Building a Context World for Dynamic Service Composition

Lian Yu, Arne Glenstrup[1], Shuang Su, Yang Zhang
School of Software and Electronics, Peking University, Beijing, 102600, PRC
lianyu@ss.pku.edu.cn, panic@itu.dk
[1]IT University of Copenhagen, Denmark

Abstract

Dynamic service composition requires responding and adapting to changes in the computing environment when orchestrating existing services into one or more new services that fit better to a composite application. This paper abstracts the changes of the environment as a context world to store the physical contexts of the computing environment, user profiles and computed results of services as well. We use ontology techniques to model the domain concepts of application contexts. Context Condition/Effect Description Language is designed to describe the dynamic semantics of the requirements and capabilities of goals and services in a concise and editable manner. Goal-driven and planning techniques are used to dynamically implement the service composition according to the domain knowledge and facts in the context world.

Keywords: Context world, Ontology modeling, Context condition/effect description language, Reasoning, Goal driven planning

1. Introduction

The goal of dynamic service composition is to realize flexible and adaptable applications by properly selecting and combining services based on user requests and environment contexts. Dynamic service composition is even able to bring forth a number of useful applications at runtime that are not envisioned at design time. Therefore, dynamic service composition is suitable for end-user applications in changing environments where physical contexts are dynamic and expected users may vary.

Existing dynamic service composition systems, such as eFlow, ICARIS, SWORD, SHOP2, request the user's requests in strict syntax formats. This makes them difficult to generate and understand. Fujii and Suda propose a semantics-based service composition architecture, which obtains the semantics of the service requested in an intuitive form, and dynamically composes the requested service based on the semantics of the service. They did not take into consideration the contexts, which reflect the situations of the computing environment and are the important factors to enable dynamic service composition.

This paper proposes to build a context world, where the formal semantics of the goal and services and the facts are stored and updated dynamically. A context-aware system can both sense and react with the help of sensors and actuators, providing services based on context information, such as temperature, location, weather, user profile, velocity and state of an entity. For the purpose of making composed *services in context* that automatically adapt to changing contexts, it is also imperative to take context information into consideration in service composition.

Context information considered in this paper includes physical context information, user context information and computing context information. Physical context information refers to the physical information that can be sensed by electronic devices, including light, noise, weather, location, and status. User context information refers to personal profile, including name, age, sex, preference, etc. Computing context information includes service availability and service computing results. The context information above can be obtained and adopted in the process of dynamic service composition.

The system architecture is depicted in Figure 1, which includes three layers: Composition/Discovery and Runner Layer, Context Reasoner Layer and Service Layer.

Service Layer: This layer is responsible for gathering context information and serving end-users. In this system, the physical context information is obtained by Physical Context Acquisition Services using sensors, e.g., infrared sensors, RFID systems, and microphones. Physical Context Acquisition Services monitor the context data in the physical data buffer, and filter, fuse and format them into unified context information that can be imported into the context world for planning. User Information Gathering Services gather the profile information of users directly by user interface. Functional Services refer to all these services that work for end-users. The semantics descriptions of all of the services are the foundation of planning.

[1] The research is supported by Danish Strategic Research Council (No 2106-08-0046) and by the National Science Foundation of China (No. 60973001).

Composition/Discovery and Runner Layer: This layer is responsible for planning a service path (i.e., a list of services) and running services. The description of goals and services is interpreted and stored in a repository. The planning process is triggered by a user requirement from a user interface. During the planning process, context Reasoner modules and relevant services will be invoked. The planner generates a service path according to the relevant goal and service definition as well as the current context for an end-user, and the runner is responsible for executing each service in the path and interacting with the end-user by communication devices.

