Using Event-Based Style for Developing M2M Applications

Truong-Giang Le[1], Olivier Hermant[2], Matthieu Manceny[1], Renaud Pawlak[3], and Renaud Rioboo[4]

[1] LISITE - ISEP, 28 rue Notre-Dame des Champs, 75006 Paris, France
[2] CRI - MINES ParisTech, 35 rue ST-Honoré, 77300 Fontainebleau, France
[3] IDCapture, 2 rue Duphot, 75001 Paris, France
[4] ENSIIE, 1 square de la Résistance, F-91025 Évry CEDEX, France
{le-truong.giang,matthieu.manceny}@isep.fr
olivier.hermant@mines-paristech.fr
renaud.pawlak@gmail.com
renaud.rioboo@ensiie.fr

Abstract. *In this paper, we introduce how to write M2M applications by using INI, a programming language specified and implemented by ourselves that supports event-based style. With event-based programming, all M2M communication can be handled and scheduled. Programmers may use existing built-in events or define their own events. We apply our approach in a real M2M gateway, which allows gathering and exchanging information between sensors and machines in the network. The results shows that our work proposes a concise and elegant alternative and complement to industrial state-of-the-art languages such as Java or C/C++.*

Keywords: event-based programming; parallel programming; domain-specific languages; M2M applications; gateway

1 Introduction

Machine-to-machine (M2M) refers to technologies that allow data communication and interaction between machine(s), device(s) or sensor(s) over a network without human intervention. The M2M connectivity market, a.k.a. the "Internet of Things", is growing worldwide. Analysts predict that there will be 25 billion connected IP devices by 2015, with M2M traffic expected to grow by 258% [2]. Another research estimates that M2M generates $35 billion in service revenues by 2016 [5]. It covers a wide array of applications including automotive, metering, remote management, IP multimedia subsystem, industrial data collection, health care, etc [16, 23]. For example, in agriculture, M2M applications are used to capture images to track crops' growth in the fields or to collect sounds to estimate insect quantity in the plants. Another example are medical centers, where the patient data such as blood pressure, heart rate, body temperature, and respiratory rate should be accumulated and sent periodically to the health care provider. In a factory, M2M sensors are used to track and monitor assets,

equipment, materials, cargo and supplies. To understand more about M2M technologies, please refer to [19, 27]. Currently, M2M industry continues to looks for better and comprehensive M2M solutions.

Although, M2M has attracted a large amount of attention over the years, developing M2M applications is still challenging. Besides, "existing M2M solutions are fragmented and usually are dedicated to a specific single application" [6]. In our work, we develop a novel programming language called INI to support developers to write M2M programs more easily. INI comes with event-based paradigm, which is an appropriate style to handle and schedule M2M communication [22]. Moreover, events handlers in INI run in parallel either asynchronously or synchronously to improve the performance and responsiveness of the system.

The rest of this paper is organized as follows. In Section 2, we give an overview of related work. In Section 3, we briefly introduce INI and how it supports event-based style. Then, the case study of a M2M gateway written in INI along with the experiment tested on the real device are presented in Section 4. Section 5 concludes the paper.

2 Related work

Many M2M infrastructures frameworks, models, paradigms and services have been proposed to ease the development of M2M systems.

In recent years, the event-driven programming has been recognized as an efficient method for interacting and collaborating with the environment in ubiquitous computing [17]. Event-based programs are generally driven by a loop that waits for events and executes the appropriate callback [15]. Using event-driven style requires less effort and may lead to better performance, simpler, more manageable, portable code, and robust software [12]. This style is strong and convenient to write many kinds of applications: M2M applications, sensors applications, mobile applications, simulation systems, embedded systems, robotics, context-aware reactive applications, self-adaptive systems, etc.

Herstad *et al.* [20] defined a service platform architecture for connected objects. The architecture exhibits a number of features to support scalability, rapid development, and technology and device independence. Liu *et al.* [24] focused on the construction of a generic manager-to-manager (M2M) interface, which enables a seamless cooperation between autonomous managers on the basis of Web services. Cristaldi *et al.* [11] presented an interface platform, which is able to collect and process data from a wide variety of sensors and exchange information supporting different communication networks and protocols. Matson *et al.*[25] tried to create a model and architecture to support networking, communication, interaction, organization and collective intelligence features between machines, robots, software agents, and humans. Currently, in order to build M2M applications, developers use classical programming languages (e.g. Java, .Net, C/C++, Perl, etc.) or their extensions. Besides, there has been several work on constructing event-based programming languages [10, 21], which can be applied to handle events happening in M2M systems.

