



# Code-Centric RFID System Based on Software Agent Intelligence

Min Chen, *Seoul National University*

Sergio González, *University of British Columbia*

Qian Zhang, *Hong Kong University of Science and Technology*

Victor C.M. Leung, *University of British Columbia*

*RFID technology could play a vital role in future smart-environment applications.*

*This code-centric RFID system uses software-agent-based intelligence to achieve faster service responses.*

**A**s an important type of wireless technology, RFID electronic identification ([www.rfid.org](http://www.rfid.org)) has increasingly received more attention in recent years. Many industries have used RFID technology to allow quick access to identification numbers or object codes. Current applications include item

tracking and tracing, inventory monitoring, asset management, supply-chain management, and e-healthcare.<sup>1-3</sup> In a typical system, which we characterize as an identification-centric RFID system (IRS), an RFID tag contains a simple antenna, a transponder, and a memory chip for saving ID information. Irradiating the transponder with electromagnetic waves enables a special RFID reader to obtain this information. Once an object's ID is verified, this object's profile can be retrieved from a database, and suitable service codes can be obtained. By following this centralized approach, the infrastructure recognizes the actions that must be performed with the object.

An IRS generally uses *passive information* about the object, such as identification and description information stored in

an RFID tag, and chooses courses of action by relying on a pre-established rule-based database. However, because this rule-based database is static, it cannot be updated in a timely fashion for new object types or environmental dynamics, thus causing a synchronization problem. Hence, an IRS has two disadvantages. First, the object's profile database must be set up in advance to allow for interactions between an RFID tag and a profile database. Second, when encountering a dynamic environment such as an emergency situation in which network access is difficult, the information stored in the database might be outdated or even unreachable, resulting in an operational delay or service failure.

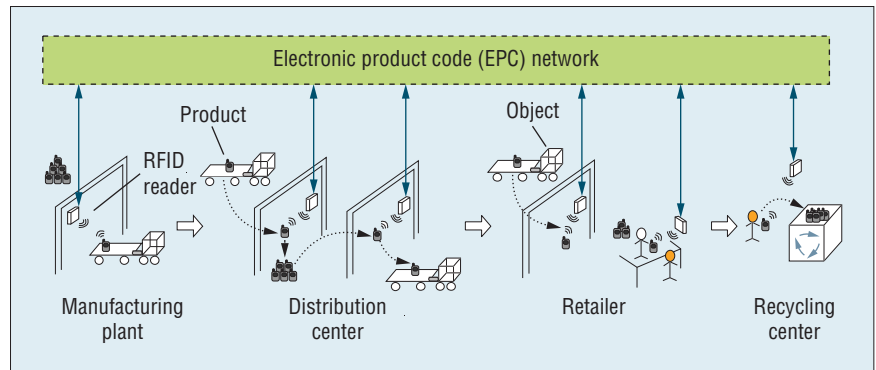
In this article, we propose a new code-centric RFID system (CRS) as a solution to

such problems. This CRS incorporates coded information that is dynamically stored in the RFID tag, making it easier for other systems to perform on-demand actions for different objects in different situations. To achieve this, we encode, in the RFID tag, mobile agents specifying up-to-date service directives that can be realized by intelligently handling various network dynamics. Because of the limited availability of memory in RFID tags,<sup>4</sup> we have designed a new compact, high-level language for coding these mobile agents. A corresponding middleware layer in the RFID reader can then interpret this language. These mobile agent-based RFID tags enable objects to instruct the system to intelligently execute actions when specific situations occur. Experimental results conducted on two testbeds confirm that the performance of our CRS is superior to that of an IRS.

The benefits of our CRS approach include the following:

- It effectively eliminates the need to retrieve associated RFID-code action information from a database and enables more robust systems for highly dynamic environments.
- It can greatly reduce overall system response time by retrieving service information from the tags.
- It helps improve the scalability of the database, which currently serves only as a backup subsystem.
- It fosters greater resilience because the system can remain operational, even during database or network failures.

