Design of a Framework for Automated Service Mashup Creation and Execution Based on Semantic Reasoning

Anna Hristoskova, Bruno Volckaert, Filip De Turck, Bart Dhoedt
IBBT, Department of Information Technology
Ghent University
9050 Ghent, Belgium
{*Anna.Hristoskova|Bruno.Volckaert|Filip.DeTurck|Bart.Dhoedt*}*@intec.UGent.be*

***Abstract*—Instead of building self-contained silos, applications are being broken down in independent structures able to offer a scoped service using open communication standards and encoding. Nowadays there is no automatic environment for the construction of new mashups from these reusable services. At the same time the designer of the mashup needs to establish the actual locations for deployment of the different components.**

This paper introduces the development of a framework focusing on the dynamic creation and execution of service mashups. By enriching the available building blocks with semantic descriptions, new service mashups are automatically composed through the use of planning algorithms. The composed mashups are automatically deployed on the available resources making optimal use of bandwidth, storage and computing power of the network and server elements. The system is extended with dynamic recovery from resource and network failures. This enrichment of business components and services with semantics, reasoning, and distributed deployment is demonstrated by means of an e-shop use case.

***Keywords*-Service Mashups; Semantic Web; planning algorithms; runtime adaptation; Quality of Service**

I. Introduction

Dynamics and efficiency are concepts of the future. The World Wide Web is undergoing an evolution from a static environment to a dynamic world in which service mashups will play a central role. A service mashup is a new service that combines functionality or content from existing sources, where the service offered by the mashup is greater than the individual participating components. These sources can be Web services, software components capable of being accessed via standard network protocols such as but not limited to SOAP over HTTP.

The arrival of these services to the ICT scene revolutionized the software architecture in the public Internet space but also in the enterprise sector. For businesses, creating a catalogue of reusable components means agile creation of new services and faster adaptation to the changing business environment. This gave birth to Software-Oriented Architectures (SOAs) [1] composed of software components, more specifically Web services. The current infrastructure for Web services has however as downside that service interfaces specify only the syntax of the provided operations without offering support for the semantics.

This issue is covered by the Semantic Web [2], [3] employing ontologies and semantic languages offering several degrees of expressiveness to describe concepts and Web services. Ontologies focus on specifying inputs, outputs, pre- and post conditions (IOPEs), and non-functional properties of services. The interaction model of the semantic languages supports choreography and/or peer-to-peer (P2P) orchestration for Web services. It enables users and software agents to automatically discover, invoke, compose, and monitor Web resources offering services, under specified constraints.

Using the semantic technologies different reasoning methods with varying complexity can be applied on Web services. At the lower level are the Matchers [4], comparing service interfaces and assigning a score depending on the extent to which services meet the requested service profile. At the level above one can find the Composers [5]–[8] adopting AI planning to accomplish a complete service composition resolving a defined goal. The highest level belongs to the Middle Agents [9]. These entities not only execute a fully automatic composition of Web services, but also take care of tasks like transformation of questions and answers, and Quality of Service (QoS).

Building on these principles, this paper proposes a mashup creation and execution environment allowing for the construction of new services departing from available functionality found on the Web or within enterprises. The developed framework disposes of a user interface (UI) for the management of semantically annotated services and the definition of user requests. Planning algorithms are designed constructing service mashups achieving these user requests. An important aspect is the runtime behavior of the framework anticipating changes (new services, failure, etc) and personalizing each request through business logic rules defined by the user.

The remainder of this paper is structured as follows: Section II presents the general concept of the mashup platform. A discussion of the current research in the field is given in Section III. Section IV exposes the development process of the platform which is evaluated based on an e-shop use case analyzed in Section V. Finally, the main conclusions are presented in Section VI and new possibilities for enhancing the platform are explored.

