Composition, but the fact is that there is up to the moment no consensus regarding the exact definition and broadness of these terms. The first contribution of this paper is the proposal of a set of parameters to classify the compositions into different levels of dynamism and automation. Based on these parameters, we then identify that most composition approaches applied in real-world contexts using traditional Web service technologies [9] (such as WSDL and BPEL) can be classified as static since the process model is created manually and the services are bound at design time. There are some attempts to apply late binding of services based on fixed interfaces and message formats, being these considered to be semi-dynamic service compositions. Approaches exposing a higher degree of dynamics can hardly be found in a real world context. Apart from obstacles such as performance and trust, the reason is clearly related to the fact that traditional Web services just partially kept their promise of being self-contained and self-describing software components.

Taking all this into account, we propose a path to increase the levels of dynamism and automatization in the service composition process, showing where resources such as ontologies and Artificial intelligence (AI) techniques could be applied using the Model Driven Architecture (MDA) approach. The strategy is then exemplified through a composition in the e-government context. Throughout the paper, our main goal is to discuss more general strategies and techniques, which can be further applied in different scenarios, instead of focusing on some specific technology or implementation. This article is organized as follows: Section 2 presents an overview of the steps of a service composition process; Section 3 compares different composition strategies and draws one path towards a fully dynamic composition process; Section 4 presents an example in the e-Government context; and finally in Section 5 conclusions and final remarks are stated.

II. THE SERVICE COMPOSITION

A composite service can be regarded as a combination of activities (which may be either atomic or composite services), invoked in a predefined order and executed as a whole. In this way, a complex service has the behavior of a typical business process. In order to build a service composition, some steps must be taken (not necessarily in this order): (1) A process model specifying control and data flow among the activities has to be created; (2) Concrete services to be bound to the process activities need to be discovered. The service composer usually interacts with a broker, e.g. a service registry, in order to look up services which match with certain criteria; (3) The composite service must be made available to potential clients. Again the broker is used to publish a description and the physical access point of the service; (4) During invocation of a composite service a coordinating entity (e.g. a process execution engine) may manage the control flow and the data flow according to the specified process model (see Section 2.3). Next we further analyze some characteristics of these steps which we will use later as criteria to classify the compositions.

2.1 *Discovery, selection and binding*

The selection of the activities which will participate in a service composition may be done either at design time or at run-time. In the former, the bindings are static, i.e. each instantiation of the composite service will be made up of the same constituent services. In the latter, the constituent services are selected at runtime, based on automatically analyzable criteria, such as service functionality, signature and QoS parameters. Late binding implies the dynamic invocation of the constituent services, i.e. a sufficient level of interoperability has to be established, either through fixed interfaces or by applying more sophisticated matchmaking and mapping mechanisms. For a service provider, the applied binding mechanism has several business implications.

In a growing service market, third party service providers may offer the same functionality at different conditions, e.g. regarding QoS parameters like price. Applying late binding, the discovery and invocation may become scalable as the number of services increases. Thus the costs of a composite service offered by a provider may decrease along with the growing competition in the associated marketplace. The cost advantage can be either handed over to the consumer or it will increase profitability at the provider's side. Furthermore, late binding may enhance fault-tolerance and thus reliability. Since the actions in a process are not hardwired to concrete services, the unavailability of a service may be compensated through the invocation of a functionally equivalent one. In addition, there are some scenarios where important service characteristics (like price) change constantly, what makes the use of run-time service discovery almost essential for the success of the composition. On the other hand, in some specific application domains, the lack of determinism, i.e. the fact that it is not possible to previously know which service is going to be selected, is not acceptable.

2.2 *Creation of the process model*

Another significant characteristic of a service composition strategy is the degree of automation in the creation of the process model. Traditional service composition methods require the user to define the data flow and the control flow of a composite service manually, either directly or by means of designer tools, e.g. in a drag-and-drop fashion. Subsequently the process description is deployed in a process execution engine. Depending on the abstraction level provided by the tools and also on the applied binding mechanism, the user either creates the process model based on concrete service descriptions or based on abstract service templates which are representatives for sets of services, i.e. for service classes. With respect to the multitude of available services and service templates, it may be a time-consuming task to manually select reasonable building blocks for the composite service.