However, these languages are not fully comfortable for M2M applications since they lack a well-defined mechanism to support scheduled operations, which are essential in M2M communication. Another limitation is that events are not constructed and handled in an intuitive manner, i.e. they are mixed with other syntaxes and notations. In our language called INI, events are defined and applied clearly. Furthermore, event actions can be scheduled easily with the help of two built-in events (e.g. `@every`, `@cron`) or by user-defined events. Another advantage is a flexible support of parallelism for events when running. Our next section will discuss in details.

3 Event-based programming with INI

Events are used to monitor changes happening in the environment or for time scheduling. In other words, any form of monitoring can be considered to be compatible with event-based style. In M2M communication, events are very frequent. Generally, three types of events are distinguished [26]:

- A timer event to express the passing of time.
- An arbitrary detectable state change in a system, e.g. the change of the value of a variable during execution.
- A physical event such as the appearance of a person detected by cameras.

For example, programmers may define an event to monitor the power level of their systems or to observe users' behaviors in order to react. They can also specify an event to schedule a desired action at preferable time. To understand more about event-based programming, please refer to [13–15].

INI is a programming language developed by ourselves, which runs on Java Virtual Machine (JVM). In INI, we support all these kinds of event as shown later. Event callback handlers (or events instances) are declared in the body of functions and are raised, by default asynchronously, every time the event occurs. By convention, an event instance in INI starts with @ and takes input and output parameters. Input parameters are configuration parameters to tune the event execution. Output parameters are variable names that are filled in with values when then the event callback is called. They can be considered as the measured characteristic of the event instance. It has to be noticed that those variables, as well as any INI variable, enjoy a global scope in the function's body. Both two kinds of parameters are optional. Moreover, an event can also be optionally bound to an id, so that other parts of the program can refer to it. The syntax of event instances is shown below:

```
id:@eventKind[inputParam1=value1, inputParam2=value2, ...]
  (outputParam1, outputParam2, ...)
    { <action> }
```

Programmer may use built-in events (listed in Table 1), or write user-defined events (in Java or in C/C++), and then integrate them to their INI programs. By developing custom events, we can process data which are captured by sensors.

Built-in event kind	Meaning
`@init()`	used to initialize variables, when a function starts.
`@end()`	triggered when no event handler runs, and when the function is about to return.
`@every[time:Integer]()`	occurs periodically, as specified by its input parameter (in milliseconds).
`@update[variable:T] (oldValue:T, newValue:T)`	invoked when the given variable's value changes during execution.
`@cron[pattern:String]()`	used to trigger an action, based on the CRON pattern indicated by its input parameter.

Table 1. Some built-in events in INI.

```
function main() {
  //Monitor the temperature of a patient
  t:@temperatureMonitor[tempPeriod = 1](temperature) {
    case {
      temperature > ... {
        //Notify to a doctor or do other automatic
        //emergency actions
        ...
      }
      //A default action
      default {
        ...
      }
    }
  }

  //Monitor the blood pressure of a patient
  b:@bloodPressureMonitor[bpPeriod = 2](pressure) {
    case {
      pressure <= ... { ... }
      ... //other conditions
    }
  }
  ...
}
```

Fig. 1. A simple health monitoring system written in INI.

To illustrate user-defined events in INI, let us consider a simple health monitoring system that must monitor several information related to the patient such as body temperature, blood pressure, etc. We can design events to deal with each task as shown in Figure 1. The event *@temperatureMonitor* named `t` has one input parameter called *tempPeriod*, which is applied to set how long the event should sleep between two consecutive checks (time unit is in hours). Be-

sides, it has one output parameter named *temperature* to indicate the current body temperature of the patient. Inside this event, based on the value of the current temperature, we can define several corresponding actions through the `n`-ary boolean case instruction (quite standard and not described here, please refer to INI Language Reference Documentation [30]). The other event *@bloodPressureMonitor* named `b` has a similar structure. All events in our program run concurrently, which means that we can handle multiple tasks at one time. To learn how to write user-defined events in INI, interested readers may refer to [30].

By default, except for the *@init* and *@end* events (see Table 1), all INI events are executed asynchronously. However, in some scenarios, a given event *e0* may want to synchronize on other events *e1,..., eN*. It means that the synchronizing event *e0* must wait for all running threads corresponding to the target events to be terminated before running. For instance, when *e0* may affect the actions defined inside other events, we need to apply the synchronization mechanism.

