LACOS: CONTEXT MANAGEMENT SYSTEM FOR A SENSOR-ROBOT NETWORK

Satoru Satake
Graduate School of Science and Technology
Keio University
3-14-1, Hiyoshi, Kohoku-ku Yokohama, Kanagawa, Japan
email: satake@ayu.ics.keio.ac.jp

Hideyuki Kawashima
Graduate School of Science and Technology
Keio University
3-14-1, Hiyoshi, Kohoku-ku Yokohama, Kanagawa, Japan
email: kawasima@ayu.ics.keio.ac.jp

Michita Imai
Faculty of Science and Technology
Keio University
3-14-1, Hiyoshi, Kohoku-ku Yokohama, Kanagawa, Japan
and Precursory Research for Embryonic Science and Technology
email: michita@ayu.ics.keio.ac.jp

ABSTRACT
The purpose of this paper is to develop a new context management system software that creates applications easily in a sensor-robot network. Following four functions are required to realize a context management system. (1) search contexts. (2) update the current context. (3) fuse two contexts. (4) reject fusion requests. To solve these problems, this paper proposes LaCoS(Layer of Context Sharing). LaCoS uses Selection and Insertion function, respectively, to solve (1) and (2). LaCoS defines new function "FUSE" to solve (3) and "REJECT FUSION" to solve (4). We implemented LaCoS and conducted experiments to confirm that LaCoS successfully equips four functions.

KEY WORDS
LaCoS, Sensor-Robot Network, Database System, Context, Interaction

1 Introduction

This paper proposes a context management system for a sensor-robot network, Layer of Context Sharing (LaCoS). The purpose of LaCoS is to manage contexts of human-robot and human-human interactions in a sensor-robot network for real world context applications such as Communityware with robots and Groupware with robots. Managing past and the current contexts of interactions enables human to find similar contexts in which he/she is interested.

But, managing contexts of interactions in sensor-robot network is difficult. Because the following functions are required: (1) search contexts, (2) update the current context, (3) fuse multiple contexts into one context, and (4) reject context fusion requests.

There are several works which tried to manage contexts of interactions in a sensor network. A network middleware[1] shares sensor information and provides the information as spatial structure. The semantic network[2] collects human interests, connects related interests and provides them for human as a directed graph.

There are two problems in both of [1] and [2]. The first problem is that they don't consider interactions between robots and human are important in a sensor-robot network. And the second problem is that they don't consider dynamically fusing nature of contexts that is a remarkable with context. Therefore previous works are not applicable for our purpose.

This paper solves these two questions. To solve the first problem, we integrate communication robots called Robovie into our design of LaCoS. And to solve the second problem, we introduce operations to fuse and reject fusion for LaCoS. Furthermore, LaCoS provides operations to search and update contexts of applications.

The rest of this paper organizes as follows. In Section 2, we describe motivation for the research and formalize the unsolved questions of previous works. In Section 3, we propose LaCoS and describes how LaCoS solves the unsolved questions. In Section 4, we evaluate the LaCoS through experiments. Finally in Section 5, we conclude this paper.

2 Requirements for applications in a sensor-robot network

2.1 Motivation

In a sensor-robot network it is essential to provide context management for human-human and human-robot interactions for applications that use information of human. I'll show one situation of which such an application is needed. Let robot R_1 show a result of an experiment to human H_1 and human H_2 in a room. And let human H_3 study in another room with a robot R_2. Suppose H_1 and H_2 found a problem in the experiment and began discussing about it. Though H_3 doesn't know the existence of the discussion, the problem is related to his research and he is potentially

interested in it.

Without context management, there is no way to tell H_3 the existence of the beneficial discussion. If the contexts of R_2 were able to browse by R_1, R_1 could provide it with H_1, and H_1 could join the discussion.

Though management of contexts is beneficial for such applications, its realization is hard since such contexts not only rapidly update, but also dynamically fuse with another context. The purpose of this paper is to present a context management system software LaCoS that supports the management of dynamically update and fuse of contexts in a sensor-robot network.

2.2 Conditions to be satisfied

A context management system software should equip the following four functions. (1)search contexts of other robots, (2)update from old context to the current context, (3)fuse two contexts, and (4)reject fusion requests for a context. Here we formalize them.

