Proceedings of the Fifth IIEEJ International Workshop
on Image Electronics and Visual Computing 2017
Da Nang, Vietnam, February 28- March 3, 2017

DEVELOPMENT OF SAKE BREWING ENTREPRENEURS SUPPORT SYSTEM IN FUKUSHIMA

Kunio Ohno† Masatoshi Hiroura††† Biro Attila†††

†Monavis IT Consulting Co. ††FB Triangel Co. †††ITware Kft.

ABSTRACT

We have developed a prototype of simple semantic model for the M2M/IoT system which can be used for small or middle sized enterprises, and will be suitable to support sake brewing entrepreneurs in Fukushima. The original basic system is Kojimori which is a cloud based sensor data management system and has already been introduced to several dozen sake brewery plants. We made an analysis to sake brewery process and created the activity−diagrams based on UML, then created a class−diagram named "Sake factory ontology". We implemented the class−diagram to a simulation program of CLOS which is very suitable to implement class−diagrams for small or middle sized semantic IoT system. The system has moved accurately and the effectiveness of the system has been confirmed. We are developing to decode the C++ based system from the CLOS system for the practical application.

1. INTRODUCTION

We have studied new business development by woman entrepreneurs in Fukushima[1][2]. This paper describes an example to support sake brewing entrepreneurs by M2M, IoT technology. The original system is named Kojimori[3]. The system we are going to develop is a semantic model system based on the Kojimori. Koji(麹) means the molt in Japanese. Kojimori means to protect, preserve, and embrace the molt. Kojimori has been developed by FB Triangle, and the system is illustrated as Figure 1.

Kojimori system is organized by sensor devices, data acquisition units (DAQs), 3G routers, OYASAI IoT Server, and customer equipment terminals of PCs, smart phones, and tablet PCs. Sensors are connected to DAQs by cables or wireless systems. Every DAQ has 3G router and the data is transmitted to the OYASAI IoT Server under the Internet Cloud. Every sensor date is stored in the database of OYASAI IoT Server, and accessed by customers' terminals. A few dozen Kojimori systems have already been introduced to Sake brewing companies, which contribute to save labors of sake factory. Especially, master brewers (Toji(杜氏) in Japanese) welcome Kojimori, because the temperature management of Koji is very important and sensible. Kojimori enables to know the temperature from distant places, then many master brewers were pleased to introduce Kojimori to their factory.

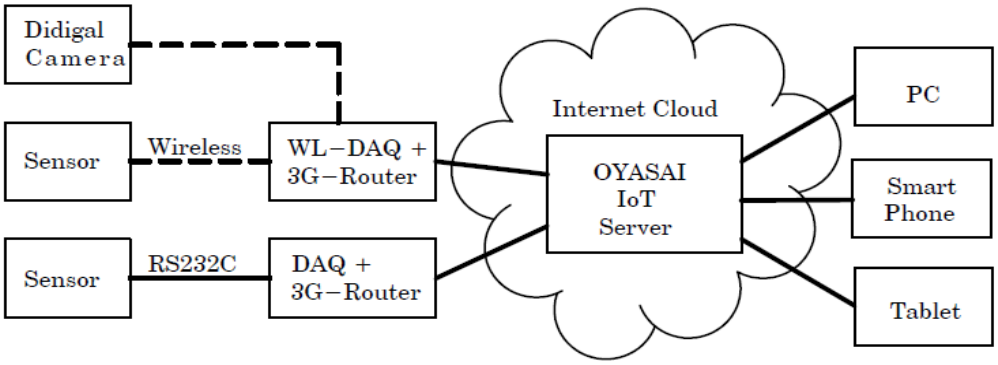

Figure 1 Outline of Kojimori System

2. ANALYSIS OF NEW REQUIREMENT

2.1 Semantic model requirement

We interviewed several master brewers at Aizu-wakamatsu city in Fukushima, and understood that more intelligent and organized system should be required. Then we started to create a semantic ontological model of Sake brewing process which may contribute to the development of new business related to local communities. Sake brewing process has been analyzed through existing document.[4] The process can be classified into four phases of preparation, initial−brewing, post−brewing, and completing. In preparation phase, rice is polished, washed, steamed and boiled.

