

MISE EN ŒUVRE D'UNE PILE SIP SUR UN SYSTEME EMBARQUE POUR LE CONTROLE DE CAPTEURS



Professeurs Responsables :
M KADIONIK Patrice
M NOUEL Patrice

Réalisé par :
ABID Mohamed
CHEBROU Anne
FAVOT Adrien
ELLERO Sébastien
KOEHLER Stéphane

Sommaire

I. Introduction	3
a) Contexte.....	3
b) Domotique.....	3
c) HomeSIP	4
d) Choix du protocole	5
e) Bibliothèque libre oSip.....	6
f) Présentation du projet	6
II. Protocole SIP	7
a) Couche réseau.....	7
b) Messages SIP.....	8
c) Relation entre les messages SUBSCRIBE et NOTIFY	13
d) Fonctionnement du message SUSCRIBE.....	13
e) Fonctionnement du message NOTIFY	14
III. Plateforme ENSEIRB	16
a) Environnement de travail.....	17
b) Plateforme de développement	17
c) Carte ARM09 et I-Button.....	18
IV. Implémentation de la gestion des évènements synchrones	20
a) Envoi de requêtes	20
b) Accusé de reception et récupération des informations	21
V. Implémentation de la gestion des évènements asynchrones	22
a) Evènements asynchrones et domotique.....	22
b) Stockage des abonnés sur le serveur.....	23
c) Organisation des tâches serveur	24
d) Envoi de requêtes du client.....	25
VI. Phase de test	26
a) Programme test client	26
b) Programme test serveur.....	26
c) Interaction entre les tâches (thread).....	26
d) Historique des tests.....	30
e) Evolution du programme.....	30
VII. Conclusion	31
VIII. Références	31

Mots Clés

Système embarqué, Linux embarqué, SIP, domotique, applications client/serveur, évènements synchrones, évènements asynchrones, traitement multitâche

I.Introduction

a) Contexte

La rapidité des progrès technologiques dans le secteur des communications multimédia, en particulier la convergence de la télécommunication et des réseaux informatiques, a entraîné l'apparition d'une nouvelle classe d'applications parfois appelée « everywhere ». Ce néologisme est formé de 2 mots : « everywhere » et (hard/soft)ware, il englobe plusieurs termes : informatique ambiante, informatique pervasive, informatique omniprésente (ubiquitous computing), intelligence ambiante... Cela signifie étendre l'informatique, qui est pour l'instant limitée aux ordinateurs, aux objets qui nous entourent et qui n'ont à priori aucune vocation informatique, afin de faciliter les tâches usuelles de la vie de tous les jours.

Cette nouvelle classe d'applications consiste à rassembler, coordonner, transmettre, traiter et réagir aux informations émanant des entités de communication (ou des objets), locales ou éloignées, fixes ou mobiles. Ces activités sont présentes dans divers domaines comprenant la médecine, l'automatique, la sûreté militaire et la maison individuelle. Cette classe d'applications nécessite un nouveau support d'exécution. De plus, un composant contrôlant la coordination entre les divers dispositifs câblés (sondes, appareils photos, etc.) est primordial. Nous appellerons ce composant l'infrastructure de communication. Cette infrastructure contrôle tous les aspects de la communication entre les entités de l'application. Elle tient compte de l'hétérogénéité des entités matérielles et des fonctionnalités des réseaux utilisés.

b) Domotique

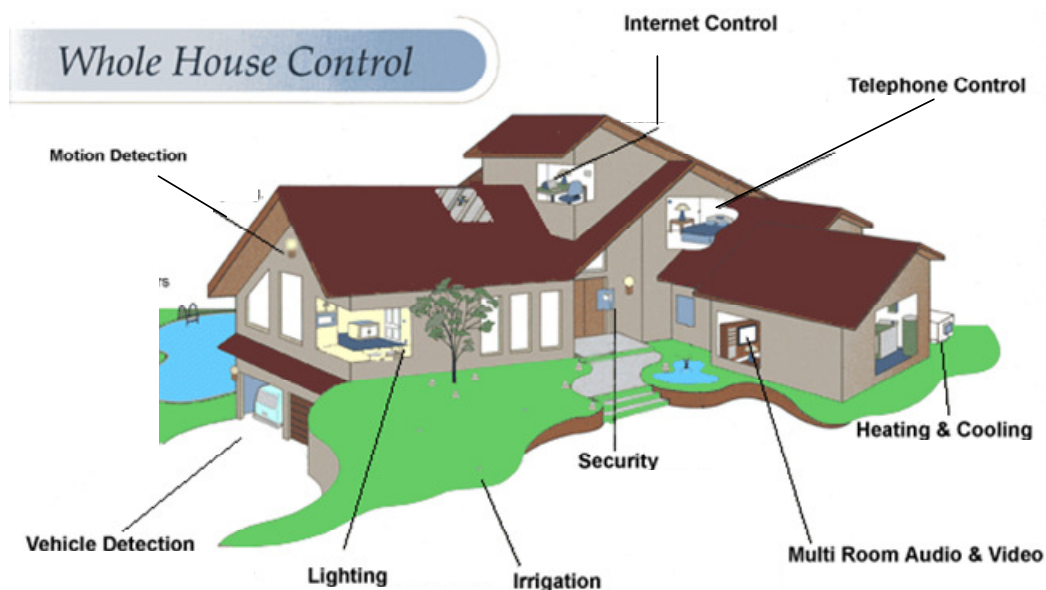


Figure 1 : Applications domotiques

La domotique désigne l'ensemble des techniques et technologies permettant de superviser, d'automatiser, de programmer, et de coordonner les tâches de confort, de sécurité de maintenance et plus généralement de service dans l'habitat. Son but est d'interfacer les « entrées/sorties » de l'habitat pour les commander de manière centralisée et parfois distante (éventuellement à l'aide de scénarii d'usage) pour procurer à l'utilisateur un certain « confort d'usage ». La figure 1 présente les applications les plus courantes comme :

- sécurité des biens et des personnes
- gestion d'ambiance (sonores ou lumineuses)
- régulation de chauffage ou de climatisation
- plus généralement, tout ce qui touche à l'électricité, l'électrotechnique, l'électronique, l'informatique, l'automatique, le multimedia, ...

c) HomeSIP



Figure 2 : Différents partenaires du projet HomeSIP

Initié début 2006, le projet HomeSIP est le fruit d'un travail de collaboration entre 2 groupes de travail des laboratoires de l'université Bordeaux I :

- groupe Cosynux du laboratoire d'électronique IXL.
- groupe Phoenix du laboratoire d'informatique Labri.

Le projet HomeSIP a pour objectif l'élaboration d'une plateforme domotique basée sur le protocole SIP. Ce projet est par nature un projet orienté embarqué, composé de différents systèmes électroniques sous Linux embarqué.