2.2.1 Definition

Before formulating the conditions, we give definitions for several terms in this paper.

Definition 1 (Context) *Let a robot be R and let an interaction of R be I at time t. Then we define a tuple consisted of R, I, and t as $context(t, R, I)$.*

Definition 2 (Fusion) *Let two robots be R_1 and R_2. We define fusion of $context(t, R_1, I_\alpha)$ and $context(t, R_2, I_\beta)$ to $context(t+1, R_1, I_\gamma)$ and $context(t+1, R_2, I_\gamma)$ respectively as $fusion(context(t, R_1, I_\alpha), context(t, R_2, I_\beta), I_\gamma)$.*

Definition 3 (Rejection) *Let a robot be R_i. We define setting contexts of R_i disable to be fused as $rejection(R_i)$.*

2.2.2 Formalization of conditions

Condition 1 (Searchablity) *We formulate being able to browse all of $context(t, R_i, I_i)$ at the current and past time t as searchable.*

Condition 2 (Updatablity) *Let a robot be R_i. We formulate enabling R_i to update $context(t, R_i, I_\alpha)$ as updatable.*

Condition 3 (Fusiblity) *Let two robots be R_i and R_j. Then, we formulate being able to execute*

$$fusion(context(t, R_i, I_\alpha), context(t, R_j, I_\beta), I_\gamma)$$

as fusible.

Condition 4 (Rejectablity) *Let a robot be R_i. Then we define being able to $rejection(R_i)$ as rejectable.*

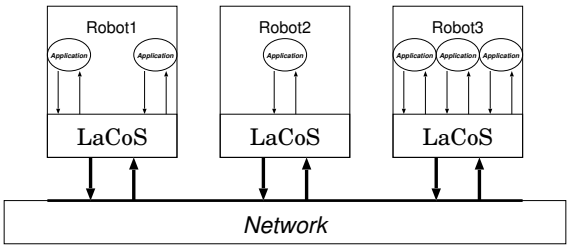

Figure 1. Robots, Network, and LaCoS

.

3 LaCoS

In this section we present a context management system for sensor-robot network named LaCoS[1]. Fig. 1 shows the overview of LaCoS. In Fig. 1, multiple robots are connected with a network and each robot has one LaCoS.

The features of the LaCoS are: (1)It satisfies four conditions defined in the previous section. We show how the LaCoS satisfies them through execution samples of LaCoS. (2)It is designed as a distributed database system to enhance durability, and (3)It has SQL interface to provide easy programming for application programmers.

3.1 Design

The system configuration of LaCoS is shown in Fig. 2. A LaCoS consists of four components. They are application interface, query issue engine, context data manager and result collector.

The application interface takes a query from an application to the query issue engine and also returns the result of the query to the application. The query issue engine finds contexts which are designated in the query, by sending broadcast message to all of LaCoSs on other hosts through network. The context data manager receives the query through the network, retrieves the required contexts

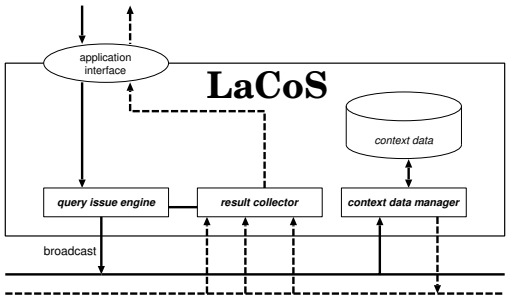

Figure 2. System Configuration of LaCoS

.

[1] **La**yer of **Co**ntext **S**haring

and returns the result of retrieval. Finally, the result collector receives the results of the query from all of LaCoSs and takes the results to the application interface. Both of the processing and the returning path for a query from an application are described in the followings.

Using a sample query, here we describe how LaCoS deals with a query. Suppose an application program issues a query such as "search contexts around me within 10 meters". For the query, in the Fig. 2, solid arrows mean processing path of the query and broken arrows mean returning path of the query respectively. At first, this query is received by the application interface and the application interface takes the query to the query issue engine. The query issue engine requires providing contexts to all of LaCoSs using broadcast message. When each query evaluation engine receives the requirement via a network, it retrieves appropriate contexts for the query on its LaCoS, and finally sends the contexts to the LaCoS which issued the requirement. For the sample query, Each query manager of LaCoS executes the query and returns all of its context which satisfies conditions of the query. Then the result collector receives all of the results. Then the selected results are taken to application interface and returned to the application program.

3.2 Operations that Satisfies Conditions

LaCoS provides the following four types of operations.

1. Searching all $context(t, R_i, I_\alpha)$.

   An operation to retrieve contexts which are within 10 meters from this robot is written as "*SELECT * FROM context_table WHERE position* $<$ *10*". In this operation, "*context_table*" has historical contexts of each robot and "*position*" is one of the attributes in the "*context_table*". This operation realizes searchability and hence LaCoS satisfies condition (1).

2. Updating $context(t, R_i, I_\alpha)$ to $context(t+1, R_i, I_\beta)$ on R_i at time t.

   An operation to update the current context of this robot is written as "*INSERT INTO context_table (context) VALUES ('new_context')*". In this operation, the current context of this robot at the *context_table* is modified to *new_context*. This operation realizes updatability and hence LaCoS satisfies condition (2).

3. Fusing $context(t, R_i, I_\alpha)$ with $context(t, R_j, I_\beta)$.

   An operation at time t to fuse the current context of this robot($robot_{this}$) with the current context of $robot_{tar}$ is written as "*FUSE TO* $robot_{tar}$ *INTO context_table (context) VALUES ('NEW CONTEXT')*". In this operation, LaCoS of $robot_{tar}$ receives the fusion request message from $robot_{this}$ to fuse with ("*NEW CONTEXT*"). This operation realizes fusibility and hence LaCoS satisfies condition (3).

```
1:  issue the insertion operation to robot_target;
2:  if( robot_target accepts insertion operation ){
3:    issue the insertion operation to robot_this;
4:    print("Fusion is accepted");
5:  }else{
6:    print("Fusion is rejected");
7:  }
```

Figure 3. Codes of the fuse operation

4. Rejecting to be fused with another $context(t, R_i, I_\alpha)$.

   An operation to reject a fusion with other contexts is written as "*REJECT FUSION REQUEST TO context_table (context)*". On the other hand, an operation to accept a fusion with other contexts is written as "*ACCEPT FUSION REQUEST TO context_table (context)*". This operation realizes rejectability and hence LaCoS satisfies condition (4).

Therefore LaCoS satisfies four conditions that should be satisfied as a context management system software for a sensor-robot network.

3.3 Implementation

In this section, we describe the implementation of LaCoS. The application interface, the query issue engine, and the result collector are originally implemented by C language and the context data manager is realized by using PostgreSQL 7.3[3]. The operations "*FUSE*", "*REJECT FUSION*", and "*ACCEPT FUSION*" are implemented by SQL[4].

Each context data manager has the same name table for contexts, described as $context\_table$. Each of $context\_table$ has the same set of attributes, but its contents are different. $robot\_name$, $context$, $position$, and $create\_time$ are attributes of a $context\_table$. Details are described table 1 in Section 4.1.

"*FUSE*" is converted to a SQL representation by "*INSERT INTO context_table (context) VALUES ('NEW CONTEXT')*". The insertion operation is sent to only $robot_{this}$ and $robot_{tar}$. The codes are described in Fig. 3. At line 1 and line 3, system issues the insertion operation to $robot_{tar}$ and $robot_{this}$ respectively. The reason why system issues the insertion operation to $robot_{tar}$ at first is that the insertion operation to $robot_{tar}$ may be rejected.

"*REJECT FUSION*" operation is converted to a SQL representation by "*CREATE TRIGGER R_FUSION BEFORE INSERT FOR EACH ROW execute PROCEDURE reject_fusion()*". This operation creates event driven function *reject_fusion()* which is invoked before *INSERT* operation in PostgreSQL. The reason why trigger performs as $reject(R_i)$ is that *reject_fusion()* stops execution of *INSERT*. The *reject_fusion()* is written by PL/pgSQL.

"*ACCEPT FUSION*" is converted to a SQL representation by "*DROP TRIGGER R_FUSION*". The operation

cancels *reject_fusion()*.