Furthermore, the creation of the data flow, i.e. the parameter assignments between the activities, can be complex and might require the user to have extensive knowledge about the underlying type representations. More advanced composition strategies actively support the user with the creation of the process model, which is often referred to as semi-automated service composition. Corresponding modeling tools may interact with a broker in order to automatically look up services which match (regarding IOPEs - Inputs, Outputs, Preconditions, Effects) with the already available control and data flow, thus facilitating and accelerating the creation of the process model. The same applies for the creation of models that are based on abstract functional building blocks (which will be bound to concrete services at run-time). Parameter assignments between these building blocks may be automatically recommended based on an analysis of the underlying types and concepts.

Fully-automated composition approaches intend to generate a service composition plan without human interaction. Mostly AI inspired methods based on formal logic are used for that matter, such as automated reasoning through theorem proving. By means of a planning algorithm a workflow graph containing available activities or concrete services is generated to satisfy the requirements established by the requestor. If there are multiple solutions for the problem, i.e. several plans satisfy the given set of requirements, a selection is made based on QoS parameters. This selection can either be made by the process designer or automatically through predefined weighting and ranking functions. Combining the latter with late service binding implies that the complete service composition (i.e. process model generation and service selection) can be performed at run-time. The question to which extent the composition procedure can be automated is subject to research. Fully automated service composition may work in narrow and formally well-defined application domains. The more complex the context however the more difficult it will be to apply the automated service composition approach in real-world applications. Again the applied degree of automation for generating the process model has significant business implications for a provider who composes services and delivers them to consumers. As mentioned above, modeling the control flow and the data flow of a composite service may be time-consuming tasks. (Semi-) automated composition techniques promise to speed up this procedure, thus bringing down the costs for developing new services. Furthermore time-to-market is accelerated since the provider may react faster and more flexible to the customer requirements. In addition the designed composite services improve in quality as the application of "intelligent" tools helps to create more efficient processes, e.g. by proposing parallel execution of functionally independent activities. One interesting (and feasible) approach for semi-automated compositions was proposed in the SATINE project [4]. It is based on self-contained activity components, which are created semi-automatically based on OWL-S [3] service ontologies.

1.3 *Execution*

When composing Web Services, two different execution models are usually applied: *Orchestration* and *Choreography*. There is not a common sense regarding these two definitions, but we can consider that in an *Orchestration* all interactions that are part of a business process (including the sequence of activities, conditional events, among others) must be described, like on a traditional workflow system. This description is then executed by an orchestration engine, which has control of the overall composition. On the other hand, a *Choreography* is more collaborative and less centralized in nature. Only the public message exchanges are considered relevant and more, each service only knows about its own interactions and behavior. Differently from Orchestration, there is not an entity that has a global view/control of the composition [12, 14]. If we refer to the origin of the words, a good comparison can be made. The first, orchestration, can be compared to a set of musicians (*services*) commanded by a conductor (*engine*). The second, choreography, can be compared to a group of dancers (*services*) that already know how to perform and that don't obey to a central coordination. Usually real scenarios involving complex systems with multi-part interactions demand both approaches. In [5] the authors propose a set of policies to regulate service compositions and establish a relationship among these policies and the execution models.

III. TOWARDS A DYNAMIC SERVICE COMPOSITION PROCESS

In this section we first present a clear classification of different composition strategies in terms of dynamism and automation. Then we draw a possible path towards a fully dynamic composition process based on a model-driven approach.

3.1 Service composition strategies

Besides the execution model (orchestration or choreography), two service composition characteristics have been examined, namely the type of service discovery/selection/binding and the degree of automation applied for the creation of a process model: service composition approaches may use early binding or late binding; the process model can be created manually, semi-automatically or automatically. As illustrated in Figure 1, these characteristic values can be used for a classification of existing service composition strategies in six main categories. The fact that the borders between these categories are not strict but fluent is made clear through the smooth transitions between the squares. Some categories may overlap, i.e. there are composition approaches that may be assigned to two or more categories. To give an example: besides early and late binding there may be several variations in between, such as the specification of a restricted set of service candidates at design time from which one service is chosen and invoked at run-time.