Le projet HomeSIP consiste donc :

- en une infrastructure matérielle composée de capteurs et d'actionneurs connectés à des systèmes embarqués qui sont communicants et qui possèdent tous une connectivité IP.
- à mettre en place les logiciels adéquats dans les systèmes embarqués sous Linux.
- à développer des langages dédiés de type DSL (*Domain Specific Language*) pour développer de nouveaux services autour de la plateforme.

d) Choix du protocole

En vue d'une application domotique, 2 types de transferts de données sont nécessaires :

- transfert synchrone : une donnée actuelle est acquise par une application, par exemple la valeur courante d'un capteur. On peut également vouloir réaliser une action, par exemple changer l'état courant d'un actionneur. Un exemple de capteur peut être un capteur de température et un actionneur, une gâchette électrique de porte. On doit pouvoir réaliser des actions « GET input » et « PUT output ».
- transfert asynchrone : une alarme est envoyée de façon asynchrone à une application en cas de survenue d'un problème quelconque. Un exemple d'alarme est la détection d'un début d'incendie. On doit pouvoir collecter des alarmes ou « TRAP ».

Que nous offre SIP pour cela ? Suivant les RFC 3261, 3265 et 3428, on peut utiliser :

- message SIP : MESSAGE, pour des actions PUT ou GET. Ce message a été originellement introduit pour la messagerie instantanée (RFC 3428).
- messages SIP : SUBSCRIBE et NOTIFY, pour les événements de type TRAP. Ces messages sont originellement prévus pour la notification de présence (RFC 3265).

Le protocole SIP possède des qualités intrinsèques pour la domotique :

- il supporte un mode d'adressage abstrait
- il apporte un niveau de confidentialité et de sécurité. Les données échangées peuvent être authentifiées et chiffrées
- il supporte différents flux d'échange et différents mécanismes de communication
- il supporte tout type de charge (payload) dans ses messages en utilisant le format MIME
- il peut s'interfacer à d'autres technologies domotiques via des passerelles : bus CAN (Control Area Network), courant porteur (norme domotique X10), ...
- il supporte la mobilité
- il réutilise l'infrastructure SIP classique

L'idée est donc d'utiliser SIP pour collecter des informations provenant de différents capteurs et de piloter aussi différents équipements (actionneurs).

➤ SIP vs SNMP

Le protocole SNMP (*Simple Network Management Protocol*) a été présenté dans le magazine Linux Magazine numéro 43 d'octobre 2002 ainsi que sa mise en oeuvre pour le contrôle par Internet de systèmes électroniques.

SNMP offre les mêmes possibilités d'échange que SIP dans un contexte domotique.

Il possède :

- Les messages SNMP PUT et GET pour des actions PUT et GET.
- Le message SNMP TRAP les événements de type TRAP.

Mais, SNMP par rapport à SIP possède des défauts majeurs :

- SNMP est un protocole complexe malgré son nom.
- Il est difficile à utiliser. L'extension d'un agent SNMP est compliquée.
- SNMP n'est pas un protocole très sécurisé surtout dans sa version 1.
- Un agent SNMP n'a pas une empreinte mémoire faible, ce qui est un handicap pour un système embarqué avec peu de mémoire.
- Il n'y a pas d'applications clientes populaires. SIP en a de nombreuses (*gaim* par exemple).

Le protocole SIP s'impose de lui-même dans un contexte domotique et de convergence électronique-informatique-réseaux.

e) Bibliothèque libre oSip



Figure 3 : Bibliothèques oSip et ExoSip choisies pour le projet

Il existe différentes piles SIP libres que l'on peut utiliser sachant qu'il convient de privilégier celles qui sont développées en langage C ou C++.

Les différents critères de choix sont : la portabilité, le respect des RFC, les couches de transport compatibles, la taille et la licence. La pile oSip est portable sur les systèmes Linux, VxWorks et Win32, elle respecte les principales RFC, elle est compatible avec les couches de transport TCP, UDP et TLS (Transport Layer Security : anciennement SSL), elle a une faible empreinte mémoire et elle possède une licence LGPL.

Le choix s'est donc porté sur la pile GNU oSIP et son extension eXosip développées par Aymeric Moizard. La pile oSIP étant écrite en langage C, elle est fortement portable et à faible empreinte mémoire. eXosip est une API basée sur oSIP pour faciliter l'écriture des applications SIP. L'API eXosip a bien sûr été privilégiée ici.

f) Présentation du projet

Notre projet consiste à recueillir des informations, issues de capteurs, à distance à l'aide d'une communication IP et des alarmes. Pour cela nous disposons d'un microprocesseur ARM contenant un noyau linux embarqué, auquel est implémenté un serveur, et qui est relié à un capteur de température "iButton" ainsi qu'à Internet. Une interface simulant la présence d'autres capteurs et de systèmes munis d'alarmes pourra être mise en place afin de concevoir des communications de type TRAP.

Ainsi, le client envoie une requête au serveur à partir d'un ordinateur quelconque relié également à Internet afin de modifier l'état d'un actionneur, d'obtenir des informations d'un capteur ou de s'inscrire à la réception d'alarmes. Le serveur interroge ou modifie l'élément concerné par la requête du client.

Enfin, on envisagera la possibilité que plusieurs clients puissent interroger le serveur en même temps.

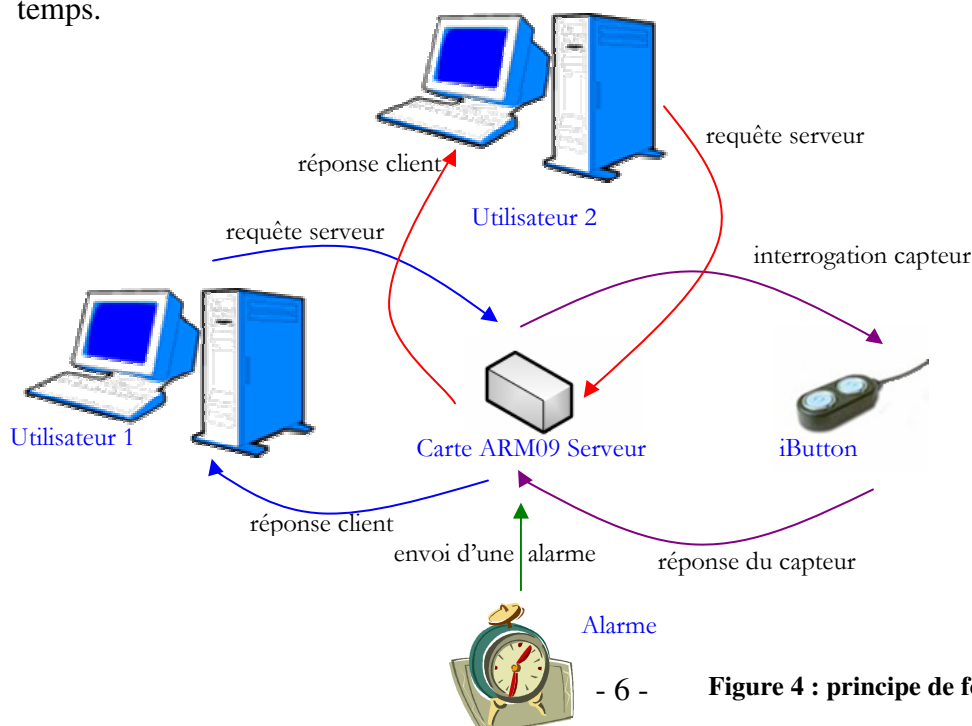


Figure 4 : principe de fonctionnement du projet

II. Protocole SIP

Le protocole SIP (Session Initiation Protocol) est un protocole de signalisation de bout en bout permettant d'établir une session entre deux équipements pour un échange de données (ou d'un flux) par Internet. Il est normalisé et standardisé par l'IETF (décrit par le RFC 3261 qui rend obsolète le RFC 2543) qui a été conçu pour établir, modifier et terminer des sessions multimédias. Le protocole SIP fait partie de la famille des protocoles Internet :

- RFC 3261 : *SIP: Session Initiation Protocol*.
- RFC 3265 : *Session Initiation Protocol (SIP)-Specific Event Notification*.
- RFC 3428 : *Session Initiation Protocol (SIP) Extension for Instant Messaging*.

Il se charge de l'authentification et de la localisation des multiples participants, mais également de la négociation sur les types de média utilisables par les différents participants en encapsulant des messages SDP (Session Description Protocol).

SIP se distingue radicalement par sa flexibilité et surtout par le fait qu'il n'est pas lié à un type de données à échanger et encore moins à un type de réseau de transport. Il ne transporte pas les données échangées durant la session comme la voix ou la vidéo. Le protocole étant indépendant de la transmission des données, tout type de données et de protocoles peut être utilisé pour cet échange. Cependant le protocole RTP (Real-time Transport Protocol) assure le plus souvent les sessions audio et vidéo. SIP remplace progressivement H323.

SIP est le standard ouvert de VoIP (Voice Over IP, voix sur IP) interopérable le plus étendu et vise à devenir LE standard des télécommunications multimédia (son, image, etc). Il n'est donc pas seulement destiné à la voix sur IP, mais à d'autres applications telles que la visiophonie, la messagerie instantanée, etc...

SIP permet l'interaction entre des éléments grâce aux messages de signalisation. Ces derniers peuvent être utilisés pour différentes fonctions :

- Enregistrer un utilisateur avec le système.
- Inviter un utilisateur à établir une session interactive.
- Négocier les limites et les conditions de session.
- Etablir un canal de média entre deux bouts ou plus.
- Terminer la session.

a) Couche réseau

Les messages SIP peuvent être transportés par différents protocoles. Le cas le plus usuel est le protocole de transport UDP. Mais on trouve également le protocole TCP, le protocole TLS (*Transport Layer Security*, RFC 3546) utilisant SSL (*Secure Socket Layer*) sur TCP ou encore le protocole SCTP (*Stream Control Transport Protocol*, RFC 2960).

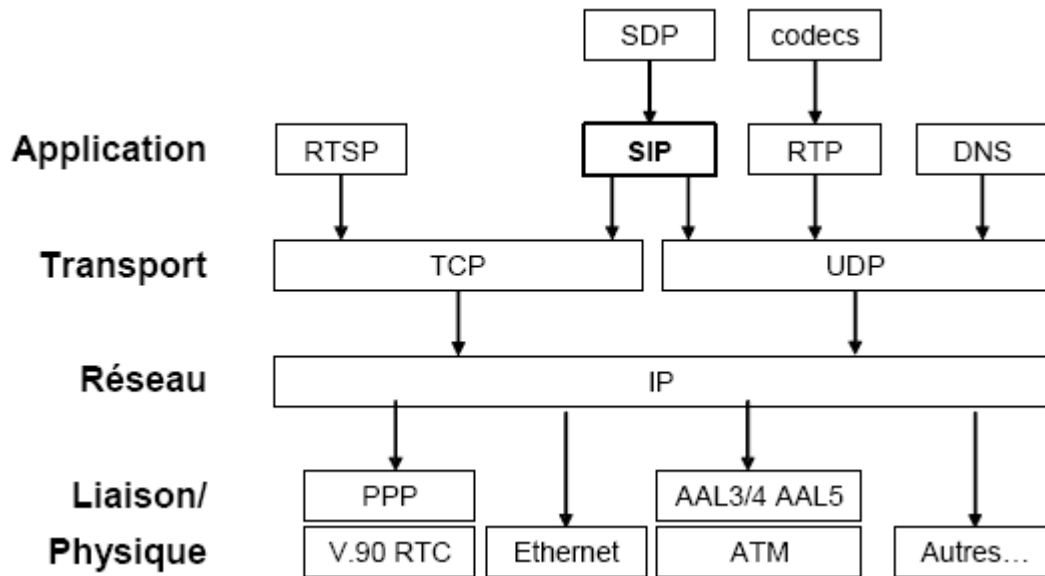


Figure 5 : Positionnement du protocole SIP parmi les autres protocoles Internet

SIP reprend des éléments techniques des protocoles SMTP (*Simple Mail Transport Protocol*) et HTTP (*Hyper Text Transport Protocol*) ; ce qui veut dire qu'il est basé sur le concept d'application client/serveur. On a une application d'extrémité cliente qui désire se mettre en relation avec une application d'extrémité serveur. SIP possède un numéro de port réservé : le 5060.

Autre point, SIP est un protocole de type commande/réponse. Ces commandes/réponses sont structurées sous forme de chaînes de caractères ASCII directement lisibles par l'humain et ressemblent étrangement à celles de HTTP (les codes d'état pour les réponses par exemple).

Enfin SIP possède un adressage pour identifier les applications SIP : c'est l'URI (*Uniform Resource Identifier*) comme on a l'URL (*Uniform Resource Locator*) pour HTTP.

Un exemple d'URI est : sip:nom@enseirb.fr.

b) Messages SIP

Les messages SIP sont codés en utilisant la syntaxe de message HTTP/1.1 (RFC2068). Contrairement au protocole HTTP, qui est basé sur TCP, SIP devra utiliser UDP pour les applications multimédia. Pour transporter plusieurs transactions à la fois, SIP peut utiliser une simple connexion TCP(mode flux) ou des datagrammes UDP(mode bloc). Seulement, les datagrammes UDP, tout en-têtes compris, ne doivent pas excéder une certaine longueur (M.T.U. pour Maximum Transmission Unit). Si la MTU est inconnue, elle est de 1500 octets par défaut. Cette taille permet l'encapsulation des datagrammes UDP ou segments TCP dans des paquets IP sans fragmentation.

En notation informatique ABNF (*Augmented Backus Naur Form*), un message SIP est :

- soit une commande
- soit une réponse

```
SIP-message
    = Request /
    Response
```

```
Une commande SIP
est de la forme :
```

```
Request      =
Request-Line
```

La figure 6 présente la structure des messages.

Ligne de départ
En-têtes
Ligne vide (CRLF)
Corps du message

Figure 6 : Structure d'un message SIP

Avec :

- **Début de ligne** = ligne de requête ou ligne d'état
- **En-têtes** = en-tête général ou en-tête de requête ou en-tête de réponse ou en-tête d'entité
- **CRLF** = balise pour indiquer la fin du champ d'en-têtes et le début du Corps du message (qui est optionnel mais nécessite un en-tête d'entité)
- **En-tête général** = Accept ou Accept-Encoding ou Accept-Language ou CALL-ID ou Contact ou Cseq ou Date ou Encryption ou Expires ou From ou Record-Route ou Timestamp ou To ou Via
- **En-tête d'entité** = Content-Encoding ou Content-Length ou Content-Type
- **En-tête de requête** = Authorization ou Contact ou Hide ou Max-Forwards ou Organization ou Priority ou Proxy-Authorization ou Proxy-Require ou Route ou Require ou Response-Key ou Subject ou User-Agent
- **En-tête de réponse** = Allow ou Proxy-Authorization ou Retry-After ou Server ou Unsupported ou Warning ou WWW-Authenticate

La figure 7 présente la structure d'un message dans le cas d'une requête.

