International Journal of Electronic Commerce Studies
Vol.6, No.2, pp.163-186, 2015
doi: 10.7903/ijecs.1329

IOT-ENABLED KNOWLEDGE SHARING-BASED COLLABORATIVE SOFTWARE MAINTENANCE DESIGN APPROACH

Rui-Yang Chen
University of Aletheia
No.32, Zhenli St., Danshui Dist., New Taipei City 25103, Taiwan (R.O.C.)
a168.cloudy@msa.hinet.net

Chao-Tsong Fangtsou
National Taipei University
151, University Rd., San Shia District, New Taipei City, 23741 Taiwan (R.O.C.)
ctfang@gmail.com

ABSTRACT

Modifying corrective programming code according to requirements modifications so that the modifications work with existing code is a challenge for software maintenance design, because it is difficult to understand the meaning of existing code. This refers to both software specifications and business implications associated with modifying a software system. The lack of a knowledge-based collaborative environment is one of the prominent problems in software maintenance design. This paper presents the IoT-enabled knowledge sharing architecture using combination and internalization for collaborative software maintenance design to derive a collaborative software design procedure from the problem-solving systematic methods TRIZ, AHP and Maturity Index on Reliability (MIR). Next, we illustrate three elements of the proposed approach in more detail: combination using MIR, collaborative tagging decision-making using AHP, and internalization of problem-solving using TRIZ. Finally, a case study and evaluation are presented to demonstrate a practical application of the proposed approach.

Keywords: Collaborative Software Maintenance Design, Knowledge Sharing, TRIZ, Agent-based, Internet of Things

1. INTRODUCTION

Software development means the creation of a new software system based on the software development life cycle. One could argue that software development quality and productivity suffers from requirements modification, however these requirements are even more difficult to fulfill during software maintenance design. For example, issues such as application domain, system architecture, programming language and software development environment, etc. all affect the ability of the software development team to remain true to evolving system requirement, especially as the software system undergoing maintenance ages over time.

Modifying corrective programming code according to requirements modifications while ensuring compatibility with existing programming code is a challenge for software maintenance design, because it is difficult to understand the meaning of existing programming code. Thus, software maintenance design is recognized as the most costly and lengthiest process in a typical software development life cycle[1]. This refers to both software specification and the business problems associated with modifying a software system because of correcting problems or adapting it to a changed software or hardware environment. So, software maintenance design is a knowledge intensive process that involves designing a specialized knowledge-based software solution for a specific business problem.

Lack of knowledge is one of the prominent problems in software maintenance. To address this problem, we adapted a knowledge sharing methodology to the specific knowledge needs associated with software maintenance. As some surveys show, knowledge management is one of the central topics in management research[2]. The ability to acquire knowledge in the software maintenance design process depends on how easily knowledge can be accumulated and shared[3]. Knowledge sharing is important for companies to be able to acquire knowledge. Software product companies are embracing the practice of knowledge sharing within virtually all phases of the software life cycle. Integration of technical knowledge with business application domain knowledge is crucial to high quality software maintenance.

Knowledge-based software development is widely discussed in several software engineering publications[4, 5, 6], but current practice also increases the knowledge needed to achieve or surpass software quality goals. Few articles discuss both the knowledge sharing and software architecture of software maintenance design, including the interactions of individual software components and the coordination among related modules in a software development life cycle.

Software architecture describes the overall organization of a software system in terms of the inter-relationships of its internal modules and data elements[7]. Developing knowledge-based software architecture therefore involves multiple tasks that require a group of agents to collaborate in order to solve the common requirement of software maintenance. Without knowledge sharing, it is difficult to share simultaneously what we know about coordinating the multitude of interactions created by maintenance design tasks. We address this problem by allowing software maintenance design processes to capture the knowledge about the software architecture from various sources using knowledge sharing as described by Nonaka[8].

Meanwhile, we utilize the TRIZ-based[9] problem-solving methodology to provide problem-solving guidance for designing software maintenance for existing systems using collaborative software networks. The current slow start of e-commerce can be attributed to the fact that it suffers from the same problems troubling electrical commerce[10]. Collaborative software design is a new concept that provides automated support for the software maintenance design process and optimizes the software development process[11]. Shen et al.[12] proposed applying collaborative engineering to product design. Collaborative design is the process that allows stakeholders with multiple perspectives to share their knowledge about software design. During the collaborative design process, related stakeholders often use knowledge to exchange design elements such as program code and technical documents.

Software development design is a collaborative process which utilizes knowledge sharing[13, 14]. Because of the need to work together to explore software design concepts, the software development teams, business users, and technical specialists (the stakeholders) must be capable of discovering effective ways to share knowledge[15]. The software design process could be plagued by quality issues[16], so Shahla et al.[17] presented a competing model of knowledge sharing that helps promote knowledge-sharing in software development teams to proactively mitigate the inadvertent introduction of quality problems during the design process.

The drawback of the traditional software design process is that designers fail to create shared understanding about design knowledge. Previous research has proposed collaboration and knowledge sharing for software development[18, 19]. Knowledge management literature showed the importance of the quality of the knowledge sharing[20]. Furthermore, knowledge transfer is required for successful knowledge sharing[21]. Knowledge sharing aims to promote the preservation of knowledge within an organization. Diego et al.[22] proposed a method for semi-automatic discovery of a knowledge-intensive process. For this reason, the

knowledge-sharing while using collaborative design can be considered an important mechanism.

In this article, the knowledge-sharing process is very important and challenging, since it heavily involves distributed locations and interaction with multiple human/physical devices among the stakeholders. A network-organization consists of some new type of sub-networks around which stakeholders organize themselves so as to reach a common objective. Due to the lack of appropriate technological support, network organizations cannot effectively implement collaborative and reconfigurable processes when their stakeholders work at some distance from each other. An emerging strategy that might enable industries to cope with rapidly changing product specifications is the implementation of reconfigurable processes[23].

Internet of Things (IoT) enabled network organizations allow stakeholders to have immediate interaction with knowledge and support the connection of physical objects in distributed locations. This immediacy of vital knowledge is especially critical when the problems that cause the failure of software are complex and multi-disciplinary. Nieto[24] analyzes the role of different types of collaborative networks in achieving product innovation. Nam[25] presented a business-aware framework for developing RFID business applications on the EPC Network.

As the IoT evolves and matures, the next-generation internet will provide for harmonious collaboration among "smart things" and humans. The main objective in the IoT is that any "thing" (object tagged) may be able to integrate with any other object[26], which may contain small chips or embedded systems depending on its intended purpose[27]. The IoT explores how enhanced and complex interaction and communication between physical objects, such as transformed digital objects, can result in the automation and streamlining of many activities.

