A SIP-Based Home Automation Platform: An Experimental Study

Benjamin Bertran, Charles Consel, Patrice Kadionik, Bastien Lamer

A SIP-based Home Automation Platform: an Experimental Study

Benjamin Bertran, Charles Consel
INRIA / LaBRI
351, Cours de la Libération
F-33405 Talence Cedex, France
{benjamin.bertran, charles.consel}@inria.fr

Patrice Kadionik
IMS / University of Bordeaux
351, Cours de la Libération
F-33405 Talence Cedex, France
patrice.kadionik@enseirb.fr

Bastien Lamer
Orange Labs
bastien.lamer@orange-ftgroup.com

***Abstract*—SIP has demonstrated its effectiveness in enabling distributed entities to exchange any media using various interaction modes. In doing so, this protocol is showing great promise to support much extended forms of telecommunication services.**

This paper reports on the use of SIP as a communication middleware to support home automation applications that consist of heterogeneous, distributed entities. We describe how SIP fulfills the requirements of home automation; we present the resulting architecture of a home automation system; and, we validate our approach with various scenarios.

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

I. Introduction

Homes and buildings are being equipped with a growing set of technologies to automate management tasks. Such tasks span from light and heating handling to energy saving and security. Although promising, the development of home automation systems raises a number of challenges. In particular, it requires to deal with widely heterogeneous entities (whether hardware or software), different modes of interaction (*e.g.,* events and streams), various kinds of data exchanges (*e.g.,* temperature measurements), and entities dynamically available. While existing software engineering approaches support the developers for the most error-prone tasks, they still lack a platform that relies on industrial standards to tackle the challenges of the home automation domain.

In this paper we present an approach and a platform that rely on the SIP protocol [1]. Commonly used in telephony, multimedia streaming, and instant messaging, SIP is a great candidate to also become a communication bus for the constituent components of a home automation system. Additionally, the open-ended nature of SIP provides a good basis to address issues that are specific to the home automation domain.

SIP is now widely deployed in both local and global telecommunication infrastructures. To leverage these infrastructures, extensions required by home automation applications need to be introduced seamlessly, preserving compatibility with existing SIP equipments such as phones, proxies, IM-agents, and video-cameras. To leverage non-SIP equipments commonly used in a home automation environment, we uniformly view them as SIP entities by mapping their interfaces into SIP communication modes.

Our approach is supported by a case study of home automation scenarios defined by Orange Labs. These scenarios exhibit the key requirements of the home automation domain. To fulfill these requirements, we developed a specific software layer over SIP. This layer factorizes the common operations needed to develop home automation applications. It has been used to successfully implement Orange Labs' scenarios.

This paper is structured as follows. Section II presents the Orange Labs' scenarios and a typical home automation architecture; it also outlines the requirements they suggest. Section III describes our approach. Section IV examines how the requirements specific to the home automation domain can either mapped directly into SIP features or expressed as SIP extensions. Section V introduces a programming support to develop home automation applications over SIP. Section VI reports on the benefits of using our approach to develop Orange Labs' scenarios. Finally, Section VII gives some concluding remarks.

II. Requirements

This section presents our scenarios and identify their needs, limitations, and risks. Additionally, these scenarios illustrate a number of concerns specific to the home automation domain. Finally, this section introduces a typical home automation architecture from which we derive our SIP-based platform.

A. Scenarios

Let us examine three scenarios involving the use of a variety of devices including alarms, video cameras, phones and televisions. These devices are connected by a home network infrastructure. Besides devices, our working scenarios also consist of external software services such as agendas and RSS feeds.

a) Advanced intercom: In this scenario, when someone uses the home intercom, it calls every phone in the house. After a period of time, if the call has not been answered, it is redirected to the mobile phone of one of the home

978-1-4244-4694-0/09/$25.00

owners. Whoever gets the call can talk to the visitor, as well as remotely open the door using the keypad of his/her phone.

