SIP as a Universal Communication Bus: A Methodology and an Experimental Study

Benjamin Bertran, Charles Consel, Wilfried Jouve, Hongyu Guan, Patrice Kadionik

SIP as a Universal Communication Bus: A Methodology and an Experimental Study

B. Bertran, C. Consel, W. Jouve
INRIA / University of Bordeaux

H. Guan, P. Kadionik
IMS / University of Bordeaux

***Abstract*—This paper describes a methodology and a programming support that use the SIP protocol as a universal communication bus in pervasive computing environments. In doing so, our work enables homogeneous communications between heterogeneous distributed entities.**

We present a classification of a wide variety of entities in terms of features, capabilities and network connectors. Based on this classification, a methodology and a programming support are described for connecting entities on the SIP communication bus. This work has been validated by applications using the SIP communication bus to coordinate widely varying entities, including serial-based sensors (RS232, 1-Wire), ZigBee devices, X10 devices, PDA, native SIP entities, and software components.

***Index Terms*—Pervasive Computing, SIP, Communication Protocol, Middleware, Embedded Systems.**

I. Introduction

Device-rich, networked environments are becoming increasingly prevalent in areas ranging from building management to healthcare. These pervasive computing environments consist of a variety of entities that are heterogeneous in many respects: (1) they are either hardware (*e.g.,* camera and telephone) or software (*e.g.,* agenda and news); (2) they rely on different network layers (*e.g.,* X10, ZigBee, and IP); (3) they interact using various modes of communication (*e.g.,* events and streams); and, (4) they exchange various kinds of data (*e.g.,* temperature measurements and video streams). Such environments are also highly dynamic with entities appearing and disappearing over time (*e.g.,* a telephone is switched on/off). Moreover, software systems managing these entities need to be open-ended to keep pace with a constant flow of technological advances.

Our research aims to address the heterogeneity and dynamicity of pervasive computing environments by generalizing SIP (Session Initiation Protocol) [1] to a software communication bus. This industry standard for Internet telephony provides a basis to address the challenges of pervasive computing environments. For example, dynamicity can be addressed by leveraging SIP's mechanism for user mobility. The heterogeneous modes of communications between entities can leverage SIP's general-purpose forms of communications, namely multimedia sessions, events and instant messaging. In our previous works [2], [3], we described how SIP addresses both advanced telephony and home automation services. We proposed a Java programming framework to develop such services.

In this paper, we present a methodology and programming support to use SIP as a universal communication bus for pervasive computing environments. Our main contributions are as follows:

- A classification of a wide variety of entities that facilitates their integration in the SIP communication bus.
- A methodology and programming support that make each class of entities SIP compliant.
- An experimental study that validates SIP as a communication bus for pervasive computing environments. This study comprises numerous entities with vastly varying features and capabilities.

This paper is organized as follows. Section II gives some background on the SIP protocol and presents its advantages in a pervasive computing context. Section III describes the general structure of SIP adapters, connecting entities to the SIP universal communication bus. The SIP middleware, which supports SIP communications, is described in Section IV. Section V introduces our experimental platform. Section VI examines our experiment results. Finally, Section VII concludes the paper.

II. A Case for SIP as a Universal Communication Bus

Let us examine the aspects that make SIP an ideal basis to form a universal communication bus.

Extensibility: SIP is an HTTP-like request/response protocol, text-based and transport-independent. Like HTTP, SIP is extensible in terms of methods, headers, and message payload. This allows the protocol to be completed with numerous standardized extensions matching specific needs, namely, instant messaging [4], [5], and events [6]. Message payload is format-independent, enabling SIP to embed any kind of data (*e.g.,* SDP [7], presence information [8], and SOAP [9]).

Interaction modes: Originally designed to deal with sessions, SIP has the potential to provide general-purpose communication forms, namely, commands (RPC-like based on instant messaging), events, and sessions of data streams [1]. These forms of communications cover what is required by an application to coordinate entities in a pervasive computing environment. More specifically, instant messaging is a one-to-one interaction mode; it can be used, for example, to query a temperature measurement from a sensor. Event is a one-to-many interaction mode; it is the preferred mechanism to propagate information such as the presence status. Finally, session is a one-to-one interaction mode with data exchanged over a period of time; it is typically used to set up a multimedia

stream between two entities, but it can be generalized to a stream of arbitrary data. For example, a GPS device produces a stream of Cartesian coordinates.

