Journal of Theoretical and Applied Information Technology
15th November 2018. Vol.96. No 21
© 2005 – ongoing JATIT & LLS

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

ASYNCHRONOUS REASONING SCHEME FOR GLOBAL ONTOLOGY MANAGEMENT IN INTERNET OF THINGS INFORMATION SYSTEMS: *ASYNCR*/GOM*

[1]YONGGOO CHOI, [2]ILKYEUN RA, [3]SANGWON LEE

[1]Professor. Department of Computer Information, Dong Seoul University, Seongnam 13117, Republic of Korea

[2]Professor. Department of Computer Science and Engineering, University of Colorado Denver, Colorado 80204, United States of America

[3]Associate Professor. Department of Computer & Software Engineering (Institute of Convergence Creativity), Wonkwang University, Iksan 54538, Republic of Korea

E-mail: [1]ygchoi@du.ac.kr, [2]ilkyeun.ra@ucdenver.edu, [3]sangwonlee@wku.ac.kr

ABSTRACT

In the open and dynamic Things of Internet (IoT), synchronization of the things is mandatory to provide their adaptable behaviors and maximum autonomies. The core goal of the synchronization is consistent context reasoning and up-to-date context maintaining in the IoT information systems. For realizing this goal, we present an asynchronous reasoning (AsyncR*) scheme, which is capable of non-stop reasoning while always maintaining up-to-date context information in the IoT information system. The *AsyncR** scheme based on semantic-timestamp and forged-version scheduling methods to preserve a serializability between concurrent ontology transactions. We also present a global ontology management (GOM) model and an ontology transaction (OT) model for efficiently governing the IoT ontology system. Finally, we talk key issues of the correctness of the *AsyncR** scheme in consideration of diverse synchronous situations.

Keywords: *AsyncR* (Asynchronous Reasoning), GOM (Global Ontology Management), Ontology System Model, Ontology Transaction Model, Internet of the Things (IoT).*

1. INTRODUCTION

The vision of ubiquitous computing has typically focused on providing people with comfortable living environments. As a key task for achieving this vision, all of the intelligent devices around human being let themselves manage overall matters related to human activities without explicit human intervention. The Internet of Things (IoT) [1] has become a key concept in industry and academic fields to achieve such key tasks.

The IoT is completely integrated into everyday life of people to provide people with diverse context-aware services and context information in an "anyone, anywhere, anytime, and anything" fashion. Therefore, all of the intelligent objects on the IoT must be able to share common knowledges, reason their situations and interoperate with each other. For these tasks, ontologies [2] have clear advantages in the IoT context model due to enabling knowledge sharing and interoperability with semantically well-defined concepts and their relationships in the open and dynamic environments.

Nowadays, the IoT technologies are widely applied to not only common workplaces, but mission-critical applications such as healthcare service, intelligent production, smart farming, security monitoring and control, and so on. In such a mission-critical IoT environments, it is very important to synchronize the things' behaviors and preserve their autonomy through adaptive context reasoning. Therefore, we propose an asynchronous reasoning (AsyncR*) scheme, which is capable of non-stop reasoning while always maintaining up-to-date context information in the IoT information system.

The design philosophy behind the *AsyncR** scheme is that a reasoning process that was once started should not be interrupted and avoided in any cases to preserve consistent behaviors of the things in the IoT, and an ontology information system for them should always maintain up-to-date context information. Analogously, it should provide a high degree of concurrent execution of ontology tasks. The *AsyncR** scheme based on semantic-timestamp and forged-version scheduling methods to preserve a serializability between concurrent ontology transactions. We also present a global ontology management (GOM) model and an ontology transaction model for efficiently governing the IoT ontology system. Finally, we talk key issues of the correctness of the *AsyncR** scheme in consideration in the global ontology system.

2. BACKGROUNDS

2.1. IoT Ontology Modeling Scheme

Ontology is the study to express the general concepts that symbolize the basic categories of objects or elements that make up the world [3]. In other words, in the IoT, the Ontology describes a way to express relationships between things clearly. This type of ontology allows the user to define the status of certain IoT objects, such as various sensors, air conditioners, fans and lamps, and to define the relationships between objects. These defined objects and their relationships can be used for search, statistics, and analysis on situation and status data, as well as analysis on their associations and links. These applications are based on semantic web technology. The semantic web is a kind of web technology for the next generation that will give computers a handle on the information that exists on the web. In other words, the semantic web makes it possible for computers to understand the meaning of web information resources and self-designate the process of handling information such as extracting, transforming, cleansing, and loading information [4]. Until now, it is known that Ontology is best suited for modelling methodologies to systematically represent and deduce these semantic web resources.