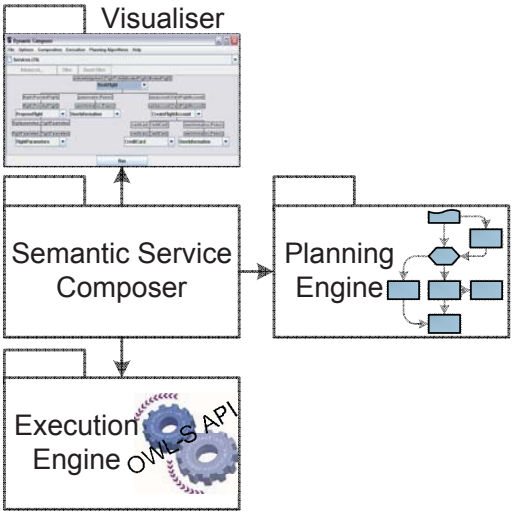

Figure 1. Dynamic Composer for Web Services

II. General Concept

The main objective of this research is to design a framework for supporting at runtime creation and execution of new service mashups without the intervention of and programming by the ICT department. Starting with a brief description of the previous system, the Dynamic Composer for Web services [10], the general idea of the developed framework is outlined. Section IV presents a more detailed discussion on its architecture.

A. Dynamic Composer for Web services

Figure 1 presents the architecture of the Dynamic Composer for Web services built around the *Semantic Service Composer*, which in turn communicates with the *Visualiser*, *Planning Engine* and *Execution Engine*.

The *Semantic Service Composer* processes initial state, goal and service descriptions annotated using OWL-S and based on semantic matching of service outputs to inputs and service effects to preconditions constructs a semantic description of the composite service mashup. The automatic composition is covered through QoS-aware planning algorithms implemented by the *Planning Engine*. Afterwards, this composition is presented by the *Visualiser*, enabling the user to manually tune it to his needs and change user preferences such as planning algorithm, execution method (e.g., sequence, parallel), etc. Subsequently, the *Execution Engine* handles the execution of the service mashup. In case of a failure, a recovery procedure is set in motion constructing an alternative execution keeping state information in mind.

B. Mashup Creation and Service Orchestration

The key difference between the Dynamic Composer and the presented platform in this article is the partitioning of the *Planning Engine* into a *Workflow Reasoner* and a *Service Mapper*. Instead of the immediate creation of a concrete service mashup from a defined goal, first an abstract composition is created by the *Workflow Reasoner* using the semantic service descriptions. The *Service Mapper* links the semantic mashup components to concrete service instances offering minimum required QoS (e.g., execution time, cost). An *Execution Environment* component is added acting as a blackboard connecting information processed by the Reasoner, Mapper and Execution Engine. In this way the mashup in construction is at runtime tuned by the three components and adapted to user defined business logic rules.

The UI was improved with an administrator's interface for the management of the service pool and extended with a logging component continually sending status updates of the different request processing stages. The user of the system may be an administrator or an end user. The service pool can be defined by the administrator but the end user is also able to add his own services. The same holds for the definition of a request. The idea of this platform is to transform into a software agent making it possible for the end user to define his own requests and select his preferred services.

III. Related Work

As mentioned in Section I the Semantic Web offers more access not only to content but also to Web services and resources. Since the information is presented in a formal way, a computer can reason about it and new knowledge can be inferred. This way the wide range of business components can be dynamically and more efficiently combined through semantic reasoning, accomplishing new service mashups. Today there are a number of popular standards and implementations [11], such as BPMN, WFF, BPEL4WS, XLANG, WSFL which define workflows. However they still exhibit a number of shortcomings: no automatic or dynamic deployment support, limited reliability guarantees, etc.

Several research projects among which some within the European Union Sixth Framework Programme for Research and Technological Development aim at creating platforms supporting the creation, management and execution of service mashups. The SODIUM [12] and OPUCE [13] projects consist of a set of languages and tools as well as related middleware, for the creation and execution of workflows composed of heterogeneous services. The MoSCA [14] middleware facilitates the development and deployment of workflows and provides for at runtime selection of the service providers that are capable of collectively delivering the composite service with the highest reliability. Unforeseen changes to such patterns are monitored, potentially triggering re-bindings during service execution.