In initial−brewing phase, some boiled rice is fermented by source molt, then Koji is produced. Both produced Koji and another boiled rice are mixed, then Shubo(酒母) is created under certain conditions. After the creation of Shubo, certain process named Shikomi(仕込) generates Moromi(醪) from Shubo during post−brewing phase. In completing phase, Moromi is separated to liquid of Sake and solid residue of Kasu(粕).

2.2 Initial brewing phase analysis

The activity−diagram of initial brewing phase is illustrated in Fig.2. In the figure, ellipses show objects, while rectangles show processes. The names of the objects and processes are written in Romaji from Japanese technical terms, because there are no suitable english terms in Sake brewing area. Then Japanese terms are written in parentheses. Mushi−mai（蒸米） is the boiled rice which is the original material of Sake. Some Mushi−mai rice called Koji−mai（麹米） is used to create Koji with Tane−koji（種麹） of source molt. It needs two or three days to create a lot of Koji through the Koji−fermentation process from Koji−mai. Temperature management is very important through the process and Kojimori has been introduced and shown the effectivity by many customers.

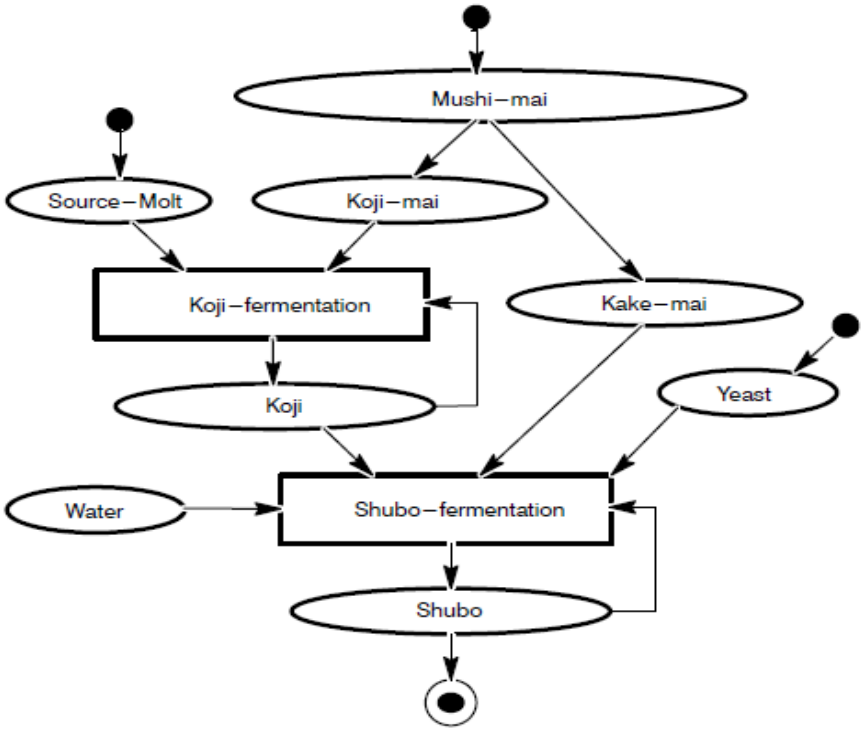

Fig.2 Activity−diagram of Initial Brewing Phase

2.3 Post brewing phase

Activity−diagram of post brewing phase is shown in Fig.3. After Shubo production, both Koji and Kake−mai（掛米） are added to original Shubo. The process to add Koji and Kake−mai is called Shikomi（仕込）. which includes three stages of Hatsu−zeo(初添), Naka−zoe(中添), and Tome−zoe(留添) before, though 4th−stage has included recently in order to improve and control the quality.

Through the effort, we heard various requirements and the solution will be a semantic model of Sake brewery process in the M2M, IoT System.