Environment dynamicity: Dynamicity is an inherent feature of home automation. SIP provides a mechanism that deals with a form of dynamicity, namely user mobility. To address this issue, SIP relies on the use of Uniform Resource Identifiers (URIs) to refer to agents, abstracting over the terminal network address. This mechanism can be used to define functional entities in a pervasive environment, abstracting over concrete entities whose availability may vary over time. As a result, the use of SIP URI shields the application code from runtime configuration changes in the environment.

Existing infrastructures: Because it is a *de facto* standard for IP telephony, SIP platforms are already widely deployed in various forms, including dedicated IP telephony systems and set-top boxes. Pervasive computing applications can thus leverage these platforms, expanding their original scope.

Convergence point: The increasingly prevalent nature of SIP makes it a converging point for many technologies. Beyond SIP phones (whether hardware or software), other SIP-compliant entities are starting to become available (*e.g.,* video camera [10]). In fact, SIP can be embedded in an increasing number of devices and software systems, representing a convergence point of a number of technologies and areas.

III. Building SIP Adapters

We have motivated the use of SIP as a universal communication bus between heterogeneous distributed entities. Let us now examine how entities need to be adapted to connect them to the SIP communication bus. This adaptation process is driven by criteria, classifying entities.

A. Entity classification

Our entity classification uses three criteria. This classification builds on our study of a large panel of entities and factorizes our experience in developing entity-specific adapters to the SIP communication bus. The first criterion is whether or not an entity is SIP native. As shown in Figure 1, a SIP-native entity is directly connected to the SIP communication bus; such entity is referred to as *type 1*. In contrast, a non-SIP entity needs an adapter. To address a non-SIP entity, a second criterion identifies whether it is IP-enabled. If so, a third criterion determines whether the entity is programmable, making it possible to introduce a SIP stack; this class of entities is of *type 2*. *Type 3* is a non-SIP, non-programmable entity; as such, it requires the use of a SIP gateway. *Type 4* is a non-SIP entity without IP capability, requiring an extended gateway. This classification of entities is summarized in Figure 2. Examples are listed in Table I. From this classification, solutions are proposed to create SIP adapters.

B. Functional architecture of a SIP adapter

We now present the layers required to adapt each class of entities to the SIP communication bus, omitting entities of type 1 that support SIP natively. To be SIP compliant, an entity must provide access to its functionalities via SIP-compliant mechanisms. To do so, access to entity functionalities are defined in terms of the three interaction modes available in SIP: commands (*i.e.,* status query and entity control), events (*i.e.,* event publishing and subscription) and sessions (*i.e.,* invitation to a session of data stream). Yet, these interaction modes need to pass and receive data that may have different formats: command-parameter values (*e.g.,* using SOAP), event values (*e.g.,* using an XML-based format [11]) and session-capability descriptions (*e.g.,* using plain text SDP).

Type	Examples	Gateway
1	SIP video camera, SIP phone, SIP softphone	No
2	PDA, Greenphone, Calendar, Monitoring entities	No
3	IP video camera, Printer	Yes
4	X10 or 1-Wire devices, Temperature sensors	Yes