In addition, there is a growing interest in integrating RFID with other technologies, such as wireless sensor networks (WSNs), Internet Protocol (IP) networks, and cellular networks, to develop scalable systems and applications



**Figure 1. Typical example of an identification-centric RFID system (IRS), as used by an entire supply chain process to help streamline the delivery of products and their appropriate disposal at a recycling center.**

that improve people's everyday lives. Such integration, coupled with continual advances in wireless communications technologies, could revolutionize a wide range of applications. By storing mobile codes in RFID tags, our CRS enables seamless integration with other systems.

### Identification-Centric RFID System

Most IRS applications are designed for object tracking, tracing, and locating ([www.autoidlabs.org](http://www.autoidlabs.org)). An example of an industrial application is tracing chemical containers stored in a warehouse. This could be unsafe for humans, owing to the intrinsic danger of exposure. Therefore, RFID tags can be attached to the chemical containers, and an RFID reader can be put on a shelf to trace them remotely.

RFID technology is becoming more widely used in daily life, such as in supply-chain management, where an IRS has proven to be an efficient approach. As Figure 1 shows, RFID can reduce information gaps by enabling real-time supply chain visibility, and is useful in purchasing, product manufacturing, shipping and receiving, storing, and selling. The example in Figure 1 also shows that it's possible to track a product's details throughout the entire supply chain process. When a product leaves the manufacturing

plant, an RFID reader obtains information, which it forwards to an electronic product code (EPC) network. When the product arrives at the distribution center or retailer, the information is stored in the EPC network's corresponding database. Eventually, after the consumer has finished using the product and disposes of it, a database connected to a recycling center could record this information.

In this application, the main functional components of the IRS are the following:

- *Rule database.* This component maintains a list of IDs and their associated rules, which the processing module accesses to formulate corresponding actions.
- *Processing module.* This module handles data-processing tasks after retrieving the incoming object's passive information from the rule database. After obtaining the corresponding rules, it checks whether any necessary condition is satisfied and performs the associated object's actions.
- *EPC network.* The EPC is a set of global technical standards aimed at enabling and sharing automatic and instant identification of items throughout the supply chain. A unique identifier of a physical object is stored in an individual RFID tag. The EPC we referenced in this

study was designed by the EPC Global Network ([www.epcglobalinc.org/home](http://www.epcglobalinc.org/home)). The EPC network has three main components: the object-naming service, the EPC information service, and the EPC discovery service.

However, a closer look at the operation of this IRS reveals some critical shortcomings. First, the ID embedded in the RFID tag doesn't directly convey any information about how to handle the object bearing this tag or what service should be applied to it. Second, any handling information must be retrieved from a database, and the data in this database is prone to becoming outdated if no manual updates are constantly performed. Third, scalability is a significant concern, given the growing number of RFID tags whose associated information must be stored in the database. Fourth, failures in the database or networking infrastructure could render the system unusable.

**Code-Centric RFID System**

In an IRS, the applications mainly help answer "where" questions—for example, where are my goods, or where is the object? Supply chain management is a typical example of this, in which the objects of interest can be conveniently tracked. However, the current status of the supply chain process and the service requirements of users or objects could change, calling for a smart environment with the intelligence to flexibly manage the quality of service (QoS). Such an environment could answer, not only the "where" questions, but also the "how" questions—for example, how to provide a certain service for a customer

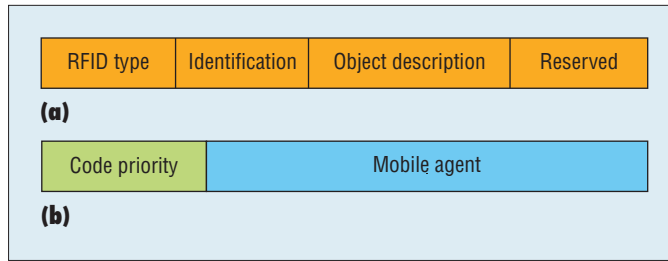
at a particular location under some specific conditions.

Our CRS meets this criterion. We call this system *code-centric* because the use of mobile-agent code is the most important feature of our system design. Introducing coded information is key to providing flexible and intelligent applications. By providing the flexibility to answer the "how" questions, the CRS facilitates a harmonious fusion of the user's requirements with the changing environment, and illustrates how RFID technology can improve the efficiency of the systems that would rely on it.

Moreover, while objects with the attached RFID tags move around, the stream-of-materials flow also provides the carrier with a code-information flow. Embedding service requirements into an RFID tag at one location can ensure that the objective is met at another location through code-centric processing. This is the rationale behind our CRS.

**Architecture**

Our proposed CRS has two main parts: the RFID tag and the code-processing and executing infrastructure in the corresponding tag reader. We introduced an extended RFID data format for the RFID tag (see Figure 2). In addition to traditional data (identification and description information, such as the object's shape, weight, color, production location, and time), the RFID tag can store a mobile agent.



**Figure 2. Extended RFID message in the code-centric RFID system (CRS): (a) original data fields used in existing systems, and (b) extension fields as proposed. In addition to traditional data, RFID tags can store mobile agents, which explicitly indicate the type of service that the object bearing the tag should receive.**

In the CRS infrastructure, the main functional components include the passive-information manager, the code-information manager, the middleware layer, the environmental-parameters provider system, and the service response system. The passive-information manager

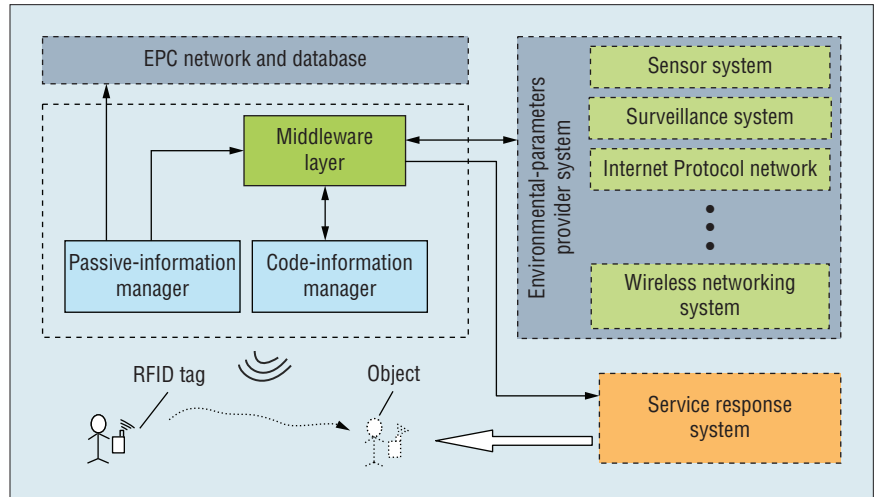
receives identification and description data from the RFID tag, and may further forward this information to the EPC network to create a record of it. The mobile-agent code is forwarded to the code-information manager, which delivers the agent to the middleware layer for interpreting. The environmental-parameters provider system sends parameters to the middleware layer so that it can take appropriate actions. The middleware layer in turn sends action commands to the service response system, which performs the desired tasks according to the decisions made for the object, as Figure 3 shows.

Although all of these modules are indispensable to the CRS infrastructure, the middleware layer is the most important because it is used to interpret the codes that form the mobile agents. The mobile agents consist of instructions or programs that the RFID infrastructure executes. Inputs to those programs may include identification or description information, as well as information provided by associated subsystems (position, sensory data such as humidity and temperature, and so on). Outputs may be any actions that the infrastructure can perform, such as interacting with a video surveillance system, writing some particular code in other tags, or issuing alarm signals. The stored code information helps the objects react to the environment more intelligently.

### Mobile-Code-Updating Mode

Providing on-demand QoS necessitates the ability to change or rewrite mobile code on-demand as well. We propose three kinds of code-updating modes: passive, active, and hybrid.

**Passive Mode.** Generally, RFID tags can be attached to human or non-human objects (products, animals, and so on). Nonhuman objects have no intelligence to update code by themselves. Thus, code updating can be performed only passively by the infrastructure (for instance, an RFID reader at some fixed location). For example, in an automatic assembly line, the operations on the product are performed step by step. If the current operation associated with the tag's code information is complete, a new code is written into the tag, meaning the object will accept the corresponding operation in the next step, and so on. If the tag's memory is large enough, all action codes can be written into it at the beginning of the assembly line. Then, the code size will continually shrink as different machine tools progressively complete the corresponding operations on the product. When multiple assembly lines are employed for some product, the RFID tag will be filled with a new set of code for performing a new series of operations in each assembly line. This implies that the size of an RFID message varies according to the different stages involved in processing the object.