This application not only illustrates the use of existing SIP features, such as audio streams and DTMF [2], but it also exhibits the need to leverage the telephony infrastructure to enable home equipments (*e.g.,* doors, lights, alarms) to be controlled remotely.

b) Anti-intrusion: This scenario is dedicated to house security. When an intrusion is detected, the application sends an SMS and an email to a house owner together with a video. It also calls the police, providing specific information (*e.g.,* number of detected intruders and area of intrusion) using a text-to-speech component.

This scenario demonstrates the need for advanced event mechanisms to bring rich information to applications (for instance, the location of the intrusion). It also points out the need to attach non-functional information to distinguish and discover entities. For example, if each video camera has an attribute specifying its location, then it is possible to dynamically and selectively record a video of the intrusion scene. Finally, this scenario shows home components interact with external services (*e.g.,* SMS).

c) Media content information: Given a user-provided set of television preferences (genres, actors...), this application sends him/her an SMS message, announcing every matching TV program. Responding to this message triggers the recording of the announced program.

This scenario underlines the need for combining home equipments and external services to perform a number of tasks. It also introduces the notion of user preferences, requiring some applications to be parameterized by the user.

Requirements: Beyond the requirements directly entailed by our working scenarios, other requirements are common to most home automation scenarios.

Heterogeneity of entities requires an approach to abstracting away entity features that are not relevant to applications (*e.g.,* protocol, model, and firmware version). This approach would allow entities that share common functionalities to be uniformly manipulated.

To do so, a key issue is to define a uniform model to interact with entities. A study of a range of existing entities suggests that three interaction modes are needed: commands, events, and sessions. Commands are used to perform actions on devices (*e.g.,* operating a light or triggering an alarm). Events enable entities to react to situations by pushing information into the application (*e.g.,* motion detection and temperature change). A session mechanism allows to configure a communication channel to exchange data over a period of time.

Making entities remotely operable raises security issues. For instance, video cameras represent a threat to the home owner's privacy. For another example, gaining control of equipments may enable an intrusion (*e.g.,* unlocking a door or opening shutters). These issues need to be taken into account for the design of our home automation platform.

B. Home Automation Architecture

Let us give an overview of our architecture and provide a preliminary assessment of its capabilities.

1) Architecture overview: For convenience, we distinguish the home environment from the rest of the world. In the *home automation architecture* depicted in Figure 1, the residential gateway makes a bridge between these two environments. This component provides a complete infrastructure to share Internet access (*e.g.,* ADSL modem, router, and firewall). It also contains an entire system with storage, and computing capabilities (*e.g.,* hard drive, general-purpose CPU, sizable memory, USB ports, *etc.*). The gateway provides the user with several services: telephony, TV (stream and recording), and Internet access.

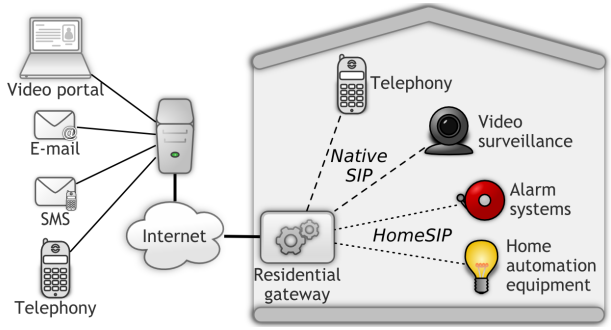

Fig. 1. Home automation architecture

The home automation platform has been designed to be integrated as part of the residential gateway. In doing so, the platform can leverage embedded components such as a SIP proxy. As such, the gateway is the center of the home infrastructure, and thus, is responsible for exposing home services to the outside world and ensuring access to external services like agendas or e-mail servers.

2) Home devices: The home environment is populated with numerous devices. Each of them provides services accessible through specific interaction modes and communication technologies (*e.g.,* Web Services, CGI, CORBA). In our approach, the constituents of the home automation environment are uniformly viewed as SIP entities. Therefore, their interfaces are mapped into our uniform interaction model, namely commands, events, and sessions.