TABLE I: Entity examples

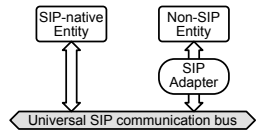

Fig. 1: Adapting entities to SIP

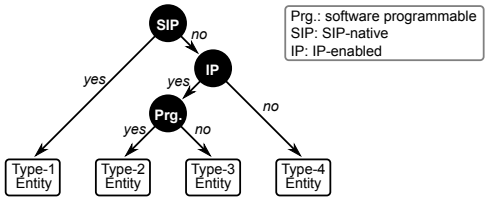

Fig. 2: Entity Classification

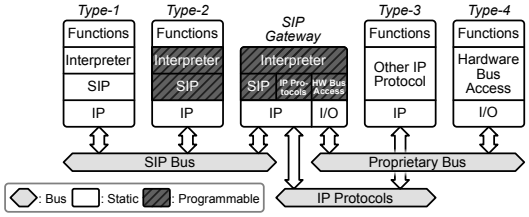

Fig. 3: Entities and Gateway Architecture

Entities of type 2: Despite SIP's rich forms of communications, the SIP communication bus needs careful parameterization to cope with a constant flow of new non-SIP entities, introducing ever changing functionalities and data formats. To address this situation, SIP adapters wrap entity functionalities with an *interpreter*. For a given SIP method, this layer extracts from the payload of a SIP message, the constituent parts of the corresponding interaction mode (*i.e.,* command, event or session). For example, a SIP request with a `MESSAGE` method corresponds to a command interaction. The payload interpreter then extracts from the request payload a SOAP message, indicating the command name (*e.g.,* `getTemperature`) and the parameter values (*e.g,* a measurement unit). The *payload*

interpreter then calls the *invocation layer* of the corresponding interaction mode with its constituent parts. This layer is responsible to invoke the target functionality in the entity (*e.g.,* an operation to measure a temperature, given a measurement unit). Figure 3 depicts the layers involved in adapting a non-SIP, programmable entity to the SIP communication bus. Because a type-2 entity is programmable, its SIP adapter can reside on the entity, making it self-contained.

Entities of types 3 and 4: When a non-SIP entity is not programmable, the SIP adapter is implemented as a hardware gateway. Note that a hardware gateway can also be used for a type-2 entity to reduce energy consumption or increase performance. A hardware gateway is mandatory for a type-4 entity to enable IP and SIP capabilities.

As illustrated in Figure 3, functionalities of entities of types 3 and 4 are accessed through ad hoc communication buses consisting of a software communication bus and an associated hardware communication bus. The hardware communication bus can be proprietary. It may simply be the processor bus of the device. Requested data can be directly accessed via registers mapped in memory. The hardware communication bus can also implement an industry standard such as X10 [12], for power line-based communication, and ZigBee [13], for wireless communication. There are low-level devices that use serial communication buses such as RS232, I2C, or 1-Wire bus [14]. In our approach, these devices are hidden behind a SIP-compliant component that directly accesses their functionalities. In fact, each time a hardware communication bus is used, the corresponding specific software communication bus must be created for hiding underlying hardware specificities.

In practice, our four classes of entities and our methodology have been successful in adapting all the devices and software components that we have encountered in developing a variety of pervasive computing applications.

IV. Enabling SIP communication

We developed a distributed SIP middleware, named DiaGen [15] that allows entities to invoke remote functionalities, receive and answer requests using the SIP communication bus. It also enables entities to interpret SIP payloads implementing the interpreter layer introduced in Section III-B. Leveraging the SIP infrastructure, the DiaGen middleware supports distributed entities with discovery and notification services.

A. Entity binding

The discovery service allows to register and look up entities. SIP provides a basis to deal with the dynamic pervasive computing environments via its support for user mobility. Specifically, SIP entities send a SIP `REGISTER` request to register their SIP URI with the registration server; this server associates entity SIP URIs with network addresses. In addition, our approach consists of using the SIP `OPTIONS` request to complete the registration process with a description of the registering entity.

Once registered, an entity can be looked up by querying the registration server. To do so, a lookup request is sent in a SIP `MESSAGE` request, containing a description of the required entity or entities. The registration server returns all registered entities matching the request.

B. Interaction modes

The notification service allows entities to subscribe and publish events. It improves the scalability of the overall platform by decoupling producers and consumers of events. The notification service receives SIP `PUBLISH` requests containing events from publishers and sends SIP `NOTIFY` requests to all entities that subscribed to the related type of events (*e.g.,* calendar event) using the SIP `SUBSCRIBE` request.

In addition to the event interaction mode, the DiaGen middleware allows entities to interact via the command and session interaction modes. In the command interaction mode, an entity sends a SIP `MESSAGE` request to operate another entity. In the session interaction mode, an entity sends a SIP `INVITE` request to negotiate session parameters and to establish a session with another entity.

Data exchanged between two entities are serialized in the SOAP format using the kSOAP [16] library and transported via both SIP request and response bodies.