Context Reasoner Layer: The context Reasoner Layer consists of reasoning related modules. The domain concept model can be constructed according to the Context Domain Description script, and the Context World can be instantiated as a facts' container. The modules of the context condition/effect builders, and the context condition/effect executors are responsible for interpreting or executing a context condition and effect script, which is written in a goal description or service description. All of the formatted context information can be imported into the context world by the context world updater. During the planning and running process, the instance set of each goal node is organized and manipulated by the Process Handler module, and services are selected by the Service Selector module. The contribution of this paper is not the reasoning engine, but is the contribution of building Context World based on top of the engine. Pellet (http://clarkparsia.com/pellet/), an open source of OWL 2 Reasoner, is utilized in our implementation.

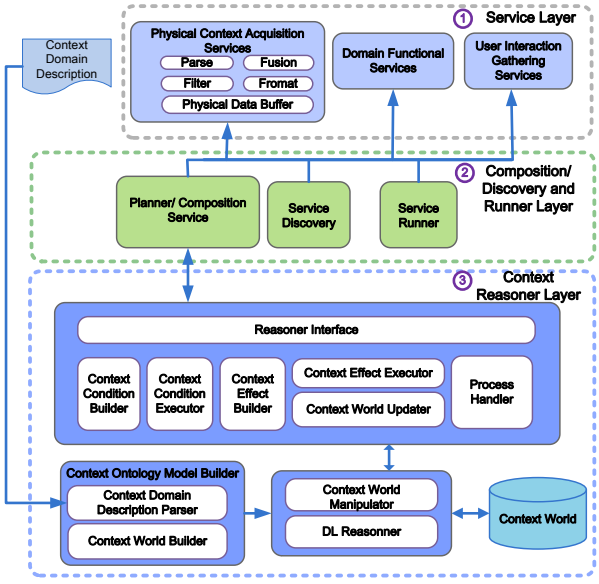

Figure 1 Architecture of CASC System

The rest of the paper is organized as follows. Section 2 presents the context modeling. Section 3 describes the context condition and context effect. Section 4 presents a survey on the related work. Section 5 concludes the paper.

2. Context Modeling

Before planning or reasoning can be performed, the domain context needs to be formally modeled. The formal model provides a definition of structures and meanings of domain concepts, based on which the information can be formatted and imported to represent the facts.

2.1. Context Ontology Meta Model

The context ontology meta-model is domain independent with several concepts defined in Figure 2.

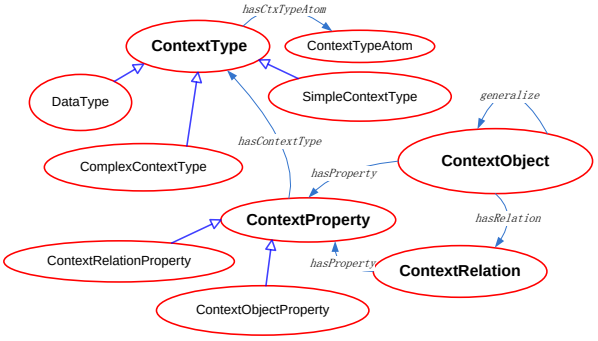

Figure 2 Context Ontology Meta Model

Context Object is defined as $o=(\iota_o, \Lambda, \Gamma)$, where ι_o is the context object name, Λ is a set of context properties, and Γ is a set of type context relations.

Context Property is defined as $p=(\iota_p, o, \tau, v)$, where ι_p is the context property name, o is the context object which the context property belongs to, τ is the context type of the property value, and v is the value of the context property. Context relation property and context object property are two subclasses of context property.

Context Relation is defined as $r=(\iota_r, s_o, t_o, \Lambda)$, where ι_r is context relation name, s_o is the source context object, t_o is the target context object, and Λ is a set of context properties.

Context Type is defined as $\tau=(\iota_\tau, \Xi)$, where ι_τ is the context type name and Ξ is a set of context type atom. There are three types of context type: simple context type, complex context type, and data type. The simple context type and complex context type are generally called as ontology context type, which contains the semantics of the data value.

2.2. Context World

The Context World is the knowledge base implementation of a specific domain, which contains domain concepts and facts. This paper uses the Context Domain Description Language to build a context world.