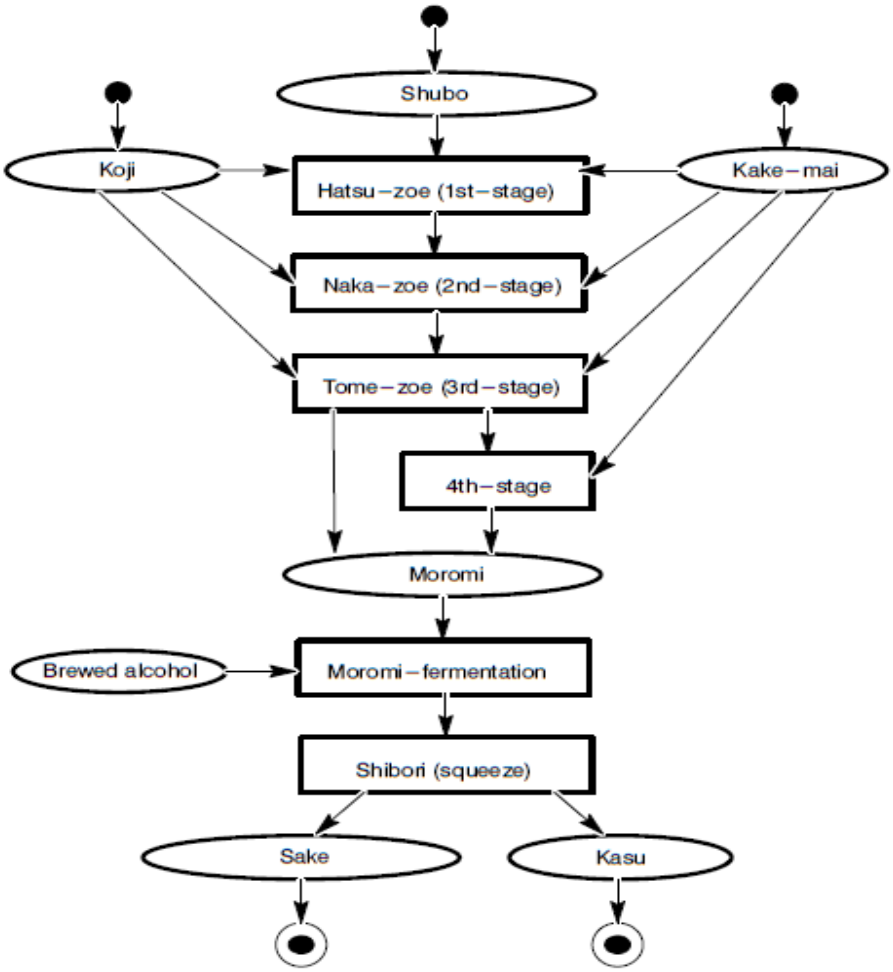

Fig.3 Activity−diagram of Post Brewing Phase

3. CLASS−DIAGRAM DESIGN AND IMPLEMENTATION TO CLOS

3.1 Hierarchical class−diagram

Hierarchical class−diagram is illustrated in Fig.4, which is designed and created from the activity−diagram shown in Fig.2 and 3. Each box represents a class, while each arrow shows the relationship between the classes, and the directed class is the super class of the start point class. By the way, from the view point of the directed class, start point class of the arrow is its subclass. Generally, superclass has more abstract or preceding concept than its base class. This relationship organizes a semantic relationship. Through the hierarchical diagram, it can be found that almost everything related to Sake is made from rice, because the superclass of almost every class is the “rice class” at the top of the hierarchical class tree. Only two classes of “source molt” and “yeast” are exception.

Then the class hierarchy seems to outline and show semantic meaning. In detailed view, for example, class “koji” is the subclass of both “koji−mai” and “source−molt” classes based on Fig.4. In the activity−diagram of Fig.2, Koji is produced through the Koji−fermentation process with Koji−mai and Source Malt.

Each class generally has its name, instance−variables, and methods as the element of programing language. Then, class “koji” is described shown in Fig.5 by UML or traditional object analysis and design template format.

3.2 Class−diagram implementation to computer

Class−diagram is very important to implement a system to computers. A class is usually the concept of something which has a certain name[5]. Then the classes represent general names of natural language in a certain aspect, and the class name corre-