In the previous discussion regarding the implications for an actor who creates and provides composite services, it was argued that composition approaches applying late binding mechanisms are more adaptable to a changing environment, where

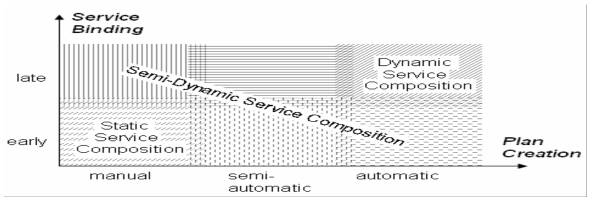
Service Binding
late
early
Dynamic Service Composition
Semi-Dynamic Service Composition
Static Service Composition
Plan Creation
manual
semi-automatic
automatic

Figure 1: Classification of Service Composition Strategies

third party providers are frequently leaving and joining. Furthermore it was argued that a high degree of automation during creation of a process model cuts down development costs and accelerates time-to-market, thus resulting in a higher flexibility of a composite service provider. In addition, quality aspects, such as reliability, have been considered. When combining the terms adaptiveness and flexibility to the more generic term dynamics, a coarse-grained and more business-oriented classification in static, semi-dynamic and dynamic service composition strategies can be made (see Figure 1). Taking into account the above mentioned attributes cost efficiency, time-to-market and reliability, it can be argued that a high degree of dynamics for service composition has positive effects on the providers' profitability. On the other hand this does not inherently mean the more automation the better. The degree of dynamics applicable in a real world context is limited by many more factors, being trust (see Section 3.2) one of the most relevant. In addition, performance issues can also represent a problem, since interacting with a broker for service discovery and match-making as well as applying sophisticated AI algorithms for automated plan generation may be time-consuming tasks.

3.2 Model Driven Approach

The Model Driven Architecture (MDA)[10] is a new approach proposed by the Object Management Group (OMG) to develop applications and write software specifications. It is based on three standards: the Unified Modeling Language (UML), the MetaObject Facility (MOF) and the Common Warehouse Metamodel (CWM). These standards should facilitate the design, description, storage and exchange of models. The MDA approach separates the specification of the operation of a system from the details of the way that system uses the capabilities of its platform. It uses abstract models to specify all the logic of the application where concepts on languages or platform are irrelevant. Later, these models are used to create new models which express the requirements of the system in a specific platform. Note that Platform independent and platform specific are not absolute concepts: what is specific to one system can be independent to another. The MDA specifies that the following models should be created during a development process [10]:

- Computation Independent Model (CIM): also called a domain model, focuses on the environment and requirements of the system. The details of the structure and processing are hidden;
- Platform Independent Model (PIM): provides a description of the system from a platform independent viewpoint, focusing only on the system functionalities;
- Platform Specific Model (PSM): describes the system combining the specifications in the PIM with the details regarding the platform where the system will run.

The great advantage in using MDA is the ability to transform a platform independent model (PIM) into a model capable of running into a great variety of technologies. MDA assumes that technology is very volatile, thus an automatic PIM-PSM transformation can save time and money by improving the efficiency of steps like implementation, integration, maintenance, testing and simulation. In an ideal world, the developer would simply submit the PIM to generators which would produce in the end executable code. But the reality is different and we are far away from this - in practice a lot of manual work must still be done. In Figure 2 we see a composition process following the MDA approach. The process starts at the definition of CIM, usually a manual task (step 1). This definition must include, among other things, the identification, specification and modeling of the composition. The UMM (UN/CEFACT Modeling Methodology) [2] is an example of methodology that could be used at this phase of the project. The first transformation takes place into CIM-PIM (step 2).

The transformation between models can be complex and, in almost every case, parameters need to be set in the source model in order to drive the transformation. After the transformation, refinements should be performed in the PIM in order to go in the direction of the executable code (step 3). In order to stay aligned with the RM-ODP, the information and computational viewpoint languages must be used. Otherwise, BPMN (Business Process Modeling Notation), UML activity diagram or EDOC profiles are also typical languages used to design the PIM in a service composition context. Next, a transformation from a PIM into a PSM takes place (step 4). This transformation plays a special role in MDA and a mapping language can be defined to automate it. In an ideal world, the same PIM could be transformed in several PSMs, for instance, a PSM-J2EE, PSM-CORBA, PSM-.NET, PSM-WS or any other. The PSM can be specified using the engineering viewpoint language defined by ISO/IEC or UML activity diagrams with extensions for a specific platform. As the previous transformation, some parameters can be set in the PIM to drive the transformation and, after it, the refinement of the PSM should be necessary (step 5). Bzivin et al. [1] describes a PIM (UML and EDOC) transformation to PSM (Java and Web Services), focusing on the static aspects of the mappings. Patrascoiu [11] presents the mapping from EDOC profiles to Web Services using a transformation language called YATL, also

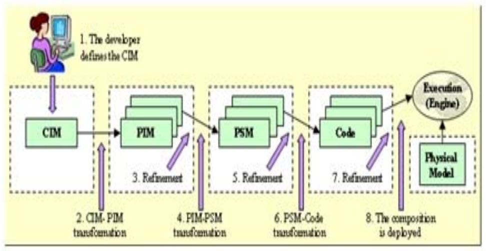
1. The developer defines the CIM
CIM
PIM
PSM
Code
Execution (Engine)
Physical Model
3. Refinement
5. Refinement
7. Refinement
2. CIM-PIM transformation
4. PIM-PSM transformation
6. PSM-Code transformation
8. The composition is deployed

Figure 2: The Composition process following an MDA approach

considering the dynamic aspects of a composition. A framework proposal for service composition using the MDA approach applied to the e-Government area was presented by Tizzo et. al.[15].

Finally, the last transformation: PSM-code (step 6). The code is the composition described in some executable language (BPEL for example). A very detailed PSM can describe the whole service composition logic. So, the gap between PSM and code can be very small. But, as the previous transformations, fine adjustments can be necessary before actually running it (step 7). The code, added to the deployment description, completes the models that are necessary to run the composition (step 8). Analyzing the MDA approach in a reverse way, the automatic code generation would start in the PSM-code transformation. When it's possible to produce a full executable code from this transformation, the code would be hidden and in fact the PSM would be executed by a virtual machine. This process is analogous to the one that happens with a traditional compiler or interpreter. Going further, if we apply this automatization within the PIM-PSM transformation, a virtual machine could execute the PIM. The next section presents the challenges and guidelines in order to archive this goal.

3.3 Using Semantics and MDA to increase the dynamism in service composition

According to the MDA (see previous section), the automation may take place in to different directions: from one model to another (model transformations) and inside a model (model refinement). During these steps, the abstraction level is gradually reduced. Note that the service compositions, which are described using abstract services, need to be binded to concrete services at a certain moment in time. The use of semantic descriptions and ontologies plays a special role in this stage. The composition described in a CIM is a description of a sequence of tasks and its data and control flow. At this point, a task is an abstract service, i.e., it is only a service description and it may not have an implementation. The information used to describe a task is composed of four fundamental elements: signature, preconditions, post conditions and information invariants [7]. Non-functional aspects can also be described. The activities in a CIM must then be detailed process done during the CIM-PIM transformation and/or PIM refinement. The automatic (or semi-automatic) transformation and refinement must rely on semantic descriptions to provide machine-readable information about each task. Algorithms based on AI techniques [13] (such as Situation calculus, PDDL, Rule-based planning or Linear Logic) can use these descriptions to decompose CIM tasks or refine the PIM. As already mentioned in Section 2.1, one of the characteristics that can be found in dynamic compositions is the possibility of selecting the participant services. Theoretically, this can be done at the PIM, PSM, code or run-time (late-binding): when it will happen can be determined by a technology restriction or by a project decision: (1) the BPEL language does not allow service selection at run-time, forcing the developer to do it before; (2) on the other hand, when the process environment changes over time, the developer can decide to do the selection as later as possible.