Luo et al.[28] proposed an algorithm known as TSOIA, which divides perception nodes into three groups to search the IoT for global optimal solutions. The IoT is a technological revolution in advanced computing with networked smart devices and digital objects[29]. Many sensor networks, such as pattern recognition, need to handle physical objects that move during the course of certain manufacturing or industrial processes[30]. The traditional view of IoT had explored connecting all physical objects to create a device-based IoT. The diverse features of devices present the opportunity to understand and model specific aspects of the interactions between humans and physical things.

The absence of a collaborative software network for digital objects makes it difficult for these digital objects to interact with other devices or embedded systems. In this article, we describe significant research efforts to define a robust IoT architecture, mainly from an agent-enabled thing-oriented perspective. Our research involves using a collaborative software design network to embed knowledge about maintenance design processes using collaborative tagging and software agents. Collaborative tagging, known as "Social Bookmarking," is stored on the server-side instead of the client-side. Recently, server-side mechanisms have gained popularity. The collaborative tagging technique is one of the important technologies developed to extend web applications with collaborative qualities. Software agents have been utilized as one approach for the development of collaborative engineering environments. This paper presents agents for software maintenance design within collaborative design environments.

Baig[31]presented multi-agent[32] based systems for protecting critical infrastructure. A robotic sensor agent-based architecture is proposed for monitoring territorial security[33]. These agents can participate in a network routing knowledge-sharing design and play a key role in collaborative design. Wang[34] proposed a new novel business model using agent-based services which efficiently integrate IoT services into business applications for 3rd party service. This shows that the efficiency of such IoT-enabled collaborative software design networks will enable the collaboration of heterogeneous networked embedded devices, both among themselves process and with external stakeholders.

New technologies such as collaborative tagging enable the creation and dissemination of knowledge in a distributed software development design network. This paper presents a novel approach to automating the bookmark decision-making process, aimed at recommending the higher priority bookmark as collaborative tagging for coordinating the interactions of maintenance design tasks when faced with complex software maintenance efforts. Software agent technique is one of the important technologies developed to enable web applications for collaboration. We focus on some specific function of personal web applications for the software maintenance design process, namely agent-based web applications, in order to enable a collaborative software maintenance design process. This process is presented in the context of a case study application.

The main purpose of this paper is to present the knowledge sharing architecture for a collaborative software maintenance design network in order to achieve combination and internalization. The rest of the paper is organized as follows. In Section 2, we present the software agents TRIZ and

AHP as they relate to this study. In the section 3, we present the proposed knowledge sharing procedure. First, we present the IoT-enabled collaborative software design network and analyze the proposed procedure according to a defined knowledge sharing architecture for collaborative software maintenance. Next, we discuss in detail the three phases of the proposed procedure: (1) combination using maturity index on reliability (MIR); (2) collaborative tagging decision-making using AHP; and (3) internalization of problem-solving using TRIZ. In the final two sections, a case study is discussed to illustrate the proposed procedure, and a concluding summary is provided.

2. RELATED WORK

2.1 Agent-oriented

According to Shoham[35], Agent-Oriented Programming (AOP) is seen as an improvement and extension of Object-Oriented Programming (OOP). A software agent is an autonomous object that performs tasks on behalf of the user[36]. Agent-Oriented Software Engineering from Wooldridge is being described as a new paradigm for the research field of Software Engineering [37]. Agents are similar to objects, but they also support structures for representing mental components. Agent-oriented programming is explained as an iterative improvement of the proven methodology of object-oriented programming.

2.2 Knowledge Sharing

A body of knowledge is normally built bottom-up from data to information, culminating in knowledge at the top of the hierarchy. Knowledge sharing happens when the experience and know-how of an organization has an impact on another department. Knowledge can be shared both tacitly and explicitly. Tacit knowledge sharing occurs without anyone directly articulating the knowledge acquired. Explicit knowledge sharing occurs when one party shares knowledge with another party about a specific practice which has been proven successful. Nonaka[8] proposes a framework for knowledge sharing illustrated in Figure 1.

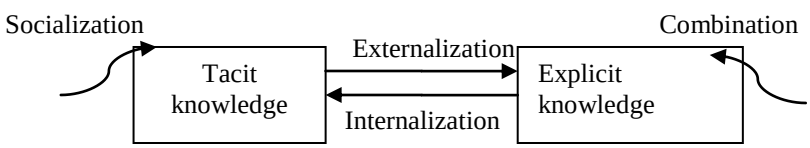

Figure 1. Knowledge sharing according to Nonaka[8]

Socialization is the process of sharing tacit knowledge within a group of knowledge workers. *Externalization* is the process of converting tacit knowledge into explicit knowledge. Through externalization, a knowledge worker may express what he knows, and this knowledge may then be circulated among a group. *Combination* is the process of combining various sources of explicit knowledge to create a new one. *Internalization* is the process of converting explicit knowledge into tacit knowledge. In software maintenance, there may not be many examples of this as creating explicit documentation is not often performed. As mentioned above, tacit knowledge is deeply rooted in the complex nature of software maintenance. Thus, the knowledge conversion from combined explicit knowledge to tacit knowledge is crucial. In this paper, we explore how combination and internalization affect knowledge sharing behavior in software maintenance.

2.3 AHP

The analytic hierarchy process (AHP) method was developed by Saaty[38]. It is a hierarchy process to solve complex problems involving multiple attributes by constructing the problem into goal, attribute and alternatives for the decision-maker. The evaluation of business alternatives is a multiple decision-making problem. AHP is used to assign relative weights to the performance indices. This method is based on pair-wise comparison between criteria and alternatives. The relative importance among the alternatives is not the same, and priority weights are chosen on a selective basis. Thus the following method is used to create the pair-wise comparison matrix.

2.4 TRIZ

The theory of inventive problem solving (abbreviation derived from the Russian title: TRIZ) was developed to provide access for engineers to the knowledge of inventors. TRIZ provides an inventive process to solve any given problem. Creativity is understood as actions that improve the ability to define the problem and to generate new ideas from the TRIZ matrix. It is integrated with knowledge for design engineers to handle conflicting situations during the inventive design problem solving process. The core components of TRIZ are the contradictions, 40 inventive principles, and the matrix. A contradiction means that a worsening engineering parameter results in an improving engineering parameter. There are 39 engineering parameters represented in a 39x39 matrix denoting the 1482 contradiction types.