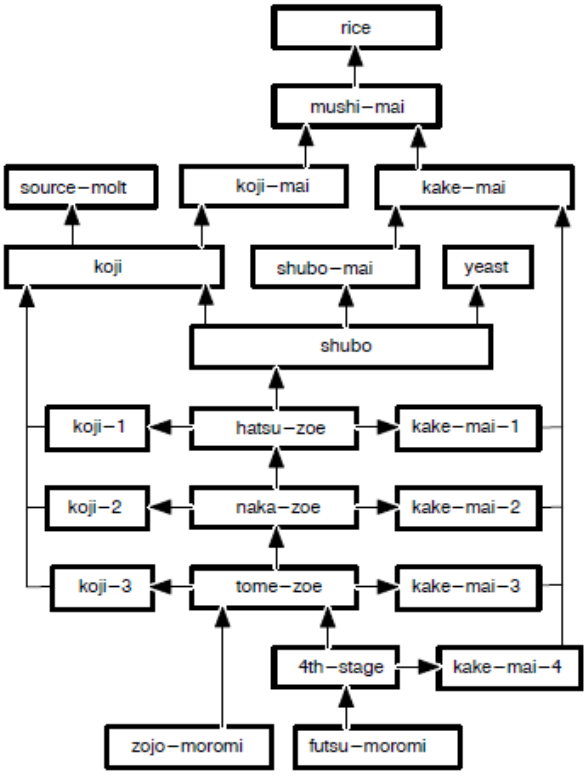

Fig.4 Class−diagram

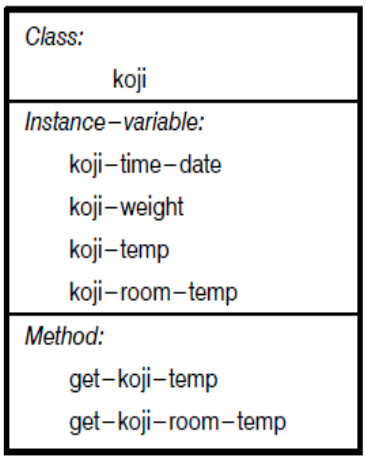

Fig.5 Class template of koji

sponds to the name (noun) of the concept. The instance−variable of a class relates to the property or attribute of the class. If we suppose the class name to the name of some concept, which means that class names are nouns of natural language, instance−variable should be some adjectives or adverbs of natural language to the class name which corresponds to a noun.

Therefore, class−diagram should be designed based on natural language. Then class−diagram should be independent of programming language or markup language as XML. When we implement the class−diagram to computer, there exist variations depend on the language. If the language were XML, the class should be based on OWL, because the concept of the class based on natural language mentioned above is the semantic concept of ontology. To realize OWL ontology from the class−diagram, such a ontology editor as Protege will be convenient. However suitable general practical tool has not developed yet.

If the language is programing language, class inheritance specification is important. There are two inheritance mechanisms of single inheritance and multiple inheritance. Java, Smalltalk, and Objective−C support single inheritance, while Common Lisp (CLOS) and C++ support multiple inheritance.

According to the class−diagram of Fig.4, several classes have multiple superclases, then multiple inheritance system will be suitable for the class−diagram. When comparing CLOS and C++, CLOS is much suitable for prototyping the class−diagram. Then we have chosen CLOS to implement the system.

3.3 CLOS program implementation

Though Common Lisp was standardized initially in 1984, this version did not include object system, because there were several object systems of lisp dialect at that time. The second version was standardized in 1990, which included object system.[6]

The object system of Common Lisp (CLOS) was much refined and improved compared with the previous various ones. The merit of CLOS is generic function mechanism based on generalized object model. The syntax of the method in any class is completely the same as general functions, which is called as generic function. This mechanism effectively realized the polymorphism, which enabled the semantic sense naturally.

Class definition of the CLOS is made by the class−name, instance−variables, and optional parameters as initial value , default value, access function name, etc. Class koji of Fig.5 is defined as Program−1.The program code can be created by the template of Fig.5, except the accessor functions. The template is very convenient to implement the class−diagram of a prototype system. This class definition method through templates is suitable for

```
(defclass koji (koji−mai source−molt)
    ((koji−time−date :accessor koji−time−date?)
     (koji−weight :accessor koji−weight?)
     (koji−temp :accessor koji−temp?)
     (koji−room−temp :accessor koji−room−temp?)))
```

Program−1　Class−definition of koji in CLOS

not only in CLOS programing but also C++ programing, because the concept of class definition and creation is almost similar. But in case of Java and Objective−C is something different, because they do not support multiple−inheritance.