Although these platforms focus on the runtime behavior of the composite services, the design time composition is usually a manually created workflow. Platforms like INFRAWEBS [15] and Amigo [16] propose approaches, in which the process of finding appropriate services is guided

by algorithms for decomposition of user goals into sub-goals and discovering the existing services able to satisfy these sub-goals. MashWeb [17] goes further than goal matching through the creation of dataflows controlling the output-input flows and workflows controlling the execution sequence of the specific services.

The presented platform adopts mashup creation and execution techniques from both worlds. Planning algorithms generate a service mashup starting from goal conditions through matching of service effects to required service preconditions. The platform automatically defines the service providers offering the minimum defined QoS for execution. Several iterations of mashup configuration and execution are possible as intermediary results are used as feedback to further tune the designed service mashup. The novelty of the platform is the use of business logic rules defined by the user which enable further tuning and personalization of the user's request.

IV. Platform Design and Implementation

This section focuses on the design of the presented platform. Firstly, a discussion on the different building blocks of a semantic mashup is presented. Afterwards, we look more closely into the different modules needed for the mashup creation and execution process.

A. Mashup building blocks

There are two types of components present in the system: *abstract semantic types* and *concrete service instances*. Existing *services instances* are enriched with semantic annotations using OWL-S. As several semantically equivalent services (matching inputs, outputs, and if necessary preconditions and effects) can exist, their semantic descriptions are grouped into one *semantic type* referring to all the equivalent *service instances*.

During the reasoning process, the *semantic types* are used to construct the composition of a new mashup type. Hereafter, the mapping process selects the *concrete service instances* corresponding to the utilized semantic types, which are capable of offering minimum required QoS.

1) Semantic Types: The *semantic types* used by the system are *goal types*, *service types* and *mashup types*. The intention is to find a common way of presenting their semantic description (IOPEs) so that the Workflow Reasoner has a general way of working with them.

- The *goal type* defines initial (inputs and valid conditions) and goal (required outputs and conditions) state information of a workflow provided by the user. Based on these, a *mashup type* is composed out of *semantic types* taking the initial state information as input while reaching the specified goal state.
- The *service type* consists of a definition of its IOPEs in terms of OWL concepts and properties. Services with effects are *world-altering services*. In contrast with *information providing* services (only outputs, e.g., sensors) which can be executed at any time especially during the planning process, *world-altering services* are only executed at composition execution time as they alter the state of the world.
- The *mashup type* is a composite service description using control constructs that can be represented by a *goal type*. To the outside world, it is just another service with its IOPEs. In this way, this new *service type* is used as part of other mashup compositions.

All three *semantic types* are expressed in the same manner by defining their IOPEs. In this way a flexible platform is created where a semantic mashup is used as a building block for other mashup constructions.

2) Service Instances: A *semantic type* is an abstract entity. In order for a service to be executed, at least one *concrete service instance* must be available at service request time. After the reasoning process, the *semantic types* are mapped to the *concrete service instances*, collectively delivering the composite service mashup, with a defined minimum of QoS.

B. Architectural Modules

Figure 2 presents the main components of the architecture. All requests (goal types and if necessary business logic rules) pass through the **Frontend** and are handled by the **Core**. The requests are sent to the **Request Scheduler** which in turn dispatches them to the Request Portal for further mashup creation and execution.

The **Request Portal** keeps track of the whole reasoning, mapping and execution process for a single user request. This object presents the user with the composite mashup, the utilized resources for execution, intermediary results, etc.