Programmers may use the following code to ensure that the event *e0* synchronizes on N target events:

```
$(e1,e2,...,eN) e0:@eventKind[...](...)
  { <action> }
```

Events in INI may also be reconfigured at runtime in order to adjust their behaviors when necessary to adapt to changes happening in the environment. Programmers can call the built-in function *reconfigure_event(eventId, [inputParam1 = value1, inputParam2 = value2,...])* in order to modify the the values of event's input parameters. For example, at some time during running, we want to collect patient's temperature data more frequently, we can adjust the input parameters of `t` by callings: `reconfigure_event(t, [tempPeriod = 0.5])`. Now our event will gather data for every 30 minutes instead of one hour as before. Moreover, we also allow programmers to stop and restart events with the built-in functions `stop_event([eventId1, eventId2, ...])` and `restart_event([eventId1, eventId2, ...])`. For instance, we may stop all data collection processes when the energy level of the system is too low and restart them later when the energy is charged again.

Last but not least, events in INI may be used in combination with a boolean expression express the requirement that need to be satisfied before an event is executed. Programmers may use the syntax below:

```
<event_expression> <logical_expression>
  { <action> }
```

For example, if we want the event `@bloodPressureMonitor` to be executed only when the temperature is higher than some threshold:

```
@bloodPressureMonitor[bpPeriod=2](pressure) temperature >...
  {...}
```

To understand more about the above mechanisms and other aspects of INI (e.g. semantics, rules, type system, type checking, built-in functions), programmers may have a look at [30].

4 Case study: A M2M gateway program

Many M2M applications are required to send data to a M2M server through network and a M2M gateway is a typical example [28]. In other words, a M2M gateway allows different types of networks to communicate with each another in order to provide data [29]. For example, users may use this device for industrial data collection or for surveillance purpose [13]. Since this kind of tasks does not create much added-value and maybe in dangerous or remote environment, taking advantages of M2M technologies is a good solution.

In this section, we show how to apply INI to write a M2M gateway program, containing two basic steps as shown in Figure 2:

- Collecting data (e.g. images, sound, etc.), which are captured by sensors or peripherals.
- Transmitting data to the server through the network.

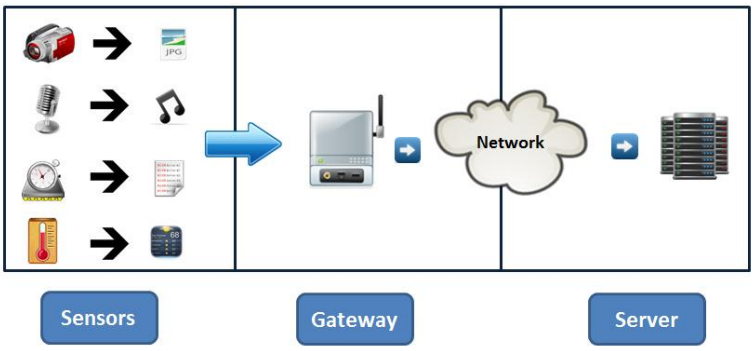

Fig. 2. The role of a gateway.

All these operations above can be scheduled easily with the help of the two built-in events `@every` and `@cron` (see Table 1. The event `@every` can be applied to do an action periodically. The event `@cron[pattern:String]()` event occurs on times indicated by the UNIX CRON pattern expression. CRON is a task scheduler that allows the concise definition of repetitive task within a single (and simple) CRON pattern [3]. A UNIX crontab-like pattern is a string split in five space separated parts, including minutes sub-pattern, hours sub-pattern, days of month sub-pattern, months sub-pattern, and days of week sub-pattern.

Some examples:

- `"5 * * * *"`: This pattern causes the event instance to occur once every hour, at the begin of the fifth minute (00:05, 01:05, 02:05 etc.).
- `"* * * * *"`: This pattern causes the event instance to occur every minute.

– `"59 11 * * 1-5"`: This pattern causes the event to occur at 11:59 AM on every weekday (i.e. Monday, Tuesday, Wednesday, Thursday and Friday).

```
1 function main() {
2   //Initialization
3   @init() {
4     capturedDataFolder = file("data")
5     //Create a new data folder in case that it does not exist
6     case {
7       !file_exists(capturedDataFolder) {
8         mkdirs(capturedDataFolder)
9       }
10    }
11    zipFile = file("data.zip")
12    keepParentFolder = true
13  }
14
15  //Capture image from a camera using the library gphoto2
16  e1:@every[time = 60000]() {
17    exec("gphoto2␣--capture-image-and-download␣--filename
18 ␣␣␣␣␣data/img" + time() + ".jpg")
19  }
20
21  //Capture sound from a microphone using the library alsa
22  e2:@every[time = 30000]() {
23    exec("arecord␣-d␣30␣-f␣cd␣data/sound" + time() + ".wav")
24  }
25
26  //Upload data to a FTP server by schedule
27  @cron[pattern = "0␣09-18␣*␣*␣1-5"]() {
28    stop_event([e1,e2])
29    zip(capturedDataFolder,zipFile)
30    upload_ftp("server_address", "user_name",
31      "password", zipFile, to_string(time()) + "data.zip")
32    delete_file(zipFile)
33    delete_folder(capturedDataFolder, keepParentFolder)
34    restart_event([e1,e2])
35  }
36 }
```

Fig. 3. A M2M gateway program written in INI.