2.5 EPC/EPCIS

An electronic product code (EPC) is a universal electronic code that provides a unique identifier for an embedded item or process in a supply

chain. EPCs are also used to create the radio-frequency identification (RFID) tags that make it possible to register related data such as product and lot number for triggered events. Triggered events consist of internal and external events which electronically move the tagged items in the supply chain. An EPC information system (EPCIS) collects and filters EPCs, making them available for correlation with RFIDs. An EPCIS aims to enable EPC-related data transferring within and across the supply chain using the EPCglobal standard[39]. EPCglobal allows for sharing of information related to product problems by integrating RFID technology into the EPC standard. From a business perspective, an EPCIS warehouses EPC information and makes it available for supporting business applications which manage products central to a company's operation. The EPCIS application is an information repository which captures events and tasks facilitating integration with data queries from RFID/NFC middleware. Thakur et al.[40] proposed a new methodology for modeling traceability information using the EPC information system (EPCIS) framework.

3. METHODOLOGY

Knowledge sharing is the major application for a collaborative software design network (CSDN), which exploits stakeholder mobility for software development and their creative nature to transmit software R&D experience based on IoT. To facilitate the development of IoT-enabled CSDN, a generic system framework is essential. This framework should provide a set of mechanisms for a systematic network, software design behavior analysis and knowledge sharing among agent-enabled objects. We propose a conceptual framework for IoT-enabled CSDN systems in Figure 3. This framework comprises the following four modules: distributed devices, software design procedures, intelligent collaborative agents, and knowledge sharing network. The knowledge sharing network is essential to enhancing the linkage and interaction with each of the software design stakeholders. These stakeholders have different benefits and objectives because of different backgrounds and project roles. For example, end users desire specific functions to meet business needs, while programmers focus on solving technical problems and completing program codes. As a result, the framework allows an autonomous robot to transfer knowledge, resulting in the ability of the multi-agent system to autonomously react to decisions in the collaborative software design network.

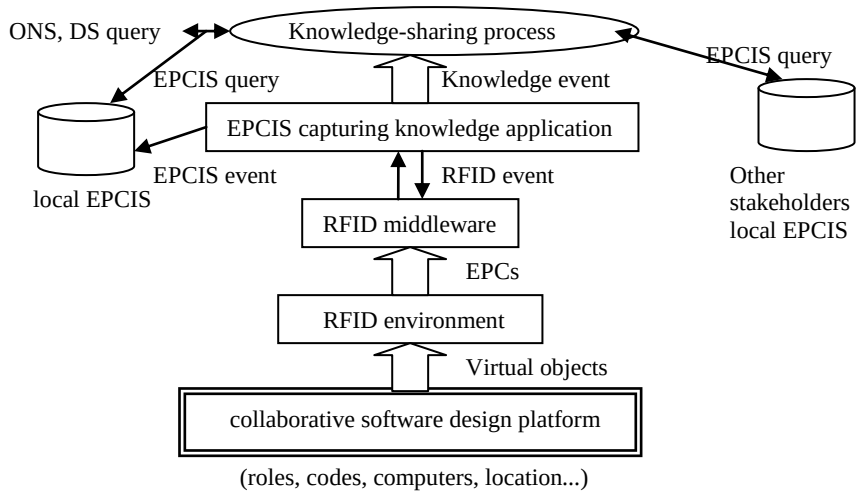

Figure 2. The knowledge-sharing process on EPC network

We define the term "virtual object" as a digital object comprising a series of interactive objects for the collaborative software design network. The interaction between virtual objects and physical objects can be very useful for automating activities. For this purpose, the virtual objects must be able to integrate autonomously using a multi-agent mechanism. A virtual object may have a series of embedded actions, which are used to integrate connected sensors to get location and position data. The virtual objects reference multiple data elements stored in an EPC repository database. These data may be processed by a collaborative software design process that contains the design logic. Using the IoT-enabled collaborative software design network, the virtual objects should be able to integrate and interact with other physical objects, devices and stakeholders.

The EPC Network is an open global standard for RFID. Figure 2 shows the knowledge-sharing system on the EPC network. The RFID middleware collects and filters data and generates information after the EPCIS capturing knowledge application requests an RFID event from the RFID middleware. The EPCIS capturing knowledge application generates EPCIS events that contain data, information and knowledge and stores them in the EPCIS repository. New knowledge arising from innovation in the knowledge-sharing process must be transferred by stakeholders to the capturing knowledge application.

Currently, the complexity of development in the IoT-enabled collaborative software design network is increasing and evolving quickly, with the addition of physical devices and software architecture. Multi-agent middleware for the IoT-enabled network should be able to provide

mechanisms to support collaborative applications with a variety of objects in order to support interactions among multiple tasks or activities for the software development design process. Traditionally, middleware layers have been used to overcome the gap between physical layers and application layers and facilitate efficient communications in distributed networks. Our multi-agent middleware approach addresses collaborative software design in pervasive embedded networks by means of knowledge-sharing services.

The knowledge-sharing services allow creating and exposing the IoT-enabled network, which is the starting point for achieving the goal of efficient software development design. From the system architecture perspective, this IoT-enabled proposed network is a highly dynamic and distributed system of coordinating intelligent objects. These objects, acting as autonomous entities, will organize themselves into collaborative networks for knowledge sharing and perform software design tasks. Therefore, this requires the application of a multi-agent approach to the software design process. Moreover, a multi-agent service will not only coordinate among intelligent objects of the IoT, but also integrate efficiently the IoT architecture into the software development design process.

The architecture we propose relies on an agent-based web application with combination and internalization in the collaborative software maintenance design process for knowledge sharing as illustrated in Figure 3. Knowledge sharing involves transferring knowledge from one specific context to another. Combination is the mechanism of combining several pieces of explicit knowledge to create a new one, and internalization is the mechanism by which one transfer tacit knowledge.

In this architecture, we address the MIR process and TRIZ analysis of existing collaborative software maintenance design. By using the MIR process to effect combination, a system analyzer (SA) may express explicit knowledge (e.g., writing a manual), and this knowledge may then be combined with user requirements. TRIZ provides an process for extracting knowledge to solve any software maintenance problem. By using TRIZ analysis as a mechanism for internalization, explicit knowledge of the system analyzer is transferred to tacit knowledge of the system designer (SD). Conflicting problems are handled with parameters during the contradiction matrix analysis. For the knowledge sharing of this software maintenance, we define collaboration as using agent-based web applications to communicate with each other's actions during the collaborative software maintenance design procedure. RIA-based web applications communicate the software maintenance design procedure to the web client using collaborative tagging, but keep the bulk of the data back on the application server. In contrast to one-word tags, we use XML (extensible markup

language) tags describing name, actions, role, problem, and requirement of the software maintenance design procedure in this paper.

