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

YONGGOO CHOI, ILKYEUN RA, SANGWON LEE
1Professor. Department of Computer Information, Dong Seoul University, Seongnam 13117, Republic of Korea
2Professor. Department of Computer Science and Engineering, University of Colorado Denver, Colorado 80204, United States of America
3Associate Professor. Department of Computer & Software Engineering (Institute of Convergence Creativity), Wonkwang University, Iksan 54538, Republic of Korea
E-mail: ygchoi@du.ac.kr, ilkyeun.ra@ucdenver.edu, 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. BACKGROUND

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).

![TemperatureSensor](https://example.com/TemperatureSensor.png)

**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](image)

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 LOs 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 model, 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.

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 $OT = \{S, P, T, A\} \subseteq OT$, and $\{Reasoning, Insertion\} \in T$, where $S$, $P$, $T$, and $A$ are the abbreviations for Start, Place, Traverse, and Adapt respectively (Figure 4).

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).

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 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).

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 \( OT(STS) \geq \max(OBN(d\text{-time})) \), 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 \( OT_1: \text{Reasoning} \) and \( OT_2: \text{Reasoning} \)): 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 \( OT_1: \text{Reasoning} \) and \( OT_2: \text{Insertion} \)): 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 \( OT_1: \text{Insertion} \) and \( OT_2: \text{Reasoning} \)): 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 \( OT_1: \text{Insertion} \) and \( OT_2: \text{Insertion} \)): 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.

![AsyncR* Scheduling Algorithm](image)

**Table 1: Timestamp Compatibility Matrix for AsyncR* Scheduling**

<table>
<thead>
<tr>
<th>( OT_1 \text{.time}_1 )</th>
<th>( OT_2 \text{.time}_2 )</th>
<th>AsyncR* scheduling</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reasoning</td>
<td>Reasoning</td>
<td>Sequence</td>
</tr>
<tr>
<td>Reasoning</td>
<td>Insertion</td>
<td>Forged-version</td>
</tr>
<tr>
<td>Insertion</td>
<td>Reasoning</td>
<td>Discard Insert</td>
</tr>
<tr>
<td>Insertion</td>
<td>Insertion</td>
<td>Sequence</td>
</tr>
</tbody>
</table>
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:

$$GO = \sum_{(i,j) \in \mathcal{G}} \sum_{i} \mathcal{UO}_i \mathcal{LO}_j,$$

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 $GO = \{ \mathcal{UO} = \{ \mathcal{LO}_{11}, \mathcal{LO}_{12}, \ldots, \mathcal{LO}_{1n} \}, \ldots, \mathcal{LO}_{m1}, \ldots, \mathcal{LO}_{mn} \}$, where $\mathcal{LO}_i$ is the ontology blocks in global ontology, and each $\mathcal{LO}_i$ has seamlessly semantic relationships with each other according to the upper-level ontology schema rules.

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 $LO_{1j}$ 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 becomes 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_{i1}, LO_{i2}, ..., LO_{in}\} \subset GO, LO_{i1} > LO_{i2} in time sequence. Now, consider the following schedule history H:

1. The AsyncR* scheduler receives LO_{ii}, where LO_{ij} denotes that an ontology transaction OT_i retrieves the LO_{ij} contained in LO_{ij}.
2. The AsyncR* scheduler places LO_{ij}, where LO_{ij} denotes that OT_j inserts the new LO into LO_{ij}.
3. The AsyncR* scheduler places in LO_{ij}.

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

**Proof:** The serial graph of H is represented as OT_i \rightarrow OT_j \rightarrow OT_l. The AsyncR* scheduler receives an absolute position of LO_{ii} from the global ontology and has a start timestamp t_i according to ontology state block (OSB) in Fig. 3. The R_i of LO_{ii} execution gets LO_{ii} \subset OL_{ij} (Case 1). Now, suppose that OT_j at the almost same time t_j requests an absolute position of LO_{ij} so as to insert a new ontology block from local ontology repositories into LO_{ij}. The insertion task of OT_j could not be immediately granted according to Lemma 1. Therefore, the AsyncR* creates a forged-version LO_{ij}^* of LO_{ij} at time t_j and inserts a local ontology block into LO_{ij}^* (Figure 7). The OT_j waits until the reasoning task of OT_l is finished. Right after finished at the time t_j, forged-version ontology block inserts into global ontology repository at the time t_j (Case 2 of Rule 3 and Rule 4) (Fig. 5). Therefore, the serial graph of H executes in serial as OT_i \rightarrow OT_j and reasoning tasks of OT_l executed in non-stop fashion (Lemma 1).

**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.
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.

REFERENCES: