# Context-Aware Design of Semantic Web Services to Improve the Precision of Compositions

Angelo Furno and Eugenio Zimeo

University of Sannio, Department of Engineering, Benevento 82100 Italy
{angelo.furno,eugenio.zimeo}@unisannio.it

**Abstract.** Service-based systems are usually conceived and executed in highly dynamic environments. To support their automatic adaptation to this variability, execution context should be considered as a first-class concept during their design.

This paper proposes a design approach that exploits semantics for modeling contexts and related systems' behaviors. The context model extends the OWL-S ontology to enrich the expressiveness of each section of an OWL-annotated service, by means of conditions and adaptation rules. These additional descriptions can be exploited by a discovery/-composition tool to automatically find the services better-tuned to the requestor's behaviors and the particular situations of the environment.

**Keywords:** Context-aware Computing, Context Modeling, Semantic Web Services, Service Design, Service Discovery, Service Composition.

## 1 Introduction

A characterizing feature of service-based systems is their dynamicity in selecting the functions satisfying user requirements. Service composition plays a fundamental role in this kind of software systems. So far, many researchers have investigated techniques to support automatic generation of service compositions from a set of published services (*domain*), given a goal to reach (*problem*). In many cases, composition techniques and related tools exploit IOPE (*Input*, *Output*, *Preconditions* and *Effects*) predicates that characterize structural (WSDL [5]) and semantic (e.g. OWL-S [1]) service descriptions to generate the compositions.

However, the potential of SOA (Service Oriented Architecture) could be better exploited if such ability of building an application by composing (even on the fly) existing functions were augmented with the awareness of the surrounding context where 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, 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. This is a desirable property in SOA, where

P.C. Vinh et al. (Eds.): ICCASA 2012, LNICST 109, pp. 97–107, 2013.

services, differently from components, should be implemented to be exploited in contexts that could change even during the same execution.

This paper proposes an approach to design context-aware services by extending the OWL-S ontology with the context dimension. Services designed according to the proposed model can be discovered and composed by dynamically tailoring a service search space to the specific user needs or preferences and the current situation of the environment where the services have to be executed. In particular, the paper presents a model for context representation and its implementation in OWL [2] and an extension of OWL-S for allowing context-awareness in semantic service descriptions and their adoption during composition.

The rest of the paper is organized as follows. In Section 2, an application scenario is introduced to highlight the importance of context-awareness in Web Service composition. Section 3 reports on related work about context-awareness. In Section 4, the conceptual model for context representation is described, while in Section 5, the OWL-Ctx ontology based on that model is detailed, together with the extension of the OWL-S ontology for service description. Section 6 describes our context-aware service composition system and discusses it by using an example. Section 7 concludes the paper.

## 2 Motivating Scenario

*Bob likes TV-Shows. He wants his home media server to automatically check the availability of new episodes every day, by connecting to the websites of the major broadcast companies or authorized external media providers. When download is not possible, since the provider only allows for streaming, or disk space is limited, only minimal information about the show (e.g. title, air date, stream URI, etc.) is available for selection, while playing has to be performed by connecting to the stream when required. For each retrieved TV show, if the language is different from Bob's native one, subtitles for that episode should be retrieved. Bob wants his media-player to prompt the list of available new TV-Show episodes, allowing him to choose the one to play. He prefers subtitles to be shown in a large font size. Also, blinds and artificial lights in Bob's TV room should be automatically adjusted to guarantee the best light conditions for watching TV.*

A single service could not satisfy Bob's complex goals, but they may be achieved by composing some of the services available. Current context (e.g. time, location, profile information, etc.) and user preferences (subtitle language or font size, favorite TV-Shows, lighting levels, etc.) have to be taken into account to fully implement the scenario above.

## 3 Related Work

Context-awareness in information systems represents an enabling solution for handling adaptation [6, 14] and it is crucial in service-based systems [13, 18].

In [19], the authors propose a context modeling approach based on ontologies to dynamically handle context types and values. Ontologies enhance the meaning

of user's context values and allow for automatically retrieving relations among them. However, context-aware service composition is not addressed in the paper.

The authors of [11] propose meta-models and an aspect-based pattern for context-awareness of services. A *Context* is a set of *Parameters* and *Entities* that can be structured in *SubContexts*. *ContextAwareServices* are services with associated *ContextViews*, containing adaptation rules and actions. Based on this model and the Aspect-Oriented Paradigm (AOP), context-aware adaptations can be dynamically performed by means of adaptation aspects dynamically woven into the core services. Despite some similarities with our model, their focus is on AOP in service modeling and implementation, instead of semantic composition.

AOP is also exploited in [12] to support context-aware semantic service composition, by weaving context aspects, defined by means of ontology concepts, within plain compositions. Weaving is performed statically, before starting the execution of the main service. However, automatic composition is not considered: plain compositions already exist and are modified by adding context services.

Exploiting context is also essential in pervasive environments [17, 20], where most service discovery or composition requests are implicitly driven by state changes. In [20], the authors introduce a design process and an architecture for building context-aware pervasive service compositions, while, in [17], an approach for personalized service discovery in pervasive environments, based on a multi-dimensional context space (*Hyperspace Analogue to Context*), is proposed. The approach is only limited to event-driven service discovery.

## 4 Context Model

By the term *application state* we mean the set of variables and corresponding values 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 and are not accessible outside it. 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 is 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.

*Context-Aware Application*s may exhibit dependencies from the context. They may interest both application pre- and post- conditions. When the properties above apply to Web Services, we call them *Context-Aware Web Services* (CAWS).

## 5 Context-Aware Semantic Service Design

To support design and composition of context-aware services we propose: an OWL ontology (OWL-Ctx), supporting the description of sets of contexts in

specific domains (sub-section 5.1) and an extension of the OWL-S ontology for services (OWL-SC, sub-section 5.2), allowing for the specification of OWL-Ctx-based context adaptation rules.

Context-aware descriptions can be exploited during service composition to automatically generate compositions better-tuned to the requestor's behaviors and preferences and to the particular situations of the surrounding environment.

### 5.1 OWL-Ctx: An OWL Ontology for Modeling Context

Fig. 1 shows OWL-Ctx, an extensible OWL ontology for describing contexts according to our model (Section 4).

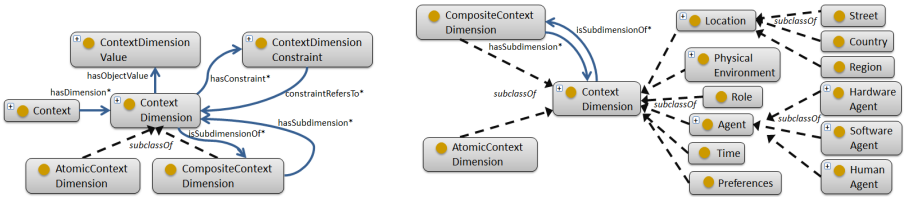


**Fig. 1.** OWL-Ctx: ontology language (a) and (partial) middle ontology (b)

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 different level of abstractions, according to the specific requirements in the considered application domain. In this sense, we distinguish between composite or atomic *ContextDimension*s.

Differently from an *AtomicContextDimension*, a *CompositeContextDimension* can be further decomposed in one or more sub-dimensions (*hasSubdimension* property), helpful for a better characterization of the associated context aspect. A *ContextDimensionValue* can be defined for each *ContextDimension* to describe complex values. Otherwise, if a concrete XML-Schema built-in datatype (e.g. *string*, *int*) is sufficient for representing the context dimension values, the *hasValue* datatype property (not shown in the figure) can be used, by introducing a sub-property having the required xsd type as range. *ContextDimensionConstraint*s allow for specifying constraints applying to *ContextDimension*s.

In the partial middle ontology of Fig. 1(b), *Time*, *Agent*, *Location*, *Preferences*, *Role* and *PhysicalEnvironment* describe generic context dimensions, typically relevant in many application domains.

Considering the scenario illustrated in Section 2, a designer could specify the properties of possible contexts in a Media ontology as follows:

```
<owl:Class rdf:about="&Media;MediaContext">
   <rdfs:subClassOf rdf:resource="&OWL-Ctx;Context"/> ...
   <owl:equivalentClass><owl:Class><owl:intersectionOf rdf:parseType="Collection">
      <rdf:Description rdf:about="&OWL-Ctx;Context"/>
      <owl:Restriction><owl:onProperty rdf:resource="&OWL-Ctx;hasDimension"/>
      <owl:someValuesFrom><owl:Class><owl:unionOf rdf:parseType="Collection">
         <rdf:Description rdf:about="&OWL-Ctx;Agent"/>
         <rdf:Description rdf:about="&Media;MediaContent"/>
         <rdf:Description rdf:about="&OWL-Ctx;PhysicalEnvironment"/> ...
```

*MediaContext* is a specialization of *OWL-Ctx:Context*, with a restriction over the range of the *hasDimension* property to *Agent*, *PhysicalEnvironment* and *MediaContent* (a concept from the Media ontology), since the user (Bob), the devices (Bob's TV, Bob's PC), the media content (TV-Shows) and the location (TV-Room) represent the relevant context dimensions.