Ligne de requête
En-tête général ou de requête ou d'entité
Ligne vide (CRLF)
Corps du message

Figure 7 : Structure d'un message de requête

Avec

- **Ligne de requête** = Méthode **SP** Requête URI **SP** version SIP **CRLF**

Le caractère SP sert de SéParateur et le caractère CRLF de terminateur de ligne. La version actuelle de SIP est la SIP/2.0.

La figure 8 présente la structure d'un message dans le cas d'une réponse.

Ligne d'état
En-tête général ou de réponse ou d'entité
Ligne vide (CRLF)
Corps du message

Figure 8 : Structure d'un message de réponse

Avec

- **Ligne d'état** = version SIP **SP** Code d'état **SP** Reason-Phrase **CRLF**

Certains champs d'en-têtes sont toujours présents dans les requêtes et les réponses, et forment l'en-tête général :

- **Call-ID** : ce champ d'en-tête contient un identificateur globalement unique pour un appel.
- **Cseq** : c'est un identificateur qui sert à rapprocher.
- **From** : il identifie l'appelant et doit être présent dans toutes les requêtes et les réponses.
- **To** : il indique la destination et doit être présent dans toutes les requêtes. Il est simplement recopié dans les réponses.
- **Via** : il est utilisé pour enregistrer la route d'une requête, de manière à permettre aux serveurs SIP intermédiaires de faire suivre un chemin exactement inverse aux réponses.
- **Encryption** : champ d'en-tête qui spécifie que le corps du message et éventuellement certains en-têtes ont été chiffrés.
- **Content-Type** : champ d'en-tête qui décrit le type de média contenu dans le corps du message.
- **Content-Length** : nombre d'octets du corps du message.
- **Content-Bodie** : corps du message.

Les messages de requêtes sont envoyés d'un client SIP à un serveur SIP :

INVITE :	requête SIP pour établir une session entre 2 agents SIP
ACK :	acquiescement d'une requête INVITE
BYE :	fin d'une session entre 2 agents SIP.
CANCEL :	message pour annuler la requête INVITE en cours
INFO :	information de session en cours.
OPTION :	demande d'informations sur le serveur.
REGISTRER :	message d'enregistrement d'un agent SIP à un Registrar Server.
MESSAGE :	permettre l'envoi de messages instantanés
NOTIFY :	notification d'événements.
PRACK :	implémente le mécanisme spécial de sécurisation des réponses provisoires.
PREFER :	permet la redirection d'appels.
SUSCRIBE :	souscription d'événements.
UPDATE	

La figure 9 présente le format des requêtes SIP.

INVITE SP sip;john@domain.com
Via: Call-ID: From: To: Call-ID: Cseq:
Subject:
Content-Type: Content-Length:
Ligne vide
Données SDP

Figure 9 : Allure d'une requête SIP

Un serveur SIP répond à une requête SIP au moyen d'une ou plusieurs réponses. Celles-ci, dont les codes sont de la forme 2xx, 3xx, 4xx, 5xx et 6xx, sont des réponses finales et terminent la transaction courante. Celles dont la forme est 1xx sont des réponses provisoires et ne terminent pas la transaction courante.

1xx	–	Message d'information.
2xx	–	Message de succès.
3xx	–	Message de redirection.
4xx	–	Réponse d'erreur sur le client indiquant que le message n'a pas été délivré correctement.
5xx	–	Réponse d'erreur

La figure 10 présente le format des réponses SIP

SIP/2.0 302 Moved temporarily
From: To: Call-ID: Localization: Expires: Cseq:
Ligne vide
Données de réponse (SDP vide, SDP chiffré, etc)

Figure 10 : Allure d'une réponse SIP

Pour établir un simple appel entre deux agents, on utilise seulement 3 requêtes : **INVITE**, **ACK**, **BYE** auxquelles on obtient 3 réponses. Une réponse SIP est de la forme :

Response	=	Status-Line *(message-header) CRLF [message-body]
Status-Line	=	SIP-Version SP Status-Code SP Reason-Phrase CRLF

On retrouve ni plus ni moins la structure d'une réponse HTTP. La figure 11 présente un enchaînement typique pour établir une communication VoIP par SIP.

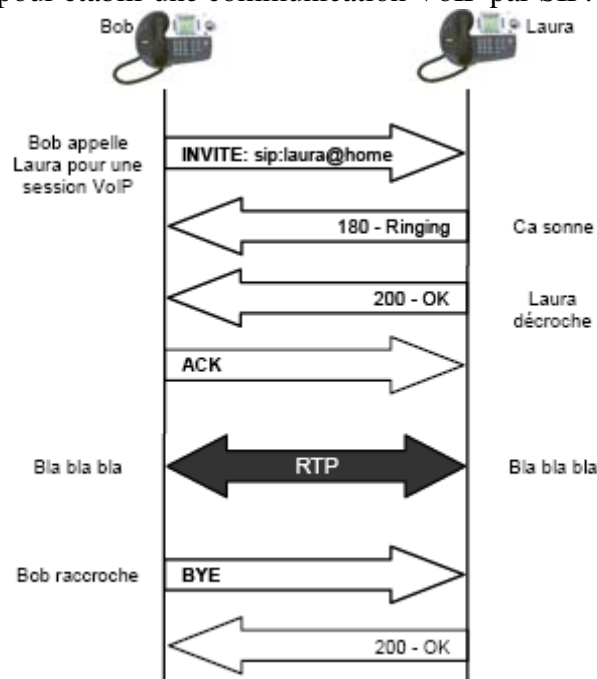


Figure 11 : Exemple d'échange lors d'une communication téléphonique VoIP

Plus précisément, voici un exemple de traces d'une requête SIP INVITE :

<p>• Requête SIP INVITE: INVITE sip:laura@home.com SIP/2.0 Via: SIP/2.0/UDP pc11.work.com Max-Forwards: 70 To: Laura <sip:laura@home.com> From: Bob <sip:bob@work.com>;tag=1928301774 Call-ID: a84b4c76e66710 CSeq: 314159 INVITE Contact: <sip:bob@pc11.work.com> Content-Type: application/sdp Content-Length: 131 v=0 o=Bob 289123451 289123451 IN IP4 111.22.33.44 s=Let us talk for a while c=IN IP4 111.22.33.44 t=0 0 m=audio 20002 RTP/AVP 0 a=rtpmap:0 PCMU/8000</p>	<p>• Réponse SIP: SIP/2.0 200 OK To: Laura <sip:laura@home.com>;tag=a6c85cf From: Bob <sip:bob@work.com>;tag=1928301774 Call-ID: a84b4c76e66710 CSeq: 314159 INVITE Contact: <sip:laura@222.33.44.55> Content-Type: application/sdp Content-Length: 131 v=0 o=Laura 289123444 289123444 IN IP4 222.33.44.55 s=Let us talk for a while c=IN IP4 222.33.44.55 t=0 0 m=audio 41002 RTP/AVP 0 a=rtpmap:0 PCMU/8000</p>
---	--

Le corps d'une commande/réponse respecte le format MIME (*Multipurpose Internet Mail Extensions*) dont l'en-tête MIME est sur l'exemple :

```
Content-Type: application/sdp
Content-Length: 131
```

et le corps MIME :

```
v=0
o=Laura 289123444 289123444 IN IP4 222.33.44.55
...
```

On respecte dans le cas présent, le format SDP (*Session Description Protocol*) qui sert à décrire la session établie :

- Nom de la session.
- Paramètres des flux audio, vidéo...
- Adresses IP et numéros de port à utiliser.
-

c) Relation entre les messages SUBSCRIBE et NOTIFY

La communication entre le client et le serveur est établie à l'aide des requêtes "SUBSCRIBE" et "NOTIFY".