The communication protocol for a selection operation between LaCoSs is implemented as followings. At first, the query issue engine sends an udp broadcast packet which contains a tcp port number. After receiving the udp packet, each LaCoS returns an ack message to the result collector using the tcp port number and invokes the context data manager component. On the other hand, query issue engine invokes its result collector component which detects ack messages and host names of other LaCoSs. Whenever the result collector detects an ack message and its host name by waiting the tcp port, the query issue engine issue the selection operation to the context data manager of the host. The result collector stops the detection of ack messages, when it can not detect an ack message for 10 msec. And the result collector collects results of the selection operation and takes them to the application interface. If the tcp port number is already used by another query, the query issue engine uses the next port.

4 Evaluation

4.1 Behavior of LaCoS

Environment In this paragraph we show environments of experiments of LaCoS. All of behavior tests were executed under Linux 2.6 and FreeBSD 4.8.

In the environment of behavior test, we assume that there are three robots connected through a network. To test this assumption, we used three machines which are named Mars, Cygnus and Antares. Mars is a Linux 2.6.5 machine which has 4G bytes memories and a 3.0 GHz Pentium 4 processor. Cygnus is a Linux 2.6.5 machine which has 2G bytes memories and a 3.0 GHz Pentium 4 processor. Antares is a FreeBSD 4.8 machine which has 2G bytes memories and a 2.0 GHz Pentium 4 processor. Mars and Antares are connected with 1G bps Ethernet, but Cygnus is connected with 100M bps Ethernet. To simplify the behavior test, we use one dimension for locational coordinates. Let the location of Mars be at 0 meter, the location of Cygnus be at 5 meters, and the location of Antares be at 100 meters.

We define context_tables named CTable on each LaCoS to represent each context. A tuple in the CTable has a context of a robot, the position of the robot, and time when the tuple is created(table 1). A context is represented by text data, such as "Discussion about robot planner". The context is bounded with its creation position and time. And initial states of CTables are shown in table 2.

Searching contexts In this paragraph we describe three search results of LaCoS. They are simple search, conditional search, and search with trouble.

At first, we show what happens to LaCoS when Mars requires all of contexts. Then, Mars issues a query to a built-in LaCoS and get the result as follows. From this result, Mars could have known that "*Cygnus is discussing LaCoS*" and "*Antares is explaining about LaCoS*".

Table 1. Context Table and its Attributes

Attribute name	Attribute type	Explanation
robot	text data	the name of robot
context	text data	key word of context
pos	real number	position of robot
time	timestamp	creation time

Table 2. Initial CTable

robot	pos	context	time
mars	0.0	Introduce partner mars	15:20
cygnus	5.0	Discussion about LaCoS	15:00
antares	100.0	Explain about LaCoS	15:15

```
mars> select robot, pos, context from CTable;
-----------------------------------------
robot    pos   context
mars     0.0   Introduce partner mars
cygnus   5.0   Discussion about LaCoS
antares  100.0 Explain about  LaCoS
```

Secondly, we show what happens to LaCoS when Mars requires all of contexts that are located less than 10 meters from Mars. Then, Mars issues a query to a built-in LaCoS and get the result as follows. From this result, Mars could have known that "*only Cygnus is near Mars*" and the result didn't include the inappropriate context of Antares. This result shows that LaCoS can remove inappropriate contexts.

```
mars> select host, pos, context \
from CTable where pos < 10.0;
-----------------------------------------
robot    pos   context
mars     0.0   Introduce partner mars
cygnus   5.0   Discussion about LaCoS with mars
```

Finally, we show what happens to LaCoS when Mars requires all of contexts when a robot is in trouble. Assume that a trouble occurred at Cygnus and therefore Cygnus isn't connected to network. We simulate the network trouble of Cygnus by stopping its LaCoS daemon. In this case, the result of a query is influenced by the trouble. This example shows LaCoS performs even when network trouble exists. What we'd like to note here is that LaCoS is a distributed system and hence it is more durable compared with concentrated system architecture.

```
mars> select host, pos, context from CTable;
-----------------------------------------
robot    pos   context
mars     0.0   Introduce partner Mars
antares  100.0 Explain about  LaCoS
```

Fusing contexts(acceptance and rejection) In this paragraph we show two fusions of LaCoS. They are a acceptance of a fusion and a rejection of a fusion.

