Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 52 (2015) 436 – 443

Procedia Computer Science

The 6th International Conference on Ambient Systems, Networks and Technologies (ANT 2015)

LISA: Lightweight Internet of Things Service Bus Architecture

Behailu Negash[a,*], Amir-Mohammad Rahmani[a], Tomi Westerlund[a], Pasi Liljeberg[a], and Hannu Tenhunen[a, b]

[a]*Department of Information Technology, University of Turku, Turku, Finland*
[b]*School of ICT, Royal Institute of Technology (KTH), Stockholm, Sweden*

Abstract

A critical challenge faced in Internet of Things (IoT) is the heterogeneous nature of its nodes from the network protocol and platform point of view. To tackle the heterogeneous nature, we introduce a distributed and lightweight service bus, LISA, which fits into network stack of a real-time operating system for constrained nodes in IoT. LISA provides an application programming interface for developers of IoT on tiny devices. It hides platform and protocol variations underneath it, thus facilitating interoperability challenges in IoT implementations. LISA is inspired by the Network on Terminal Architecture (NoTA), a service centric open architecture by Nokia Research Center. Unlike many other interoperability frameworks, LISA is designed specifically for resource constrained nodes and it provides essential features of a service bus for easy service oriented architecture implementation.

Peer-review under responsibility of the Conference Program Chairs

Keywords: NoTA; LISA; Interoperability; IoT; ESB; SOA; 6LoWPAN; RIOT

1. Introduction

The Internet has evolved from a research project to a huge hub of knowledge and its benefit to human beings increases as more devices get connected to access or provide services. It is estimated that more than 50 Billion devices will be connected by 2020[1]. A large proportion of these devices will be small embedded devices which will be identified uniquely and interact with other devices. This network of embedded devices is referred as Internet of

* Corresponding author. Tel.: +358-456-014324
E-mail address: behneg@utu.fi

Peer-review under responsibility of the Conference Program Chairs
doi:10.1016/j.procs.2015.05.010

Things (IoT). It is believed that enormous economic and social benefit can be gained by utilizing IoT in different domains.

Simple applications designed for specific tasks have been developed and used before complicated integration requirements arise. One of the solutions to integrate such applications is by making these applications to expose their functionality as a service, known as Service Oriented Architecture (SOA). However, the implementation of SOA leads to large amount of interaction between these services. In addition, each service instance has to be aware of the location of the other end and interface exposed during communication. Enterprise service bus (ESB) gives an infrastructure for SOA implementation for managing the communication between services. It also provides a common medium for communication between different standards[13].

Internet of things is facing obstacles for growth towards the envisioned level of success[1]. One of these hurdles is the lack of a dominant protocol, which led to manufacturers using different protocols and platforms. In addition, the nature of nodes and applications in this scenario is different in that they are too resource constrained. This creates network and application silos each with its own protocol and platform. There are few efforts to bridge these silos and build an interoperable larger network. AllSeen Alliance has built a framework for interoperability of IoT devices[5, 6]. The device service bus by Medeiros et al[8] is also another effort for interoperability. However, these frameworks do not address the larger proportion of devices, which are constrained in the available resources.

LISA is a light weight service bus which is inspired by NoTA[3] concept, targeting these tiny devices, built with resource constraints in mind from the beginning. LISA is different in that it contributes the following key features for the interoperability of devices and applications in IoT:

- LISA is lightweight and targets extremely resource constrained nodes having a few kilobytes of memory.
- LISA supports low power operations with an ultimate goal of supporting most of the interconnection protocols for interoperability.
- LISA provides familiar, socket like, interface for application programmers to interact with the API regardless of the platform and protocol used.
- LISA targets to fit to a bare metal implementation and it will be released as an open source project.

The rest of the paper is organized as follows. We start by the motivation of the resource constraints in devices in Section 2, followed by differences in the protocols and platforms in IoT that led to the need for interoperability. We also discuss related works in the same section. In Section 3, the preliminaries of LISA such as its basis of architecture (ESB, SOA and NoTA) are discussed. In Section 4, the details of LISA including the implementation are explained followed by the demonstrator discussion in Section 5. Section 6 is dedicated to the conclusion and future works for the service bus.