The **Workflow Reasoner** accepts a *semantic type* together with case specific business logic rules and creates a *mashup type*. This *semantic type* can be a defined goal, a service type or a mashup type as they are all expressed in terms of IOPEs. The following functionality is provided by the Workflow Reasoner:

- *Mapping* of parts of a type description to existing *service types*. This is useful during *mashup type* reconfiguration as we might not want to perform reasoning on the whole mashup again.
- *Construction* of a *mashup type* through semantic matching of IOPEs: a service provides outputs used as inputs for another service and effects accomplishing preconditions required for the execution of services.
- *Adaptation* of the designed mashup to user defined *business logic rules*. These rules are defined using SWRL and OWL concepts.
- *Mashup types* can be *cached* for reuse. In case of repeating goals the performance of the reasoning is improved and the mashup can be used as building block for new compositions.

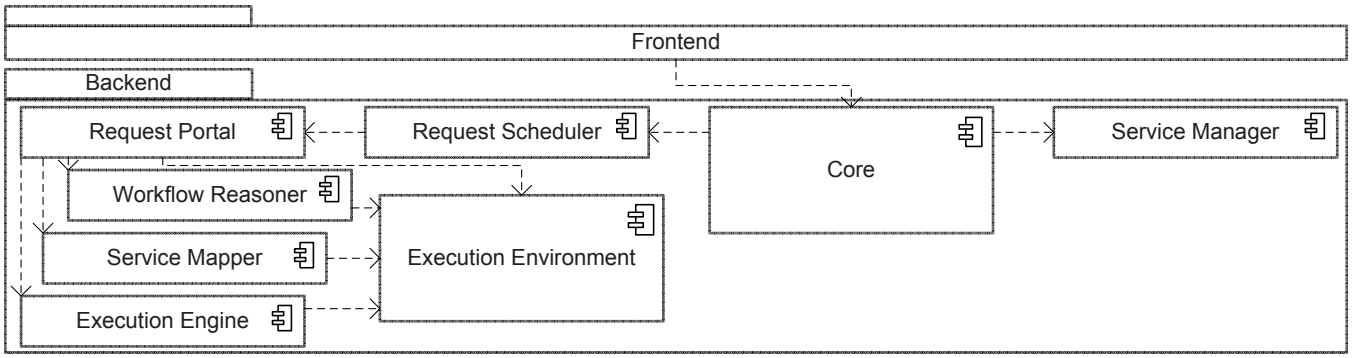

Figure 2. Main building blocks of the mashup creation and execution environment.

The **Service Mapper** instantiates the *mashup type* by mapping *service types* to *service instances* keeping in mind the required QoS (in this case minimal execution time and cost). The Mapper should be able to execute alternative service instances depending on the existing network and load of the available resources.

The QoS of a specific *service instance* consists of a QoS type, QoS Value, QoS Comparator. A QoS Type can be the cost for executing a service, the execution time, etc. Each QoS Type disposes of a QoS Value and a specific QoS Comparator for comparing the actual QoS Values.

The **Execution Engine** handles the execution of the mashup through the execution of the separate service instances. State information of the mashup is stored in case of failure in the Execution Environment. This state information is present in the effects of the world-altering services.

The **Execution Environment** acts as storage for business logic rules, inputs, goals, results, composite mashups, etc. Data is gathered by the Workflow Reasoner and Service Mapper to guide the reasoning and mapping process of the new service mashup at design and runtime. In case information-providing services should be evaluated during the construction of the mashup, the Execution Engine stores their intermediary results in the Execution Environment. State information from world-altering services is stored in a similar fashion during the effective execution of the mashup in case of failure. Figure 3 presents a basic matching principle where services are executed using inputs and conditions from the Execution Environment and service effects and outputs are produced and added to this Environment. This results into a dynamic system where new knowledge is evaluated and added at runtime.