### 5.2 OWL-SC: An OWL-S Extension for 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. 2(a).

Each of the three OWL-S core concepts for describing a *Service* (i.e. *Profile*, *Process* and *Grounding*) can be associated (Fig. 2(b)) to a context adaptation rule (*CtxAdaptationRule*), by means of the proper *has*AdaptationRule* object property.

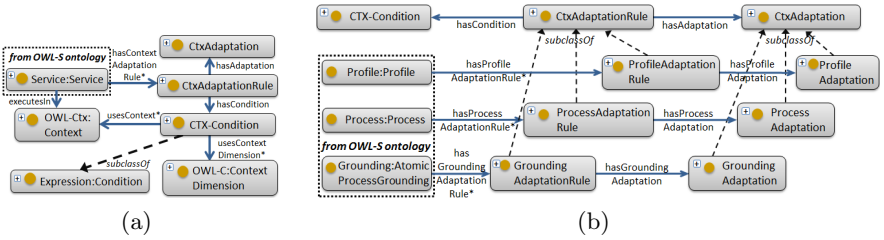


**Fig. 2.** OWL-SC: ontology language (a) and (partial) middle ontology (b)

A service designer may be aware of a set of contexts or context dimension values, which can be used to specify at design-time 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 *CtxAdaptationRule* (and its sub-concepts) is defined by means of a context condition (*CTX-Condition*) and a context adaptation action (*CtxAdaptation*). It prescribes the context-dependent condition to be satisfied in order to apply the associated adaptation to 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.

The main context adaptations over OWL-S sections currently supported are:

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

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

- `current_ctx matches ref_ctx`
- `current_ctx includes "concept hasValue 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 [9].

Currently, we are using the Semantic Web Rule Language [3] (SWRL) to express the last three kinds of condition. An example of SWRL context condition, related to the *Media* context specialization previously defined, 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 the disk of a media server has not enough space ($\leq$ 500MB) to download some file.

## 6 Context-Aware Service Composition

Context-aware compositions are supported by a semi-automatic tool, the composer (Fig. 3), presented in [7] and extended in order to support context-aware semantic descriptions of services and problems. Part of a more complex system, namely Semantic Autonomic Workflow Engine (SAWE) [16], the composer can be used either for the initial definition (plan) of a service composition or for re-planning an already defined composition, to be changed completely or in part in order to react to external events.

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

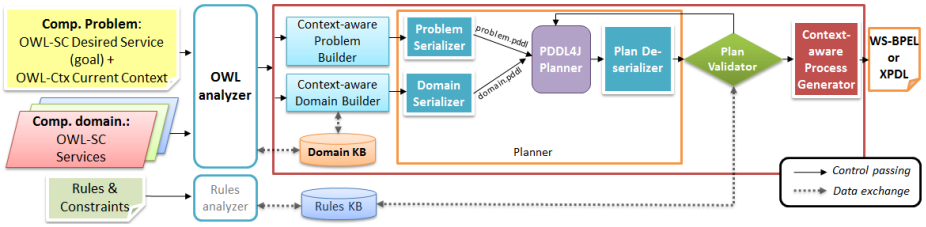


**Fig. 3.** Composer Architecture

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 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 [10]). Similarly, the *Context-aware Problem Builder* augments the problem representation given by the user, by injecting relevant context information derived from our SAWE monitoring support.

The *Planner* is the component deputed to the processing of contextualized domain and problem for producing an action plan satisfying the problem. The current implementation of the tool uses PDDL4J [15], based on the Graphplan algorithm [8]. The *Context-aware Process Generator* generates a concrete (WS-BPEL [4]) representation of the plan (binding phase), executable on a process engine, by exploiting the available contextualized grounding information.