2. Motivation and related works

As a basis for discussion, let us categorize devices with network interface in to three classes based on resources they have. The first class contains those devices having multiple high speed processors with gigabytes of memory and multiple networking options, such as PC, smart phones and tablets. Those devices which have few megabytes of memory and used in different Industrial or home automation belong to the second class. The last class contains devices which are too small in processing power, memory and usually have a single network interface. These devices might operate on batteries which are meant to be used for longer period of time. Typical device in this class include sensor nodes which could be as small as 8bit microcontrollers having 32Kbyte of RAM and 512Kbyte of flash memory.

There are few choices of frameworks to build interoperable system for the first two classes of devices mentioned above[5, 6]. There exists protocols which are tailored for specific type of application especially for the first class of devices working in the current Internet domain. However, the constraints in the third class of devices make it difficult to build an interoperable system using same protocols and frameworks. Hence, there is limited applicability of the techniques used to integrate systems in the current Internet domain. These devices also have real-time requirements which should be met by their operating system.

As mentioned above, there are different frameworks developed for interoperability. However, most of these are working for devices capable of supporting the standard Internet protocols. The framework introduced by AllSeen[5, 6] is promising in both the applicability and the community behind it, AllSeen Alliance. One advantage of the framework is the support of multiple programming languages and familiar platforms in the PC and mobile device domain. However, this work is not focused on the resource-constrained devices which are targeted by LISA. A device service bus presented by Medeiros et al[8] uses web services for interoperability based on Device Profile for Web Services (DPWS). However, similar to the above discussion, this device service bus is also targeted for non-resource-constrained nodes.

The initial design of the current Internet was not for service centered operation. The work presented by Nordstrom et al[7] addresses the resulting mobility problem accessing the Internet. By introducing a middle layer between the network and transport layer, they provide a means of addressing services independent of the network address. This has been a motivation for LISA to include a discovery mechanism which is customized for the IoT domain.

The target of LISA is to fill the interoperability gap by providing a simple API for programmers of these resource constrained devices enabling developers to build a distributed and interoperable system. In the following section, we discuss the roots of overall architecture of LISA.

3. Foundation architecture of LISA

There are different levels of integration requirements[12, 14]; leading to multiple options of integration patterns. LISA addresses the integration requirements which arise from platform and protocol differences. It enables the development of SOA by providing the infrastructure for services to work together. Selected architecture and integration patterns which are related to the design of LISA are discussed below.

3.1 Service Orientated Architecture and Enterprise Service Bus

Before discussing the architecture of LISA, it is necessary to explain the basis of architecture style for LISA; service orientation and service bus. Service oriented architecture (SOA) is an architectural style where individual functionalities of a system are exposed as autonomous services so that consumers of a service can easily access it. This has been extensively used to solve the problem of interoperability in enterprise systems regardless of the platform on which the service runs. The services have defined interfaces which are described using standard specification such as web service description language (WSDL). The format of exchanged message is usually either XML or JSON[13]. LISA has also similar approach; service nodes define the interface they expose using WSDL like description language which is adapted from NoTA without customization. In contrast to interface description specification, the message format of LISA is customized from that of NoTA.

The autonomous services in SOA need a means of communication and sharing of the interface they expose for other services or a client. Enterprise service buses provide the communication channel to access the services exposed in SOA. Service buses enable the discovery of services by other clients. There are many service buses, either open source or proprietary, to choose from[4]. Service bus, unlike message brokers, is not based on store and forward type of servers. LISA serves as a service bus in a similar protocol subnetwork and works as a broker where the communication is outside the boundary.

3.2 Network on Terminal Architecture

Network on Terminal Architecture (NoTA) is a service based open architecture designed by Nokia Research Center. The initial objective for the design of this architecture was to facilitate the development of mobile devices by minimizing the time required to integrate different modules. It also minimizes the coupling of system modules. The architecture has its roots in the area of Network on Chip (NoC) and SOA as it focuses on interconnection of services and applications utilizing these services. It is a platform and protocol independent architecture. The reference implementation of NoTA, which was release as an open source implementation of this architecture, has support for variety of network protocols and few platforms.