The Context Domain Description Language is a simple and easy-to-understand script language, which uses the constructs of Context Ontology Meta Models to define the

domain concepts.

(1) Context Domain Description Language

Context Domain Description Language is a simple and easy-to-understand script language, which using the constructs of Context Ontology Meta Model to define the domain concepts. Figure 3 shows an example of a context Domain description script.

```
ContextWorld{
    ContextType {
        SimpleContextType {
            Direction {
                Atom { North; Sourth; East; West; }
                General { Northeast=North&East; } } } }
        ComplexContextType {
            DrugEffect {
                Atom{ analgesic, antibiotics, painkiller}
                General { CA=analgesic&antibiotics;
                          SA=antibiotics|!painkiller; } } }
        DataType {
            LocationPoint {
            Definition { com.ss.pku.datatype.LocationPoint; } } }
        Fun {
            outer { Definition { com.ss.pku.CoorCom.outer} } }
    ContextObject {
        Patient {
            Generalize{Person;}
            ContextProperty {
                name:string;
                sec:Security; }
            ContextRelation {
                locWith:InnerArea (some=true,inverse=true,
                    trsitive=true,symmetric = true) {
                    direct:Direction; } } }
    }
}
```

Figure 3 Context Domain Description Script

The language is defined with B.N.F. described in Table 1.

Table 1 Context Domain Description Language Definition

CWODL	:=	{ "ContextType" "{" CTXTYPEDEF "}" }
		{ "ContextObject" "{" CTXOBJDEF "}" }
CTXTYPEDEF	:=	{ "SimpleContextType" "{" STYPEDEF "}" }
	|	{ "ComplexContexType" "{" CTYPEDEF "}" }
	|	{ "DataType" "{" DTYPEDEF "}" }
	|	{ "Fun" "{" FUNNAME "{ Definition" "{" CLASSNAME "}}"}
STYPEDEF	:=	TYPENAME "{" "Atom" "{" { ATOMVALUE ";" } "}"
		"General" "{" GENERALEXP ";" } "}"
CTYPEDEF	:=	TYPENAME "{" "Atom" "{" { ATOMVALUE ";" } "}"
		"General" "{" TYPENAME "=" GENERALEXP ";" } "}"
GENERALEXP	:=	GENERALEXP OPT TYPEEXP
OPT	:=	("&" | "|")
TYPEEXP	:=	["!"] TYPENAME
CTXOBJDEF	:=	{ OBJNAME "{" OBJGEN { CTXPPT } { CTXREL } "}" }
OBJGEN	:	"Generalize {" { OBJNAME ";" } "}"
CTXPPT	:=	"ContextProperty" "{" { PPTDECLARE ";" } "}"
CTXREL	:=	"ContextRelation" "{" { RELDECLARE } "}"
PPTDECLARE	:=	PPTNAME ":" CTXTYPENAME
RELDECLARE	:=	RELNAME ":" OBJNAME "{" { PPTDECLARE ";" }

(2) Context World Construction

The Context World can be constructed with a Context World Builder, which is depicted in Figure 4.

Context Domain Description Parser: The context domain description is firstly parsed by the document parser. The context object, types, properties and relations are realized and relevant data structures are constructed, and finally a context world model is constructed.

Context World Model: The Context World Model is the domain concept model with the domain concepts defined according to the context domain description script. The Context World Model can be instantiated to one or more context world.

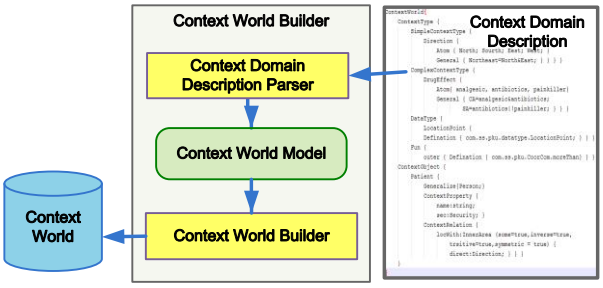

Figure 4 Context World Builder

Context World Builder: The Context World Builder instantiates the context world model to a concrete context world. The Description Logic Reasoner is used to create the ontology classes and properties according to the definitions in the context world model. With the Context World built, the facts can be added and the reasoning can be performed during the planning and running process.