When home automation devices use proprietary protocols, their deployment in our architecture requires the development of *wrappers*. This software layer translates proprietary protocols into SIP. Figure 2 focuses on the internal structure of the residential gateway and the devices. As can be noticed, wrappers can be located either (1) inside the residential gateway as a module, (2) in an adaptation gateway serving multiple devices, or (3) directly in the device, whenever possible.

3) Applications: In our study, applications are hosted in the home network (either in the residential gateway or in a dedicated device) and provide various services to the user. These applications control, manage, and coordinate devices and software components (whether inside or outside the home);

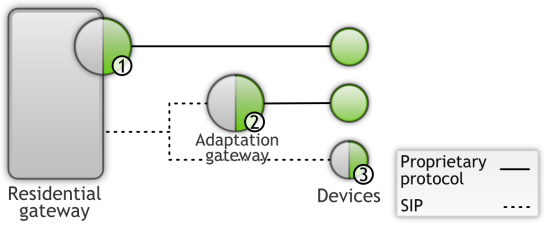

Fig. 2. Device wrapping

they are seen as SIP agents. This configuration allows some services to keep on running in case of network connection failure (WAN side of the residential gateway), such as local home automation ones (*e.g.,* light control).

III. Our approach

Because SIP is HTTP-inspired, it is highly versatile and extensible. These features make this protocol amenable to be used in a range of extended forms of communications such as instant messaging [3], [4], presence [5], as well as converged applications combining telephony with software systems [6]. Furthermore, SIP enables to transport a range of data formats by leveraging such protocols as SDP [7], PIDF [8], and SOAP. More generally, SIP is widely used in ToIP infrastructures, ranging from small domestic environments to country-wide telephony networks.

Our approach aims to leverage a SIP-based infrastructure and its key benefits to developing home automation applications. To do so, several adaptations are needed to fulfill the home automation requirements. These adaptations must be in conformance with the SIP protocol and reuse existing extensions whenever possible. These two constraints are critical for a seamless integration of our approach in already-deployed SIP infrastructures. These adaptations are described in the next section.

To facilitate the use of our approach by developers, we provide them with a complete programming framework on top of SIP and our adaptations. Thanks to this support, developers manipulate high-level concepts (*e.g.,* service discovery) in Java. As well, a developer does not have to forge SIP messages nor manage SIP transactions, preventing him from writing boilerplate code.

Because this programming support is generated with respect to a description of the home environment, it guides the development of applications with typed operations and methods to be implemented.

IV. SIP Adaptations

This section presents how some requirements can be directly mapped into SIP and what adaptations are needed to address the remaining ones. Our proposed adaptations revolve around the use SOAP to transfer arbitrarily rich data. SOAP messages are embedded in SIP message payloads. These adaptations are designed to be in conformance with SIP platforms and SIP-native agents.

A. Registration and discovery

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. We can use this mechanism to identify an entity, whether hardware or software, by its name expressed as a URI. Entities can thus be viewed as SIP agents that can be found via their URI locally as well as throughout the Internet. To achieve this, SIP agents register to SIP proxies that store basic network information (IP address, port) associated to their URI.

Fig. 3. Registration (a) and discovery (b) processes

Although rudimentary, this SIP mechanism can be used to achieve entity discovery. We propose to extend the registration process of SIP agents with a semantic description of entities they correspond to. To do so, we decompose this process in two steps. The first one is the normal registration procedure and uses the `REGISTER` message. This allows native SIP agents to register and provide their URIs and network information, as mentioned earlier. Unfortunately, a URI does not give any information about the nature of its associated entity (*e.g.,* device type and location). To obtain additional information from an entity, we query the entity with the `OPTIONS` message. If the entity belongs to our approach, it returns an enriched response describing itself in terms of attributes and its type name; this response is stored for later use. An attribute is property-value pair characterizing the entity. The type name denotes a set of entities that share the same functionalities (*e.g.,* a light, a fan, and an alarm), allowing the application to manipulate them uniformly.