There are two type of nodes in NoTA; service node (SN) and application node (AN). The application node is a client that request for a service from service node. The interaction of these nodes pass through a stack called the Device Interconnect protocol (DIP), shown in Figure 1 (a). *DIP* has two layers; High (*H_IN*) and Low (*L_IN*) interconnect layers. Application and service nodes communicate either using message passing (control plane) or a node streams data to a target (data plane).

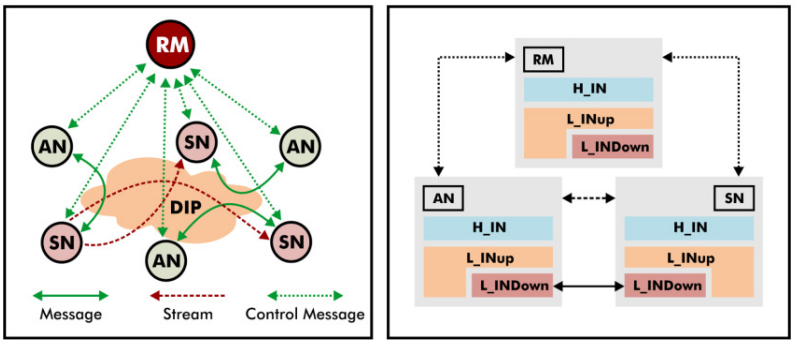

Figure 1 (a). NoTA nodes and message types; (b) Inside NoTA DIP

The High interconnect layer (*H_IN*), shown in Figure 1 (b), is the device interconnect layer that is closer to the user application and it provides Berkeley socket (BSD) like interface for the programmers. The interface introduces one additional function prior to the call for *HSocket* (equivalent of Socket function in BSD sockets); the programmer needs to call *Hgetinstance* to get a pointer to an *H_IN* instance which is passed in subsequent calls. *H_IN* is where the NoTA protocol is implemented with functionalities such as service discovery, activation and registration. On the other end of *H_IN*, it communicates with the underlying layer, *L_IN* to interact with the specific network protocol.

The lower interconnect layer (*L_IN*) has two sub layers, *L_INup* and *L_INdown*, as shown in Figure 1-b. The upper layer (*L_INup*) communicates with *H_IN* and the lower layer (*L_INdown*) provides specific implementation of different protocols. Different L_INdown implementations are enabled for a specific node based on the available network interface and in some cases multiple *L_INdown* can be enabled at the same time.

NoTA also provides a special type of service node, which can be enabled depending on the size of the network. This service node is called Resource Manager, which is used to handle dynamic discovery of services by application nodes. NoTA also provides stub generator, which is an easy way of handling the process of creating and using NoTA sockets. The services in NoTA are described using an XML based service description file, similar to web service description language (WSDL). The format of messages in NoTA is different from that of web services. NoTA defines service signal types[3] that specify message parameters for nodes to understand the content. The service description technique and message format of NoTA are used in LISA as-is.

4. LISA-the lightweight embedded service bus

In this section, we explore the details of LISA. As discussed in the previous sections, it is designed specifically for resource constrained devices. In most implementations, these devices use a lightweight operating system to take care of low level details while user applications focus on specific business requirements. There are many flavors of such operating systems; for instance Contiki, TinyOS, RIOT[9] and FreeRTOS are widely used. Regardless of the internal design of these operating systems, they are all designed to be lightweight. One of the requirements of interoperability is the ability to use a combination of these operating systems in a large system. In addition, each node might also have a different network interface. LISA unifies these differences in platform and protocol by enabling the implementation of SOA. LISA is designed to be portable to these operating systems with minimal configuration. In some cases however, there are applications that run without an operating system. LISA also has the

goal of enabling bare metal implementation in future releases. The initial version of LISA, however, targets RIOT which is discussed in the following section.

4.1 Target platform

RIOT is a micro kernel based, real-time, multi-threaded and modular operating system specifically designed taking resource-constrained nodes in to consideration[9]. It is designed to bridge the gap between the full-fledged operating systems (e.g. Linux and Windows) which are easier to program and the smaller operating systems for sensor nodes (e.g. Contiki and TinyOs) by providing easy programmability for the lower end. It supports standard C programming and provides inter-process communication facilities with partial POSIX compliance. It has a lightweight network stack with 6LoWPAN and RPL support. The overall features provided for module development and easy portability to multiple boards and CPU attracted us to make RIOT the first target platform to test the concept of LISA. As shown in Figure 2, LISA runs on top of the transport layer and partially crosses to reach the network layer.

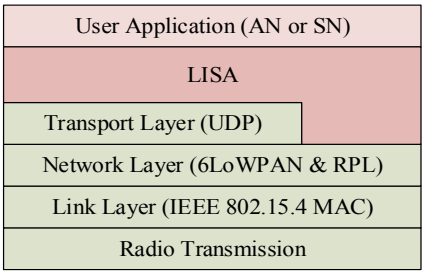

Figure 2. LISA in the network stack of RIOT

4.2 High level architecture

LISA has three node types; application nodes (AN), service nodes (SN) and a manager node. It is organized in a federated manner. Autonomous sub networks of AN's and SN's are managed with a single manager node. Following the node types, there are different levels of uptime. Application node is in sleep mode most of the time and initiates communication whenever it wakes up. Service nodes can also go to a sleep state, but they are alive for longer time than application nodes. Application and service nodes receive manager advertisement which contains the address of the manager. Service nodes are identified with unique ID and register with the manager as they wake up. Application nodes request the manager for the address of a service during discovery phase. Once the address is resolved the application and service node can start to communicate.

When there is a service request from outside smaller networks the manager node represents the node in passing the request to manager of the destination network. Service requests inside a sub network occur simply in a client-server fashion once the channel is established with the help of the manager. For ease of discussion, Figure 3 shows five subnetworks each having a manager (red nodes in Figure 3). Assuming each of these subnetworks use different operating system and different protocols (such as 6LoWPAN and Bluetooth low Energy), each of these nodes can communicate locally with the help of the manager node during discovery. However, if a node in one subnetwork using 6LoWPAN requests a service which is using Bluetooth low energy, the managers of these sub networks act on behalf of the individual nodes in transferring messages.

The target platforms for LISA can vary in the internal structure in terms of thread support, type of inter-process communication and the protocol used. The link with the platform has a uniform interface for managing threads and inter-process communication. The communication with the underlying network stack is also made through another interface implemented for each target protocol. On the upper end of LISA is the interface exposed for user applications to utilize its functionality. This interface resembles a typical BSD socket interface except that it requires the programmer to acquire an instance of the service bus for the following standard requests. This segment of the bus handles the specific protocol needed to send requests to the destination or serve other requests coming from clients.

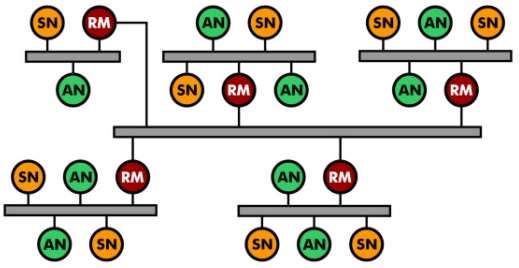

Figure 3. Federated architecture of LISA

4.3 LISA protocol

When an application node requests a service, it has to first locate the destination of the service. In one subnetwork, service nodes and application nodes do not have to know the protocol specific address to communicate, but only a simple service ID assigned by the user. The communication is message based and the manager node only works during the initial network setup phase. However, if the desired target is outside the local network, the manager acts as a message broker across the network, as discussed above. There are two types of messages, setup message during the initial phase (such as discovery, registration and access request) and the user message. There are five types of setup messages which are used as a standard for the bus to be aware of another segment of the bus. These messages are used for service discovery, service registration, application authentication and service deactivation. Each of these messages affect the state of the bus segment in the current node. Registration request message (*SRP_REQ*) is acknowledged with registration confirmation (*SRP_CNF*) upon successful event. An application node requesting this service sends service discovery request (*SDP_REQ*) specifying the service ID. There might be multiple service nodes with the same service id but the manager reply discovery confirmation with the address of the registered services (with that ID) randomly. Once the application node receives the address, the application node sends an access request to the service node which replies with a simple access code.