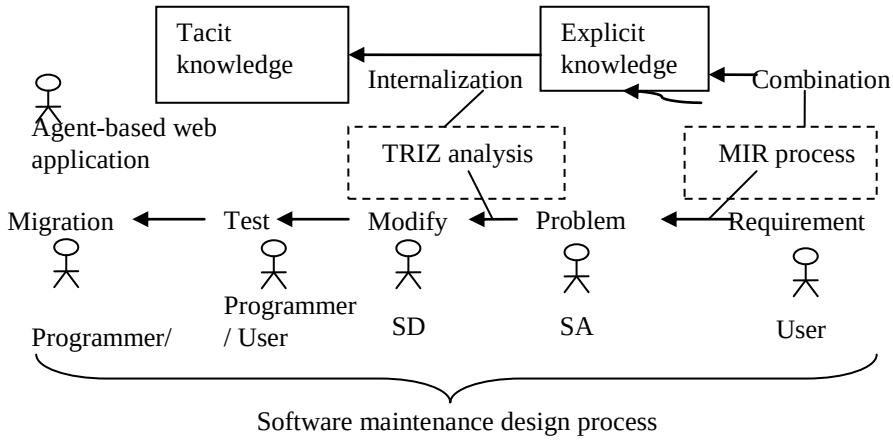

Figure 3. Knowledge sharing architecture

Based on the above analysis, the following stepwise motivational development is proposed using internalization and combination to motivate knowledge sharing in collaborative software maintenance as described in Figure 4. This approach consists of combination using MIR, collaborative tagging decision-making using AHP, and internalization of problem-solving using TRIZ phases. The combination phase is to establish integrated customer requirement feedback into software maintenance design as explicit knowledge using the MIR process, which is utilized for the agent-based web application of combining explicit knowledge into collaborative tagging in order to achieve a collaborative mechanism. The collaborative tagging decision-making phase is to make decisions among some collaborative tags, which are utilized in the AHP process of considering both software system and business requirements in order to derive the weights of the higher priority collaborative tagging. The internalization of problem-solving phase is used to extract explicit knowledge for the higher priority collaborative tagging of converting problems encountered in software maintenance design experiences into creative solutions using TRIZ, which is utilized for problem-solving in order to derive tacit knowledge.

3.1 Combination Using MIR

The customer feedback phase is used to understand green product problems for the user. The MIR scale uses four levels to assess the quality of information in identified loops. Based on the above, an MIR model was developed for the software maintenance design process.

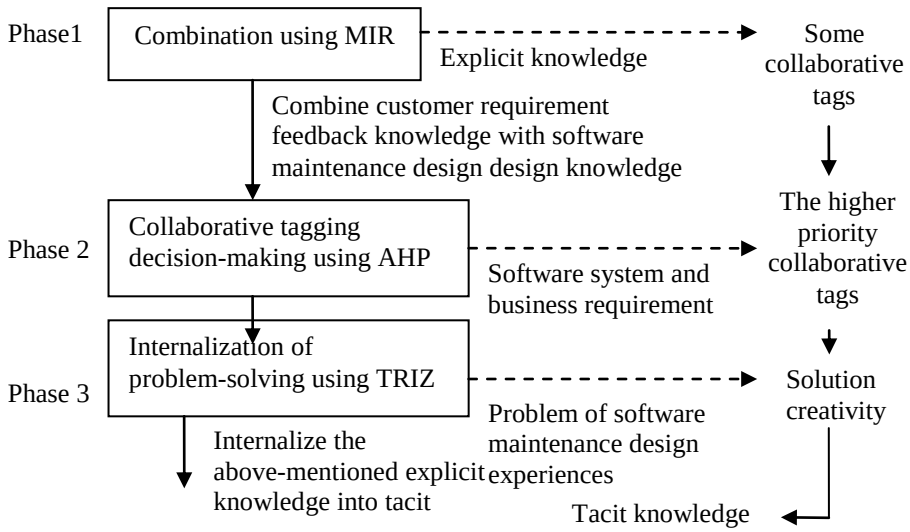

Figure 4. The procedure of the proposed Knowledge sharing architecture

We use the user requirements and software systems knowledge to combine the most valuable explicit knowledge. Using the MIR model it is possible to not only extract the problem-solving knowledge, but also combine the knowledge with experiences derived from related roles in the software maintenance design process.

The four level MIR scale is described as follows:

MIR level 1 indicates adequate measurements: What is the scope of the software maintenance design problem for the user's requirement? The programmer has quantitative evidence of the process output in terms of the maintenance design problem, but the origin of the problem is unknown.

MIR level 2 indicates the cause of the problem: The relevant maintenance design problem with sufficient information to determine some causes can be identified. The system designer has quantitative evidence of the process output, knows the origin of the problems (such as coding errors), but does not know what actually causes the problems.

MIR level 3 find root-cause: The inference root-cause of the software maintenance design can be determined for the mentioned problem. The system analyzer has quantitative evidence of the user's requirement, knows the origin of the problems, knows what actually causes them, and is able to solve to problems, but is not able to prevent similar events from happening in the future.

MIR level 4 indicates preventive measurements: Adequate information is available to analyze the solution. The feedback-based model using MIR is

given in Figure 5. Combining problem-solving knowledge can be used to analyze and solve the software maintenance design problem.

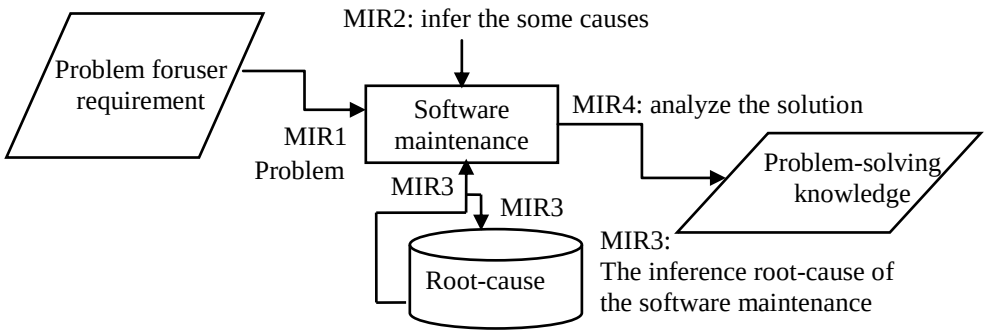

Figure 5. The feedback-based model on MIR

3.2 Collaborative Tagging Decision-making Using AHP

We show that integrating collaborative tagging into knowledge sharing in the software maintenance design using the AHP method takes into account the higher priority collaborative tags. The AHP deconstructs the knowledge sharing objective into a hierarchical structure of criteria and alternatives as shown Figure 6. The knowledge sharing objective can be divided into two sub-objectives: software system and customer requirement. Knowledge sharing must integrate the collaborative mechanism into the software maintenance process.