**Figure 3. Functional components of the CRS.** The main parts of the system are the passive-information manager, the code-information manager, the middleware layer, the environmental-parameters provider system, and the service response system.

**Active Mode.** If the object is a human being, he or she might have specific requirements concerning service types and their quality. Users can update the codes actively by using portable RFID readers, in which case they finish the code-updating process before the objects arrive at the service provider's location.

**Hybrid Mode.** This mode combines passive and active modes. In addition to users actively setting codes, code updating can be passively performed near some stationary RFID reader (for example, at the entrance of a store).

**Pattern Classification.** Table 1 categorizes code-updating patterns by object type. For nonhuman objects, the goal is usually to perform some operation on the object or adjust the surrounding environment that best suits that

object's needs. For human objects, on the other hand, the goal might be to provide user-specific services.

### Middleware Design for the CRS

The middleware subsystem should be capable of processing all the code directives and actions specified by the corresponding application. In addition, its programmability should ensure, to a large extent, that applications can be enhanced or upgraded by deploying new codes without having to constantly alter the middleware. Furthermore, the code should encapsulate all the underlying operations, at both the network and database levels. This high-level approach lets programmers focus on the operational aspects of the application code being deployed without worrying about lower-level intricacies. However, all these features should be enacted by

**Table 1. Pattern classification of code updating in the CRS.**

Code-updating mode	Object type	Main goals	Example application
Passive	Product, animal, and other nonhuman objects	Perform operations on an object Adjust the surrounding environment	Automatic assembly line
Active	Human	Provide user-specific services	Smart house
Hybrid	Human	Provide user-specific services Adjust the surrounding environment	E-healthcare

code that is compact enough to fit in memory-constrained RFID tags.

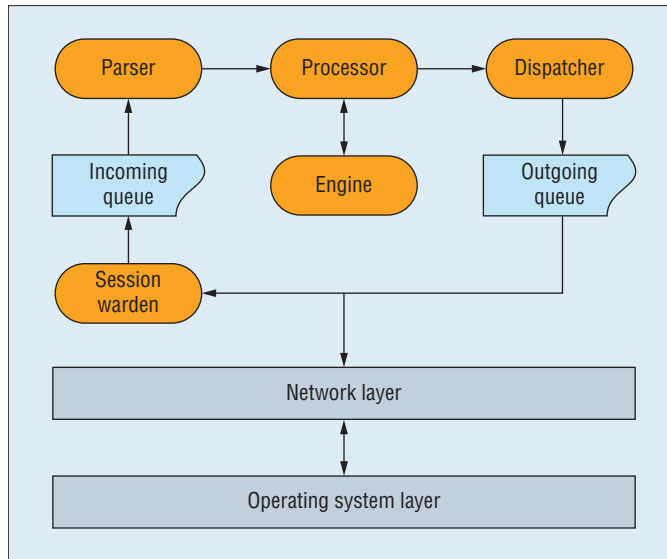
In an earlier work, we outlined the most important characteristics for middleware design in WSNs, which are formed by devices with severe memory and data-processing limitations.<sup>5</sup> These same limitations clearly exist in our CRS, where an event-driven scheme prompts the deployment and execution of compact mobile codes, with a sufficient degree of flexibility to support a variety of applications.

Therefore, we developed the Wiseman middleware system as a proof of concept for deploying mobile codes.<sup>6</sup> By introducing minor adjustments, Wiseman can provide a solid middleware foundation for supporting our proposed CRS. In its current form, Wiseman already provides all the middleware functionalities we deem essential. Its architecture is based on processing text-based action scripts, which can be dynamically modified.

### Middleware Architecture

Wiseman contains four main components: an incoming-agent queue, a code parser, a processor block, and an agent dispatcher. It also includes two helper components: an engine and a session warden. Figure 4 shows these components and their interactions.