CLOS is generally much more flexible than C++, then is convenient to implement a prototype system from class hierarchy diagram as class−diagram of UML. Method of "get−koji−temp" is also implemented as Program−2.Every method of CLOS

```
(defmethod get−koji−temp ((obj koji))
  (let* ((old−data (slot−value obj 'koji−temp))
         (add−data
          (list (get−time)
            (progn
              (format t "Input koji−temperature:  ")
              (read−from−string (read−line))))))
    (setf (slot−value obj 'koji−temp) (cons add−data old−data))))
```

Program−2　Method definition of get−koji−temp in CLOS

must be a generic function of Common Lisp, then the syntacs of the method is completely the same as general functions of Common Lisp specification.

3.4 Classes for initial brewing phase

Classes for initial brewing phase are described in Table 1. and 2. Table 1 shows the classes of rice, mushi−mai, kake−mai, source−molt, and koji of the class−diagram shown as Fig.4. This table can be

Table 1　Classes of initial brewing phase (Process for koji production)

Class−name	rice	musi−mai	kake−mai	source−molt	koji
Superclass		rice	musi−mai		koji−mai source−molt
instance variable	product−name rank refine−percentage refine−date wshing−date boil−date	(mix−in)	kake−mai−time−date kake−mai−weight	molt−name molt−maker molt−prod−date molt−purch−date	koji−time−date koji−weight koji−temp koji−room−temp
method					get−koji−temp get−koji−room−temp

thought as the template summary of classes. Class definition template of koji in Fig.5 is just the right most item of Table 1. The class musi−mai contains neither any instance−variable nor any method. Such empty classes are called as "mix−in"[7], because the role is just programing convenience. In this case, the instance−variables of the rice and musi−mai is the same, but the semantics of rice and musi−mai (boiled rice) is different. Then musi−mai is newly defined. If some factors like washing time or steam condition might be needed, new instance−variables can be added in the musi−mai class. Likewise, the classes of koji−mai and shubo−mai are also mix−in. Class koji−mai may manages

the information for class koji, koji−1 through koji−3 in future. Therefore the concept is different from the class mushi−mai, then newly defined.

Table 2 Classes of initial brewing phase (Process for shubo production)

Class-name	koji-1	koji-2	koji-3	yeast	shubo
Superclass	koji	koji	koji		koji shubo-mai yeast
instance variable	koji-time-date1 koji-weight1 koji-temp1 koji-room-temp1	koji-time-date2 koji-weight2 koji-temp2 koji-room-temp2	koji-time-date3 koji-weight3 koji-temp3 koji-room-temp3	yeast-name yeast-maker yeast-prod-date yeast-purch-date	shubo-time-date shubo-water shubo-temp shubo-room-temp shubo-bome shubo-tou shubo-san shubo-amin
method	get-koji-temp1 get-koji-room-temp1	get-koji-temp2 get-koji-room-temp2	get-koji-temp3 get-koji-room-temp3		get-shubo-temp get-shubo-room-temp get-shubo-bome get-shubo-tou get-shubo-san get-shubo-amin

Table 2 shows the classes of koji−1 through 3, yeast and shubo, which are related to the koji fermentation process and the shubo fermentation process in Fig.2. Subscript 1〜3 of the koji1 through 3 correspond to the processes of Hatsu−zoe (1st stage), Naka−zoe (2nd stage), and Tome−zoe (3rd stage). Class shubo is mix−in as described in the previous sentence, which is a subclass of kake−mai. The concept of shubo−mai is different from kake−mai.

3.5 Classes for post brewing phase

Classes for post brewing phase are described in Table 3. and 4.Table 3 shows the classes of kake−

Table 3 Classes of post brewing phase (Process for moromi production)

Class-name	kake-mai-1	kake-mai-2	kake-mai-3	kake-mai-4	hatsu-zoe
Superclass	kake-mai	kake-mai	kake-mai	kake-mai	shubo koji-1 kake-mai-1
instance variable	kake-mai-time-date1 kake-mai-weight1	kake-mai-time-date2 kake-mai-weight2	kake-mai-time-date3 kake-mai-weight3	kake-mai-time-date4 kake-mai-weight4	hatsu-time-date hatsu-water hatsu-temp hatsu-room-temp hatsu-bome hatsu-tou hatsu-san hatsu-amin
method					get-hatsu-temp get-hatsu-room-temp get-hatsu-bome get-hatsu-tou get-hatsu-san get-hatsu-amin