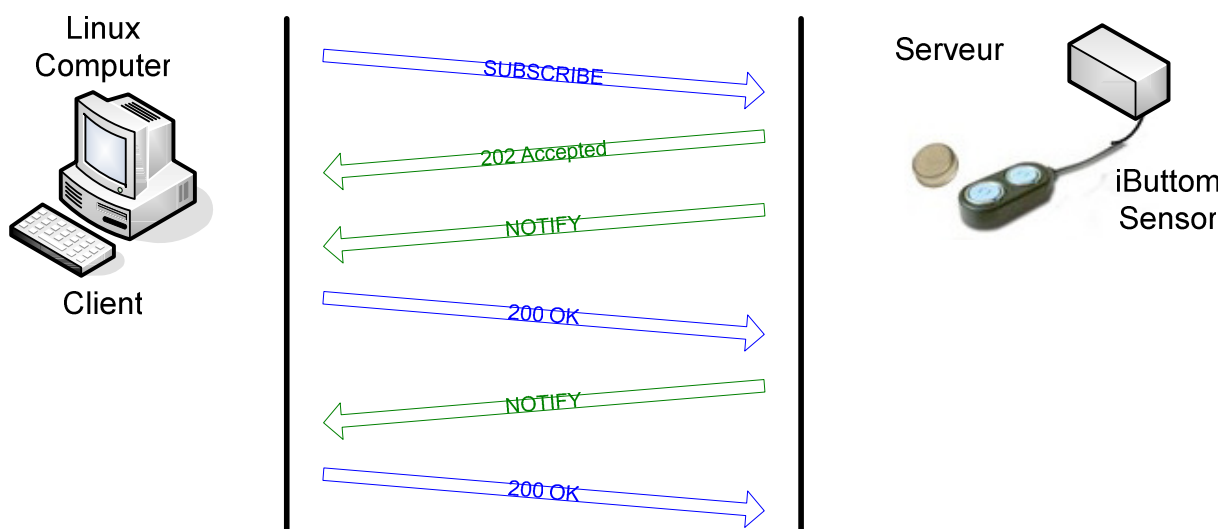


Figure 12 : Communication SUBSCRIBE / NOTIFY

d) Fonctionnement du message SUBSCRIBE

Le client interroge tout d'abord le serveur au moyen d'une requête SUBSCRIBE.

Si celle-ci est acceptée, le serveur en retourne le message "200 OK" et une communication est établie pendant une durée déterminée par le champ "EXPIRE" contenu dans le "Subscribe".

- initialisation

Au lancement du programme gérant le "Subscribe", il faut tout d'abord initialiser le contexte, c'est-à-dire la pile Osip, puis préciser le protocole utilisé pour le transport.

Nous utilisons ainsi le protocole UDP (car il n'y a pas de version de eXosip utilisant TCP) avec le port 5000, qui est réservé au protocole SIP.

- envoi de la requête Subscribe

Une fois les conditions de communication établies, il faut construire la requête en indiquant l'adresse IP du microprocesseur, celle de l'ordinateur client, le type de "Notify" que l'on veut (car il en existe 3 différents en fait et nous avons choisis un "Notify" de type "présence"), puis la durée de la communication. Pour cela, nous utilisons une fonction de la librairie qui va placer ces différents éléments dans une structure adaptée. Ensuite, on place le message à envoyer dans le corps de la requête, qui est "La température est de x", puis on précise le type de message envoyé, qui dans notre cas est "text/plain". Il ne reste alors plus qu'à envoyer le tout au serveur.

La procédure d'envoi de requête SIP est assez simple avec la bibliothèque eXosip2.

Les différentes étapes sont les suivantes :

- ✓ On construit le message qu'on veut envoyer.
- ✓ On verrouille les ressources d'eXosip2 car on ne peut envoyer qu'un seul message à la fois (cas de plusieurs User Agent en concurrence).
- ✓ On envoie le message.
- ✓ On déverrouille les ressources.

➤ réception de la réponse

Une fois la requête envoyée, on lance une boucle infinie qui va durer tout le temps de la communication, puis on attend une réponse que l'on traite suivant le cas.

En principe, le serveur envoie en premier lieu un "200 OK", puis ensuite le "Notify" comportant la température dans le "corps" de la requête (il s'agit du "case" : *EXOSIP_SUBSCRIPTION_NOTIFY*).

On affiche alors le résultat à l'écran.

e) Fonctionnement du message NOTIFY

Une requête de notification est émise par un système auquel on demande des informations à travers une souscription.

Ainsi, le serveur va envoyer des requêtes de notifications comportant les informations à une certaine fréquence pendant la durée annoncée par le client.

➤ initialisation

Tout comme pour le "Subscribe", il faut initialiser le contexte en premier lieu.

On utilise donc également le protocole UDP avec le port 5000.

➤ lancement de l'écoute

Une fois le contexte de fonctionnement initialisé, on peut "lancer" l'écoute. Celle-ci se caractérise par une boucle infinie avec l'attente d'un événement, qui dans notre cas est une requête "Subscribe".

Cette dernière peut être :

- ✓ soit une nouvelle requête de "Subscribe"
- ✓ soit une demande de mise à jour d'un "Subscribe"

➤ réception d'une requête

Une fois le corps du programme conçu, il faut déterminer les actions à effectuer lorsqu'on reçoit une des 2 requêtes vues ci-dessus, en complétant les 2 "case" du "switch" présent dans le code précédent.

Si on reçoit une requête de "Subscribe", on doit tout d'abord informer l'émetteur de la bonne réception de sa requête en lui envoyant une requête SIP de type "200 OK". Celle-ci joue un rôle d'accusé de réception (ACK) dans un dialogue SIP.

Le principe d'envoi est le même que pour la fonction "Subscribe".

➤ envoi de la requête Notify

La première étape consiste à récupérer la durée de la demande de souscription, en lisant le champ associé dans la requête "Subscribe" reçu.

Ensuite le serveur envoi des requêtes "Notify" à l'émetteur à une fréquence d'une toutes les 5 secondes pendant toute la durée du "Subscribe". La procédure utilisé pour envoyer les requêtes est la même que celle présenté précédemment pour le "Subscribe" même si les fonctions utilisé ne sont pas les mêmes.

