Enhancing Web Process Self-awareness with Context-aware Service Composition

Angelo Furno
Department of Engineering
University of Sannio
Benevento, 82100 - Italy
angelo.furno@unisannio.it

***Abstract*—Web processes are usually conceived and executed in highly dynamic environments, including unstable or initially unknown users' requirements, preferences or behaviors, continuously changing properties of interacting devices or physical components, etc. To address this dynamicity, service-based systems should be designed in a flexible way, allowing the possibility to model slightly different versions of the same service in relation to different possible execution contexts. Also, according to the autonomic computing vision, the process should be able to dynamically self-change its structure in order to reach successful termination, even if unexpected anomalies or context changes occur during execution.**

A design approach based on a semantic model for context representation and an extension of the OWL-S ontology are proposed to enrich the expressiveness of each section of a typical OWL-S semantic service description. Context conditions and adaptation rules may be attached to the service description to express how it changes according to specific context configurations. During the planning or re-planning phases of an autonomic workflow, these descriptions can be exploited by a discovery/composition tool to automatically find the atomic or composite services better-tuned to the current requestor's behaviors and to the particular situations of the surrounding environment.

***Keywords*-Service Composition; Context-awareness; Self-awareness; Autonomic Workflow; Semantic Service Design**

I. Introduction and Related Work

Modern systems are required to be able to understand the context in which they are going to be assembled, deployed or executed and configure and change themselves accordingly, even at run-time. In this sense, it appears essential to design them in such a way they can be dynamically forged to users behaviors or expectations and to the evolving states of the surrounding physical environment.

Autonomic Computing (AC) [1] is a natural solution to allow systems to autonomously manage themselves. These systems should not only be able to take automated actions, but also exhibit advanced ability to sense and respond to changes, by means of self-learning and self-managing capabilities.

The focus of this paper is about designing context-aware semantic services, exploitable to automatically generate complex Web processes (i.e. workflows), satisfying users' goals in specific contexts. The composition process is an essential part of the lifecycle of an autonomic workflow [2], since it supports the system ability to autonomously and dynamically change its structure in order to reach its objectives, even if external events change the execution context.

So far, many researchers have investigated different techniques to support the automatic generation of service compositions from a set of published services (domain), given a specific goal to reach (problem) [3], [4]. In many cases, composition techniques and the related tools exploit IOPE (Input, Output, Preconditions and Effects) predicates that characterize structural (WSDL) and semantic (often OWL-S) services descriptions to generate the desired compositions.

The potential of SOA could be better exploited if such ability of building an application by composing existing functions were augmented with the awareness of the surrounding context where the composition takes place. This way, services and related compositions could be forged to adapt their malleable aspects to the specificity of the environment. This impacts the design phase of the services that characterize a given domain, but also the definition of the problem and the goal used to drive a specific composition.

Designing services with this new approach means to extend their semantic descriptions with new attributes and rules that are able to slightly change the structure and behavior of the services according to the needs emerging in the specific context where they will be used.

Context-aware services are semantically described by using an extension of the OWL-S ontology, taking into account the context dimension. The services designed according to the proposed model can be discovered and composed by dynamically tailoring a (possibly distributed) service search space to user needs, preferences and the current situation of the environment where the services have to be executed. In particular, the paper reinforces the notion of autonomic workflow by presenting a model for context representation, its implementation in the OWL language [5], and an extension of OWL-S [6] for allowing context-awareness in semantic service descriptions and their adoption during composition.

The rest of the paper is organized as follows: Section II briefly reports on the notion of autonomic workflow, implemented by our Semantic Autonomic Workflow Engine (SAWE); in Section III, the approach for designing context-aware services is outlined, together with a description of the ontologies supporting it; Section IV gives details on how context-aware descriptions are exploited in SAWE for context-aware automatic service composition, also sketching an example related to a pervasive environment. Section V concludes the paper.

II. Autonomic Workflows

In [2], the notion of Autonomic Workflow (AW) has been introduced to deal with the problem of increasing complexity during the whole lifecycle of traditional workflows (i.e. service compositions). Fig. 1 depicts the lifecycle of an autonomic workflow.

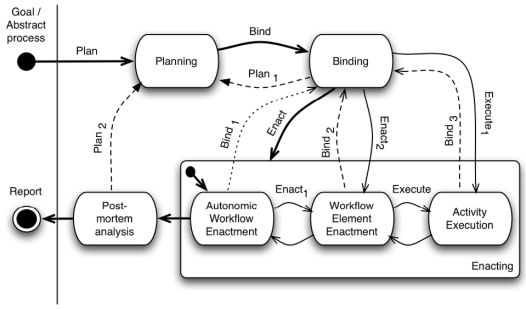

Figure 1. Autonomic Workflow lifecycle [2]

The first two phases (*Planning* and *Binding*) are related to the configuration of an adaptive composite service. An abstract workflow, containing the abstract specification of the activities to be performed, is generated during the *Planning* phase by exploiting the knowledge of the goal objectives (functional requirements), the domain of candidate services for problem resolution (both goal and domain are characterized in a semantic way, by the use of OWL-S profiles), the problem initial state, the domain constraints and the current knowledge of the context. After the abstract process definition, during the *Binding* phase, atomic activities (or workflow elements) are associated to concrete service operations. When an activity has been enacted and bound, it can be executed by the workflow engine.

Together with *the plan-bind-enact-execute* flow (*direct action flow*), the AW lifecycle provides a *reaction flow*, which is intended to implement the adaptive behavior of the autonomic workflow, in case of: (a) low-level malfunctioning (network issues, hardware failures, etc.); (b) logical reasons (agreed quality levels not fulfilled anymore in service provisioning, occurrence of monitored environmental conditions/events, etc.).

When such needs are detected, a set of adaptive strategies may be performed. The simplest one consists in re-binding the running activity ($Bind_3$ edge) in order to find another service implementation compatible with the activity to be executed.

Besides re-binding, the authors consider the possibility to use re-planning ($Plan_1$ edge in the lifecycle figure). When no concrete implementation can be found for executing a specific activity, planning is performed in order to find a new workflow having the same initial and final states (pre/post-conditions) of the activity and the same input/output characterization. If no plan exists, satisfying the activity goals from its initial conditions, re-planning may be used to change the remaining part of the workflow to be executed. This requires to solve a planning problem identified by the original workflow goal objectives and, as the initial state, the current state of the process execution. Planning can also be used during the post-mortem analysis phase of the lifecycle ($Plan_2$ edge) to improve the AW by using the knowledge acquired from the execution.

In the following sections, we focus our attention on the *Planning* and *Re-planning* phases, since they are strictly related to service composition, while the *Binding* phase is considered for the aspects related to the generation of executable workflows.

A. Semantic Autonomic Workflow Engine (SAWE)

SAWE (Semantic and Autonomic Workflow Engine) [2], [7] is the system, developed at the University of Sannio, supporting the notion of autonomic workflow. It exploits meta-information to automatically perform several activities that, in traditional systems, are executed by users and administrators.

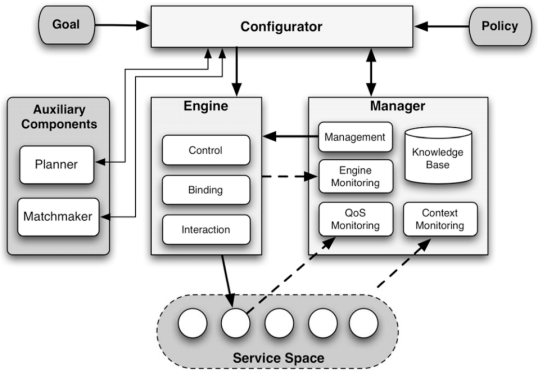

Figure 2. SAWE conceptual architecture [2]

The system comprises three main components: Configurator, Engine and Manager (Fig. 2). The Engine is responsible for process execution. The Manager is in charge of monitoring the environment. The Configurator feeds the engine with workflows or sub-workflows on the basis of inputs derived from users, designers or the manager.