The type name of entities, together with their attributes, are used by the application to discover entities in a given home environment. A query for entity discovery takes the form of a `MESSAGE` message including the search criteria. Then the response message contains a list of matching entities. This exchange is built as a command invocation, described below. Figure 3 summarizes the registration process and the entity discovery.

B. Commands

Home automation devices often make their functionalities accessible via an RPC-like command, which is a one-to-one operation between two services. A command invocation consists of a name and argument values; it produces a return value. We implement this mechanism in SIP using extensions for instant messaging [3], [4]. A command invocation is encoded using SOAP. The caller builds a `MESSAGE` message with a SOAP payload. The targeted service decodes the SOAP message and executes the command code. When the

execution completes successfully, the result is encoded into a 200 `MESSAGE` response. Otherwise, an error message is returned.

C. Events

Events are based on the publish/subscribe paradigm [9], [10]. In this model, an entity publishes its events to an *event notifier*, which in turn notifies the subscribing entities. This is a one-to-many interaction, where the publisher does not know subscribers (illustrated in Figure 4). A subscriber targets a particular event coming from a specific entity (*i.e.,* a URI). Several existing SIP extensions address this interaction mode [1], [11] but none allows arbitrary payloads for event subscription and publication.

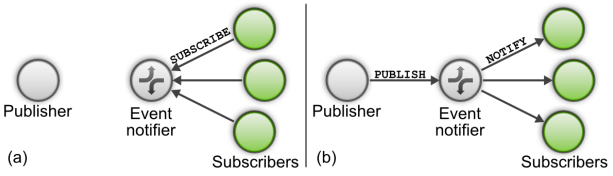

Fig. 4. Event interaction mode: (a) subscription (b) publication

We lift this limitation by introducing SOAP over SIP event messages. `PUBLISH` request body consists of the SOAP-encoded event name and event value. Similarly, `SUBSCRIBE` message consists of the SOAP-encoded URI of the publisher and the event name. In addition to the contents of a `PUBLISH` payload, a `NOTIFY` payload includes the URI of the event publisher to pass this information to the event subscribers.

D. Sessions

Sessions, combined with SDP, are essential to exchange streamed data such as audio and video. They represent a flexible interaction mode because they allow parameters of the streamed data to be negotiated between the parties. We generalized the negotiation process, again by using SOAP-encoded payloads. This generalization allows, for example, to negotiate the rate at which a temperature sensor sends its measurements.

V. The DiaSpec Approach

DiaSpec [12], [13], [14] is a lightweight Architecture Description Language [15] (ADL) dedicated to the pervasive computing domain. From a DiaSpec specification, the DiaSpec compiler generates a dedicated programming framework. The generated support provides the developer with high-level programming mechanisms, abstracting over the underlying communication layer, namely SIP. This makes it possible to write applications and device wrappers, without knowing about this protocol. Figure 5 illustrates the DiaSpec approach.

A. The DiaSpec Language

A DiaSpec specification defines a taxonomy of entities dedicated to the target application area. It consists of declarations of classes of entities, each declaration gathers entities

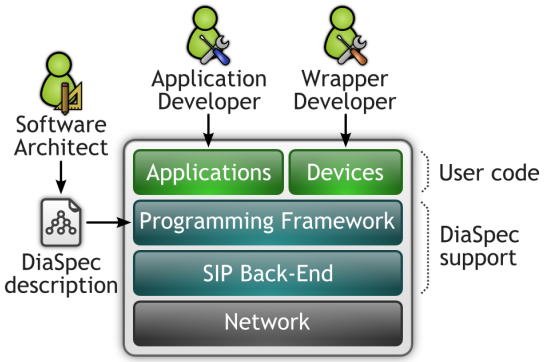

Fig. 5. The DiaSpec software structure

that share commonalities; their differences are expressed by attribute declarations; and, three connector declarations are used to define their interactions with other devices, namely, events, commands, and sessions. Attributes can represent a constant property (*e.g.,* a color or a range) or a dynamic state, such as the current location of a mobile object.