III. Plateforme ENSEIRB

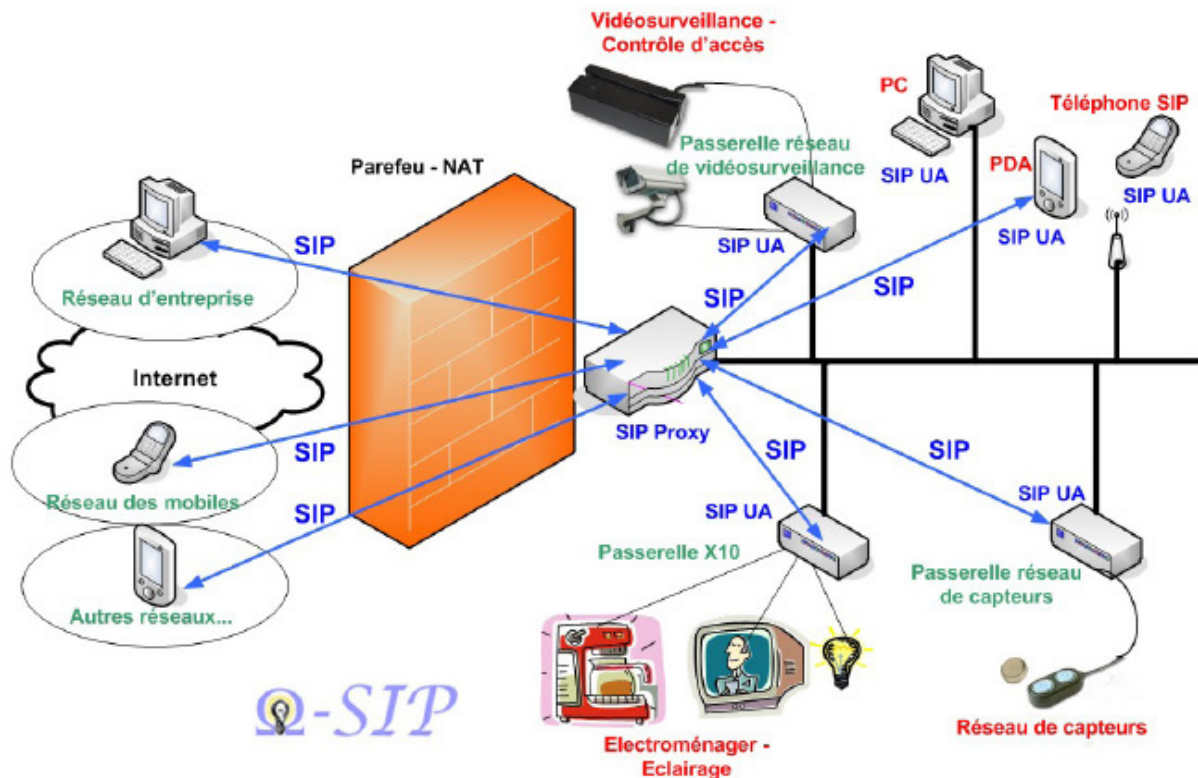


Figure 13 : Plateforme matérielle HomeSIP

La plateforme matérielle est basée sur l'architecture SIP de l'ENSEIRB. Un routeur central assure la redistribution sur le réseau Ethernet de l'école suivant le protocole SIP. Le réseau Ethernet de l'ENSEIRB sert d'ossature à la plateforme HomeSIP. Toutes les communications utilisent ce protocole. L'avantage du SIP est qu'il permet de franchir aisément le firewall assurant la sécurité du système informatique. Par l'intermédiaire de différentes passerelles il est ainsi possible d'avoir accès à certains éléments basés sur X10, technologie très répandue en domotique par la communication d'appareils par courant porteur. Ceci est matérialisé par le pilotage d'une cafetière ou encore de la commande de l'éclairage. Il est aussi possible d'accéder à différents capteurs de température ou encore contrôler la vidéo surveillance.

Ainsi la globalité et puissance de cette architecture, permet d'évaluer concrètement l'intérêt du protocole SIP dans le domaine de la domotique.

Ce dernier en possède des qualités intrinsèques:

- Il supporte un mode d'adressage abstrait.
- Il apporte un niveau de confidentialité et de sécurité. Les données échangées peuvent être authentifiées et chiffrées.
- Il supporte différents flux d'échange et différents mécanismes de communication.
- Il supporte tout type de charge (*payload*) dans ses messages en utilisant le format MIME.
- Il peut s'interfacer à d'autres technologies domotiques via des passerelles : bus CAN (*Control Area Network*), courant porteur (norme domotique X10)...
- Il supporte la mobilité.
- Il réutilise l'infrastructure SIP classique

a) Environnement de travail

Pour développer ce système électronique et l'application logicielle nécessaire à la mise en œuvre de ce système nous avons mis en œuvre différents outils afin de mieux matérialiser cette architecture.

La plateforme de développement est très proche de la l'architecture du système. Elle est constituée de deux PC fonctionnant sous linux (distribution FEDORA) relié par un HUB à une carte cible à base d'un processeur ARM9 développée par la société EUREKA. Le rôle de cette carte sera tout d'abord d'interfacer le capteur de température et le réseau Ethernet sur la base du protocole SIP. Ensuite, elle permettra de router toutes les connections. Une table d'IP est stockée dans la mémoire de la carte. Ainsi, il est possible de faire un lien entre les adresses IP et le nom des machines locales (ici Host 1 & Host 2).

Enfin la fonction principale en fait, et le traitement des requêtes et la réponse à ces dernières.