An ontology modelling component consists of a large number of concepts, relationships between concepts, and constraints. In other words, it helps us classify the things that are in this world into a common class and explicitly articulate relationship between them [5].

- Concept: It is a basic unit of ontology used to abstract the properties of an object in a particular area. Things in the same category are called classes.
- Relationship or attribute: This is an ontology unit for expressing meaningful associations between classes.
- Objects or instances: This refers to the physical object of the abstract concept, and the value at the bottom of the ontology hierarchy.
- Communication: This formats relationship among the ideas of the Ontology and is used to clearly infer information.

To further explain the Ontology concept, see a sensor Ontology example as shown in Figure 1.

Example 1(*Sensor Ontology*): The fact that ‘*TemperatureSensor* is a kind of *Sensor*’ is a true proposition and knowledge that everybody has agreed to. *TemperatureSensor* and *Sensor* are the central concepts used to express knowledge, and their relationship is expressed as *is-a* (Figure 1).

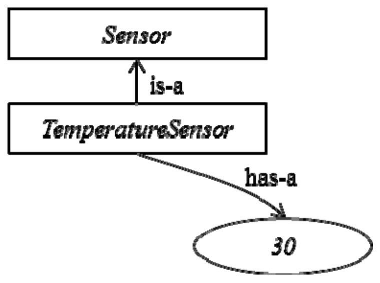


Figure 1: Hierarchical Representation of the Partial Sensor Ontology

TemperatureSensor and *Sensor* can be expressed as meaningful hierarchical relationships through the relationship *is-a*. And the *TemperatureSensor* concept can be expressed as a *hasValue* relationship, and the property can be expressed as an integer. It is true that *TemperatureSensor* has a value of *30*, and when you format it, it becomes an axiom.

End of Example 1.

IoT ontology modeling technology requires an annotation technique to convert sensor big-streaming data into a semantic basis. In other words, it defines the detailed meaning of the status and circumstances of objects for interoperability and situation analysis through semantic based reasoning of IoT data. To express things on a semantic web, abstract IoT and sensor streaming data (such as user, object, time, location, service, and situation) are expressed in an on-demand knowledge base and various status information is deduced. Major Ontology includes SSN (Semantic Sensor Network), Onto-Sensor, and SensorData [6, 7, 8, and 9]. They are primarily an extension of the data recognition model of the Open Geospatial Cooperative (OGC) SWE (Sensor Web Enablement) [8]. However, these IoT ontology models are designed to serve the interoperability of things, so there is a limit to the real time sharing of objects. Accordingly, synchronization techniques are necessary for IoT multiple products to jointly use a single knowledge base, Ontology.

2.2. IoT Ontology Sharing Scheme

In the open and dynamic context of the IoT, the purpose of synchronization among the things is to ensure the maximum autonomy of things and to enable the thorough interoperability of things at the same time. To preserve consistency in the context information based on the IoT Ontology knowledge, concurrency control schemes are required. They are necessary between devices to add new knowledge classes to the ontology to keep things in synchronization, and devices that use knowledge in real time from the ontology. In the IoT, the purpose of the situational synchronization technique is to give things a context-knowledge view of the current situation, and to do so. To do so, Ontology storage should reflect the context of the current application on a real-time basis from various sensors around IoT. In IoT situations, the purpose of synchronization is to ensure proper behavior and to maintain independence of objects. That is, to ensure that things are as independent as possible while maintaining their thorough synchronization.

The basic principle of IoT Ontology sharing is to set the sequence of approaches to things. That is, it is necessary to control the concurrency control for the use of the status information of the different objects in the Ontology management system to ensure effective interoperability through synchronization of devices. Using the Ontology status information for real-time big data analysis in IoT environment, there are two works; changing the Ontology schema structure to add a new instance to the Ontology and searching for the Ontology schema. This environmental approach requires access to a number of classes in the semantic hierarchy structure of the Ontology.

Example 2(*Undesirable Behavior when Approaching the IoT Ontology Semantic Hierarchical Structure*): Let's assume that, in a humidity sensor *HumiditySensor* in an object *Object* in the IoT environment, 70% of a value measured at humidity sensor 1 *HumiditySensor#1* H1 is about to be added while 78% of a value measured at a specific time is used in a ventilator 1 *Ventilator#1* (Figure 2).

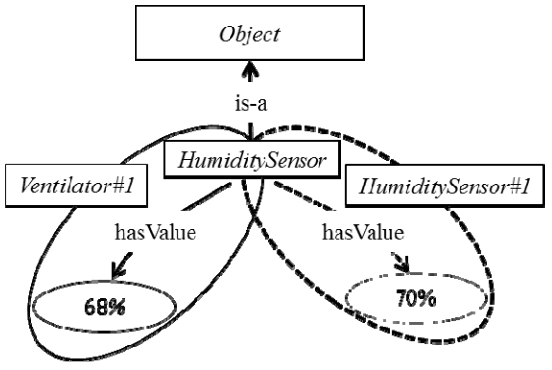


Figure 2: Concurrent Access of an Element in Ontological Semantic Web

Figure 2 shows that various things use the context awareness information in the IoT Ontology. While *Ventilator#1* recognizes that the humidity instance is *68%*, the *70%* of humidity instance of *HumiditySensor#1* recently measured can be added to the IoT Ontology. Real-time access to Ontology repository for IoT context-aware services is more likely to cause phantom phenomena [10], which can negatively affect the real-time interoperability of intelligent things in IoT.

End of Example 2.

In the concurrent access to IoT Ontology, a phantom phenomenon means that when a particular object is searched for the same ontology blocks repeatedly, it results in different resulting values. That is, this typically occurs when adding new classes or instances during the course of an ontology context data search. In IoT Ontology services, a data inconsistency for phantom phenomenon can lead undesirable behaviors to an intelligent device. When multiple things simultaneously approach the same ontology, undesirable behaviors in the IoT may occur.

To prevent these undesirable behaviors from a phantom phenomenon there are conservative and optimistic concurrent-control schemes. The conservative schemes are to have a locking method in advance to prevent other objects from accessing the IoT context information when an object in the IoT are accessed. On the other hand, the optimistic scheme is to allow multiple objects to access a single IoT Ontology simultaneously, but to grant the final result through a consistency check at the end of the access.

3. GLOBAL ONTOLOGY MANAGEMENT MODEL

3.1. Ontology System Model

In order to apply the IoT information system to mission-critical applications, it is necessary to integrate the local ontologies for overall governments for them. This integrated local ontology defines as global ontology [11]. The global ontology is composed by upper-level ontologies (UO) and local ontologies (LOs) for a unified and coherent reasoning. The *UO* specifies the semantic representation of common concepts and their semantic relationships of the things such as object, users, context, services, and so on [12]. On the other hand, the *LO*s are a semantic representation of the concepts and their relationships for any specific goals in domain-specific situations with the local IoT environments. The repetitive blocks of the local ontologies include in a global ontology according to the upper-level ontology schema rules that is specified in semantic relationships between ontology concepts or their instances. In the mission-critical IoT applications, providing consistent behaviors of the things and reasonable performances for efficient interoperability of them comes from a global ontology management (GOM) model (Figure 3).

We suppose that the local ontology blocks are inserted into the global ontology repository by a middle-ware solution over a span with a certain period of time. Such large global ontology bases are necessary to drive semantic context-awareness services in the intelligent business applications and scientific terminologies [13]. In the *GOM* system, the context reasoning rules provide ways to make high-level contexts which are more meaningful from the low-level context [14] and used by the context reasoner. The context awareness is a task that recognizes a specific situation by combining the reasoning results with other contexts. The *GOM* scheduler processes in the execution order between a context reasoning task and an insertion operation. Finally, the service adaptation [15] plays a role in executing adaptive services related to global ontology.

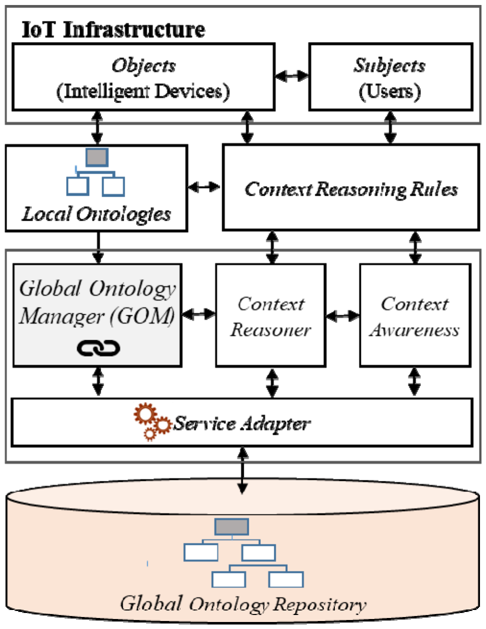


Figure 3: Global Ontology Management System

3.2. Ontology Transaction Model

To preserve a consistent information system in IoT environments, it is important to have a well-defined transaction model to be adequate to the IoT information systems. In fact, the context features of the IoT information systems are time-series and time-less [16] in the case of governing both their contexts and applications. The time-series context information is to record up-to-date context data every time the situation information of the things is changed. The other hand, the time-less context information is applied to predicting future situations of the things based on analyzing their past status using the context information. This double-sidedness features of the context information can make ontology-transaction model simple. Therefore, the action tasks in an ontology transaction consist in reasoning through traversing existing ontology blocks, and insertion for submitting new ontology blocks in the global ontology repository. That is why traversing and inserting ontology blocks for governing up-to-date situations is meaningful, but updating and deleting existing ontology blocks for ontology management is meaningless because of the double-side context information features.