For instance, a context object *Drug* has an instance *d1*, and the context object *Drug* has a context property *effect* defined in the context domain description script, and the property *effect* has context type *DrugEffect*. The concepts and facts in Context World are depicted in Figure 5. All of the ontology classes (including *ContextProperty*, *DateTime*, *Drug*, *DrugEffect* and its atoms *Analgesic* and *Painkiller*), and ontology properties (including *objppt_Drug_effect* and *objppt_Durg_effect_inv*) are created by the Context World Builder according to the definitions in the Context Domain Description script. At the planning time or running time, relevant instances are created by Context World Updater to represent facts. When an instance *d1* is created, an ontology individual *d1_effect* and *d1_effect_value* are created by Context World Updater to represent the context property *effect* and its value.

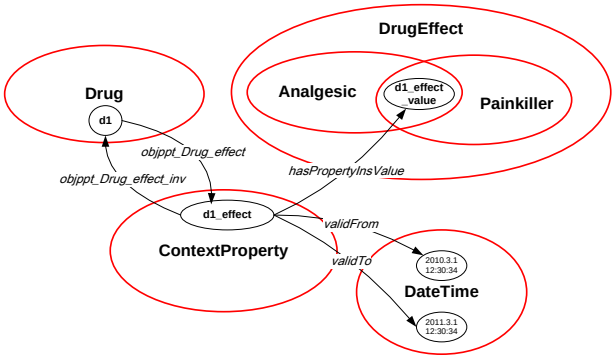

Figure 5 Context Property in the Context World

Another illustration is depicted in Figure 6. The context

objects *Patient* and *InnerArea* have context relation *Patient_LocWith*. When their instances *p1* and *r1* are asserted to be related within *Patient_LocWith*, an ontology individual *p1_locWith_1* is created to represent the relation instance, and ontology individual *p1_locWith_1_direct* and *p1_locWith_1_value* are created to represent the context property *direct* and its value.

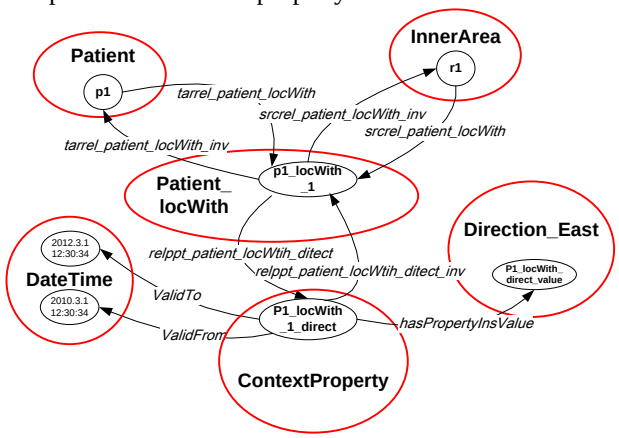

Figure 6 Context Relation in the Context World

Note that in the examples of the context world, both drug and patient are considered as contexts because they are interacting and impacting with each other.

2.3. Planning, Composition and Running with Context World

Having the Context World constructed, planning/composition and running can be performed. This paper proposes a Context Condition to describe a domain constraint or selective condition, and proposes Context Effect to describe the service performance to the context. Context Condition and Context Effect is the easy-to-use script language, which can be edited and modified during domain analysis. In the planning process, context condition is commanded to be executed and the result can be obtained according to the facts in a context world, and context effect is commanded to update the facts in the context world. The updated context world will affect the result obtained by the context condition. Context condition and context effect are organized in the goal description script and service description script, and the planning result will be gotten with the execution of them.

3. Context Condition and Context Effect

This section will describe the context condition and context effect, including the definition and the interpretation process and execution process.