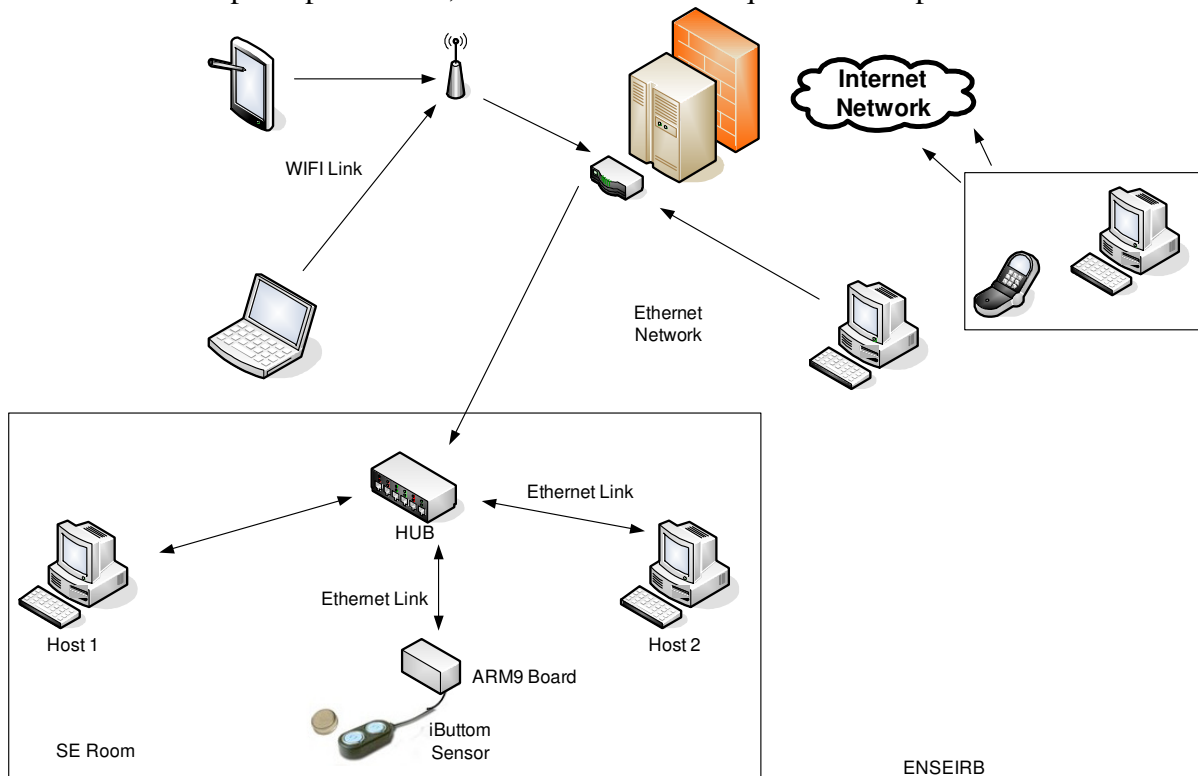


Figure 14 : Matérialisation de l'architecture Home Sip

b) Plateforme de développement

La compilation croisée permet de développer le logiciel et de la mettre au point sous un environnement graphique plus convivial qu'une simple console. Ainsi nous disposons de tous les outils de débogage. Le répertoire de travail principal est situé dans la mémoire ROM de la carte ARM et ainsi en utilisant le système de fichiers NFS, le PC hôte principal a un accès en lecture/écriture aux différents fichiers de ce répertoire.

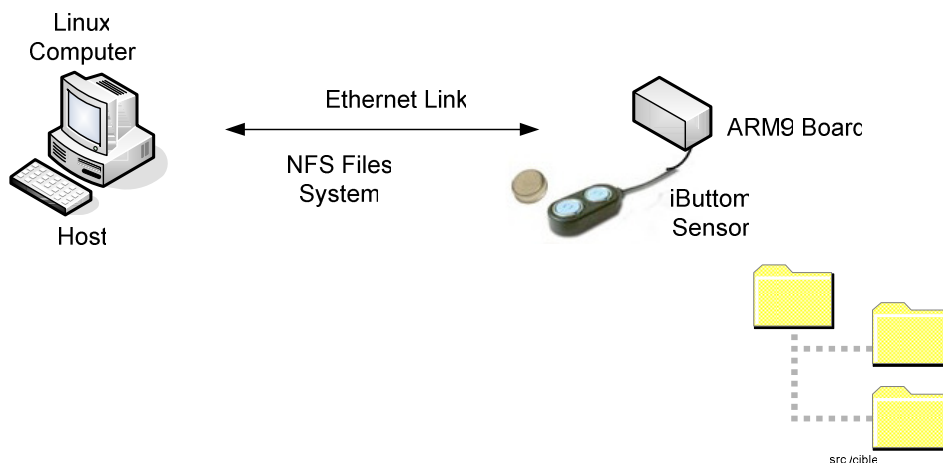


Figure 15 : Environnement et compilation croisée

La compilation des différents fichiers pour l'architecture ARM est réalisée à l'aide du compilateur "*arm-linux-gcc*".

Nous avons développé une petite interface permettant la commande des différents capteurs et actionneurs. En effet, afin d'être le plus exhaustif possible et surtout que le système soit proche de l'architecture générale, nous avons créé différents capteurs virtuels. L'interface permet de piloter différents organes tels que les capteurs de température, les volets de la maison (virtuelle), l'arrosage ou encore la climatisation. Un détecteur d'intrusion permet de déclencher une alarme.

Le développement est effectué en premier lieu sur le PC afin de tester les programmes. En phase finale, l'emprunte mémoire est optimisée et les versions finales sont chargées sur la carte.

Il existe deux façons de charger le compilateur dans la cible :

- soit par ligne série
- soit par tftp (*trivial file transfert protocol*)

Nous avons opté pour la 2^{ème} solution car elle est plus rapide.

c) Carte ARM09 et I-Button

La passerelle SIP/réseau de capteurs est composée de matériels du commerce pour ne pas avoir des développements hardware à réaliser. On retrouve une carte ARM9 d'Eureka. Son grand avantage est qu'elle supporte bien sûr Linux mais qu'elle possède de nombreuses E/S pour pouvoir connecter différents capteurs et actionneurs : liaisons série, bus I2C, E/S, bus USB...

La carte possède aussi une interface réseau pour pouvoir remplir son rôle de passerelle.

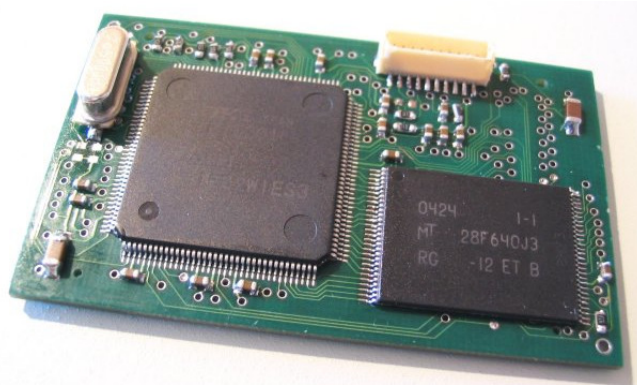


Figure 16 : carte fille sur la plateforme de développement : Microcontrôleur ARM09

En ce qui concerne les capteurs de température, le capteur *iButton*® DS1920 de Dallas Semiconductor possède toutes les caractéristiques nécessaires pour la matérialisation de notre mini-architecture avec une interface pour le connecter à une liaison série. L'interface de pilotage d'un *iButton*® étant standard, il est possible d'en rajouter d'autres.



Figure 17 : Capteur iButton DS1920

IV. Implémentation de la gestion des événements synchrones

a) Envoi de requêtes

Les requêtes entre l'ordinateur de l'utilisateur (client) et la carte ARM (serveur) se font grâce à l'envoi de « MESSAGE » défini avec les bibliothèques *osip* et *eXosip*.

La création d'un message se fait en plusieurs étapes :

- Déclaration d'une structure de type *osip_message_t* qui va être renseignée par les différents paramètres et envoyée au serveur.

```
osip_message_t *message;
```

- Renseignement de la variable à propos de la méthode (« MESSAGE », « PING »,...) du destinataire, de l'expéditeur et optionnellement de l'entête du chemin. La fonction utilisée vient de la bibliothèque *eXosip*, elle retourne '0' si il n'y a pas d'erreurs.

```
eXosip_message_build_request (&message, "MESSAGE", adresse_cible,  
adresse_host, NULL);
```

- Renseignement de la structure avec les paramètres tels que l'expiration, les données à envoyer ainsi que le type des données. On utilise ici trois fonctions de la bibliothèque *osip*.

```
osip_message_set_expires (message, "120");  
osip_message_set_body (message, buf, strlen (buf));  
osip_message_set_content_type (message, "text/plain");
```

- Envoi du message une fois qu'il est renseigné grâce à une fonction de la bibliothèque *eXosip*, mais avant il faut bloquer les ressources de la pile sip grâce à une fonction marchant comme un sémaphore. Une fois l'émission du message effectuée, on débloque les ressources grâce à une autre fonction. La fonction d'envoi du message renvoi '0' si il n'y a pas d'erreurs.

```
eXosip_lock ();  
eXosip_message_send_request (message);  
eXosip_unlock ();
```

b) Accusé de réception et récupération des informations

Si le serveur a bien reçu les requêtes alors il renverra un accusé de réception de type « 200 OK », ensuite il enverra un message vers l'ordinateur de l'utilisateur (client) contenant les informations que l'utilisateur avait demandées.