In the rest of the paper, we focus our attention on the configurator (Fig. 3), the sub-system in charge of imple-

menting self-configuration of service compositions through the knowledge coded at design-time or collected at run-time.

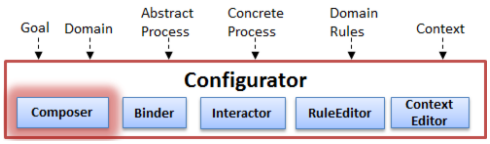

Figure 3. SAWE Configurator

The configurator acts on every aspect of a concrete service composition by changing the overall composition graph to make it executable within the new conditions. To this end, it can: change a link between an activity and a concrete service (re-bind); insert, delete or replace an activity; change the endpoints of a transition; substitute an activity with a sub-process that is able to perform the same actions and to produce the same effects on the external world. Its main component, the context-aware composer, is described in Section IV.

III. Context-aware Semantic Service Design

To support the design of context-aware composite services we consider: a model of the context (sub-section III-A), describing our notion of context in relation to distributed applications and Web services; a corresponding OWL ontology (OWL-Ctx), supporting the description of sets of contexts in specific domains (sub-section III-B); an extension of the OWL-S ontology for services (OWL-SC, sub-section III-C), allowing for the specification of OWL-Ctx-based context adaptation rules.

A. Context Model

By the term *application state* we mean the set of variables and corresponding values that the application is able to access or modify. We distinguish between *internal* and *external application state*.

The *internal state* is the set of variables only visible to the application itself. They are created, used and eventually destroyed by the application (which can be an ensemble of components, having visibility of them) and are not accessible outside the application. Besides *input* and *output* parameters, the application may have predicates to be satisfied in order to execute it (*pre-conditions*) and predicates that are satisfied after the application has been executed (*post-conditions*).

The *external state* is the set of variables accessible also outside the application. They can be read or modified by users, devices or applications other than the one the state is referred to. This set of variables represents the ***context*** in our model: it includes every attribute that characterizes a user and/or the (smart) environment a distributed application interacts with.

An application may exhibit dependencies from the context and in this case we call it *Context-Aware Application* (CAA). In our vision, context dependencies are related to application pre- and post- conditions. A CAA may or may not change the current context during and after its execution. In the latter case, post-conditions include references to context variables. When the properties above are related to Web Services, we speak about *Context-aware Web Services* (CAWS).

Considering the context model defined above, we propose the OWL-Ctx context ontology to extend OWL-S [6] for Web Services design, thus enabling automatic reasoning about context and context-aware composition. By using the OWL-Ctx, a designer may include relevant context-aware information in OWL-S extended semantic service descriptions, supported by our ontology OWL-SC. Context-aware semantic descriptions can be exploited during the service composition process to automatically generate compositions better-tuned to the requestor's behaviors and preferences and to the particular situations of the surrounding environment.

B. OWL-Ctx: an OWL Ontology for Modeling Context

Fig. 4 shows OWL-Ctx, an extensible OWL ontology for describing contexts according to our model.

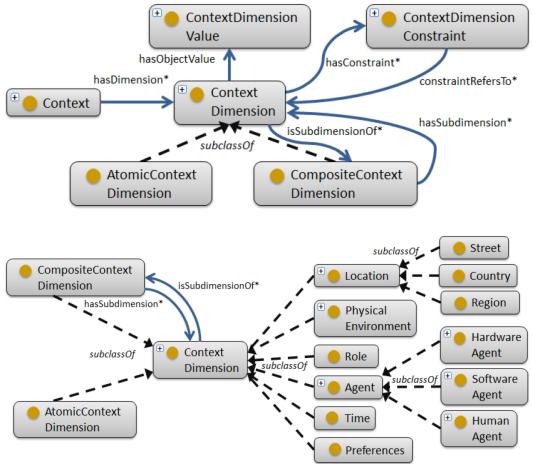

Figure 4. OWL-Ctx: ontology language (a) and (partial) middle ontology (b)

Context, *ContextDimension*, *ContextDimensionValue* and *ContextDimensionConstraint* are the main concepts in this ontology (Fig. 4(a)). The *Context* related to a software system is composed of a set of *ContextDimensions*, each describing one relevant aspect, or dimension, of the environment enclosing the particular software system. Each dimension can be modeled at a different level of abstraction, according to the specific requirements related to the application domain to consider. In this sense, we can distinguish a *ContextDimension* as a composite or an atomic dimension. Differently from an *AtomicContextDimension*, a *CompositeContextDimension* can be further decomposed in one or more sub-dimensions that can be helpful for a better characterization of the associated context aspect.