V. Experimental platform

Our universal communication bus has been developed in the context of a home automation project. The goal of this project is to design and implement a home automation platform based on SIP. Experiments have been made in a real environment, depicted in Figure 4. This environment was populated by various home automation entities, ranging from telephony equipments to home appliances.

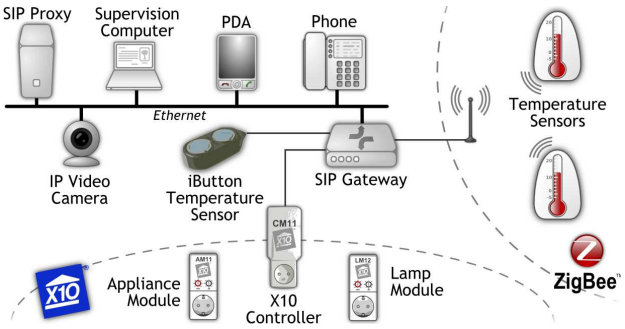

Fig. 4: Experimental platform

This platform serves as a vehicle to experiment with various scenarios. For example, we have developed a surveillance application that involves IP video cameras, X10 alarms, SIP phones and PDAs. Another example is an application displaying various information of interest on a screen, including appointments and weather conditions.

VI. Experimental study

In this section, we validate our use of SIP as a universal communication bus. This validation is done in the context of our experimental platform, equipped with entities belonging to all the types discussed earlier. First, we examine the adaptation work required for each type of entities. Then, we present and analyze performance measurements.

A. Entity adaptations

Let us examine the development work required to make each entity type SIP compatible.

Type-1 entity: By design, the DiaGen middleware is fully compatible with SIP-native entities. Application code developed with the DiaGen middleware can thus directly interact with these entities. This situation allows to leverage existing SIP infrastructures (*e.g.,* OpenSER server) and entities (*e.g.,* SIP video cameras, SIP phones and softphones).

Type-2 entity: There exists a wide variety of existing entities with programming capabilities, ranging from PDAs to software calendars. Our approach consists of providing the developers with a Java programming framework to create invocation layers and to connect entity functionalities to the SIP communication bus. Developers rely on high-level operations to (1) register and lookup entities and (2) implement and invoke entity functionalities. Our Java programming framework abstracts over the intricacies of the underlying technologies and prevents developers from writing boilerplate code, *e.g.,* SIP method creation, payload marshalling/unmarshalling and concurrency handling.

Type-3 entity: The type-3 category consists of non-programmable entities, supporting IP protocols (*e.g.,* HTTP and RTSP for IP video cameras). Making these entities SIP compliant amounts to develop adapters mapping their protocol into SIP. Such adapters form a SIP gateway. Our programming framework provides support for the developers to build such gateway. We propose two approaches to implement a gateway. The first approach is based on Java and requires adequate resources in the platform. The second approach is less resource-demanding: it relies on a C version of our programming framework. We chose this second approach and embedded a C-based gateway into a small Single-Board Computer (SBC) (*e.g.,* an ARM-based board [17]). The functional architecture of a SIP gateway is shown in Figure 5.

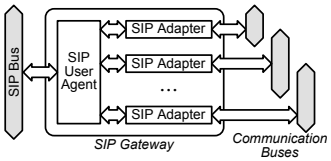

Fig. 5: Software architecture of a SIP gateway

To develop our SIP gateway, we have first ported Linux 2.6 with its own root file system to the SBC board. The GNU oSIP library [18] has then been ported to the SBC board, and a SIP user agent has been developed on top of this library. When deployed, the SIP user agent registers each entity it serves.

To illustrate the use of our SIP gateway, consider the IP video camera. A surveillance entity sends an `INVITE` request to the SIP gateway of the IP video camera to establish a session of video stream with a PDA. The SIP gateway extracts appropriate information from the SIP message and sends an RTSP request to the camera. Once the communication is established, the SIP gateway is no longer involved and the video is streamed directly from the camera to the SIP client of the PDA using the RTP protocol.

Type-4 entity: The type-4 entities represent the majority of devices deployed in a typical home environment. This type consists of entities that are non-programmable and communicate with a non-IP protocol. An adapter needs to be developed for every entity relying on a new protocol. This type of adapter is difficult to write because it involves low-level communication operations. We gather the adapters for non-IP protocols into a SIP gateway (Figure 5). This gateway resides in another SBC board with specific interfaces (*e.g.,* ZigBee and X10). For example, a specific ZigBee SIP adapter gets the temperature measurement from the ZigBee temperature sensor via the serial ZigBee base connected to the SBC board.