It is worth to note that, in general, the context can be extremely dynamic, evolving so rapidly that it could be not possible to complete the composition process without taking into account the mutated environmental conditions. However, in the present work, we assume context variations to be reasonably slower than the time required for a typical composition problem to complete. By this assumption, composition may be completed in relation to the contextualized domain and problem, being the final composition still meaningful. Also, in the current SAWE implementation, context changes, following the composition process, may be addressed by re-planning, according to a reactive approach to adaptation. Alternatively, a proactive strategy, based on probabilistic techniques (e.g. Bayesian networks, Markov processes), might be used for learning context transition probabilities from past experience. This way, alternative compositions could be computed in advance for highly probable context transitions, minimizing (with respect to reactive strategies) the time the running system is unavailable when context actually changes.

### 6.1 Example

A synthetic version of the scenario described in Section 2 has been considered in order to evaluate our context-aware composer.

In Fig. 4(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 OWL-Ctx current context representation as input, which can be automatically acquired and updated by our SAWE system. In Fig. 4(b), a subset of the OWL-SC service domain is shown together with the relevant dependencies on the context, used in the adaptation rules.

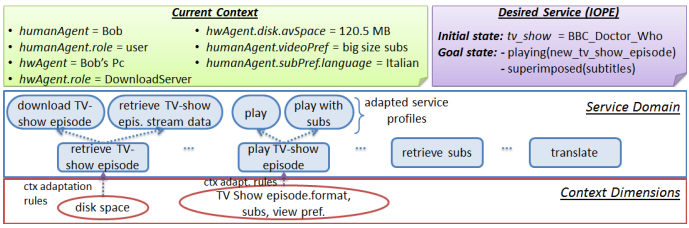


**Fig. 4.** Problem description: current context, goal IOPE (a) and domain (b)

The OWL-SC service for retrieving a TV-Show episode provides the effect *newEpisodeAvailable(TV-ShowEpisode)* and, in the basic case, the download of the last *TV-Show* episode file (effect *downloaded(TV-ShowEpisode)*).

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 5.2. The adaptation replaces the downloaded result with the effect *streamDataAcquired(TV-ShowEpisode)*. The same condition also controls a grounding adaptation rule, changing the concrete service from the one used to download the file to the one for retrieving the stream information.

Since the disk space amounts to 120.5 MB in the current context of Fig. 4(a), the rule is activated and the adaptation applied by the *Context-aware Domain Builder*. The stream retrieval service (i.e. *RetrieveTV-ShowEpisodeStreamData* in Fig. 4(b)) 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 domain action, thus reducing the actual size of the domain, with benefits over composition performances.

The *PlayTV-ShowEpisode* service contains pre-condition *newEpisodeAvailable(TV-ShowEpisode)* and parameter *TV-ShowEpisode* as an input. The effect *playing(TV-ShowEpisode)* is provided. A context-dependent adaptation is included in the OWL-SC description for the grounding section: depending on the *streamDataAcquired(TV-ShowEpisode)* or *downloaded(TV-ShowEpisode)* condition a different concrete service will be grounded to the service process. The

grounding adaptation rule does not generate different services in the domain. Instead, two groundings are stored for being later used by the Process Generator when the final WS-BPEL process has to be created. The profile and atomic process sections of the *PlayTV-ShowEpisode* service contains an adaptation rule for subtitle rendering. Depending on the availability of subtitles, they have to be considered as an additional input and their superimposition as an additional effect. Since at the moment of domain building it is not known whether the subtitles are going to be available or not, the Context-aware Domain Builder includes both the two services to the domain (the one including subtitles as input and the one not including them). Also, since view preferences are another input of the service and part of the context, the Context-aware Problem Builder expands the set of known inputs for problem solution with them.

The *RetrieveSubtitles* service only downloads English subtitles and a *Translate* service to the user's language (i.e. Italian) is also available in the domain.

After contextualized domain and problem conversion to PDDL, the Graphplan planner generates an abstract PDDL plan, containing the sequence of *RetrieveSubtitles* and *Translate* in parallel with *RetrieveTV-ShowEpisodeStreamData*. The parallel is in sequence with the *PlayWithSubs* service. The translate service appears 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 his/her goals.

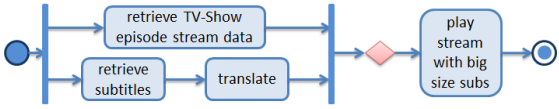


**Fig. 5.** Resulting concrete service composition

The WS-BPEL generator produces the resulting concrete process of Fig. 5 from the abstract plan, using the available grounding and context knowledge.