The incoming-agent queue temporarily stores agents arriving from the wireless interface for immediate processing. Wiseman does not support multithreading capabilities, because the interpreter is designed for operation in WSN nodes with stringent hardware limitations. The parser tokenizes individual instructions for gradual execution by splitting the



**Figure 4. Architecture of the Wiseman system. The incoming queue, the parser, the processor, and the dispatcher are the main components of this system, and the engine and the session warden are helper modules to these main components.**

code that forms the programs into two segments: head and tail. The head is the code segment that is ready for immediate processing; the tail comprises the rest of the agent's code that will be subsequently processed. The next instruction is obtained by tokenizing the first code segment from the tail if the outcome of the head's execution is successful. This process halts at any time if

- the current operation is unsuccessful,
- an explicit agent termination operation is indicated, or
- an agent hop operation is encountered.

The first and second cases are self-explanatory. For the latter case, control of the execution process is passed to the dispatcher module, which forwards the agent's tail to another node (or set of nodes). For our proposed CRS, the dispatcher forwards the agent's tail to a WSN node, as specified by the system's mobile codes.

### Language Constructs

Wiseman's language includes variables, rules, operators, and delimiters.

Its text-based lexemes of reduced size allow agents to be dynamically modified as needed, make the codes readable by humans, and occupy usually a few hundred bytes to implement the desired actions.

**Variables.** Wiseman implements three kinds of variables: numeric, character, and mobile. The first two are for storing numeric and single characters at the local node; the third is used by agents to bring values along as they traverse the

networked system. Typically, mobile variables are labeled  $M$  ( $M1$ ,  $M2$ , and so on), which is semantically similar to the way private variables are labeled in object-oriented programming. In addition to these user-defined variables, environmental variables provide information about the current execution environment. For example, the identity variable  $I$  stores a read-only value of the local node's ID number, and the predecessor variable  $P$  stores the ID number of the node from which the agent came. Similarly, the link variable  $L$  stores the label identifier of the virtual link that the agent used for hopping (if it exists).

**Operators.** Wiseman provides a variety of both general-purpose and system-specific operators, such as regular arithmetic operators ( $+$ ,  $-$ ,  $*$ ,  $/$ , and  $=$ ) and comparison operators ( $<$ ,  $<=$ ,  $==$ ,  $=>$ ,  $>$ , and  $!=$ ). The hop forwards the agent to another location, as specified by the value appearing on the right side of the  $\#$  character. Alternatively, an agent can be copied or moved to one or more locations associated with a virtual link, as indicated by the

value on the left side of the operator. In the latter case, the agent is cloned with as many copies as the existing destination nodes associated with the virtual links. After hopping, the agent's execution thread resumes at the point at which the process had been suspended. (A full description of Wiseman operators is available elsewhere.<sup>6</sup>)

**Rules.** Wiseman implements three rules for manipulating an agent's execution flow. The repeat rule  $R$  indicates that the codes delimited by curly brackets will be cyclically executed until a certain condition is found. AND and OR rules (denoted  $A$  and  $O$ ) control an agent's execution by checking whether the codes delimited by square brackets yield a true or false value for each code segment. Otherwise, the entire rule itself returns a false value, which halts the agent's execution.