C. Scenario description

The composition and execution process is presented in Figure 4 using a pipes & filter flow. Starting from a definition of a general goal, an *abstract mashup type* is constructed by the *Workflow Reasoner* providing the needed calculations for achieving this goal. Next the *Service Mapper* maps the different components of the composition to *concrete service*

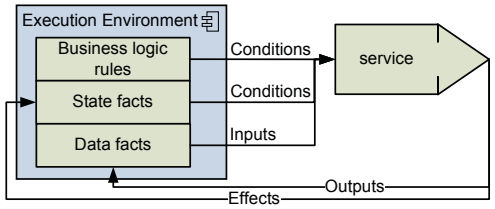

Figure 3. Inference through matching of service preconditions and inputs and returning service effects and outputs.

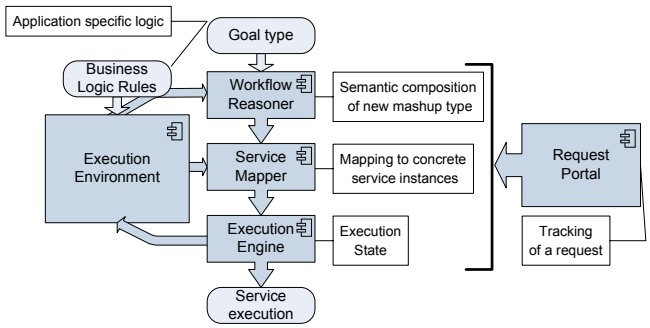

Figure 4. Workflow of the composition and execution process.

instances keeping in mind QoS constraints and requirements. The whole is executed by the *Execution Engine* where the recovery procedure kicks in in case of external errors, such as network failure or a service delivering erroneous results. The reasoning and mapping process gather the needed information through the *Execution Environment* where case specific business logic rules and intermediary results are stored for further adaptation of the composite mashup.

A *Request Portal* keeps track of the whole process for a single user request. Information stored into this module is used to present the service mashup to the user, which he can further tune to his specific needs, present the utilized resources for execution, return intermediary results, etc.

The idea for this system is to resemble an expert system. This offers a framework able to dynamically react at runtime to changing context, optimizing the composition and execu-

tion process. This divides the reasoning process into two steps. First through backward chaining a general composition of service types is achieved specifically resolving user defined goals. A forward chaining procedure further tunes this composition utilizing the defined business logic rules and the intermediary results of already executed information-providing services.

V. Use Case Example

This section describes functionality provided by an e-shop system including the different iteration steps of the reasoning and execution process. For this purpose an e-shop ontology was created defining the different concepts needed for the annotation of the service IOPEs. Next, we developed the e-shop services needed for the use case and from their WSDL interfaces created OWL-S descriptions using the e-shop ontology.

A. E-shop description

A sale consists of a customer buying one or more products. Traditionally, this means that:

1) The customer orders the products, selected from a list of possible products (the catalogue).
2) An amount of the customer's money, equal to the price, is transferred to the e-shop.
3) The products are delivered to the customer. This can happen in several ways:
   - Digital products, such as music and software, are conveyed over the Internet.
   - Physical products are transported to the customer's delivery address or to a proxy point of the customer's choice (usually close to his location) where they can be collected.

B. Design of the e-shop workflow

1) Trigger: A potential customer browses to the e-shop handled by the service *WebShopCatalogue*.

2) Initial state: The e-shop and customer info is known. This includes account information necessary to make payments to the e-shop.

3) Goal description: The composition is successfully executed, when the following effects are reached:

- The customer selected product(s).
- The customer paid the price of the product(s).
- The product(s) is(are) delivered to the customer.