Figure 6 describes the architecture of a light regulation application. The environment is composed of lights, light sensors, and a controller able to receive information from sensors and to trigger operations on lights. This architecture illustrates both the command and event interaction modes. As can be noticed, each declared connector is optionally refined with the class of entities it may interact with. For example, the `LightSensor` component provides the `Luminosity` event *to* `LightController` entities. The `LightController` component requires the `Variation` command *from* the `Light` entities.

```
component Device(String building, String room) { }

component Light extends Device {
  provides command Variation to LightController;
}

component LightSensor extends Device {
  provides event Luminosity to LightController;
}

component LightController  {
  requires command Variation from Light;
  requires event Luminosity from LightSensor;
}

icommand Variation {
  void increase();
  void decrease();
  void setLevel(int value);
}
```

Fig. 6. A DiaSpec specification

B. Programming Support

From a DiaSpec description, the DiaSpec compiler generates a dedicated programming framework. Conforming to the specified environment, it provides the developer with a Java programming support, facilitating creation of applications and wrappers. The generated support is independent of a

given communication technology. Developers only manipulate high-level distributed programming concepts (*e.g.,* registration, discovery and remote calls).

The generated programming framework is highly customized with respect to a given DiaSpec description. Figure 7 shows the use of this generated support. Implementation of `MyLightController` constructor looks for `LightSensors` in the building `A29`, then subscribes to the `Luminosity` event. The notify method is called if the `LightController` receives a notification of a `Luminosity` event. In this implementation, lights of the room are dimmed or brightened depending on the luminosity event value.

```
public class MyLightController extends
    LightController {

  public MyLightController() {
    LightSensorComposite sensors =
      select(lightSensorsWhere().building("A29"));
    sensors.subscribeLuminosity();
  }

  @Override
  public void notify(Proxy servicePublisher,
      Luminosity event) {
    LightComposite lights =
      select(lightsWhere().
      building(event.building).room(event.room));
    if (event.luminosityValue < 5000)
      lights.increase();
    else if (event.luminosityValue > 6000)
      lights.decrease();
  }
}
```

Fig. 7. A `LightController` implementation using supplied framework

This code fragment illustrates discoveries, subscriptions and commands. Every object used in this implementation comes from the dedicated support, generated from the DiaSpec description shown earlier (Figure 6). This support hides the underlying communication technology, here SIP. Each technology is addressed by its own back-end.

C. SIP Back-End

We have implemented a SIP back-end for DiaSpec. It is responsible for mapping DiaSpec concepts into the ones of SIP. To do so, our back-end makes use of the SIP adaptations described earlier. We describe this phase by giving an example of its output.

```
MESSAGE sip:Light.Kitchen@home.com SIP/2.0
From: <sip:MyLightController@home.com>;tag=cefd113d
To: <sip:Light.Kitchen@home.com>
Call-ID: 2ad17cb28971a961a669411c6acc2c64@home.com
CSeq: 11 MESSAGE
[...]
User-Agent: DiaSpec v1.1
Content-Type: application/soap+xml
Content-Length: 327

<v:Envelope
 xmlns:i="http://www.w3.org/2001/XMLSchema-instance"
 xmlns:d="http://www.w3.org/2001/XMLSchema"
 xmlns:c="http://www.w3.org/2001/12/soap-encoding"
 xmlns:v="http://www.w3.org/2001/12/soap-envelope">
 <v:Header />
 <v:Body>
  <n0:increase id="o0" c:root="1" />
 </v:Body>
</v:Envelope>
```

Fig. 8. A SIP message generated by the back-end

Figure 8 shows a SIP message, generated by the instruction `lights.increase();` included in Figure 7. For each light, the SIP back-end generates and sends a message containing the call to the `increase()` method.

VI. Discussions

Several scenarios, including the ones previously described, were successfully implemented and deployed. These experiments allowed us to validate our SIP-based platform in practice. We now discuss the main lessons learned doing this work.

a) Service discovery: The discovery mechanism has proved its effectiveness in dealing with highly-dynamic environments where appliances are frequently introduced/turned on and removed/turned off.