When modeling contexts in a specific application domain, it is possible to specify the dimensions deriving from the *AtomicContextDimension*, those specializing the *CompositeContextDimension* concept and the relationships among them by means of multiple *hasSubdimension* properties. Also, for each *ContextDimension*, a *ContextDimensionValue* can be defined to represent complex values associated to the specific dimension. A *ContextDimensionConstraint* can be used to specify domain-dependent constraints applying to *ContextDimensions*. In the partial context middle ontology of Fig. 4(b), *Time*, *Agent*, *Location*, *Preferences*, *Role* and *PhysicalEnvironment* describe some domain-independent context dimensions, typically used for specifying context properties with respect to any software application.

C. OWL-SC: An OWL-S Extension for describing Semantic Context-aware Service

The OWL-Ctx ontology is exploited by the OWL-SC ontology, our extension of the OWL-S service ontology for describing context-aware semantic services. The most relevant elements extending OWL-S are reported in Fig. 5(a).

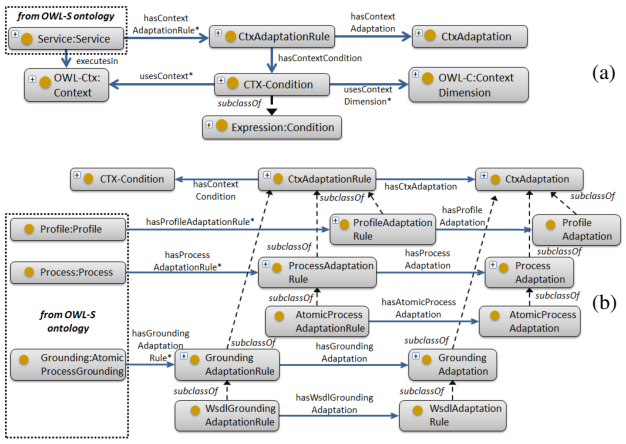

Figure 5. OWL-SC: ontology language (a) and (partial) middle ontology (b)

Each of the three core concepts, provided by the OWL-S ontology to describe a *Service* (*Profile*, *Process* and *Grounding*), can be associated (Fig. 5(b)) to a *ContextAdaptationRule*, by means of the *hasContextAdaptationRule* object property.

A service designer may be aware of a set of contexts or context dimension values, which can be used, during the design phase, to specify conditions for the service to change its basic features (i.e. profile, process or grounding properties). A reference to the current context may be used to relate context conditions to the situation at the moment the extended service description will be analyzed (relationship *executesIn* between *Service* and *Context*) for discovery or composition.

A *ContextAdaptationRule* (and its sub-concepts) is defined by means of a context condition (*Ctx-Condition*) and a context adaptation action (*ContextAdaptation*). It prescribes the context-dependent condition to be satisfied in order to apply the associated adaptation for the specific section of the OWL-S description. If at least one *ProfileAdaptationRule*, *ProcessAdaptationRule* or *GroundingAdaptationRule* is specified in the description, the service is a CAWS.

Currently, we have defined the following context adaptation rules in our language:

- Defaulting an input/output parameter (profile/process section);
- Nulling a parameter, not applicable for a specific context condition (profile/process);
- Changing the owls <*process:parameterType*> of an input/output parameter to a different ontology concept (process);
- Replacing pre-conditions or effects of the basic OWL-S service description (profile/process);
- Changing the *WsdlAtomicProcessGrounding* input/output section of an atomic Process with a new Wsdl MessageMap (grounding);
- Changing the *WsdlAtomicProcessGrounding* section of an atomic Process with a new WSDL operation and/or WSDL portType (grounding).