Our complete program is shown in Figure 3. The `main` function is composed of four events. The event `@init` (lines 3-13) is invoked to initialize necessary variables that will be used later. The variable `capturedDataFolder` indicates the folder where we put the collected data. If this folder does not exist, we create

it (lines 7-9). The variable `zipFile` denotes a zipped data file. We compress the data before uploading to save network bandwidth.

The next two events are `@every` event kind. They are used to collect image and sound data (e.g. in a field or in a factory). The event `e1` (lines 16-19) is invoked every minute to get a picture captured by a camera (by using the library `gphoto2` [4]). The event `e2` (lines 22-24) is invoked every 30 seconds to use a microphone to record sound (by using the library `alsa` [1]). All files (i.e. pictures and sounds) are saved into the data folder. These collecting processes (i.e. two events) run in parallel to take advantage of multithreading for better performance.

The last event is a `@cron` (lines 27-35) event, which is employed to upload the data to a FTP server every hour during working hours (i.e. 9:00 AM - 6:00 PM) on every weekday. Inside this event, first, we stop the two events `e1` and `e2` (line 28). Next, we compress the data folder before uploading (line 33). Then we upload the compressed data to a FTP server (lines 30-31). Since the gateway is only an data-exchanging device with limited storage capacity, after uploading, we delete the data to save storage (lines 32-33). Finally, we restart the two events `e1` and `e2` so that we can collect data again (line 34).

Now let us compare our INI program with a Java program that do the same tasks. In order to makes all operations running in parallel in Java, we need to create some explicit threads for different tasks: capturing pictures, recording sounds, uploading data and time scheduling for all operations. Scheduling is a nontrivial task since it is not well-supported in Java. In addition, we also need some thread pools in order to manage and synchronize those threads when needed. Although Java has a powerful support for concurrency, writing correct concurrent applications in this language is still challenging and error-prone, especially for novice programmers [18]. To decrease the difficulty and hide the unnecessary complexity, INI separates the thread issue (implemented in Java), and the event-handling issue (implemented in INI). This separation of concerns helps in making the INI approach clearer and less error-prone than a pure-Java program.

We tested our program on the real gateway in the scope of the MCUBE project [7]. The device is provided by Webdyn [9], a company dedicated to design, develop, and market this kind of product. We use Oracle Java SE Embedded for establishing the runtime environment of INI on the gateway. This platform is optimized for mid-range to high-end embedded systems and offers a high-performance virtual machine, full high-performance graphics support, deployment infrastructure, and a rich set of features and libraries [8]. During the experiment, our program worked well. All the data are captured and transmitted properly.

5 Conclusion and future work

In this paper, we presented how to write M2M applications using event-based style through INI, a programming language developed by ourselves. INI can be

seen as an Architecture + DSL (Domain-Specific Language) for (multi-threaded) event handling and coordination. INI allows programmers to construct and define events in a convenient and straightforward way. Besides, events in INI run in parallel to perform multiple tasks concurrently. For testing, we built a gateway program and when running on the real device, this program completes required tasks adequately and appropriately.

For the future work, we will apply INI in more M2M applications and also in other domains like robotics or manufacturing systems. We also have a plan to evaluate quality and performance of INI programs. Currently, we are trying to better a tool called `INICheck`, which can convert a major subset of INI to Promela, the input modeling language of the model checker SPIN. Then SPIN can be used to verify some properties of INI programs. This tool allows the programmer to have insurances on his code and its behavior.

Acknowledgments

The work presented in this article is supported by the European Union. Europe is committed in Ile-de-France with the European Regional Development Fund.