From the viewpoint of the collaborative mechanism, this knowledge sharing can be divided into two tasks: software system and customer requirement. The software system includes coding, migration and testing. The customer requirement includes application domain, modification analysis and function training. Therefore, the software system was categorized as technical knowledge criteria and the customer requirement as business knowledge criteria. The AHP method was used to determine the relative importance of the knowledge variables such as software system and customer requirement. The weight of each decision element was derived using eigenvalue computations. Three alternatives such as collaborative tagging were used to evaluate the relative importance of these variables. Data were gathered by interviewing the evaluators using a questionnaire with a scale of importance which included choices of 1(Equally), 3(Weak), 5(Strong) and 7(Extreme). Expert choices were used as input for the AHP method, which consists of prioritizing collaborative tagging to represent the knowledge sharing object in the software maintenance process.

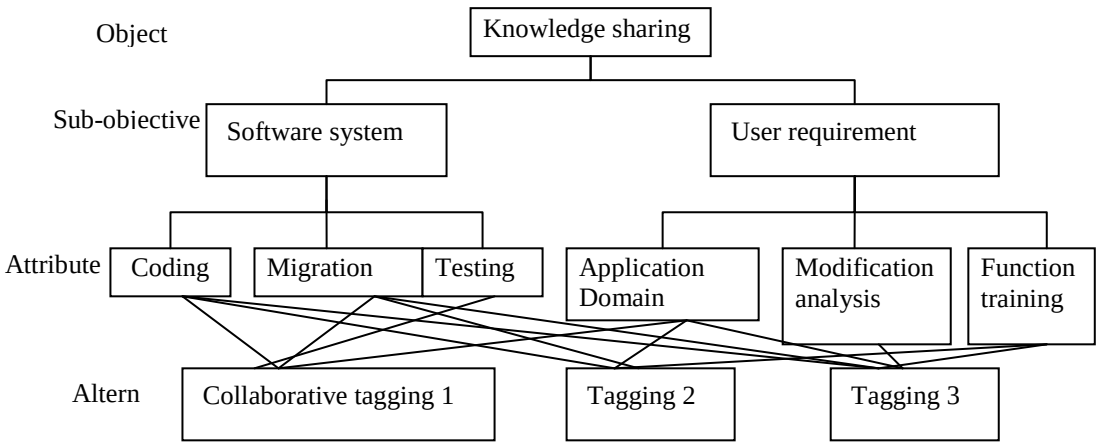

Figure 6. AHP for knowledge sharing

3.3 Internalization of Problem-solving Using TRIZ

In addition to the customer requirement emphasized by the customer complaint, we also highlighted the need to consider the cause of the software maintenance design problem. The problem elements could be represented as follows. A problem item is described by a collection of the error. The error is described by a collection of the code bug, and each error is classified into only one of the mutually exclusive data based on the cause of the problem. From the above elements of the problem, one key point of the software maintenance design problem for classifying the cause of the problem could be found. These contradictory terms of increasing user requirement complexity and increasing maintenance design reliability are accompanied by the need to shorten the maintenance design times. The above elements of the problem have identified four major problem-efficiency elements for the TRIZ matrix which are integrated with the 39 engineering parameters in order to internalize the above-mentioned explicit knowledge into tacit knowledge.

We present the software maintenance design problem-solving using the TRIZ approach. It is different from the traditional TRIZ approach which is mainly applied to product engineering problems. However, this approach does not consider the services nature of the problem. As each element is identified or more elements are identified simultaneously, it produces high problem-solving efficiency. To integrate problem-efficiency elements into the TRIZ matrix, we illustrate the relationship between all elements of problem-efficiency and the 39 TRIZ engineering parameters in the Table 1. This value is also used to illustrate the relative importance of these elements and is desired to be lower for higher relative importance.

Table 1. Table for relationship of problem-efficiency elements

TRIZ engineering parameters	Problem-efficiency elements		
	Error	Bug	Cause
1 Weight of moving object 2 Weight of non-moving object	A. error of the problem	B. code bug of the error	C. cause of the problem

Following the steps shown in the Figure 7, the service-based TRIZ approach is shown as follows:

Step1. Find problem-efficiency elements

When fixing a problem during software maintenance, the solution of the problem should be considered as creatively as possible. However the problem cannot easily support a creative approach if the only solution considered is a strict problem-solving method. The usual problem solution is to address the defect with an engineering trade-off. The creative solution should be to classify the relationship between the type of problem and the design engineering parameters.

Step2. Find corresponding engineering parameters for TRIZ via Table1

At this step, the specific problem of designing a solution could be generalized to an abstract engineering problem from the contradiction matrix. Therefore, utilizing Table 1 the corresponding TRIZ engineering parameters can be found. In this case, engineering parameters come from task division. For example, one of the main problems in software maintenance design was the programming code context was too difficult to understand because of poor documentation. Accordingly, the product development task during which the documentation is developed to comply with the software specification standards must be revised. We can divide the task into some parameters such as "length of coding template," which transfers from the parameter "length of moving object."

Step3. Decide fuzzy relative importance using IF set

Quite often, it is easy to obtain the parameters when there is only a contradiction. However, if there are several parameters, the fuzzy relative matrix can assist the designer in acquiring the most favorable parameters. The fuzzy relative weight indicates that the IF set can be used to represent relative importance. The IF set comes from the previously mentioned problem formulation.

Step4. Examine and obtain the favorable inventive principles

From the corresponding parameters, the contradiction matrix suggests some inventive principles which are supposed to be useful in solving the problem. These suggested principles provide useful rule and hints to finding a problem-solving solution. By doing so, designers could more easily produce creative solutions.