At first, we describe what happens to LaCoS when Mars requires to fuse its context with the context of Cygnus. Note that, a rejection of a fusion is not set on Cygnus in this case.

```
mars> fuse to cygnus into CTable (context) values \
      ('Discuss about LaCoS with Mars & Cygnus');
-------------------------------------------
Fusion is accepted
```

In this case, Cygnus accepts the fusion request. Therefore Mars can invite its partner to the discussion. After that, CTable is updated as follows.

```
mars> select host, pos, context from CTable \
      where time='latest';
-------------------------------------------
robot   pos  context
mars    0.0  Discuss about LaCoS with Mars & Cygnus
cygnus  5.0  Discuss about LaCoS with Mars & Cygnus
antares 100  Explain about LaCoS
```

Secondly and finally, we show what happens to LaCoS when Mars requires to fuse its context with the context of Antares. Note that, however, a rejection of a fusion is set on Antares in this case since Antares wants to continue the current context. In this case, the fusion request is rejected and CTables of Mars and Antares weren't changed.

```
antares> reject fusion request to CTable (context)
mars> fuse to antares into CTable (context) values\
      ('Explain about LaCoS with Mars');
-------------------------------------------
Fusion is rejected.

mars> select host, pos, context from CTable \
      where time='latest';
-------------------------------------------
robot   pos  context
mars    0.0  Discuss about LaCoS with Mars & Cygnus
cygnus  5.0  Discuss about LaCoS with Mars & Cygnus
antares 100  Explain about LaCoS
```

4.2 Scalability

We had two experiments to evaluate scalability of LaCoS. In the first experiment, we evaluated execution time for a query that searches contexts when the number of robots increases. And in the second experiment, we evaluated the execution time of a query when the number of queries issued simultaneously increases in the condition that all of the queries are issued to the same robot. We used four machines which are Mars, Cygnus, Antares and the machine that has the same spec as Antares.

For these two experiments, we stored 10000 contexts in each LaCoS. To simplify experiments, we used text data such as "1", "2", . . . , "9999" and "10000" as context and queries that partially matches between contexts. And a query is set to search one of the contexts randomly. We measured an execution time of a query 100 times and calculated the average.

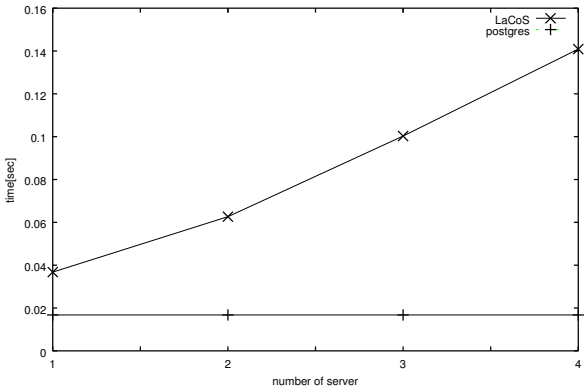

Figure 4. The relation of number of LaCoS and searching time

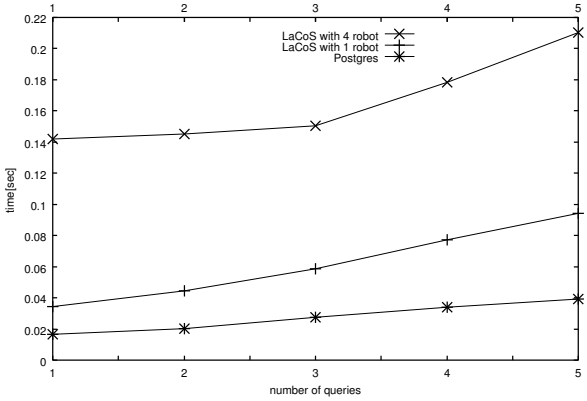

Figure 5. The relation of number of queries and its search time

The result of the first experiment is shown in Fig. 4. In Fig. 4, the x axis shows the number of LaCoS and the y axis shows the search time for a query. And Fig. 4 shows as the number of LaCoS increases, search time of a query increases. The result of PostgreSQL in Fig. 4 shows the execution time of the context data manager component in LaCoS.

Fig. 4 clearly indicates that LaCoS can be useful for a few robots, but it doesn't have high scalability. When the number of robots is four, LaCoS returns the result of a query within 0.14 sec. In case of one robot, LaCoS needed about twice as much search time as PostgreSQL. And with four robots, LaCoS needed about seven times as much costs as PostgreSQL.