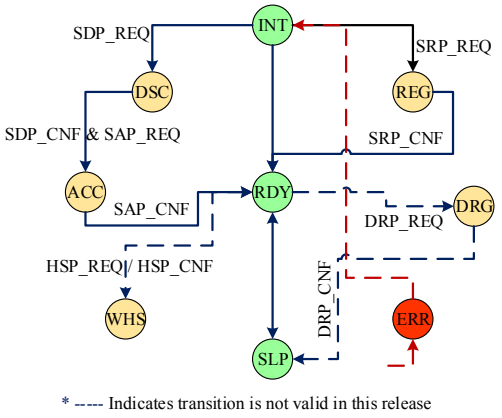

Figure 4. State transition of LISA

The setup messages affect the state of the bus, and hence the state of the underlying network interface. The interface and the bus go into a sleep mode whenever there is no transmission. Advanced synchronization between a service node and application node can be achieved by gathering the registration time and discovery request time from service nodes and application nodes. This feature can provide better power management in the overall sub network[11]. The state transition of the bus segment is handled as shown in the Figure 4. The bus is in transient state if

it is in one of the yellow circles (in Figure 4) and can be utilized by the user application only when it is in the ready state. The left path is taken by application nodes, the middle is taken by manager nodes and the right path is for service nodes. There are states which are not fully implemented in the current version (such as QUD and WHS) as LISA supports message type communication only.

4.4 LISA user messages

As mentioned earlier, the user message format for LISA is adapted from NoTA. Service interface describes what parameters are required to use a service and the type of parameters passed. Considering a sample health monitoring application domain[10], where application nodes (sensors) send readings to a service running on another node, the logging service might accept requests in the form SensorID (unsigned integer), PatientID (unsigned integer) and Reading (float). The type, length and values of each parameter will be ordered accordingly and copied to the buffer in the body of the message. Hence, the above sample service receives messages with content starting with 0x11, code for 8 bit unsigned integer [3], for type of SensorID, followed by the actual value. The receiving end identifies user message from the header and parse the values according to the service definition exposed for the application node.

5. LISA demonstrator

The IoT configuration for the demonstrator utilizes all the three node types: application, service and manager. To setup the network between these nodes, we create three tap interfaces (tap0, tap1 and tap2) and three terminals each running an instance of RIOT inside Linux in a native mode. The first instance acts as an application node, the second one as a service node and the last one as a manager. The manager node is always in the listening state to handle requests from application and service nodes. As shown in Figure 4, the manager goes to ready state directly. Service node starts and sends a registration request with service ID 4 (chosen randomly). In this sample application, we do not have the manager advertisement, and hence we provide manager address when running application and service nodes.

The manager receives the registration request and replies with confirmation once it stores the address of the service node. Once the service is done with registration, it is in a ready state waiting for clients. Whenever the service wants to go into a sleep state, it sends a deregister request for the manager to remove it from the list. Similarly, an application node sends first a service discovery request for the manager. The request is acknowledged with the address of the service if it is found in the current network. The manager sends its own address if the requested service is not in the local network. The current version of the manager node cannot forward to final destination when the service is not local, so the demonstrator concentrates only for local connections. Figure 5 describes the interaction of the nodes and the exchanged messages.

Table 1. Size of LISA vs NoTA.

API name	Size (KB)
LISA only	~22
LISA with RIOT	~130
NoTA	~260

The size of the whole application including the operating system and the required modules is less than 130 Kbytes (as shown in Table 1), from which LISA takes 22 Kbytes (less than 20%). Looking at the running processes from RIOT shell commands, LISA creates only one thread which registers itself for handling incoming packets. This improves power and memory management.