## 7 Conclusion

The paper has presented a model to design context-aware services that can be exploited as a flexible domain to automatically generate context-aware compositions by means of a specific tool. Besides extending the services, the tool exploits an extension of the problem definition that includes also the context representation (defined by designers or implicitly inferred by the system).

The analysis conducted during the definition of the model has highlighted a further problem to address: by exploiting context for problem expansion and evaluation of adaptation rules before planning, valid composite solutions might

be indirectly excluded during domain construction if, for an abstract service, more than one post-condition may fit the context, but only one of them satisfies the pre-conditions of the next services (e.g. there is disk space for retrieving a HD file but the player is not able to play it). This requires the contextualized expansion of services to be performed during planning to take into account the state dependencies when context rules are applied.

## References

1. OWL-S: Semantic Markup for Web Services, http://www.w3.org/Submission/OWL-S/
2. OWL: Web Ontology Language Overview, http://www.w3.org/TR/owl-features/
3. SWRL: A Semantic Web Rule Language Combining OWL and RuleML, http://www.w3.org/Submission/SWRL
4. Web Services Business Process Execution Language Version (WS-BPEL) 2.0., http://docs.oasis-open.org/wsbpel/2.0/wsbpel-v2.0.html
5. Web Services Description Language (WSDL) 1.1., http://www.w3.org/TR/wsdl
6. Abowd, G.D., Dey, A.K., Brown, P.J., Davies, N., Smith, M., Steggles, P.: Towards a Better Understanding of Context and Context-Awareness. In: Gellersen, H.-W. (ed.) HUC 1999. LNCS, vol. 1707, pp. 304–307. Springer, Heidelberg (1999)
7. Bevilacqua, L., Furno, A., di Carlo, V., Zimeo, E.: A tool for automatic generation of ws-bpel compositions from owl-s described services. In: 2011 5th International Conference on Software, Knowledge Information, Industrial Management and Applications (SKIMA), pp. 1–8 (September 2011)
8. Blum, A.L., Furst, M.L.: Fast planning through planning graph analysis. Artificial Intelligence 90(1), 1636–1642 (1995)
9. Bolchini, C., Curino, C.A., Orsi, G., Quintarelli, E., Rossato, R., Schreiber, F.A., Tanca, L.: And what can context do for data? Commun. ACM 52(11), 136–140 (2009)
10. Ghallab, M., Isi, C.K., Penberthy, S., Smith, D.E., Sun, Y., Weld, D.: PDDL - the planning domain definition language. Tech. rep., CVC TR-98-003/DCS TR-1165, Yale Center for Computational Vision and Control (1998)
11. Hafiddi, H., Baidouri, H., Nassar, M., Kriouile, A.: An aspect based pattern for context-awareness of services. International Journal of Computer Science and Network Security 12(1), 71–78 (2012)
12. Li, L., Liu, D., Bouguettaya, A.: Semantic based aspect-oriented programming for context-aware web service composition. Information Systems 36(3), 551–564 (2011)
13. Maamar, Z., Benslimane, D., Narendra, N.C.: What can context do for web services? Commun. ACM 49(12), 98–103 (2006)
14. Pascoe, J.: Adding generic contextual capabilities to wearable computers. In: Second International Symposium on Wearable Computers, Digest of Papers, pp. 92–99 (October 1998)
15. Pellier, D.: PDDL4J (2011), http://sourceforge.net/projects/pdd4j/
16. Polese, M., Tretola, G., Zimeo, E.: Self-adaptive management of web processes. In: 2010 12th IEEE International Symposium on Web Systems Evolution (WSE), pp. 33–42 (September 2010)

17. Rasch, K., Li, F., Sehic, S., Ayani, R., Dustdar, S.: Context-driven personalized service discovery in pervasive environments. World Wide Web 14, 295–319 (2011)
18. Truong, H.L., Dustdar, S.: A survey on context-aware web service systems. International Journal of Web Information Systems 5(1), 5–31 (2009)
19. Xiao, H., Zou, Y., Ng, J., Nigul, L.: An approach for context-aware service discovery and recommendation. In: 2010 IEEE International Conference on Web Services (ICWS), pp. 163–170 (July 2010)
20. Zhou, J., Gilman, E., Palola, J., Riekki, J., Ylianttila, M., Sun, J.: Context-aware pervasive service composition and its implementation. Personal Ubiquitous Comput. 15(3), 291–303 (2011)