The result of the second experiment is shown in Fig. 5 and Fig. 6. In both of Fig. 5 and Fig. 6, x axis shows the number of queries issued simultaneously. In Fig. 5, y axis shows search time with four robots, one robot, and PostgreSQL. In Fig. 6, y axis shows the search time of

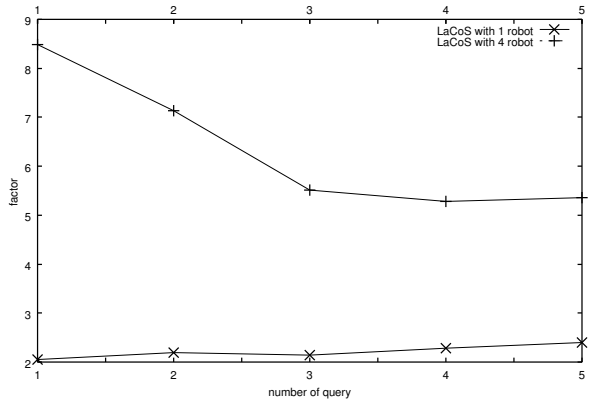

Figure 6. The relation of number of queries and factor compared with PostgreSQL

LaCoS divided by the search time of PostgreSQL.

Fig. 5 seems to indicate that LaCoS with four robots has few scalability. However, Fig. 6 indicates that LaCoS with four robots has more scalability compared with one robot. In Fig. 6, as the concurrency of queries increases, the difference of factors becomes smaller. When the concurrency of the query was one, the difference of the factor was 6.5, but when the concurrency was five, the difference decreased to 3.6.

The reason why this phenomenon occurred is because of applying broadcast messages. LaCoS makes connection in the order of return messages. The order depends on network and the host condition, therefore it may be changed at the query time. That is why it showed higher utilization in case of four robots than in case of one robot.

5 Conclusion

In this paper, we proposed the context management system in a sensor-robot network called LaCoS. LaCoS was designed to achieve four functions, (1) context searchable, (2) context updatable, (3) context fusible and (4) context rejectable. To support (1), LaCoS provided *"SELECT"* operation. To support (2), LaCoS provided *"INSERT"* operation. To support (3), LaCoS provided *"FUSE"* operation. And to support (4), LaCoS provided *"REJECT FUSION"* operation. It should be shown from the results of experiments that LaCoS equips four functions and has higher durability, but a few scalability compared with concentrated architecture.

There are several future works for LaCoS. First, we should introduce the context division function to LaCoS, which is a reversing idea of context fusions. It is also future work to solve few scalability. And we must improve the performance. Then, we must introduce LaCoS to Robovie[5] to evaluate effects of LaCoS, because we evaluated the performance of LaCoS without motor control modules and sensor modules which require high CPU power. When these works were done, robots in the same context would share sensor data from a sensors network and the sensors equipped by the robots to accomplish their interactions. And finally, we will create applications of communication robot with LaCoS.

References

[1] Hiroshi NOGUCHI, Taketoshi MORI, and Tomomasa SATO, Network Middleware for Utilization of Sensors in ROOM, *Proc. of the 2003 IEEE/RSJ International Conference on Intelligent Robots and Systems*, Las Vegas, USA, pp. 1832-1838, 2003

[2] Stephen Peters and Howard E. shrobe, Using Semantic Network for Knowledge Representation in an Intelligent Environment, *In PerCom '03: 1st Annual IEEE International Conference on Pervasive Computing and Communications*, Ft. Worth TX, USA, March 2003.

[3] PostgreSQL, *http://www.postgresql.org*

[4] Abraham Silberschatz, Henry F. Korth, and S. Sudarshan, SQL, *DATABASE SYSTEM CONCEPTS 4TH EDITION*, (New York, McGraw-Hill, 2002) pp. 135-187

[5] T. Kanda, H. Ishiguro, T. Ono, M. Imai, and R.Nkatsu, Development and Evaluation of an Interactive Humanoid Robot "Robovie", *IEEE International Conference on Robotics and Automation*, Washington D.C., USA, pp. 1848-1855, 2002