In order to perform the selection of services and considering the late-binding scenario, the result of step 8 (Figure 2) would be a composition of Abstract Services, to be bound to Concrete Services at run-time. In Figure 4, this process is illustrated. An engine receives the composition description (i.e. the code), starts to run it and for each abstract service (described with semantic information), a service repository (or a service broker) is contacted in order to discover which service will be the responsible for executing that activity, always considering the associated ontology.

IV. AN EXAMPLE

In order to apply the techniques presented in Section 3, we consider now an example in the e-Government context. The goal of the composition is to provide birth certificates to citizens through an e-Government portal. First of all, the portal checks the municipality where the citizen was born (city of birth). This is an essential step in order to correctly determine which services will participate in the composition. For different municipalities, even though the CIM is the same, different services should be selected and a different sequence of activities might be performed. That is also an important fact to justify the use of a dynamic strategy in this scenario. Furthermore, each one of the activities involved in the process can be performed by services located in different organizations. This would also create the necessity of considering aspects such as privacy, autonomy and security, omitted in the example for not being in the scope of this paper. An interesting strategy for handling these cross-border issues is the use of Interaction Policies [5]. Referring back to the strategies presented in Section 3, and considering a unique CIM as input, the information ”city of birth” may be used in one of the following manners:

- Considering that each activity of the CIM could be implemented in a different way for different cities, the ”city of birth” is used to determine the result of the CIM-PIM transformation and refinement. That is, each city might have a different associated PIM;

- If besides the same CIM, the PIM is also identical for all municipalities, the differences might occur in the PSM. It means that the activities are exactly the same for all cities, but each city may rely on a different technological platform;

- when the CIM, PIM and PSM are identical, only one composition is built, deployed, and the ”city of birth” is used as parameter for selecting the correct concrete services to perform each of the activities in a given execution instance.

Note how one single variable, depending on the strategy adopted and on the characteristics of the problem, may have a great impact on the result composite service. Remember that in our example, the citizen requests the emission of a birth certificate at an e-Government portal. The steps of this process described in the CIM (Figure 5) are:

1. The citizen has to pay for the emission of the certificate;

2. If the payment succeeds, the process continues and the request is forwarded to a public servant (step 3). If not, the citizen must be notified about the problem;

3. A municipality servant must check the pending certificate emission request and validate them. In case of his approval, the request proceeds to step 4. If the servant, for any reason, denies the request, the citizen must be notified;

4. Finally, the certificate must be emitted and the citizen notified about the status of his request.

Other requirements of the process could also be modeled in the CIM (in fact they are necessary if we desire to automatically proceed to next step in the MDA process), but it is not in the scope of this article to further detail them. As shown in Figure 2, the next steps in our strategy are then the transformation of the CIM into a PIM and the PIM refinement (Figure 6). We consider in our example that each municipality may have its own PIM, and that the information ”city of birth” will be important in all transformations and refinements and also in the execution phase (to correctly select and bind the abstract services to concrete implementations). Therefore, the PIM-PSM and PSM-Code transformation would be analogous to the CIM-PIM transformation illustrated in Figure 6.

In order to implement these transformations we consider the associated ontology, the services registry and also a database with previous existing activity models. This model database is particularly important because it contains, at each level, a description of the possible activity mappings to the immediate lower abstraction level in the composition (see Figure 3). Note that if we do not rely on this strategy, we must them adopt some AI technique (Section 2.2) to perform the abstraction level change.