Context-dependent conditions, currently supported by our OWL-SC ontology, include:

- `current_ctx matches ref_ctx`
- `current_ctx includes "concept hasValue ontology_individual"`
- `current_ctx includes "concept.datatype_property = value"`
- `current_ctx includes "concept.object_property hasValue individual"`

where `current_ctx` is the context reference for the context at the moment in which the composition domain is explored for finding a solution to the submitted problem. Both `current_ctx` and `ref_ctx` specifications can be verified according to an approach like the one used in [8].

Currently, we are using the Semantic Web Rule Language (SWRL) [9] to express the last three kinds of condition. An example of context condition, defined in SWRL and related to a specialization of the OWL-Ctx ontology for multimedia (*Media* context ontology), is given:

```
OWL-Ctx:hasDimension(current_ctx, ?hwAgent) ∧ Media:
HWAgent(?hwAgent) ∧ OWL-Ctx:hasSubdimension(?hwAgent,
    ?role) ∧ OWL-Ctx:Role(?role) ∧ sameAs(?role,
Media:DownloadServer) ∧ OWL-Ctx:hasSubdimension(?hwAgent,
 ?disk) ∧ Media:Disk(?disk) ∧ Media:diskMBSpace(?disk,
   ?avSpace) ∧ swrlb:lowerThanOrEqual(?avSpace, 500)
```

The condition verifies whether one disk of a media server has not enough space ($\leq$ 500MB) to download some file.

IV. Context-aware Service Composition

Context-aware compositions are supported in SAWE by the composer (see Fig. 3), presented in [10], [11] and recently extended to support context-aware semantic descriptions of services and problems. Fig. 6 describes the logic flow and the main components of the Composer.

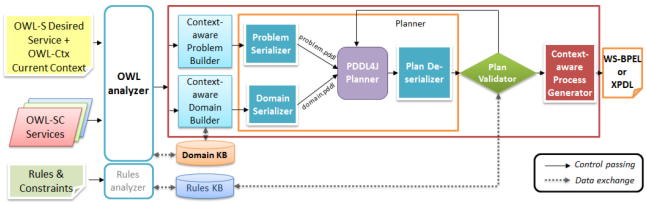

Figure 6. Composer Architecture

A traditional composition process consists of exploring a set of candidate Web Services (the service domain) in order to find a flow of activities (a plan) that, starting from the provided description of the initial state (i.e. predicates true before the task beginning), is able to reach the goal state (i.e. predicates to be true at the end of the service chain).

When performing context-aware composition, the set of semantically described candidate services may include CAWSs (the OWL-SC domain) and is provided as an input to the system (manually or automatically via a matchmaking process), together with the semantic description of the initial state of the environment and the goal. OWL input files are analyzed and used by the **OWL Analyzer** to build the internal representation of the composition problem. Then, context is exploited for preparing the planning phase. The **Context-aware Domain Builder** evaluates the context dependencies in pre-conditions, effects and adaptation rules of the service descriptions, with respect to the current context, and generates a contextualized instance of the domain, suited to be converted into the planner language (PDDL [12]). Similarly, the **Context-aware Problem Builder** augments the problem representation given by the user by injecting into it relevant information from the context, which is derived by the monitoring support available in SAWE.

The **Planner** is the component deputed to the processing of contextualized domain and problem inputs in order to produce a plan of domain actions satisfying the problem. The current implementation of the tool uses PDDL4J [13], based on the Graphplan algorithm [14].

The **Context-aware Process Generator** is in charge of generating a concrete (WS-BPEL) representation of the plan (binding phase), executable on a process engine, by exploiting the available contextualized grounding information.

A. Pervasive Scenario Example

In Fig. 7(a), a user requests an abstract service for retrieving and playing the last episode related to some specified TV-show; subtitles are requested to be superimposed to the video. Besides the desired service IOPE description, our composer is fed with the current context representation as input. This information can be automatically acquired and updated by the system, without being controlled or explicitly known to the service requestor, and is represented according to the OWL-Ctx ontology language. We assume context variations to be reasonably slower than the time required for a typical composition problem to complete. Context changes, following the composition process, may be addressed by re-planning.