Step5. Design consideration for problem-solving

As the principle is to prefer creative solutions to prevent customer complaint problems from recurring, giving the relationship between elements of problem-efficiency and TRIZ engineering parameters which are the closet match for the designers can be advantageous. That is why classifying the cause of the problem according to Table 1 is helpful. Therefore, the designers, according to TRIZ, associated with efficient problem solving can provide very helpful input on how to speed up creative designs for resolving the customer complaint. Therefore to use the service-based TRIZ as an internalization method in maintenance design project, we need to not only extract implicit knowledge, but also transfer tacit knowledge.

Because of this, software maintenance design is already more difficult to resolve with quality than the original software design process was. It follows that software maintenance design is difficult to manage using system architecture and instead is typically driven by reactive tasks. Reactive tasks came out as a reaction to the knowledge sharing that occurs when the system analysts gather explicit knowledge from user feedback and the experience of software engineers.

The analysis of the proposed TRIZ process provided the internalization for transferring explicit knowledge to tacit knowledge. It illustrates the main steps needed for all systematic knowledge-sharing analysis. Software maintenance design has to surmount several kinds of process problems. In this context, process is defined as an activity to accomplish a task. The transfer of TRIZ to the field of management is referred to as ‘service-based TRIZ’. Based on the intangible process, a three-step process to analyze software maintenance design problems in a well-structured form based on TRIZ was recommended. The first step was to apply TRIZ tools through direct analogy to non-technical problems. In a classical TRIZ interpretation, substances are any kind of physical product. In the following, substances are considered to be ‘tasks’ in the service-based TRIZ. A task includes explicit and implicit knowledge. They are fundamental for solving process problems. Explicit knowledge is of systematic structure and easily

transferable. Implicit knowledge applies to problem solving know-how with a less visible structure.

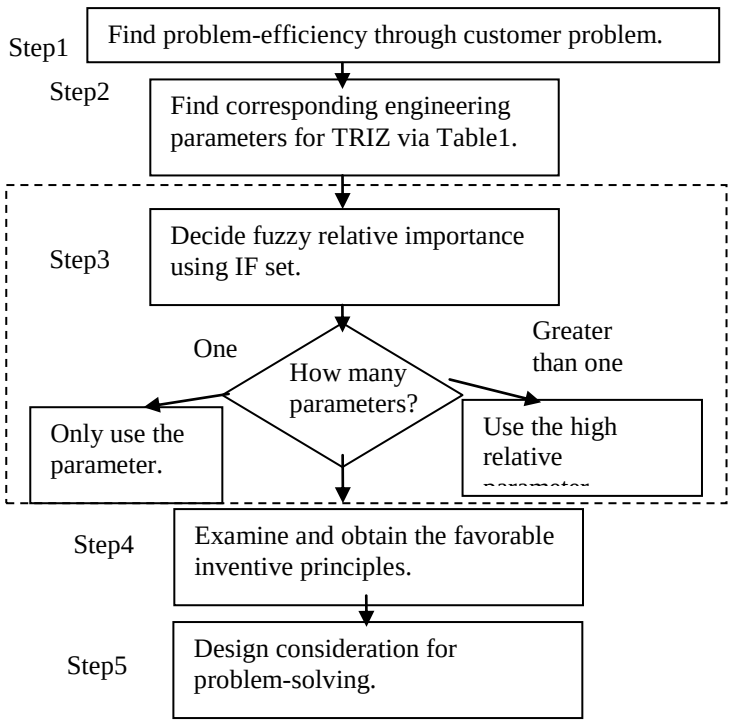

Figure 7. Flowchart of fuzzy problem-solving process

4. CASE STUDY AND EVALUATION

A case study concerning a software design company was used to illustrate the proposed architecture. Both experiments and their results will be discussed here. We conducted interviews for three small maintenance design projects with individuals who had tested well on related roles and were responsible for the maintenance design of legacy software systems. These three software maintenance design projects are based on enterprise resource planning (ERP) systems with customized programming for three different kinds of add-in localized modular business applications, such as bond, invoice and order entry process in Taiwan.

The three business processes aim at developing a software maintenance design to meet the company's special processing needs based on the legacy ERP system framework. In this experiment, data on the software maintenance design were gathered through existing projects. This experiment examined the test time reduction, gap degree elimination and prevention of problem recurrence in the collaborative software maintenance design procedure.

Collaborative software architecture (CSA) is an automated system that enables related roles (including system designers, system analyzer, programmer and user) to collaborate and interact in the software maintenance design process with collaborative tagging. The agent helps the related roles participate in collaborative tasks by advising the other roles, when needed, to take the appropriate tasks and create a collaborative tagging as a web application tool for collaboration. The collaborative tagging is to provide the related role with the ‘‘right’’ task at the ‘‘right’’ time.

The proposed architecture integrates related roles, agents, and tasks, which enables the intelligent assistant agent to effectively achieve the user’s requirement and software system design in CSA. There are two steps of operation offered by the agents based in CSA. Firstly, the agent-based web application provides information during the role’s interaction with a specific application. This information was represented by collaborative tagging. Secondly, the collaborative tagging supports the role’s collaboration in the maintenance design task. In addition, the web application exploits the knowledge sharing, which utilizes knowledge about the role’s current working applications.

Knowledge about the problem-solving is a necessity for related roles which are inactive in the collaborative software architecture. Our approach to using web applications for problem-solving in collaborative software maintenance design is to adopt XML-based tagging. The XML-based tagging within the context of maintenance design tasks is designed to support the agent’s knowledge sharing process. It aims at modeling a sequence of actions with explicit and tacit knowledge. The XML-based tagging consists of three components such as relationship, interaction and context components. The relationship component is capable of understanding relationship degree among collaborative tags. The interaction component provides four sub-steps: (1) to find the user’s requirement topics related to past problem-solving efforts; (2) to combine user feedback knowledge that the user might be interested in; (3) to extract tacit knowledge from problem-solving experiences; (4) to internalize tacit knowledge, such as problem-solving knowledge that contributes to finding a solution based on the agent’s coordination of the role’s experiences.

With regard to the above discussion, this study applies the concept of IoT-enabled knowledge sharing networks to the problem of collaborative software development design. The designed model for IoT-enabled networks consists of four dimensions: MIR1, MIR2, MIR3 and MIR4.

We surveyed a group of 50 individuals involved in three software development projects, yielding 30 usable responses. Survey questionnaires designed based on the MIR model were delivered to 30 select project

members through a brainstorm group discussion during which the questionnaires were explained and answered. Thirty valid questionnaires were analyzed for consistency index and ratio of each survey, and they are all less than 0.1. Table 2 lists the resulting weightings and rankings of the four dimensions and 12 key points from FAHP analysis. The MIR4 dimension is by far the highest weighted dimension with a weighting of 0.516, which accounts for over half of the total weighting of 1. This indicates that the critical software design relies heavily on the MIR4 to utilize knowledge-sharing to good effect. On the other hand, the MIR3 is the lowest-ranking dimension, with a weighting of 0.100. Under the MIR4 dimension, *prevent similar problems in the future* is the overall highest ranking with a weighting of 0.225.