4) Design of the workflow: Figure 5 presents a workflow of the different e-shop services from selection to payment and delivery of the selected products. The effect of the selection is implied in the output of the *WebShopCatalogue*, which is the set of selected products. If the customer fails to select one or more products, the execution of the composition is prematurely ended. Otherwise a *FOR* step is required iterating over each product. A decision is made whether the product is in stock or should be ordered

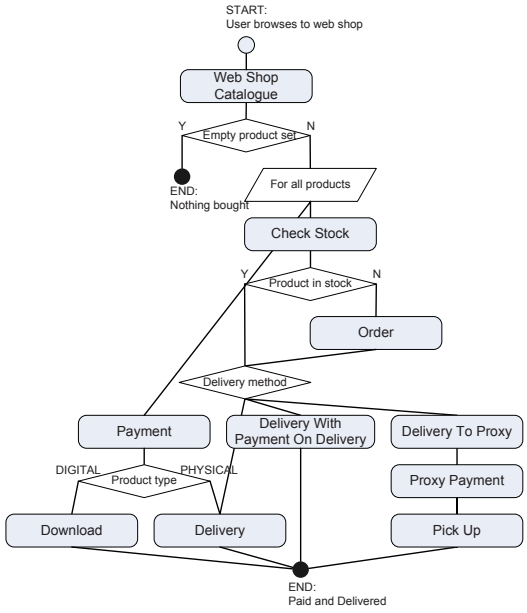

Figure 5. Workflow of an e-shop.

followed by payment and delivery. A *Delivery method* is added having as result one or more payment and delivery options in which, according to the configurable rules, the purchase is made. This result is not known at composition time but can be defined through business logic rules by the user, being a customer or an e-shop manager. If the result is one delivery method defined by the e-shop manager, the purchase is made in that way. If it is more than one, the customer chooses amongst all the possibilities and the execution path depends on his decision. The result of this interactive choice cannot always be known at composition time: the customer makes a choice after being presented with the different execution paths. Consequently, the complete e-shop composition exposes a decision point with multiple possible branches. When the composition is executed, the correct branch is chosen and followed.

C. Automatic optimization of the workflow execution

Before execution, the e-shop workflow is further pruned through the execution of *information-providing services*. Depending on their output, further decisions are made, reducing the execution paths. For example, by executing the *WebShopCatalogue* service, the Reasoner decides whether there are any selected products and if they are digital or physical. Then, the *CheckStock* service verifies whether the physical products if any are in stock. This way the *Download* or *Delivery* and/or *Order* services are automatically removed.

D. Runtime adaptation of the workflow

In order to execute the e-shop composition, the e-shop manager needs to define business logic rules expressing which *Payment* and *Delivery method* should be chosen or

the customer should choose from the offered possibilities. Once the choice is made, the reasoning process configures the e-shop composition automatically at runtime through the removal of the decision point and the selection of only one Payment and Delivery path. For example if one selects *Payment followed by Delivery* all the other options like *Payment on Delivery* and *Delivery to Proxy* are excluded.

VI. Conclusion and Future Work

This paper focuses on the study of an framework for dynamic composition and execution of the building blocks of service mashups. Based on semantic descriptions of Web services, reasoning algorithms are developed for automatically composing new service mashups realizing defined goals. These algorithms construct for a planning system satisfying several QoS constraints and requirements. This system is optimized for the dynamic response to changing context such as new business logic, failure or overload of network elements or services. We implemented an e-shop system to validate this framework and illustrate the workflow execution optimizations.

In the future the planning and execution system will be extended with a distributed deployment component which will execute the different service instances depending on the available resources making optimal use of bandwidth, storage and computing power of the network and server elements. Furthermore, techniques will be studied to take into account trends in user and resource behavior, in order to optimally design context-aware service mashups.

Acknowledgment

Anna Hristoskova would like to thank the Special Research Fund of Ghent University (BOF) for financial support through her PhD grant. This work is partly funded by WTEPlus, an IWT project on the definition of an open architecture that allows the creation, sharing and composition of service mashups, seamlessly combining functionality found on the Web, the enterprise or within the 'walled garden' of the telecom operator.

References

[1] M. P. Papazoglou, P. Traverso, S. Dustdar, and F. Leymann, "Service-Oriented Computing: State of the Art and Research Challenges", IEEE Computer Society, 2007, 40(11): 38-45.