3.1 Context Condition

The Context Condition consists of variable declarations and context condition atoms. Each condition atom has one or more decision operation. Firstly, we will introduce the Decision Operation with which the context condition can be defined. Table 2 shows the semantics of Decision Operation.

Table 2 Semantics of Decision Operation

Expression	Semantics
x :- y	Decide whether x belongs to y. The operator can only be used when x is the variable or constant with the type of Ontology Context Type. The result of the execution of the decision expression could be true, unknown or false.
x :include y	Decide whether x includes y.
x :exact y	Decide whether x exactly equals y.
x :exist	Decide whether x exists. There are three type the existence decision: Target exist: decide whether the target of a context object instance within a context relation exists; Relation exist: decide whether the relation of two context object instance exists; Property exists: decide whether the property value is not unknown.
x :more y	Decide whether x is more than y. x, y should be context property variable/constant or context property value variable/constant. The context type of x, y should be data type. If the value of x or y is unknown, the result is unknown.
x :less y	Decide whether x is less than y.
x:moreEqualy	Decide whether x is more or equals than y.
x :lessEqual y	Decide whether x is less or equals than y.
x :equal y	Decide whether x equals than y.
x :notEqual y	Decide whether x does not equals than y.

The decision operation can be added one or more prefixes to modify its semantics. Prefix includes “#t”, “#f”, “#u” and “#r”. “#t” means that unknown should be translated to false. “#f” means that unknown should be translated to true. “#u” means that unknown should be translated to true, and true should be translated to false. This prefix is commonly used to decide whether the result of the expression is unknown. “#r” means that true should be translated to false, and false should be translated to true. Table 3 shows the B.N.F of Context Condition.

Table 3 Definition of the Context Condition Language

CDTNBUNDLE	:=	{ CDTNEXP “;’ }
CDTNEXP	:=	CDTNOPD DECISIONOPT [CDTNOPD]
	|	“:” FUNNAME “(“ { CDTNOPD “,” } “)”
CDTNOPD	:=	(CDTNATOM | CTXPPTVALCONSTEXP | CTXPPTVALVAR)
CTXPPTVALCONSTEXP	:=	CTXPPTVALCONSTEXP OPT TYPEEXP
OPT	:=	(“and” | “or”)
TYPEEXP	:=	[“not”] TYPENAME
CTXOBJVAR	:=	(CTXOBJVARNAME | CTXOBJVARDECLARE)
CTXOBJVARDECLARE	:=	“(“ “?” CTXOBJVARNAME “:” CTXOBJ “)”
CTXOBJPAIR	:=	“<” CTXOBJVAR “,” CTXOBJVAR “>”
CTXRELEXP	:=	“.” CTXRELNAME [“[“ { CTXRELRESTRICT “,” } “]”]
CTXRELRESTRICT	:=	“.” CTXPPTNAME DECISIONOPT CDTNOPD
	|	“:” FUNNAME “(“ { (CDTNOPD | “.” CTXPPTNAME) “,” } “)”

An example of a context condition is shown as follows. The declaration block defines the variables to which certain instants will be assigned before the execution of the context condition. The property of both context object

and context relation can be visited with the pattern of ".<property name>". The decision operator can be used to express judgments with property values. For instants, "pa1.name : equals 'Bob'".

```
declare{    pa1:Patient; do1:Doctor;    }
condition{   <pa1,do1>.medicalCare :exist;
<pa1,do1>.medicalCare.medRegisterID :equal 'A102';
<do1,(?rfid:RFIDItem)>.hasRFID.status :-
RFIDAssignStatus.Expired;
pa1.medicalCare.workingRoom.locWith.location :equal 'Q141';
pa1.medicalCare[.medRegisterID:equal
'A102', .status:-MEDCareStatus.Available].workingRoom
:exist;    }
```

A context relation is expressed with a pair of context object variable names. A user can also use the pattern of "<context object name>.<context relation name>" to express the target context object instances of the relation. Then the relation names can be linked as a sequence to get the target context object instances that do not have a direct relation with the source context object instance. Figure 7 shows the process of interpretation and execution of the context condition.