Table 2. The resulting weightings and rankings

Dime-nsion	Weights	Key point	Weig-hting	Ranking
MIR1	0.150	How much software maintenance problem	0.115	3
		User requirement	0.035	9
MIR2	0.234	cause of the problem	0.114	4
		causes can be identified	0.070	6
		origin of the problems	0.050	7
MIR3	0.100	inference root-cause of the software design	0.035	9
		knows what actually causes	0.025	10
		solve problems	0.040	8
MIR4	0.516	prevent similar problems in the future again	0.225	1
		analyze the solution problem-solving	0.100	5
		knowledge base	0.175	2
		quantitative evidence of the maintenance	0.016	11

Table 3. A comparison of the proposed and traditional software design

	The number of occurring problem	Response time(hours)
The proposed design	21	0.61
The traditional design	30	1

The efficiency of the IoT-enabled collaborative software design network was also analyzed. We checked that this network improved the response time of the solution when encountering a contradiction because of a software design problem. The response time of the solution was reduced by 39% on average. We also found that the contradiction problem was reduced 30%, decreasing the number of occurring problems from 30 to 21 compared with the proposed software design and traditional design in the Table 3.

The traditional design is based on the normal software maintenance process. Such a software maintenance process is not considered to be IoT-based and a knowledge sharing environment. It mainly follows event-driven maintenance by programming code errors and partially corrects this maintenance problem according to personal IT technical experiences.

To demonstrate the efficiency of the proposed approach, we developed a software application with/without using CSDN and compared them with the knowledge-sharing application for software maintenance process. We selected two criteria such as maintenance time and the number of recurrences of the same problem. Further, to obtain a more accurate result, seven examples are implemented in each software maintenance program. Figure 8 shows a comparison of the maintenance time. It is seen that the software designer saves 49% of maintenance time in the CSDN approach.

Item	Average maintenance time (7 examples) (minutes)
CSDN approach	15
Non-CSDN approach	28
Saving time ratio	49%

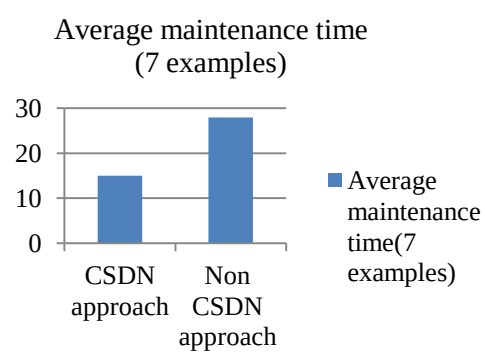

Figure 8. Comparison of maintenance time

5. CONCLUSION

This study focused on a knowledge sharing architecture for collaborative software maintenance. The architecture we propose relies on an agent-based web application with combination and internalization in the

collaborative software maintenance design process for knowledge sharing. This paper presents a novel approach to automate the bookmark decision-making process, aiming at recommending the higher priority bookmark as collaborative tagging to coordinating and interacting maintenance design tasks when facing multi-tasks of software maintenance. There are significant benefits among software development stakeholders that can explore how the collaboration is managed and knowledge sharing can be achieved. Collaborative multi-agents led to corresponding developments in the field of autonomous, ubiquitous and intelligent computing. Future work might examine the architecture for other case types. Studies in other knowledge-based problem-solving process would add to the proposed approach of the results.

6. REFERENCES

[1] N.F. Schneidewind, The state of software maintenance. *IEEE Transactions on Software Engineering*, 13(3), p303-310, 1987. http://dx.doi.org/ 10.1109/TSE.1987.233161.

[2] A. Serenko and N. Bontis, Meta-review of knowledge management and intellectual capital literature. Knowledge and Process Management, 11(3), p185-198, 2004. http://dx.doi.org/10.1002/kpm.203.

[3] L. Argote, B. McEvily, and R. Reagans, Managing knowledge in organizations: An integrative framework and review of emerging themes. *Management Science*, 49(4), p571-582, 2003. http://dx.doi.org/10.1287/mnsc.49.4.571.14424.

[4] K.C. Desouza, Barriers to effective use of knowledge management systems in software engineering. *Communications of the ACM*, 46(1), p99-101, 2003. http://dx.doi.org/10.1145/602421.602458.

[5] P. Robillard, The role of knowledge in software development, *Communications of the ACM*, 42(1), p87-92, 1999. http://dx.doi.org/10.1145/291469.291476.

[6] I. Rus, and M. Lindvall, Knowledge management in software engineering. *IEEE Software*, 19(3), p26-38, 2006.

[7] M. Shaw, and D. Garlan, *Software architectures: Perspectives on an emerging discipline*. USA: Prentice-Hall, 1996.

[8] I. Nonaka, and Hirotaka Takeuchi, *The knowledge-creating company*. USA: Oxford University Press, 1995.

[9] G. Altshuller, *And Suddenly the inventor appeared: TRIZ, the theory of inventive problem solving technical innovation center*. Worcester, MA: Technical innovation center Inc., 1996.

[10] K. Siau, and Z. Shen, Building customer trust in mobile commerce. *Communications of the ACM*, 46(4), p91-94, 2003. http://dx.doi.org/10.1145/641205.641211.

[11] L.H. Wang, W. M. Shen, H. Xie, and J. Neelamkavil, A. Pardasani, Collaborative conceptual design— state of the art and future trends. *Computer-Aided Design*, 34(13), p981-996, 2002. http://dx.doi.org/10.1016/S0010-4485(01)00157-9.

[12] W.M. Shen, Q. Hao, and W.D. Li, Computer supported collaborative design : retrospective and perspective. *Computers in Industry*, 59(9), p855-862, 2008. http://dx.doi.org/10.1016/j.compind.2008.07.001.

[13] Y. Fang, and D. Neufeld, Understanding sustained participation in open source software projects. *Journal of Management Information Systems*, 25(4), p9-50, 2009. http://dx.doi.org/10.2753/MIS0742-1222250401.

[14] K.D. Joshi, S. Sarker, and S. Sarker, Knowledge transfer within information systems development teams: Examining the role of knowledge source attributes. *Decision Support Systems*, 43(1), p322-335, 2007. http://dx.doi.org/10.1016/j.dss.2006.10.003.