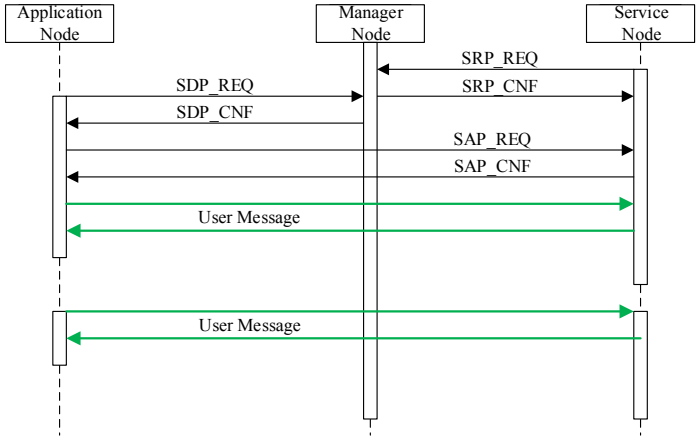

Figure 5. Application, manager and service nodes their message flow

6. Conclusion and future work

We introduced a lightweight embedded service bus to facilitate the implementation of service oriented architecture in Internet of Things, thereby eliminating the challenges of interoperability. Our implementation has been discussed and demonstrated to be compact for resource constrained devices. LISA offers service discovery, registration and authentication that are essential features to setup and communicate application messages between nodes. It also sets a foundation for future extensions and enhancements. The future research, beside the addition of more protocols and platform support, includes studying the possibility of running LISA without any operating system, adding more features for manager node to activate services and implementing inter-subnetwork communication. The future work will also contain the development of an optional configuration for quality-of-service messaging depending on the application type. We will release LISA as an open source project thereby providing a community of developers and in house contribution to enhance the features of LISA.

References

1. CERP-IOT; Vision and Challenges for realizing the Internet of Things; European commission, Information society and media, March 2010.
2. Hauke Petersen, Emmanuel Baccelli, Matthias Wahlisch, Interoperable Services on Constrained Devices in the Internet of Things; *W3C Workshop on the Web of Things*, Berlin, Germany, Jun 2014.
3. Dirk-Jan C. Binnema*;* NoTA programming guide; Nokia Research Centre, 2009.
4. Abdullah S Alghamdi, Iftikhar Ahmad, Muhammad Nasir; Selecting the Best Alternative SOA Service Bus for C4I systems Using Multi-criteria Decision Making technique*; IEEE Region 8 SIBIRCON-2010*, Irkutsk Listvyanka, Russia, 2010.
5. Allseen Alliance; Introduction to the AllJoyn Framework*;* [Online], Available: https://allseenalliance.org, February 28, 2014.
6. Allseen Alliance; Introduction to AllJoyn Thin Library; [Online], Available: https://allseenalliance.org, June 30, 2014.
7. Erik Nordstr¨om, David Shue, Prem Gopalan, Robert Kiefer,Matvey Arye, Steven Y. Ko, Jennifer Rexford, Michael J. Freedman; Serval: An End-Host Stack for Service-Centric Networking; *9th USENIX Symposium*, NSDI 12, 2012.
8. Gustavo Medeiros Araújo, Frank Siqueira; The Device Service Bus: A Solution for Embedded Device Integration through Web Services; *symposium on Applied Computing*, 2009.
9. Emmanuel Baccelli, Oliver Hahm; RIOT: One OS to Rule Them All in the IoT*;* [Research Report] RR-8176, 2012. <hal-00768685v1>
10. Amir-Mohammad Rahmani, Nanda Kumar Thanigaivelan, Tuan Nguyen Gia, Jose Granados, Behailu Negash, Pasi Liljeberg, Hannu Tenhunen; Smart e-Health Gateway: Bringing Intelligence to Internet-of-Things Based Ubiquitous Healthcare Systems; *IEEE Consumer Communications and Networking Conference*, pp. 826-834, USA, 2015.
11. Adam Dunkels, Joakim Eriksson, Nicolas Tsiftes; Low-power Interoperability for the IPv6-based Internet of Things; *Wireless Ad-hoc Networks,* Stockholm, May 2011
12. Gregor Hohpe, Bobby Woolf; *Enterprise Integration Patterns - Designing, Building and Deploying Messaging Solutions*; Addison Wesley, October, 2003.
13. Martin Keen, Amit Acharya, Susan Bishop, Alan Hopkins, Sven Milinski, Chris Nott, Rick Robinson, Jonathan Adams, Paul Verschueren; *Patterns: Implementing an SOA Using an Enterprise Service Bus*; [Online]. Available: http://www.redbooks.ibm.com/ , 2004
14. Jussi Kiljander, Matti EteHipera, Janne Takalo-Mattila, Juha-Pekka Soininen; Opening information of low capacity embedded systems for Smart Spaces; *Intelligent Solutions in Embedded Systems (WISES), 2010 8th Workshop on ,* vol., no., pp.23,28, 8-9 July 2010