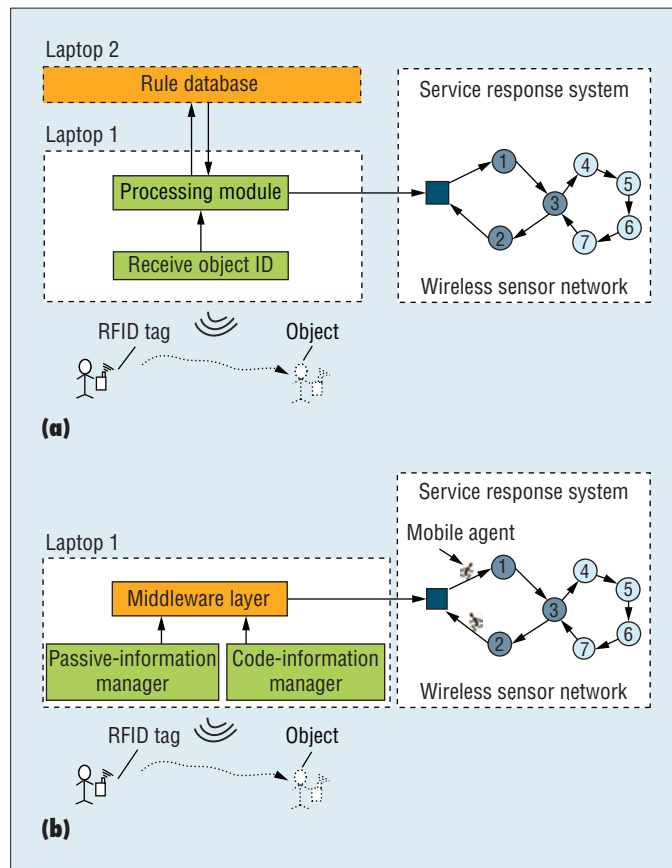
### Important Wiseman Features for the CRS

Wiseman supports three agent migration techniques that are particularly useful in our proposed CRS. The first is known as *explicit path hopping*, which provides the necessary functionality to enable the explicit definition of the path that an agent will follow. The second, *variable-target hopping*, employs mobile or numeric variables on the right side of the hop operator to indicate the agent's current destination target. The third method, *labeled-path hopping*, allows the creation of labeled paths for

emulating multicast transmissions from the local node. The main advantage of this third method is that the agent's programmer does not need to know in advance the identity of the destination nodes. This can be a significant advantage when multiple RFID tags must be read. In addition to these functionalities, Wiseman allows a programmable execution flow that is flexible enough to support events that might arise in the CRS, such as simplified if-then-else statements.

### Comparison of CRS and IRS Approaches

To test the effectiveness of our CRS, we compared it to an IRS by evaluating how long each system took to complete a set of tasks.



**Figure 5. Testbed scenario for (a) the IRS and (b) the CRS. Both architectures include the service response system. The testbed itself provides a good example of how differently these systems are realized in an actual setting.**

### Experimental Setup

We set up two testbeds: one for the IRS, and one for the CRS. As Figure 5 shows, the common part of the two architectures was the service response system, which in this experiment was a WSN based on MICAz Mote (with 128 Kbytes of instruction memory and 4 Kbytes of data memory), and we employed TinyOS v1.1 as our wireless sensor platform. We used Wiseman as our middleware system, as discussed earlier. (Wiseman was implemented in the nesC language in our previous work.<sup>5</sup>)

According to the topology of the sensor network shown in Figure 5, we employed the following action code:  $1\$n; \#1; \#3; \#4; \#5; \#6; \#7; \#3; \#2; 1\$n$ . The numbers appearing on the right side

of the hop operator  $\#$  indicate the identity number of the node to which the agent is set to migrate next (for instance,  $\#1$  migrates the agent to node 1). The middle segment of the action code ( $\#3; \#4; \#5; \#6; \#7; \#3$ ) hops through the links while performing some specific actions. In addition, agents toggle on the green LEDs as a visual aid through the WSN by means of the  $1\$n$  operation. In this operation, the 1 character signifies an LED operation, and the  $n$  signifies that a local LED is toggled on. There are two  $1\$n$  code segments in the action code, to mark the beginning and end of the agent's migration. We record the action delay between the time when the first LED is toggled on and the time when the second LED is toggled on.

The main difference between the simulated IRS and CRS is the way the system processes code information retrieved from the tag.

**Simulated IRS.** A cell phone sends a message, including the object’s ID, to laptop 1 through the Wi-Fi access point. The ID information goes to laptop 2, which serves as the rule database. The rule database server runs an FTP and a Web server that are accessible through a regular Internet connection. Hence, the rule database connects to the processing module through an FTP connection to update and retrieve the required rules. The rules are stored in the rule database’s hard drive. If laptop 1 successfully accesses the rule database and obtains the action codes by searching laptop 2 for rules, it distributes the codes to the WSN’s gateway (implemented in a MICAz Mote attached to an MIB510 interface board), as Figure 5a shows. The gateway then dispatches a mobile agent to the WSN to carry out the task.