In the ontology transaction model, a set of ontology tasks is $OT = \{S, P, T, A\} \subset OT$, and $\{Reasoning, Insertion\} \in T$, where S, P, T, and A are the abbreviations for *Start*, *Place*, *Traverse*, and *Adapt* respectively (Figure 4).

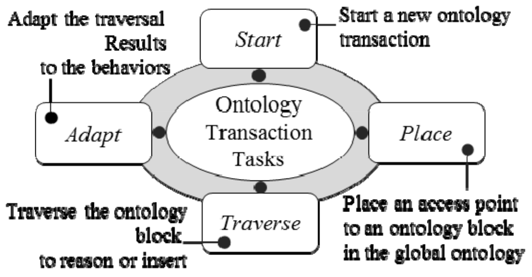

Adapt the traversal Results to the behaviors
Start
Start a new ontology transaction
Ontology Transaction Tasks
Adapt
Place
Traverse the ontology block to reason or insert
Traverse
Place an access point to an ontology block in the global ontology

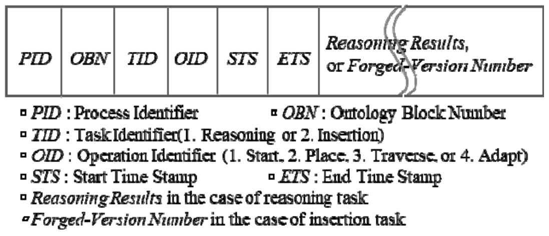
Figure 4: Tasks in the Ontology Transaction

In the *Start* step in an ontology transaction, *GOM* system creates a new ontology process to begin the ontology transaction. And then the *Place* step directly moves an access position on a target ontology block in the global ontology using primary context types [17] such as ontology identifier, time, location, state, etc. The *Traverse* step searches ontology blocks located in the *Place* step for reasoning or insertion. Depending on the traversal results, the *Adapt* step performs context-awareness services based on reasoning results, or actually inserts the ontology block from local ontologies into global ontology repository.

4. ASYNCHRONOUS REASONING SCHEME

4.1. Ontology Process State Block

The first design philosophy behind *AsyncR** is that a reasoning process that was once started should not be stopped in any case to preserve the consistency of the behaviors of the IoT objects, while the global ontology is always maintaining up-to-date context information under the dynamic environments. Simultaneously, it provides a high degree of concurrent execution of ontology tasks for preserving autonomous behaviors of the intelligent devices on the IoT. To elucidate the *AsyncR** scheme, we assume that the global ontology resides in computing main memory after being parsed in accordance with ontology schema rules, and each domain ontology in the global ontology has ontology block numbers (OBNs) to distinguish them from the others. We also assume that an ontology state block (OSB) has the ontology process state information such as ontology transactions, domain-ontology blocks, their associated time-stamps, and etc. To deal with such *OSB*, *AsyncR** scheduler maintains the following data structure (Figure 5).



PID	*OBN*	*TID*	*OID*	*STS*	*ETS*	*Reasoning Results*, or *Forged-Version Number*

▫ *PID* : Process Identifier ▫ *OBN* : Ontology Block Number
▫ *TID* : Task Identifier(1. Reasoning or 2. Insertion)
▫ *OID* : Operation Identifier (1. Start, 2. Place, 3. Traverse, or 4. Adapt)
▫ *STS* : Start Time Stamp ▫ *ETS* : End Time Stamp
▫ *Reasoning Results* in the case of reasoning task
▫ *Forged-Version Number* in the case of insertion task

Figure 5: Ontology State Block

In Figure 5, when an ontology transaction is started, the *OSB* is immediately created by a *GOM* system, and the *PID* has a unique number to be able to distinguish the process from other processes. The *OBN* assigns the ontology block number in the global ontology to be accessed by an ontology transaction. The *TID* records a number of the task types defined in an ontology transaction. It only has one of two digits: *1* means a reasoning task, or *2* means an insertion task. The *OID* is assigned a number of an operation in an ontology transaction. It has one of four digits: *1* means start, *2* means place, *3* means traverse, or *4* means adapt operation. The *STS* and the *ETS* records each start and end time of the ontology transaction. The other part in the *OSB* has reasoning results in the case of the reasoning task, or a forged-version number in the case of the insertion task when required by *AsyncR** scheduler.