mai1 through 4, and hatsu−zoe, which are related to produce the origin of moromi. By the way, the classes of koji1 through 3 are also related to this process, which have already been prepared in previous process. The class hatsu−zoe is the first stage of Sandan−Jhikomi (three stages preparation). Sandan−Jikomi (三段仕込) is the traditional japanese sake completing process, which ferments the shubo adding koji and kake−mai step by step three times as a rule. The names of three stages are hatsu−zoe (初添), naka−zoe (仲添), and tome−zoe (留添). But recently, one stage named yondan (4段：4th stage) has been added for futsu−shu (普通酒) case. For zojo−shu (増醸酒) case, 4th stage is not applied. Table 4 shows the classes of naka−zoe, tome−zoe, 4th−stage, futsu−moromi, and zojo−moromi. Many methods are implemented in those classes, because the management of moromi state is essential for the sake product quality.

4. RESULT AND THE FUTURE PLAN

4.1 Simulation result

Table 4 Classes of post brewing phase (Process for shikomi phase)

Class-name	naka-zoe	tome-zoe	4th-stage	futsu-moromi	zojo-moromi
Superclass	hatsu-zoe koji-2 kake-mai-2	naka-zoe koji-3 kake-mai-3	tome-zoe kake-mai-4	4th-stage	tome-zoe
instance variable	naka-time-date naka-water naka-temp naka-room-temp naka-bome naka-tou naka-san naka-amin	tome-time-date tome-water tome-temp tome-room-temp tome-bome tome-tou tome-san tome-amin	4th-time-date 4th-water 4th-temp 4th-room-temp 4th-bome 4th-tou 4th-san 4th-amin	futsu-time-date futsu-water futsu-temp futsu-room-temp futsu-bome futsu-tou futsu-san futsu-amin	zojo-time-date zojo-water zojo-temp zojo-room-temp zojo-bome zojo-tou zojo-san zojo-amin
method	get-naka-temp get-naka-room-temp get-naka-bome get-naka-tou get-naka-san get-naka-amin	get-tome-temp get-tome-room-temp get-tome-bome get-tome-tou get-tome-san get-tome-amin	get-4th-temp get-4th-room-temp get-4th-bome get-4th-tou get-4th-san get-4th-amin	get-futsu-temp get-futsu-room-temp get-futsu-bome get-futsu-tou get-futsu-san get-futsu-amin	get-zojo-temp get-zojo-room-temp get-zojo-bome get-zojo-tou get-zojo-san get-zojo-amin

The prototype system has completely been implemented by CLOS. The preparation phase and the completing phase are simple while the initial brewing phase and the post−brewing phase are rather complex.

The necessary data through the four phases are stored in the instance of futsu−moromi or zojo−moromi, at the lowest class of the CLOS program. The system has been simulated based on the existing data.

Though there exist some insufficient classes and methods, the system moves accurately and the effectiveness of the system has been confirmed.

The class diagram of Fig.4 and the related information of Table 1 through Table 4 organizes compact semantic information for small or middle sized sake brewing enterprises, which organize an ontology of sake−brewing process domain. Then we call those information as “Sake factory ontology”, which “Tsukuri−zakaya Ontology（造り酒屋オントロジ）” in Japanese. By the way, “Tsukuri−zakaya” means small or middle sized sake brewing factory.

4.2 Further development

The concept of ontology is a semantic model of a certain system in its domain. Then the ontological model helps to implement the IoT system, because the ontological model is semantically similar to the real world process and data. And the model is independent of programing language and data format.

Though the prototype is written in CLOS, practical system should be written in C++, because existing OYASAI IoT Server is written in C and C++ language. Then a possible development may be to decode the CLOS codes to C++, because C++ supports multiple inheritance as CLOS. However, C++ object is not suitable for persistent data which is necessary for practical system and should be stored in an appropriate database.

5. DISCUSSION

Our goal is to establish and support certain IoT system suitable for small or middle sized enterprise based on Kojimori. This prototype system to Sake brewing business is the first possible trial to make a simple semantic model for the IoT system.

Semantic systems should be designed and modeled through the object analysis and design methodology[8], and the Universal Modeling Language (UML) has been one of the concluded standard. Though strong trend of recent semantic system implementation may be based on the web ontology language OWL, the tools and the programing method is rather complicated, then other methodology based on object−oriented programing language should be selected.