For X10 entities, we have ported the Heyu open source project [19] to the SBC board. A specific SIP adapter has been written. A USB CM11 module, which handles several X10 devices, is connected to the SIP adapter. It receives X10 commands from the adapter and sends them to X10 entities connected to the power line network.

For iButton temperature sensor entities, we have modified an open source library developed by Dallas Semiconductors [20] and ported it on Linux. A specific iButton SIP adapter has been written.

B. Results and Discussion

This section assesses the validity of our approach. To do so, we have conducted experimental studies to measure the performance of our platform and its scalability. We omit the analysis of type-1 entities because they are SIP native and provide the required performance by design. In practice, all type-2 entities we encountered offer enough computing power to map functionalities into the operations supported by an entity. As a result, this category of entities incurs negligible overhead.

Type-3 and type-4 entities both require a SIP gateway. However, type-4 entities are the most demanding in terms of computing power because they translate a high-level protocol, namely SIP, into a low-level one, such as ZigBee or iButton. Moreover, the type-4 entities represent the vast majority of the devices deployed in a typical home environment. Consequently, our experimental study concentrates on the type-4 entities.

Our experimental platform includes a SIP gateway that adapts two ZigBee temperature sensors, two X10 entities and an iButton temperature sensor to the SIP communication bus. For the implementation, we used a 180 MHz ARM9 processor running Linux 2.6.20 with 32 MB SDRAM and 8 MB flash memory.

First, we measure the memory footprint of the run-time support of our implementation, using the Exmap-console tool [21]. This measurement was performed on the adapters and the user agent of the SIP gateway. Their sizes are shown in Table II. On our resource-constrained platform, the memory footprint of the entire SIP gateway is 518 KB, representing less than 2% of the total available memory of the SBC board (32 MB). Note that this SIP gateway comprises three SIP adapters.

	SIP adapters			SIP user agent	SIP Gateway
	ZigBee	iButton	X10		
Memory footprint	113 KB	107 KB	326 KB	350 KB	518 KB

TABLE II: SIP gateway Memory footprint

Run Time	Mode	User agent	Adapters	Total
ZigBee read	IM	15.3 ms	175 ms	190.3 ms
	PUB	6.4 ms	175 ms	181.4 ms
iButton read	IM	15.3 ms	557 ms	572.3 ms
	PUB	6.4 ms	557 ms	563.4 ms
X10 write	IM	15.3 ms	373 ms	388.3 ms

TABLE III: SIP gateway run-time overhead
IM: Instant messaging for command
PUB: Publish for event

These figures demonstrate that adapters for non-IP protocols incur minimal overhead, making our approach amenable to resource-constrained platforms.

In our implementation, a command or an event are encoded in SOAP. Like SIP, SOAP uses textual representation. As a result, message processing is much more computation intensive than binary encoding such as BER [22]. However, SOAP deals with complex data structures, facilitates interoperability and enables extensibility.

Table III reports on the run time of our SIP gateway. The first column lists read and write operations on ZigBee, iButton and X10 devices. The second column gives the mode of the read/write operation, which can either be implemented as an instant message or an event publication. The remaining columns provide the execution time of the implementation mode, the adapter and the total time, respectively.

We observe that the SIP user agent executes an event (less than 7 ms) twice as fast as a command (less than 16 ms). This is due to the fact that a command produces a full-fledged return value, whereas an event returns a status. Examining the measurements of the adapters, we note that their run times vary widely. This variation depends on the nature of the non-IP protocols. Specifically, the iButton sensor uses a 1-Wire bus that is much slower than the other communication buses. This results in making the iButton adapter a bottleneck (more than 550 ms), compared the processing of SIP messages performed by the user agent (less than 16 ms).

In fact, one can notice that the processing time of the user agent is 10 to 90 times faster than the adapters. This observation leads us to introduce a multithreaded SIP gateway to optimize the SBC board resources. We used POSIX threads to cache values of sensors. It allows to increase scalability of our SIP gateway. Our implementation deals with more than 60 commands (1308 bytes per command on average) or 150 events (1346 bytes per event on average) per second. Based on the interactions we had with our industrial partners in the telecommunication domain, this performance fulfills the requirements of realistic home environments.