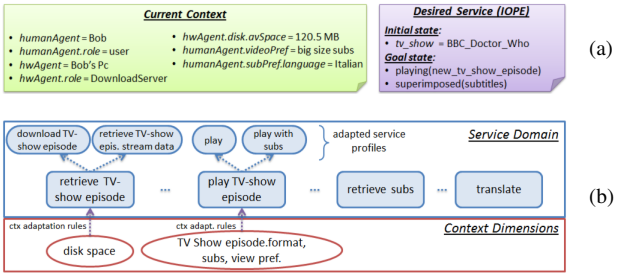

Figure 7. Problem description: current context, desired service IOPE (a) and domain (b)

In Fig. 7(b), a subset of the OWL-SC service domain (Media domain) is shown together with the relevant dependencies on the context, used in the adaptation rules. As an example, the OWL-SC service for retrieving a TV-show episode (*RetrieveTV-showEpisode*) presents a profile having a *TV-show* as an input and a *TVShowEpisode* as an output and providing the effect *newEpisodeAvailable(TVShowEpisode)*. In the basic case, the service also produces the result to download the file containing the last episode of the specified *TV-show* (effect *downloaded(TVShowEpisode)*).

The profile and its service atomic process have been extended with a context adaptation rule, changing the process result in case the download server has not enough space to store the episode file. The context condition used to control this profile/process adaptation in the OWL-SC description is the SWRL condition reported at the end of Section III-C. The adaptation replaces the downloaded result with the effect *streamDataAcquired(TVShowEpisode)*. Also, a different grounding is associated with the same condition in a grounding adaptation rule, because the concrete service for retrieving the stream information is different from the one used to download the file.

Since the disk space amounts to 120.5 MB in the current context of Fig. 7(a), the rule is activated and the adaptation applied by the *Context-aware Domain Builder*. The stream retrieval service (i.e. *RetrieveTVShowEpisodeStreamData* in Fig. 5) is included within the contextualized domain, while the download one is not. When converting to PDDL, the stream retrieval service is the only to appear as a PDDL action, with its inputs, outputs, pre-conditions and effects, thus reducing the actual size of the domain, with benefits

over the composer performances.

After contextualized domain and problem conversion to PDDL, the Graphplan planner generates an abstract PDDL plan. The plan considered in the example contains the sequence of *RetrieveSubtitles* and *Translate* in parallel with *RetrieveTVShowEpisodeStreamData*. The parallel is in sequence with the *PlayWithSubs* service. The translate service is included in the first sequence because the user's preference about subtitle language (Italian) has been added to the goal of the problem by exploiting available context knowledge. Without context-awareness, the composer would have not been able to find such a suitable composition, according to the assumption that *RetrieveSubtitles* only retrieves English subtitles and the user has not explicitly indicated the Italian language preference in its goal. Starting from the plan, the WS-BPEL generator produces the resulting concrete process of Fig. 8, using the available grounding information and context knowledge.

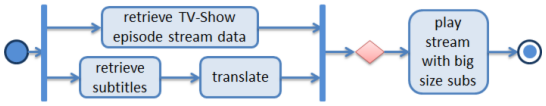

Figure 8. Resulting concrete service composition

V. Conclusion

In the paper, we have presented a model to design context-aware services that can be exploited as a flexible domain to automatically generate context-aware compositions. An OWL ontology for modeling contexts (OWL-Ctx) and an extension of the OWL-S ontology for semantically describing services (OWL-SC) have been proposed. Also, our SAWE context-aware composer, performing planning over contextualized instances of problem and service domain, has been described.

Our approach represents a solution for enhancing the autonomic capabilities of Web Processes, since planning and re-planning may be used for automatically taking into account new user requirements, unexpected exceptions or evolutions of the surrounding environment, occurring during the process execution, by changing its structure accordingly.