**Simulated CRS.** A cell phone sends a message, including the object’s ID and mobile codes, to laptop 1 through the Wi-Fi access point. The mobile code is interpreted directly in the middleware layer of laptop 1. Then the code is forwarded to the WSN gateway, which loads it into a mobile agent and dispatches the agent to the sensor network.

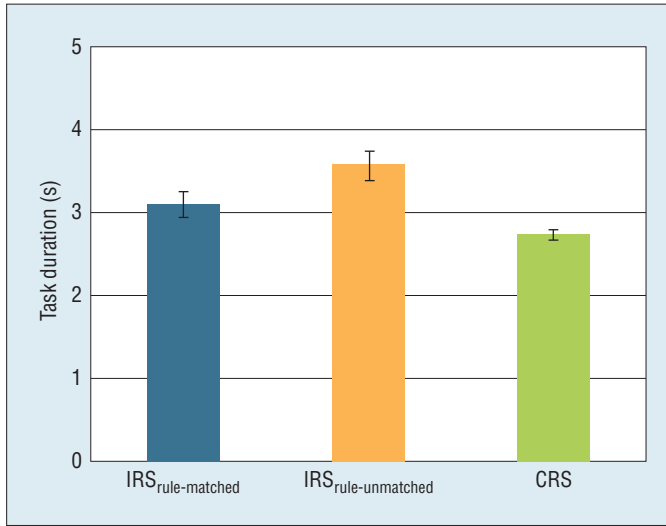
**Experimental Results**

Because the IRS requires searching for rules and accessing the database, failures can occur when the link

between laptop 1 and laptop 2 is broken or when the rule entry has not yet been pre-established in the database. Thus, we further divide the testbed realization of the IRS into two schemes: rule matched and rule unmatched. The process flows of all three schemes are as follows.

**Rule-Matched IRS.** First, the ID is forwarded to the rule database, which sends the action code to the processing module. Next, the mobile agent migrates to the sensor network. According to the action code, the agent itinerary for visiting WSN nodes is 1-3-4-5-6-7-3-2.

**Rule-Unmatched IRS.** This process is the most complicated among the three schemes. First, the ID is forwarded to the rule database, but then a message of “no matching” is returned to the processing module. So, a message with updated code information is sent to the rule database. (This procedure would incur an operational delay in actual situations.) Next, the rule database sends action code to the processing module, and an agent is dispatched.



**Figure 6. Comparison of task duration for a rule-matched IRS (IRS<sub>rule-matched</sub>), a rule-unmatched IRS (IRS<sub>rule-unmasked</sub>), and the CRS. Although the rule-matched IRS was more efficient than the rule-unmatched IRS, the CRS was clearly faster than both.**

**CRS.** Data processing in the CRS is the simplest of all three cases. Once laptop 1 receives the ID and code information, the agent is dispatched directly to migrate along the itinerary indicated by the action code.

**Task Duration Comparison.** We measured task duration as our performance parameter of interest. This is the time from when the tag was forwarded to laptop 1 to when the actions or operations associated with the rule were finished.

In RFID systems, the time delay for transmitting a tag’s information to an RFID reader is a few milliseconds. Thus, the task duration is mainly related to the process flow’s complexity. As Figure 6 shows, compared to the two IRS schemes, the CRS had the lowest task duration. This is because the CRS processes the action code locally without accessing the rule database. Note that the current testbed does not fully exhibit the advantages of the CRS; in actual situations, the amount of time a human would take to process the document would be greater.