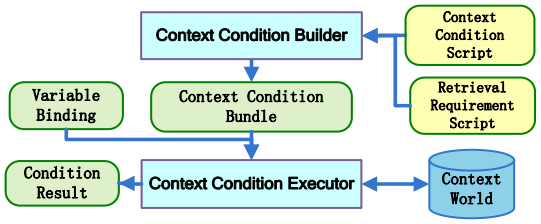

Figure 7 Interpretation and Execution of Context Condition

Context Condition Script: The text script of a group of context conditions.

Retrieval Requirement: The text script of a group of retrieval requirements.

Context Condition Builder: This module loads and verifies the text scripts, and interprets them according to the context model. Finally the context condition bundles are generated.

Context Condition Bundle: The inner structure of a context condition. A context condition bundle can be saved in a repository.

Variable Binding: Before executing context conditions, the pre-binding variables of the context condition need to be assigned with instances. Variable binding is a set of variable-instance pairs, according to which the corresponding variables in the context condition will be assigned with the instance.

Context Condition Executor: When a context condition bundle is commanded to be executed, the Context condition executor will load the context condition bundle and assign its pre-binding variables according to variable bindings. The Pellet [5] ABox Query Engine and Pellet Service Programming Interface (SPI) are used to perform the execution.

Context Condition Result: The context condition result includes the result status (true, unknown or false), the accepting binding set, pending binding set and the rejected binding set.

3.2 Context Effect

An example of a context effect script is shown as follows.

```
declare{    pa1:Patient; do1:Doctor; docName:XSDString; }
effect{ pa1 <<; do1<<; }
effect{ pa1.name <<'Smith'; do1.name<<docName;
pa1.medicalCare<< do1;
<pa1,do1>.medicalCare.status<<MEDCareStatus.Available;}
```

In the declaration block, all of the variables appearing in the effect expressions are declared. The effect expressions are written in the effect blocks. The instance binding of the variable can come from the process structure of goals and services in the CASC system. At execution time, the effect atoms will be executed with the sequence appearing in the script. During the execution of an effect block, the variable binding will not be modified; instead the variable binding modification will be performed just after each effect block is executed. Figure 8 shows the execution process and corresponding modules of context effects.

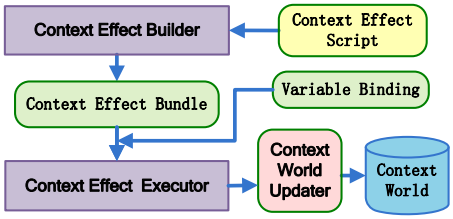

Figure 8 Execution Process

Context Effect Script: The text script of a group of context effects.

Context Effect Builder: The effect script is interpreted into the structure called effect bundle by the effect builder module, and the effect bundle can be stored in a certain repository. In the interpretation process, the effect script is realized and verified according to context world model, and an update command list will be constructed.

Context Effect Bundle: The inner structure of a context effect. A context effect bundle contains the declaration set and an update command list generated by the Context Effect Builder.

Variable Binding: The variable binding is a set of variable-instance pair, according to which the corresponding variables in the context condition will be assigned with instances.

Context Effect Executor: When the effect execution command arrives, the effect bundle will be loaded and executed by the Context Effect Executor. The variables in the context effect bundle will be assigned according to the variable binding, and the executor will invoke the context world updater model to execute the update commands

sequentially.

Context World Updater is responsible for updating the context world according to the update command, which consists of a batch of update operations. Update operations include creating or removing of instances of context objects or context relations, and modifying context property values.

In the planning process, sometimes it is necessary to rollback the update, which means undoing update operations and returning to previous states of the context world. This system adapts the Logging Update Strategy, recording all of the update operations in logs, and using a history repository to record history values. If the rollback command arrives, the Context World Updater reads logs and performs related reverse operations. For the remove or value-modify operation, the old value will be read from the history repository.