Besides extending our implementation of the context-aware composer with support for additional context-dependent conditions and adaptation rules, we are currently investigating the possibility to improve the contextualized expansion of services. Our idea is to perform the expansion directly during planning (i.e. context-aware planning), instead of doing it before (and after) the traditional planning process. This way the state dependencies among domain activities could be evaluated right in the moment when context rules are applied, thus avoiding the possibility to exclude valid compositions, as a consequence of an anticipated (blind) context expansion.

References

[1] J. Kephart and D. Chess, "The vision of autonomic computing," *Computer*, vol. 36, no. 1, pp. 41 – 50, jan 2003.

[2] G. Tretola and E. Zimeo, "Autonomic internet-scale workflows," in *Proceedings of the 3rd International Workshop on Monitoring, Adaptation and Beyond*, ser. MONA '10. New York, NY, USA: ACM, 2010, pp. 48–56. [Online]. Available: http://doi.acm.org/10.1145/1929566.1929573

[3] J. Rao and X. Su, "A survey of automated web service composition methods," in *In Proceedings of the First International Workshop on Semantic Web Services and Web Process Composition, SWSWPC 2004*, 2004, pp. 43–54.

[4] S. Dustdar and W. Schreiner, "A survey on web services composition," *International Journal on Web and Grid Services*, vol. 1, no. 1, pp. 1–30, 2005.

[5] D. L. McGuinness and F. Van Harmelen, "OWL Web Ontology Language overview," *W3C Recommendation*, vol. 10, pp. 1–19, 2004. [Online]. Available: http://www.w3.org/TR/owl-features/

[6] D. Martin, M. Burstein, E. Hobbs, O. Lassila, D. Mcdermott, S. Mcilraith, S. Narayanan, B. Parsia, T. Payne, E. Sirin, N. Srinivasan, and K. Sycara, "OWL-S: Semantic markup for web services," Tech. Rep., Nov. 2004. [Online]. Available: http://www.w3.org/Submission/OWL-S/

[7] M. Polese, G. Tretola, and E. Zimeo, "Self-adaptive management of web processes," in *Web Systems Evolution (WSE), 2010 12th IEEE International Symposium on*, sept. 2010, pp. 33 –42.

[8] C. Bolchini, C. A. Curino, G. Orsi, E. Quintarelli, R. Rossato, F. A. Schreiber, and L. Tanca, "And what can context do for data?" *Commun. ACM*, vol. 52, no. 11, pp. 136–140, Nov. 2009. [Online]. Available: http://doi.acm.org/10.1145/1592761.1592793

[9] I. Horrocks, P. F. Patel-Schneider, H. Boley, S. Tabet, B. Grosof, and M. Dean, "Swrl: A semantic web rule language combining owl and ruleml," World Wide Web Consortium, W3C Member Submission, 2004. [Online]. Available: http://www.w3.org/Submission/SWRL

[10] L. Bevilacqua, A. Furno, V. di Carlo, and E. Zimeo, "A tool for automatic generation of ws-bpel compositions from owl-s described services," in *Software, Knowledge Information, Industrial Management and Applications (SKIMA), 2011 5th International Conference on*, sept. 2011, pp. 1 –8.

[11] L. Bevilacqua, A. Furno, V. di Carlo, and E. Zimeo, "Automatic generation of concrete compositions in adaptive contexts [to appear]," *Mediterranean Journal of Computers and Networks*, 2012.

[12] M. Ghallab, C. K. Isi, S. Penberthy, D. E. Smith, Y. Sun, and D. Weld, "PDDL - the planning domain definition language," CVC TR-98-003/DCS TR-1165, Yale Center for Computational Vision and Control, Tech. Rep., 1998. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.37.212

[13] D. Pellier. (2011) PDDL4J. Last checked: July 2011. [Online]. Available: http://sourceforge.net/projects/pdd4j/

[14] A. L. Blum and M. L. Furst, "Fast planning through planning graph analysis," *Artificial Intelligence*, vol. 90, no. 1, pp. 1636–1642, 1995.