**O**ur novel CRS makes it possible for mobile code to handle context-aware situations. Benefits include improved system scalability and automated monitoring. In addition to improving performance in terms of task duration, the CRS also provides the flexibility for users to change code information on demand. Hence, this information remains accurate and up-to-date,

## THE AUTHORS

while reflecting user requirements and network dynamics. However, to successfully deploy CRS-based architectures, several remaining challenges must be addressed, including knowledge representation and situation-aware code interpretation. In the future, we plan to enhance the system capabilities of situation-aware code delivery, interpretation, and update to handle heterogeneous ubiquitous objects in the real world. ■

### Acknowledgments

This project was supported in part by Research Grants Council (RGC) grants under contracts 622508 and 623209; National Natural Science Foundation of China RGC grant N\_HKUST609/07; Innovation and Technology Fund project ITP/023/08LP; the Ministry of Knowledge Economy (MKE), Korea, under the Information Technology Research Center support program supervised by the National IT Industry Promotion Agency (NIPA-2009-C1090-0902-0006); the IT R&D program of the MKE Korea Evaluation Institute of Industrial Technology (2008-F-034-02); and the Canadian Natural Sciences and Engineering Research Council under Strategic Project grants 365208 and 364962.

### References

1. B. Nath, F. Reynolds, and R. Want, "RFID Technology and Applications," *IEEE Pervasive Computing*, vol. 5, no. 1, 2006, pp. 22–24.
2. Q.Z. Sheng, X. Li, and S. Zeadally, "Enabling Next-Generation RFID Applications: Solutions and Challenges," *Computer*, vol. 41, no. 9, 2008, pp. 21–28.
3. M. Chen et al., "A 2G-RFID-Based E-Healthcare System," *IEEE Wireless Comm. Magazine*, vol. 17, no. 1, 2010, pp. 37–43.
4. *Compliant FRAM Embedded High-Speed RFID LSI FerVID Family: MB89R118*, ASSP ISO/IEC 15693, Fujitsu Microelectronics, data sheet

**Min Chen** is an assistant professor in the School of Computer Science and Engineering at Seoul National University. His research interests include multimedia and communications, wireless sensor networks, ad hoc networks, and wireless body area networks. He has a PhD in electrical engineering from South China University of Technology. He is a senior member of IEEE. Contact him at minchen@ieee.org.

**Sergio González** is a post-doctoral fellow in the Department of Electrical and Computer Engineering at the University of British Columbia. His research focuses on wireless sensor networks and ad hoc networks. He has a PhD in electrical and computer engineering from the University of British Columbia. He is a member of IEEE and the IEEE Communications Society. Contact him at sergiog@ece.ubc.ca.


**Qian Zhang** is an associate professor in the Department of Computer Science and Engineering at the Hong Kong University of Science and Technology. Her research interests include multimedia networking, wireless communications and networking, and overlay networking. She has a PhD in computer science from Wuhan University. She is a senior member of IEEE. Contact her at qianzh@cse.ust.hk.

**Victor C.M. Leung** is a professor and the incumbent Telus Mobility Research Chair in Advanced Telecommunications Engineering in the Department of Electrical and Computer Engineering at the University of British Columbia. His research focuses on wireless networks and mobile systems. He has a PhD in electrical engineering from the University of British Columbia. He is a Fellow of IEEE. Contact him at vleung@ece.ubc.ca.

DS04-33101-4Ea; <http://edevice.fujitsu.com/fj/DATASHEET/e-ds/e433101.pdf>.

5. M. Chen, S. Gonzalez, and V.C.M. Leung, "Applications and Design Issues of Mobile Agents in Wireless Sensor Networks," *IEEE Wireless Comm. Magazine*, vol. 14, no. 6, 2007, pp. 20–26.
6. S. Gonzalez, M. Chen and V. Leung, "Design, Implementation and Case Study of WISEMAN: Wireless

Sensors Employing Mobile Agents," *Proc. 2nd Int'l Conf. Mobile Wireless Middleware, Operating Systems, and Applications* (Mobileware 09), LNCS 7, Springer, 2009, pp. 366–380.

 Selected CS articles and columns are also available for free at <http://ComputingNow.computer.org>.



**IEEE Intelligent Systems**

THE #1 ARTIFICIAL INTELLIGENCE MAGAZINE!

IEEE Intelligent Systems delivers the latest peer-reviewed research on all aspects of artificial intelligence, focusing on practical, fielded applications. Contributors include leading experts in

- Intelligent Agents • The Semantic Web
- Natural Language Processing
- Robotics • Machine Learning

Visit us on the Web at [www.computer.org/intelligent](http://www.computer.org/intelligent)