4.2. Scheduling Algorithm

In this *AsyncR** scheduling algorithm, our primary concerns focus on how to preserve the non-stop execution of context reasoning tasks without interference of any other tasks, and how to gain higher concurrency for mission-critical context awareness services in the IoT information systems. For this goal, a semantic timestamp and forged-version control scheme are used in the *AsyncR** scheduling algorithm (Figure 6).

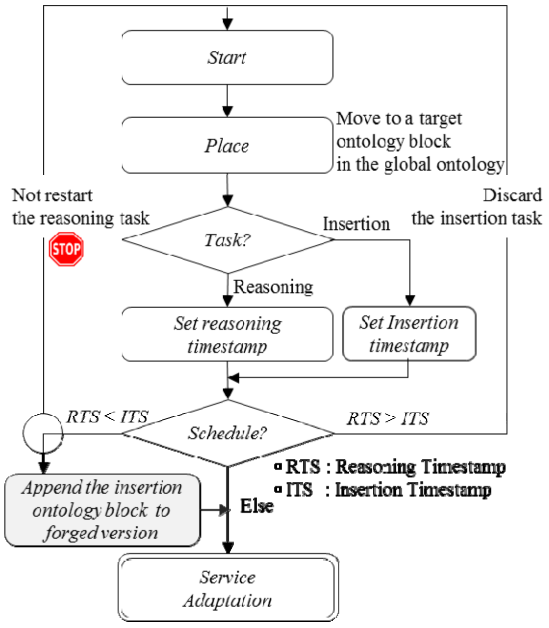

Start
Place
Move to a target ontology block in the global ontology
Not restart the reasoning task
STOP
Task?
Insertion
Discard the insertion task
Reasoning
Set reasoning timestamp
Set Insertion timestamp
RTS < ITS
Schedule?
RTS > ITS
▫RTS : Reasoning Timestamp
▫ITS : Insertion Timestamp
Append the insertion ontology block to forged version
Else
Service Adaptation

Figure 6: AsyncR Scheduling Algorithm*

In *AsyncR** scheduling algorithm in the Figure 4, each ontology transaction progresses according to the following scheduling rules.

- **Rule 1**(*Start scheduling*): If an ontology transaction begins in *Start* step, the *GOM* system creates an *OSB* and assign a *PID*, *OBN*, *TID*, *OID*, and *STS* respectively to the *OSB* as shown in Fig. 3.
- **Rule 2**(*Place scheduling*): According to the context reasoning rules and ontology schema, the *GOM* system directly moves an access position on a target ontology block in global ontology using primary context types. The rule of finding an access position of an ontology block is the same as *OT*(*STS*) >= max(*OBN*(d-time)), where d-time is the time assigned to the ontology block.
- **Rule 3**(*Semantic timestamp scheduling*): If an ontology transaction is successfully initiated, *AsyncR** scheduler assigns the ontology task type to the *TID* and the starting time to *STS* in the *OSB* respectively. If two different ontology transactions access the same ontology block in the global ontology, the *AsyncR** scheduler processes as follows.

- **Case 1**(*In the case of service requests in the order of OT_1: Reasoning and OT_2: Reasoning*): The ontology transactions OT_1 and OT_2 run a "first-come, first-serve" basis in time sequence because of non-conflict serializability.
- **Case 2**(*In the case of service requests in the order of OT_1: Reasoning and OT_2: Insertion*): In case that while OT_1 is executing a reasoning task on a specific ontology block in the global ontology, OT_2 is supposed to insert a new ontology block at the same position, the insertion ontology block issued by OT_2 leads a phantom block to OT_1. In this case, after the insertion task in the OT_2 is completed, the OT_1 should be restarted for preserving serializability of the ontology transactions. However, since restarting the reasoning task of the OT_1 results in inconsistent interoperability among IoT objects, OT_1 should not be interrupted in any case. Therefore, for asynchronous reasoning of OT_1, the *AsyncR** scheduler temporally made a new forged-version for OT_2, and insert it into the global ontology repository as soon as OT_1 have finished.
- **Case 3**(*In the case of service requests in the order of OT_1: Insertion and OT_2: Reasoning*): The OT_1 is cancelled because the ontology block of the insertion in the OT_1 is unnecessary for any other applications.
- **Case 4**(*In the case of service requests in the order of OT_1: Insertion and OT_2: Insertion*): Ontology transactions run a "first-come, first-serve" basis in time sequence of pseudo-conflict serializability.