To evaluate our SIP gateway, we also measured the run time of our implementation, varying the processor frequency from 180 MHz down to 80 MHz. We observed that decreasing the processor frequency increases the run time to handle a command or event, almost linearly. However, the execution time to read a value in entities is almost constant, since this operation depends on the nature of the target proprietary bus. Thus, with threads, the maximum bandwidth provided by our SIP gateway for command or event is practically linear in the processor frequency. It allows users to scale the hardware to meet the requirements of the target environment.

VII. Conclusion

We have presented an approach to enabling homogeneous communications between heterogeneous distributed entities. This approach relies on the use of SIP as a universal communication bus for pervasive computing environments. We described a methodology and programming support to adapt heterogeneous entities to the SIP communication bus. Our approach has been used to make a wide variety of entities SIP compliant. These entities have then been integrated into a number of applications for home automation. Finally, our experimental study has proved that our approach is realistic for all classes of entities, and that our SIP gateway can run efficiently on resource-constrained platforms.

References

[1] Rosenberg, J. et al., "SIP : Session Initiation Protocol," IETF, RFC 3261, Jun. 2002.

[2] W. Jouve, N. Palix, C. Consel, and P. Kadionik, "A SIP-based programming framework for advanced telephony applications," in *Proceedings of The 2nd LNCS Conference on Principles, Systems and Applications of IP Telecommunications (IPTComm'08)*, 2008.

[3] B. Bertran, C. Consel, P. Kadionik, and B. Lamer, "A SIP-based home Automation Platform: an experimental study," in *13th International Conference on Intelligence in Next Generation Networks*. IEEE, 2009.

[4] B. Campbell, J. Rosenberg, H. Schulzrinne, C. Huitema, and D. Gurle, "Session initiation protocol (SIP) extension for instant messaging," IETF, RFC 3428, 2002.

[5] "SIP for Instant Messaging and Presence Leveraging Extensions. IETF Working Group, http://www.ietf.org/html.charters/simple-charter.html."

[6] A. B. Roach, "Session Initiation Protocol (SIP)-Specific Event Notification," IETF, RFC 3265, Jun. 2002.

[7] M. Handley and V. Jacobson, "SDP: Session Description Protocol," IETF, RFC 2327, 1998.

[8] H. Sugano, S. Fujimoto, G. Klyne, A. Bateman, W. Carr, and J. Peterson, "Presence Information Data Format (PIDF)," IETF, RFC 3863, 2004.

[9] X. Wu, P. Koskelainen, and H. Schulzrinne, "Use of session initiation protocol (SIP) and simple object access protocol (SOAP) for conference floor control," IETF, Internet Draft, September 2003.

[10] "Mobotix SIP Cameras, http://www.abptech.com/products/Mobotix/."

[11] J. Rosenberg, "A presence event package for the session initiation protocol SIP: Session Initiation Protocol," IETF, RFC 3856, 2004.

[12] "X10 communication protocol, http://www.x10.org."

[13] "The ZigBee Alliance, http://www.zigbee.org."

[14] Maxim, "The 1-Wire Bus, http://www.maxim-ic.com/products/1-wire/."

[15] D. Cassou, B. Bertran, N. Loriant, and C. Consel, "A generative programming approach to developing pervasive computing systems," in *Proceedings of the 8th International Conference on Generative Programming and Component Engineering (GPCE'09)*, 2009.

[16] "kSOAP 2, http://ksoap2.sourceforge.net."

[17] "Eukréa SBC Board, http://www.eukrea.com."

[18] "The GNU oSIP library, http://www.gnu.org/software/osip/."

[19] "The Heyu project, http://heyu.tanj.com/."

[20] "iButton SDK, http://www.maxim-ic.com/products/ibutton/."

[21] "Exmap-console tool, http://labs.o-hand.com/exmap-console."

[22] "Asn.1 encoding rules: Specification of basic encoding rules (BER), canonical encoding rules (CER) and distinguished encoding rules (DER)." ITU-T X.690.