Due to class-based definitions, the discovery mechanism allows applications to dynamically and transparently integrate new kinds of devices. This capability enables the home automation system to evolve and grow without requiring a new support to be generated.

b) Interaction modes: Our proposed interaction modes covered all the encountered situations. Choosing the appropriate interaction mode is simple in practice. Commands are naturally used for simple one-to-one interaction to operate devices or software components. Events are particularly well-suited to push values into all interested entities when some situation occurs. Last, in our set of working scenarios, sessions are not used beyond audio streams.

c) Programming abstractions: As mentioned earlier, the SIP adaptations are introduced by a back-end. It factorizes the boiler-plate code to map our programming support to a SIP platform (*e.g.,* message construction). The generated programming framework allows the logic for both the application and the wrapping of devices to be developed, without requiring SIP knowledge.

Code generated by our back-end mainly sends and receives SIP messages, including creating and parsing these messages. This processing critically relies on Jain SIP [16], [17]. In addition, kSOAP [18] is used to encode and decode SOAP messages.

d) Application creation: Relying on a programming framework allowed us to quickly develop, adapt, and check our applications. As well, it facilitates the creation of new applications. For example, the home architecture described in Section II gives a structured basis to integrate new applications into the residential gateway.

e) Proprietary protocol integration: Our experiments were made using real devices. To do so, we wrapped several proprietary protocols into SIP, including X10, to control binary and dimming devices, ZigBee, to take advantage of various

sensors (*e.g.,* temperature, light), HTTP, to control video cameras (*e.g.,* motion, zoom, snapshot), and Web Services, to get agenda information or TV programs.

In terms of performance, we noticed that resource consumption induced by SIP is often very low compared with the driver itself. In most cases, performance is bounded by the wrapped technology. For example, the ZigBee latency is about fifty times higher than processing SIP messages.

The most difficult part during technology wrapping is the adaptation of our SIP concepts into the target technology. For example, a ZigBee sensor has to be solicited to give its current value. This requires to introduce a polling process to check the sensor value. This situation often requires extra code and introduces an overhead.

VII. Conclusion

We have presented a home automation platform based on SIP and other industrial standards. Our platform not only fulfills the requirements of home automation, but it also leverages the infrastructure of a telecommunication carrier. It enables the coordination of heterogeneous entities, whether or not SIP-based. The programming of home automation applications is greatly facilitated by a customized programming framework. Our approach has been successfully used to develop and test home automation scenarios from Orange Labs.

References

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

[2] ITU-T. Recommendation Q.23: Technical features of push-button telephone sets, http://www.itu.int/rec/T-REC-Q.23/.

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

[4] G. Klyne and D. Atkins. Common Presence and Instant Messaging (CPIM): Message format. RFC 3862, IETF, 2004.

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

[6] 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)*, Heidelberg, Germany, jul 2008. Best Student Paper Award.

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

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

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

[10] J. Rosenberg, H. Schulzrinne, and O. Levin. A session initiation protocol (SIP) event package for conference state. RFC 4575, IETF, 2006.

[11] A. Niemi and D. Atkins. Session initiation protocol (SIP) extension for event state publication. RFC 3903, IETF, 2004.

[12] W. Jouve, J. Lancia, N. Palix, C. Consel, and J. Lawall. High-level programming support for robust pervasive computing applications. In *Proceedings of the 6th IEEE Conference on Pervasive Computing and Communications (PERCOM'08)*, pages 252–255, Hong Kong, China, mar 2008.

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

[14] DiaSpec, http://diaspec.bordeaux.inria.fr.

[15] Nenad Medvidovic and Richard N. Taylor. A classification and comparison framework for software architecture description languages. *IEEE Transactions on Software Engineering*, 26(1):70–93, 2000.

[16] JAIN-SIP, JAVA API for SIP Signaling, https://jain-sip.dev.java.net.

[17] Sun Microsystems. The JAIN SIP API specification v1.1. Technical report, Sun Microsystems, June 2003.

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