Considering the above cases, *AsyncR** timestamp scheduling algorithm can streamline them as shown in Table 1.

Table 1: Timestamp Compatibility Matrix for AsyncR Scheduling*

$OT_1.time_1$	$OT_2.time_2$	*AsyncR** scheduling
Reasoning	*Reasoning*	Sequence
Reasoning	*Insertion*	Forged-version
Insertion	*Reasoning*	Discard Insert
Insertion	*Insertion*	Sequence

- **Rule 4**(*Forged-version scheduling*): If an ontology transaction requests a traveling on an ontology block for an insertion task while other transaction is traveling on the same ontology block for a reasoning task, the *AsyncR** scheduler decides to create a forged version of the ontology block and waits until the reasoning task is finished. Right after finished, the forged-version ontology block is inserted into a global ontology repository.
- **Rule 5**(*Service adaptation*): If an ontology transaction scheduling is completed successfully, the *AsyncR** scheduler assigns ending time of the transaction to ETS in the OSB and begins ontology tasks: context awareness, or insertion task.

4.3. Proof of Correctness

In the academic field, a correctness criterion pertaining to concurrency control schemes has been widely used in serializability among transactions [18]. To prove the correctness of *AsyncR** scheme, we will show that the *AsyncR** scheduler always produces a serial execution history. Due to preservation requirement of creation order of any conflicting insertion ontology transactions, it is necessary to show that the forged-version timestamp scheduling scheme for the *AsyncR** always processes them in the order of their creation sequence.

Definition 1(*Global ontology structure*): When a global ontology (GO) consists in an upper-level ontology (UO) and several local ontologies (LOs), it can express in follow:

$$GO = UO \propto \sum_{i=0}^{m} \sum_{i=0}^{n} LO_{ij}, \qquad (1)$$

where i is the number of a specific ontology domain, and j is the number of a ontology block in the block i. By the Equation the global ontology presents as GO = { $UO \propto$ { { LO_{11}, LO_{12}, …., LO_{1n} }, ..., LO_{ij}, ..., LO_{mn} } }, where LO_{ij} is the ontology blocks in global ontology, and each LO_{ij} has seamlessly semantic relationships with each other according to the upper-level ontology schema rules.

Definition 2(*Context integration and integrity constraints of a global ontology*): The global ontology should preserve ① context integration that completely converges the local (or domain) ontologies to share the global ontology among the things in the IoT, and ② context integrity that entirely contains all the essential context information pertaining to context reasoning.

Suppose 1(*Uniform access to an ontology block in the global ontology*): Access to an ontology block in the global ontology is uniform in main memory ontology.

Lemma 1(*Non-stop reasoning*): In the IoT information systems, context reasoning for context-awareness services should never be interrupted and avoided in any other situations for consistent interoperability among the things in the IoT environments.

Proof) Let's recall Definition 1. If in the IoT dynamic environments, an ontology transaction interrupts and avoids reasoning process on LO_{ij}, at the moment, $LO_{i(j+1)}$ may be added in another ontology transaction. In this case, the LO_{ij} becomes old-fashion context block in time sequence for a situation status. This schedule scheme leads to inconsistent interoperability because of context information loss, and violates the context integrity constraints in the definition 1. Therefore, a reasoning task in an ontology transaction should be always executed in non-stop fashion to combat such side-effects caused by a result of such miss-scheduling.

Lemma 2(*New-fashion placing*): If some ontology transaction intents to access an target ontology block in the global ontology, the access point should be always placed in the most recently inserted ontology block in time sequence.

Proof) The IoT applications are typically based on prompt interoperability using context information generated in a real-time fashion from various sensors on the IoT. Let's recall Definition 1 and Suppose 1. Suppose that the local ontology sets { LO_{11}, LO_{12}, …., LO_{1n} } in a global ontology are listed in chronological order, and $LO_{11} > LO_{12}$ in time sequence is established. If an ontology transaction intends to traverse an ontology block LO_{1j} for context reasoning, the LO_{11} always becomes the target ontology block. In the same manner, if an ontology transaction intends to insert a new ontology block from local ontology repositories, it become inserted into the front of LO_{11}. This new-fashion placement follows a stack data structure that has a "last-in, first-service (LIFO)" fashion.