Among various languages, CLOS was chosen and was suitable for our purpose. The key for semantic systems should be class hierarchy and the CLOS is much simpler then another method. Through the development of this system, the merit using CLOS

were proven, and the possibility for small and middle sized enterprise IoT system should be confirmed.

Practical method to create an ontological semantic prototype system is as follows.

(1) To create a conceptual hierarchical class−diagram for the prototype system.

(2) To create the class templates which contain class−names, instance−variables, and methods for all classes.

(3) Based on the templates, to create the class table and confirm the relationship among the classes.

(4) Based on the class table, to implement class definitions through CLOS.

(5) To write the code of related methods.

(6) To write contextual simulation program and try to activate the system.

From now on, semantic IoT system based on its domain ontology will be needed for various business field. The prototype development steps mentioned above will widely be applicable for various area. We believe that semantic Kojimori based on this concept will encourage entrepreneurs in local sake brewing communities.

Especially recent entrepreneurs may require such computer skills as fintech, artificial intelligence, deep learning, user experience, and so on. Semantic prototyping skill which we described in this paper will develop those skills, and it will contribute to the businesses he or she plans, tries, and engages in.

6. CONCLUSIONS

We have developed a prototype of simple semantic model for small scale M2M/IoT system which can be used for small or middle sized enterprises, and will be suitable to support sake brewing entrepreneurs. We made an analysis to sake brewery process and created the class−diagram of “Sake factory ontology”.

We have developed class templates for the classes of the class−diagram, extended the templates to a set of class tables, and implemented to a simulation program of CLOS. Our system has been operated accurately and the effectiveness of the system has been confirmed. For small or middle sized semantic IoT systems, class definition of CLOS with class tables based on the class−diagrams, which represents the physical semantic meaning of the existing field system and organizes an ontological concept seems very useful.

Prototype developing method we have tried will contribute to entrepreneurs who are interested in semantic IoT systems and to develop the simulation program for their businesses.

7. ACKNOWLEDGEMENTS

Authors would like to thank Ms. Hajdu Csilla of previous ITware manager for her effort to initiate and organize this work, and also thank to Ms. Keiko Hayashi of Suzunoe Shuzo, and Mr. Shin−ichiro Nakajima of Nakurayama Shuzo for giving us chances to visit their facilities and informing us practical brewing process of sake.

REFERENCES

[1] Kunio Ohno, Mikiko Watabe, Mitsuko Nishiguchi;”A STUDY ON THE HUMAN RESOURCE DEVELOPMENT FOR ENTREPRENEURS TOWARD FUTURE NETWORK SOCIETY”, Proc. 4th IIEEJ International Workshop on Image Electronics and Visual Computing Koh Samui, Thailand, (2014.10)

[2] Kazunori Akutagawa, Kunio Ohno,Mitsuko Nishiguchi ; ”Human Resource Development of Woman Entrepreneurs in Fukushima with Intercultural Historical View”, Proc. SIETAR Europa 2015 Congress (2015.5)

[3] Kunio Ohno, Biro Attila, Hajdu Csilla, and Masatoshi Hiroura; “Manufacturing and Market Development of M2M/IoT Product Kojimori with Intercultural Effort”, IPSJ SIG Report, Vol 2015−DC−98 No.17 (2015.7)

[4] Nihon Jozo Kyokai; “Saishin Shuzo Dokuhon”, Zaidanhojin Nihon Jozo Kyokai, (1980)

[5] Peter Wegner; “Concepts and Paradigms of Object−Oriented Programming”, OOPS MESSENGER − A Quarterly Publication of the SIG Programing Language, Vol.1, No.1 (1990.8)

[6] Guy L. Steele Jr.; “Common Lisp the Language − Second Edition”, Digital Equipment Co. (1990)

[7] Daniel Weinreb, David Moon; “Lisp Machine Manual − 4th Edition”, MIT Artificial Intelligence Laboratory, (1981.7)

[8] Ivar Jacobson, Magnus Christerson, Patrik Jonsson, Gunnar Overgaard ; “Object−Oriented Software Engineerig − A Use Case Driven Approach”, ACM press (1992)