In Figure 7 we see the UML Activity Diagram which is part of the PIM for a given municipality. Differently from the CIM, the PIM describes in more details the sequence of activities that should be performed to successfully complete the process:

- An activity called ”receiveCitizenData” is responsible for getting the citizen request and information;

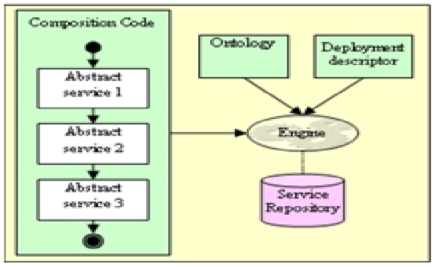

Figure 4: Composition with late (semantic-based) binding

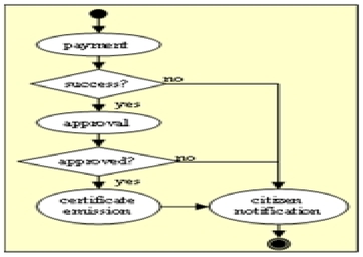

Figure 5: CIM (UML Activity Diagram)

- The payment information is sent to the activity ”receivePaymentInfo”, that checks whether the citizen will pay by Credit Card or by Payment Order, and then calls the associated activity to actually perform the payment (”debitCC” or ”paymentOrder”);
- If the payment was successful, the process proceeds to step 4. If not, an e-mail notification activity must inform the citizen the reason of he failure;
- A public servant participates in the activity ”employeeApprovalService” and checks if everything is fine with the request, validating it. In case of approval, the process proceeds to step 5. In case of failure, the citizen must be informed by the activity ”emailNotification”;
- The certificate is generated electronically (”certificate-Generator”) and then printed and stamped (”certificateEmission”), being finally the citizen notified of the success of his request and informed about the procedures to pick up the document.

Next follows a (possible automatic) transformation from the PIM into a PSM which can be, for instance, a BPEL specific one. Finally a PSM-Code transformation takes place, and the resulting composition code (made of abstract services) is deployed. At run-time, the binding to the concrete services is performed like Figure 4 illustrated.

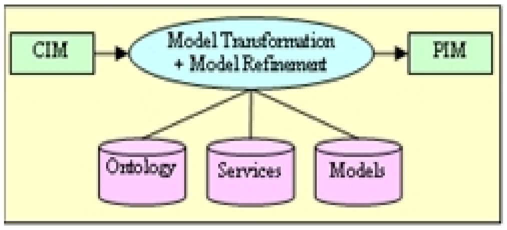

Figure 6: CIM-PIM transformation and refinement

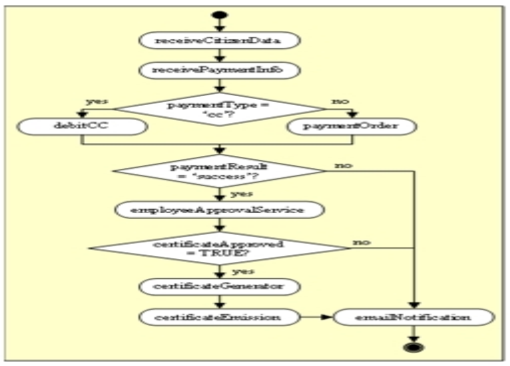

Figure 7: PIM (UML Activity Diagram)

V. CONCLUSIONS AND FINAL REMARKS

The proposal of effective solutions to enable interoperability among heterogeneous and inter-organizational systems remains a key issue in the development of new Web-based applications, especially in the e-Business and e-Government contexts. The so called Service-Oriented Architecture, and more specifically its Composition layer, appears as a promising solution to deal with these demands. Even though most part of the academia and industry considers that Dynamic (and/or Automatic) Service Composition is the next stage to be reached, there is still a lot of misuse and misunderstanding regarding these concepts. In this paper, our first contribution is the proposal of a clear classification of different composition strategies with regards to their levels of dynamism and automatization.

Besides this classification, another important contribution of our paper is the introduction of a generic model driven approach for composing services, which is further specialized to illustrate the techniques that can be applied to enable fully dynamic service compositions. Aspects like dynamic selection of services and the use of AI techniques to automatically generate the execution plans are also discussed. An example service in the e-Government context is presented to illustrate the use of the proposed techniques. As already mentioned, the goal of this paper was not to focus on any specific technology or implementation solution, but rather to critically analyze the challenges and possible alternatives regarding service compositions in general. An interesting extension to this work would be to apply the proposed strategies in some technology specific scenario and evaluate its potentials and limitations considering different domains and platform characteristics.