Theorem 1(*View serializability [19] of AsyncR* scheme*): Suppose that the local ontology sets are represented as $\{LO_{11}, LO_{12}, ..., LO_{1n}\} \subset GO$, $LO_{11} > LO_{12}$ in time sequence. Now, consider the following schedule history *H*:

(1) The *AsyncR** scheduler receives LO_{11}, where LO_{11} denotes that an ontology transaction OT_1 retrieves the LO_{11} contained in LO_{1j}.

(2) The *AsyncR** scheduler places LO_{1j}, where LO_{1j} denotes that OT_2 inserts the new *LO* into LO_{1j}.

(3) The *AsyncR** scheduler places in LO_{11}.

According to schedule history *H*, the *AsyncR** scheme is view serializability if and only if differencing R_2 from R_1 is null where R_1 is a result set of Case 1 and R_2 is a result set of Case 2

Proof: The serial graph of *H* is represented as $OT_1 \rightarrow OT_2 \rightarrow OT_1$. The *AsyncR** scheduler receives an absolute position of LO_{11} from the global ontology and has a start timestamp t_1 according to ontology state block (OSB) in Fig. 3. The R_1 of LO_{11} execution gets $LO_{11} \subset OL_{1j}$ (Case 1). Now, suppose that OT_2 at the almost same time t_2 requests an absolute position of LO_{1j} so as to insert a new ontology block from local ontology repositories into LO_{1j}. The insertion task of OT_2 could not be immediately granted according to Lemma 1. Therefore, the *AsyncR** creates a forged-version LO_{11}^{+} of LO_{1j} at time t_6 and inserts a local ontology block into LO_{11}^{+} (Figure 7). The OT_2 waits until the reasoning task of OT_1 is finished. Right after finished at the time t_5, forged-version ontology block inserts into global ontology repository at the time t_7 (Case 2 of Rule 3 and Rule 4) (Fig. 5). Therefore, the serial graph of *H* executes in serial as $OT_1 \rightarrow OT_2$ and reasoning tasks of OT_1 executed in non-stop fashion (Lemma 1). According to Case 2 of Rule 2, the inconsistent interoperability occurs in *AsyncR** scheduling rules since differencing R_2 from R_1 is always null value. This is sufficient to prove that *AsyncR** scheduler only produces serializable executions semantically.

Theorem 2(*Correctness of AsyncR* scheme*): For any history *H*, *H* is serializable if only if every transaction in *H* obeys the schedule rules of *AsyncR**.

Proof: This necessary condition is capable of being proven by following the same discussion as that of Theorem 1. From Lemma 1 and 2, we can also infer that all conflicting insertion transactions in *H* are executed in the order of their creation and *H* is serializable, if every ontology transaction in *H* obeys the concurrency control rules of *AsyncR**. Hence, if every transaction in *H* obeys the scheduling rules of *AsyncR**, *H* is serializable. Finally, *H* is serializable if and only if every ontology transaction in *H* obeys the scheduling rules for *AsyncR**.

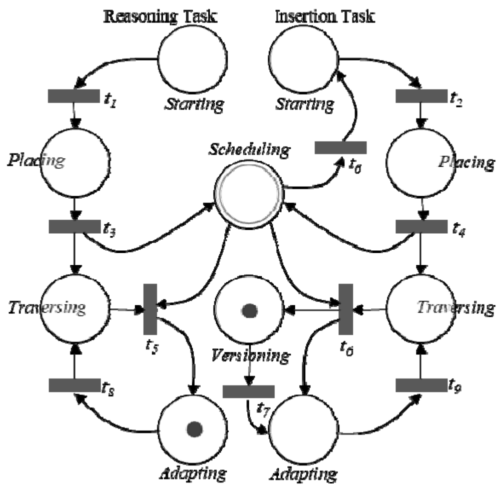


Figure 7: Proof of view serializability of the ontology transactions using Petri Nets [20]

5. DISCUSSION AND CONCLUSIONS

In this paper, we proposed an *AsyncR** scheme, which was capable of non-stop reasoning while always maintaining up-to-date context information in the IoT information system. The *AsyncR** scheme based on semantic-timestamp and forged-version scheduling methods to preserve a serializability between concurrent ontology transactions. The background of the *AsyncR** scheme was that a reasoning process that was once started should not be stopped in any case to preserve the consistency of the behaviors of the IoT objects, while the global ontology should always maintain up-to-date context information. For realizing this design philosophy, we also presented a global ontology management (GOM) model and an ontology transaction model for the *AsyncR** scheme. Finally, we talked correctness issues of the proposed scheme using Petri Nets. The AsyncR* scheme is simple, but provides a high degree of concurrency of ontology tasks.