[15] S. Sawyer, P.J. Guinan, and J. Cooprider, Social interactions of information systems development teams: A performance perspective. *Information Systems Journal*, 20(1), p81-107, 2010. http://dx.doi.org/10.1111/j.1365-2575.2008.00311.x.

[16] N. Gorla, T.M. Somers, and B. Wong, Organizational impact of system quality, information quality, and service quality. *The Journal of Strategic Information Systems*, 19(3), p207-228, 2010. http://dx.doi.org/10.1016/j.jsis.2010.05.001.

[17] G. Shahla, and D. A. John, Modeling high-quality knowledge sharing in cross-functional software development teams. *Information Processing and Management*, 49(1), p138-157, 2013. http://dx.doi.org/10.1016/j.ipm.2012.07.001.

[18] N. Levina, Collaborating on multiparty information systems development projects: A collective reflection-in-action view. *Information Systems Research*, 16(2), p109-130, 2005.

[19] L.G. Pee, A. Kankanhalli, and H.W. Kim, Knowledge sharing in information systems development: A social interdependence perspective. *Journal of the Association for Information Systems*, 11(10), p550-575, 2010.

[20] M.R. Haas, and M.T. Hansen, Different knowledge, different benefits: Toward a productivity perspective on knowledge sharing in organizations. *Strategic Management Journal*, 28(11), p1133-1153, 2007. http://dx.doi.org/10.1002/smj.631.

[21] A.A. Kane, Unlocking knowledge transfer potential: Knowledge demonstrability and super ordinate social identity. *Organization Science*, 21(3), p643-660, 2010. http://dx.doi.org/10.1287/orsc.1090.0469.

[22] D.C. Soares, F.M. Santoro, and F.A. Baião, Discovering collaborative knowledge-intensive processes through e-mail mining. *Journal of*

Network and Computer Applications, 36(6), p1451-1465, 2013. http://dx.doi.org/10.1016/j.jnca.2013.02.007.

[23] A.O. Oke, K.A. Hosseinm, and N.J. Theron, The design and development of a reconfigurable manufacturing system. *South African Journal of Industrial Engineering*, 22(2), p121-132, 2011.

[24] M.J. Nieto, and L. Santamarıa, The importance of diverse collaborative networks for the novelty of product innovation. *Technovation*, 27(6), p367-377, 2007. http://dx.doi.org/10.1016/j.technovation.2006.10.001.

[25] T. Nam, and Y. Keunhyuk, Business-aware framework for supporting RFID-enabled applications in EPC Network. *Journal of Network and Computer Applications*, 34(3), p958-971, 2011. http://dx.doi.org/10.1016/j.jnca.2010.04.021.

[26] J. Gubbi, R. Buyya, S. Marusic, and M. Palaniswam, Internet of thing (IoT): A vision, architectural elements, and future directions. *Future Generation Computer Systems*, 29(7), p1645-1660, 2013. http://dx.doi.org/10.1016/j.future.2013.01.010.

[27] M. Kranz, P. Holleis, and A. Schmidt, Embedded interaction: interacting with the internet of things. *IEEE Internet Computing*, 14(2), p46-53, 2010. http://dx.doi.org/10.1109/MIC.2009.141.

[28] S. Luo, X. Lu, and L. Cheng, TSOIA: An efficient node selection algorithm facing the uncertain process for Internet of Things. *Journal of Network and Computer Applications* , 36(2), p738-743, 2013. http://dx.doi.org/10.1016/j.jnca.2012.12.015.

[29] J. Pascual, O. Sanjuan, J.M. Cueva, B.C. Pelayo, M. Alvarez, and A. Gonzalez, Modeling architecture for collaborative virtual objects based on services. *Journal of Network and Computer Applications*, 34(5), p1634-47, 2011. http://dx.doi.org/10.1016/j.jnca.2011.04.002.

[30] C. Konstantopoulos, A. Mpitziopoulos, D. Gavalas, and G. Pantziou, Effective determination of mobile agent itineraries for data aggregation on sensor networks. *IEEE Transactions on Knowledge and Data Engineering*, 22(12), 1679-1693, 2010. http://dx.doi.org/10.1109/TKDE.2009.203.

[31] Z.A. Baig, Multi-agent systems for protecting critical infrastructures: A survey. *Journal of Network and Computer Applications*, 35(3), p1151-1161, 2012. http://dx.doi.org/10.1016/j.jnca.2012.01.006.

[32] S.U. Guan, and F. Hua, A multi-agent architecture for electronic payment. *International Journal of Information Technology and Decision Making*, 2(3), p497-522, 2003. http://dx.doi.org/10.1142/S0219622003000781.

[33] R. Abielmona, E.M. Petriu, M. Harb, and S. Wesolkowski, Mission-driven robotic intelligent sensor agents for territorial security. *IEEE Computational Intelligence Magazine*, 6(1), p55-67, 2011. http://dx.doi.org/10.1109/MCI.2010.939580.

[34] J.P. Wang, Q. Zhu, and Y. Ma, An agent-based hybrid service delivery for coordinating internet of things and 3rd party service providers. *Journal of Network and Computer Applications*, 36(6), p1684-1695, 2013. http://dx.doi.org/10.1016/j.jnca.2013.04.014.

[35] Y. Shoham, Agent-oriented programming. *Artificial Intelligence*, 60(1), p51-92, 1993. http://dx.doi.org/10.1016/0004-3702(93)90034-9.

[36] N. Boudriga, and M.S. Obaidat, Intelligent agents on the web: A review. *Computing in Science Engineering*, 6(4), p35-42, 2004. http://dx.doi.org/10.1109/MCSE.2004.13.

[37] M.J. Wooldridge, and N.R. Jennings, Intelligent agents: theory and practice. *Knowledge Engineering Review*, 10(2), p115-152, 1995. http://dx.doi.org/10.1017/S0269888900008122.

[38] T.L. Saaty, *The Analytic Hierarchy Process*. New York: McGraw-Hill, 1980.

[39] EPCIS Standard, *EPC Information Services Version 1.0.1 Specification*. Retrieved on September 21, 2007, from http://www.epcglobalinc.org/standards/epcis/epcis_1_0_1-standard-20070921.pdf.

[40] M. Thakur, C.F. Sorensen, F.O. Bjornson, E. Foras, and R.H. Charles, Managing food traceability information using EPCIS framework. *Journal of Food Engineering*, 103(4), p417-433, 2001. http://dx.doi.org/10.1016/j.jfoodeng.2010.11.012.