References

1. Alsa, `http://www.alsa-project.org/`
2. Cisco jumps into the M2M market, `http://www.networkcomputing.com/wireless/231600077`
3. Crontab, `http://crontab.org/`
4. Gphoto2, `http://gphoto.org/`
5. M2M to generate $35bn in service revenues by 2016, `http://juniperresearch.com/viewpressrelease.php?pr=243`
6. Machine-to-Machine (M2M) The rise of the machines, `http://www.juniper.net/us/en/local/pdf/whitepapers/2000416-en.pdf`
7. The MCUBE project, `http://mcube.isep.fr:8080/`
8. Oracle Java SE Embedded, `http://www.oracle.com/technetwork/java/embedded/overview/getstarted/index.html`
9. Webdyn, `http://www.webdyn.com/en/`
10. Cohen, N.H., Kalleberg, K.T.: EventScript: an event-processing language based on regular expressions with actions. In: Proceedings of the 2008 ACM SIGPLAN-SIGBED conference on Languages, compilers, and tools for embedded systems. pp. 111–120. LCTES '08, ACM, New York, NY, USA (2008)
11. Cristaldi, L., Faifer, M., Grande, F., Ottoboni, R.: An improved M2M platform for multi-sensors agent application. In: Sensors for Industry Conference, 2005. pp. 79 –83 (feb 2005)
12. Dabek, F., Zeldovich, N., Kaashoek, F., Mazières, D., Morris, R.: Event-driven programming for robust software. In: Proceedings of the 10th workshop on ACM SIGOPS European workshop. pp. 186–189. EW 10 (2002)
13. Denecke, K.: Event-Driven Surveillance: Possibilities and Challenges. Springer, Berlin (2012)

14. Etzion, O., Niblett, P.: Event Processing in Action. Manning Publications Co. (2010)
15. Faison, T.: Event-Based Programming: Taking Events to the Limit. Apress, Berkely, CA, USA (2006)
16. Fan, Z., Tan, S.: M2M communications for E-health: Standards, enabling technologies, and research challenges. In: Medical Information and Communication Technology (ISMICT), 2012 6th International Symposium on. pp. 1 –4 (march 2012)
17. Fischer, J., Majumdar, R., Millstein, T.: Tasks: language support for event-driven programming. In: Proceedings of the 2007 ACM SIGPLAN symposium on Partial evaluation and semantics-based program manipulation. pp. 134–143. PEPM '07, ACM, New York, NY, USA (2007)
18. Goetz, B., Peierls, T.: Java concurrency in practice. Addison-Wesley (2006)
19. Hersent, O., Boswarthick, D., Elloumi, O.: The Internet of Things: Key Applications and Protocols. John Wiley & Sons, Incorporated (2011)
20. Herstad, A., Nersveen, E., Samset, H., Storsveen, A., Svaet, S., Husa, K.: Connected objects: Building a service platform for M2M. In: Intelligence in Next Generation Networks, 2009. ICIN 2009. 13th International Conference on. pp. 1 –4 (oct 2009)
21. Holzer, A., Ziarek, L., Jayaram, K., Eugster, P.: Putting events in context: aspects for event-based distributed programming. In: Proceedings of the tenth international conference on Aspect-oriented software development. pp. 241–252. AOSD '11, ACM, New York, NY, USA (2011)
22. IoT-A: (Internet of Things – Architecture) Project Deliverable d3.1 - Initial M2M API Analysis (2012)
23. Kim, B.H., Ahn, H.J., Kim, J.O., Yoo, M., Cho, K., Choi, D.: Application of M2M technology to manufacturing systems. In: Information and Communication Technology Convergence (ICTC), 2010 International Conference on. pp. 519 –520 (nov 2010)
24. Liu, L., Gaedke, M., Koeppel, A.: M2M interface: a Web services-based framework for federated enterprise management. In: Web Services, 2005. ICWS 2005. Proceedings. 2005 IEEE International Conference on. pp. 2 vol. (xxxiii+856) (july 2005)
25. Matson, E., Min, B.C.: M2M infrastructure to integrate humans, agents and robots into collectives. In: Instrumentation and Measurement Technology Conference (I2MTC), 2011 IEEE. pp. 1 –6 (may 2011)
26. Mühl, G., Fiege, L., Pietzuch, P.: Distributed Event-Based Systems. Springer-Verlag New York, Inc., Secaucus, NJ, USA (2006)
27. Roebuck, K.: Machine-to-machine (M2M) Communication Services: High-impact Technology - What You Need to Know: Definitions, Adoptions, Impact, Benefits, Maturity, Vendors. Lightning Source Incorporated (2011)
28. Singh, S., Huang, K.L.: A robust M2M gateway for effective integration of capillary and 3GPP networks. In: Advanced Networks and Telecommunication Systems (ANTS), 2011 IEEE 5th International Conference on. pp. 1 –3 (dec 2011)
29. Sosinsky, B.: Networking Bible. Wiley Publishing, 1st edn. (2009)
30. Truong-Giang, L.: INI Online (2012), `https://sites.google.com/site/inilanguage/`