Current Web services, without any support from agents, still do provide the capability to seamlessly integrate different platforms. They provide an excellent choice for implementing distributed applications because they are architecturally neutral. Agent technology provides several advantages, which can be easily incorporated in existing Web services. The capability of agents to be autonomous, cooperate with other agents, be aware of the context in which they are invoked, and dynamically adapt to changes in the environment are some of the main advantages that agent have compared to current Web services. Agent-based Web services would provide clients with a fast, personalized, and intelligent service. This is turn will increase the percentage of returned customers because of higher satisfaction from services provided.

We believe that much research is still necessary and also that a fully dynamic composition may be still far from becoming a reality, except for specific and well-defined application domains. Nevertheless, the research community, having recognized the potential of the evolving Semantic Web, has spawned several activities in the direction of Semantic Web Services. With languages like the Web Ontology Language (OWL), machine-understandable Web service descriptions can be created and shared. Generic service ontologies, such as OWL-S [3] and WSMO [7], in combination with appropriate rule languages and new AI techniques lay the foundations for semantically describing the functionality and the behavior of services, and keep the road open for a future of dynamic service compositions.

REFERENCES

[1] J . Bzivin, S. Hammoudi, D. Lopes, and F. Jouault.Applying mda approach for web service platform. In Proc. of the 8th IEEE Intl Enterprise Distributed Object Computing Conference (EDOC 2004), pages 58-70, 2004.

[2] CEFACT. UN/CEFACT Modeling Methodology (UMM) User Guide CEFACT/TMG/N093.

[3] T. O. S. Coalition. Owl-s: Semantic markup for web services. White paper - http://www.daml.org/services, July 2004.

[4] A. Dogac, Y. Kabak, G. Laleci, S. Sinir, A. Yildiz, S. Kirbas, and Y. Gurcan. Semantically enriched web services for the travel industry. ACM Sigmod Record, 33(3), Sep. 2004.

[5] I. J. G. dos Santos and E. R. M. Madeira. Applying orchestration and choreography of web services on dynamic virtual marketplaces. International Journal of Cooperative Information Systems (IJCIS), 15(1):57-85, Mar. 2006.

[6] C. Feier and J. Domingue. The Web Service Modeling Language WSML. DERI International, WSML Final Draft, Apr. 2005.

[7] D. Frankel. Scaling the business process platform up. MDA journal - http://www.bptrends.com, Dec. 2005.

[8] IEEE. IEEE Standard Computer Dictionary: A Compilation of IEEE Standard Computer Glossaries. New York, NY, 1990.

[9] N. Milanovic and M. Malek. Current solutions for web service composition. IEEE Internet Computing, 8(6):51-59, Nov.-Dec. 2004.

[10] OMG. MDA Guide Version 1.01. http://www.omg.org/cgi-bin/apps/doc?omg/03-06-01.pdf, 2003.

[11] O. Patrascoiu. Mapping edoc to web services using yatl. In Proc. of the 8th IEEE Intl. Enterprise Distributed Object Computing Conference (EDOC 2004), pages 286-297, 2004.

[12] C. Peltz. Web services orchestration and choreography. IEEE Computer, 36(10):46-52, 2003.

[13] J. Rao and X. Su. A survey of automated web service composition methods. In Proc. of 1st Intl. Workshop on Semantic Web Services and Web Process Composition, July 2004.

[14] S. Ross-Talbot and N. Bharti. Dancing with Web Services: W3C chair talks choreography. http://searchwebservices.techtarget.com/, Mar 2005.

[15] N. P. Tizzo, J. R. Borelli, M. de Jesus Mendes, L. Damasceno, A. Kamata, A. Figueiredo, M. A. Rodrigues, and J. G. S. Junior. Service composition applied to e – government. In IFIP 18th World Computer Congress, Building the E-Service Society: E-Commerce, E-Business and E-Governement, pages 307-326. Kluwer Academic publishers, 2004.