4. Related Work

Gu et al. [1] propose a context model based on ontology using Web Ontology Language to address issues including semantic context representation, reasoning, knowledge sharing, classification, dependency and quality. For rapid prototyping, a SOCAM architecture is established, which has the ability to adopt various context reasoning mechanisms for various contexts.

Maamar et al. [2] use concepts of agent and context to reduce the complexity of service composition, where agent represents an autonomous entity that acts on behalf of users. Their work mainly focuses on the agent conversation mechanism to decide whether services will participate in composition.

Situation calculus is used by [3] [4] for service formalizing and automatical composition. Paper [3] adopts and extends Golog to formalize service reasoning about action. Besides using situation calculus to model dynamic systems, paper [4] also establishes some basic situation calculus planning systems. Yet situation calculus is based on First-order logic, where undecidability problem limits the reasoning for services.

The theory of Dynamic Description Logic (DDL) is established in [6] to perform service composition in logical level, where the syntax and semantic is extended from Description Logic. An action symbol is added, and the concept of possible world is introduced to give the semantic model to interpret action. An extended tableaux algorithm is used for satisfiability-checking of a formula. In their further work [7] , a service formalizing method is proposed based on DDL, and the reasoning of action is used for service matching and composition. Yet no detailed plan strategy is presented, although it is mentioned in their work.

CB-SeC [8] [9] is a framework to put forward for context-based service discovery and composition, where services are incrementally filtered and ranked according to the evaluation of contextual information for providing best composite services. The framework is layered by tasks and functionalities for flexibility and adaptability, but it is a preliminary framework without details in each part.

5. Conclusion and Future Work

This paper presents the design and implementation of a context world to support dynamic service composition. The Context World is the knowledge base implementation of a specific domain, which contains domain concepts and facts. Context information from the physical environment, user interaction, and functional service results are stored in the context world. The semantics descriptions of all of the services are the foundation of planning and dynamic service composition. Context Ontology Model is proposed to model domain concepts in context. Context Condition and Context Effect are proposed to describe dynamic semantics in an editable and easily-modified script manner. The ontology reasoning system is enhanced by accommodating context condition/effect execution.

Reference

[1] Gu, T., et al. 2004. An ontology-based context model in intelligent environments. In Proceedings of the Communication Networks and Distributed Systems Modeling and Simulation Conf., Soc. for Modeling and Simulation Int'l.

[2] Maamar, Z., Kouadri, S., and Yahyaoui, H. 2004. Symposium a web services composition approach based on software agents and context. In Proceedings of the 2004 ACM symposium on Applied computing, 14-17.

[3] McIlraith, S. A. and Son, T. C. 2002. Adapting golog for composition of semantic web services. In Proceedings of the 8th International Conference on Principles of Knowledge Representation and Reasoning, 482—496

[4] Reiter, R. 2001. Knowledge in action：logical foundations for describing and implementing dynamical Systems．Cambridge，MA：MIT Press.

[5] Evren Sirin, Bijan Parsia, Bernardo Cuenca Grau, Aditya Kalyanpur, Yarden Katz. Pellet: A practical OWL-DL reasoner [J]. Web Semantics: Science, Services and Agents on the World Wide Web. 2007,5(2): 51-53.

[6] Chang, L., Shi, Z., Qiu, L., and Lin, F. 2007. Dynamic description logic: embracing actions into description logic. In Proc. of the 20th International Workshop on Description Logics (DL'07), Italy.

[7] Chang, L., Lin, F., and Shi, Z. 2007. A dynamic description logic for semantic web service. In: 3rd Int. Conf. on Semantics, Knowledge and Grid, IEEE Press, Los Alamitos.

[8] Mostefaoui, S. K. and Hirsbrunner, B. 2003. Towards a context-based service composition framework. In Proceedings of the International Conference on Web Service, ICWS'03, Las Vagas, Nevada, 23-26.

[9] Mostefaoui, S. K., Tafat-Bouzid, A., and Hirsbrunner, B. 2003. Using context information for service discovery and composition. In iiWAS.