As this paper may ignore comparison with other approaches in terms of some examples, the *AsyncR** scheme could be strengthened by additional evidence demonstrating its strengths over the conventional approaches. The implementation of the proposed scheme and the quantitative analysis of execution times according to a real-time variance of computerized resources needed for the proposed timestamp and version scheme will be included in future works. Finally, we would like to extend our scheme to allow further theoretical investigation by using some properties such as multiple versions and operation semantics for more novel models.

ACKNOWLEDGMENTS

This paper was supported by Dong Seoul University for the research year (2016.8 ~ 2017.7) of the first author, *Yonggoo Choi*.

REFRENCES:

[1] IStrategy, I. T. U., and Policy Unit. "ITU Internet Reports 2005: The internet of things.", *International Telecommunication Union (Geneva)*, 2005.

[2] T. R. Gruber, "A Translation Approach to Portable Ontology Specifications", *Knowledge Acquisition*, Vol. 5, No. 2, 1993, pp. 199-220.

[3] J. Park, "Ontology," in Management Information Systems, *G.B. Davis (ed.) Cambridge*, Blackwell Publishing, Vol. 7, 2005, pp. 233-236.

[4] Amit Sheth and Matthew Perry, "Traveling the Semantic Web through Space, Time, and Theme", *IEEE Internet Computing*, Vol. 12, No. 2, 2008, pp. 81-86.

[5] Mustafa Jarrar, Jan Demey, and Robert Meersman. "On Using Conceptual Data Modeling for Ontology Engineering", *Journal on Data Semantics i*, Springer Berlin Heidelberg, 2003, pp. 185-207.

[6] Holger Neuhaus and Michael Compton. "The semantic sensor network ontology", *AGILE Workshop on Challenges in Geospatial Data Harmonisation*, 2009.

[7] D. J. Russonmanno, C. Kothari, and O. Thomas, "Sensor Ontologies: from Shallow to Deep Models", *Proceedings of System Theory*, SSST'05 Proceedings of the 37th Southeastern Symposium on Los Alamitos (USA), Mar. 2005, pp. 107-112.

[8] A. Sheth, C. Henson, and S.S. Sahoo, "Semantic Sensor Web", *Internet Computing IEEE*, Vol. 12, Aug. 2008, pp. 78-83.

[9] C. Michael, H. Cory, L. Laurent, N. Holger, and S. Amit, "A Survey of the Semantic Specification of Sensors", *Proceedings of the 2nd International Workshop on Semantic Sensor Networks (SSN09)* (USA), Oct. 2009, pp. 17-32.

[10] Weikum, Gerhard, and Gottfried Vossen. *Transactional information systems: theory, algorithms, and the practice of concurrency control and recovery*. Elsevier, 2001.

[11] R. F. Brena and H. G. Ceballos. "Combining Global and Local Ontology handling in a Multiagent System", *FLAIRS Conference*, 2004.

[12] M. Lenzerini, "Ontology-based Data Management", *Proceedings of the 20th ACM international conference on Information and knowledge management*, 2011.

[13] J. Seidenberg and A. Rector. "A Methodology for Asynchronous Multi-User Editing of semantic Web Ontologies", *Proceedings of the 4th International Conference on Knowledge Capture*, 2007.

[14] C. Perera, et al., "Context Aware Computing for the Internet of Things: A survey", *IEEE Communications Surveys & Tutorials*, Vol. 16, No. 1, 2014, pp. 414-454.

[15] H. Guermah, et al., "An Ontology Oriented Architecture for Context Aware Services Adaptation", *arXiv preprint arXiv:1404.3280*, 2014.

[16] C. Bettini, et al., "A Survey of Context Modelling and Reasoning Techniques", *Pervasive and Mobile Computing*, Vol. 6, No. 2, 2010, pp. 161-180.

[17] G. D. Abowd, et al., "Towards a Better Understanding of Context and Context-Awareness", *International Symposium on Handheld and Ubiquitous Computing, Springer Berlin Heidelberg*, 1999.

[18] N. S. Barghouti and G. E. Kaiser, "Concurrency Control in Advanced Database Applications", *ACM Computing Surveys*, Vol. 23, no. 3, 1991, pp. 269-317.

[19] M. Sunil, "Timestamp-Ordering Protocol for Concurrent Transactions - A Performance Study", *International Journal of Computer Applications, National Conference on Emerging Trends in Computer Technology*, 2014, pp. 24-26.

[20] T. Murata "Petri Nets: Properties, Analysis and Applications", *Proceedings of the IEEE*, Vol. 77, No. 4, 1989, pp. 541-580.