[2] Semantic Web, "Providing a common framework that allows data to be shared and reused across application, enterprise, and community boundaries.", http://www.w3.org/2001/sw/ (online) 08.02.2010.

[3] T. Berners-Lee, J. Hendler, and O. Lassila, "The Semantic Web: A New Form Of Web Content That Is Meaningful To Computers Will Unleash A Revolution Of New Possibilities", Journal of the Scientific American, 2001, 284(5): 34-43.

[4] M. Klusch, B. Fries, and K. Sycara, "Automated Semantic Web Service Discovery with OWLS-MX", In Proceedings of 5th International Conference on Autonomous Agents and Multi-Agent Systems, AAMAS 2006.

[5] E. Sirin, B. Parsia, and J. Hendler, "Filtering and Selecting Semantic Web Services with Interactive Composition Techniques", IEEE Intelligent Systems, 19(4):42-49, 2004.

[6] E. Sirin, B. Parsia, D. Wu, J. Hendler, and D. Nau, "HTN planning for web service composition using SHOP2", Journal of Web Semantics, 1(4):377-396, 2004.

[7] M. Klusch, A. Gerber, and M. Schmidt, "Semantic Web Service Composition Planning with OWLS-Xplan", Proceedings of the First International AAAI Fall Symposium on Agents and the Semantic Web, 2005.

[8] S. McIlraith and T. C. Son, "Adapting Golog for Composition of Semantic Web Services", Eighth International Conference on Principles of Knowledge Representation and Reasoning, pp. 482-496, 2002.

[9] A. Lopes and L. Botelho, "Executing Semantic Web Services with a Context-Aware Service Execution Agent", SOCASE 2007 (AAMAS Workshop) held in Honolulu, HW, United States of America, 2007.

[10] A. Hristoskova, B. Volckaert, and F. De Turck, "Dynamic Composition of Semantically Annotated Web Services through QoS-Aware HTN Planning Algorithms", Proceedings of the Fourth International Conference on Internet and Web Applications and Services (ICIW 2009), pp. 377-382.

[11] W. M. P. Van der Aalst, M. Dumas, and A. H. M. ter Hofstede, "Web service composition languages: Old wine in new bottles", Proceeding of the 29th EUROMICRO Conference: New Waves in System Architecture, pp. 298305, 2003.

[12] A. Tsalgatidou, G. Athanasopoulos, M. Pantazoglou, C. Pautasso, T. Heinis, R. Grønmo, H. Hoff, A. J. Berre, M. Glittum, and S. Topouzidou, "Developing scientific workflows from heterogeneous services", ACM Sigmod Record, 35(2):pp. 22-28, 2006.

[13] J. C. Yelmo, R. Trapero, J. M. del Álamo, J. Siene, M. Drewniok, I. Ordás, and K. McCallum, "User-Driven Service Lifecycle Management - Adopting Internet Paradigms in Telecom Services", Lecture Notes in Computer Science, 4749:pp. 342-352, 2009.

[14] L. Del Prete and L. Capra, "MoSCA: seamless execution of mobile composite services", Proceedings of the 7th Workshop on Reflective and Adaptive Middleware, pp. 5-10, 2008.

[15] G. Agre and Z. Marinova, "An INFRAWEBS Approach to Dynamic Composition of Semantic Web Services", Cybernetics and Information Technologies, 7(1):pp. 45-61, 2007.

[16] M. Valle, F. Ramparany, and L. Vercouter, "Dynamic service composition in ambient intelligence environments: a multi-agent approach", Proceeding of the First European Young Researcher Workshop on Service-Oriented Computing, 2005.

[17] H. Pfeffer, "A Underlay System for Enhancing Dynamicity within Web Mashups", International Journal On Advances in Software, 2(1):pp. 63-75, 2009.