L'information est récupérée directement dans la réponse de type « 200 OK » renvoyée par le serveur. C'est un thread qui est chargé de surveiller les événements comme la réception d'une réponse à un message. Une boucle infinie permet de rester constamment à l'écoute d'évènements.

La réponse à un message est traitée de la façon suivante :

- Création d'une variable de type *osip_body_t* qui va permettre de récupérer l'information contenu dans le corps du message.

```
osip_body_t *body;
```

- Récupération du corps du message pour pouvoir l'afficher grâce à un « fprintf » vers une sortie standard qui est l'écran.

```
switch (event->type) {
    case EXOSIP_MESSAGE_ANSWERED:
        osip_message_get_body (event->response, 0, &body);
        if (body != NULL && body->body != NULL )
        {
            fprintf (stderr,"Valeur : %s\n", body->body);
            fprintf(stderr,">");
        }
        break;
```

Vous avez donc un petit aperçu de la méthode à employer pour établir une communication entre un ordinateur (client) et une carte ARM (serveur) avec le protocole sip pour des événements dit synchrones. Pour plus de détails se référer au code complet en annexe.

V. Implémentation de la gestion des événements asynchrones

a) Evènements asynchrones et domotique

Avant de choisir une implémentation pour la gestion des événements asynchrones il faut définir le concept dans le domaine de la domotique

Pour recevoir des événements asynchrones un client doit s'inscrire à un service en utilisant le couple subscribe/notify défini dans le protocole SIP. En domotique un événement asynchrone peu par exemple être une alarme (figure 18).

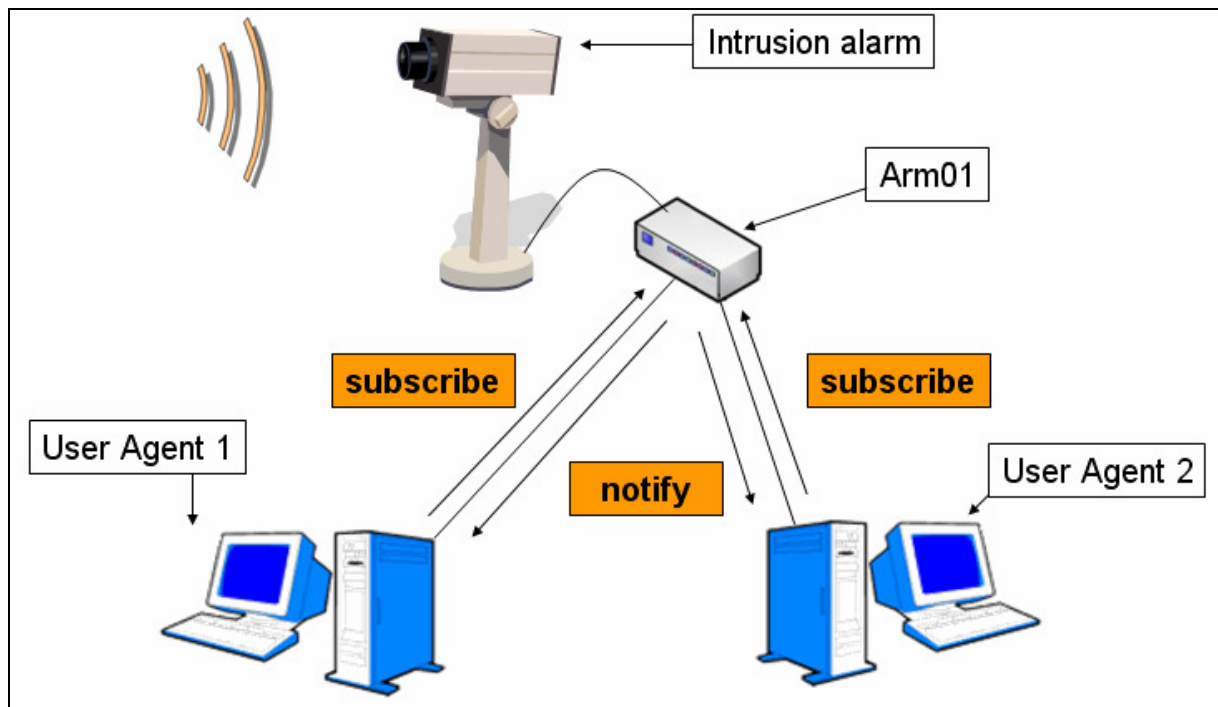


Figure 18 : inscription au service d'alarme

Dans ce cas, les utilisateurs ayant le désir d'être prévenus lors d'une intrusion doivent s'inscrire au service d'alarme qui avertira tous les utilisateurs inscrits lors d'une intrusion. L'inscription à un service est d'une durée limitée l'utilisateur devra mettre à jour son inscription s'il ne veut pas être désabonné du service.

Maintenant que le concept est défini, la question est de savoir comment implémenter cette fonction au niveau du client et au niveau du serveur.

b) Stockage des abonnés sur le serveur

Pour stocker les abonnés sur le serveur le choix d'une liste semble judicieux, en effet la liste permet de rajouter et supprimer dynamiquement des utilisateurs et de parcourir la liste lors de l'envoi du NOTIFY.

Chaque maillon de la liste représente un abonné et contient les champs suivant :

- Tid : identifiant de la transaction (utile pour 200 OK)
- Did : identifiant pour le dialogue SIP (permet d'envoyer un notify)
- Expire : permet de stocker la durée d'expiration
- Suivant : pointeur vers le maillon suivant
- Précédent : pointeur vers le maillon précédent



Figure 19 : détail des champs d'un maillon de la liste stockant les abonnés

Les abonnés sont rajoutés au début de la liste lors de leurs inscription (figure 20) et ils sont supprimés de la liste lorsque le champs expire est égal à 0.

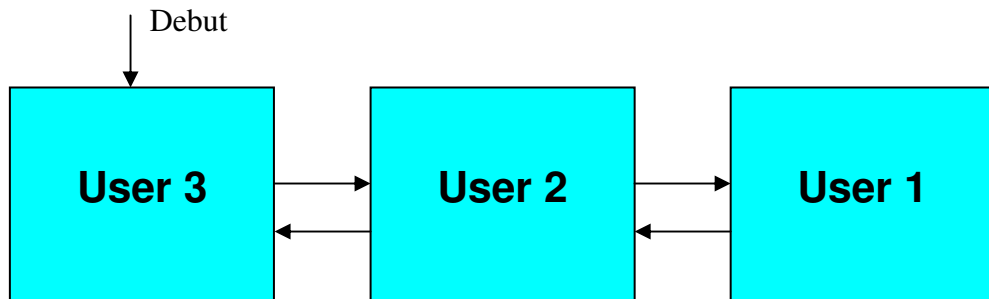


Figure 20 : Ajout d'un abonné dans la liste

c) Organisation des tâches serveur

Le serveur doit utiliser deux tâches (thread) distinctes, l'une pour traiter les demandes d'inscription et les stocker dans la liste et l'autre pour envoyer des notifications aux abonnés disponibles dans la liste.

La tâche SIP EVENT reçoit les inscriptions (SUBSCRIBE) et ajoute des éléments dans la liste.

La tâche Notify permet d'envoyer les notifications (NOTIFY) aux abonnés disponibles dans la liste.

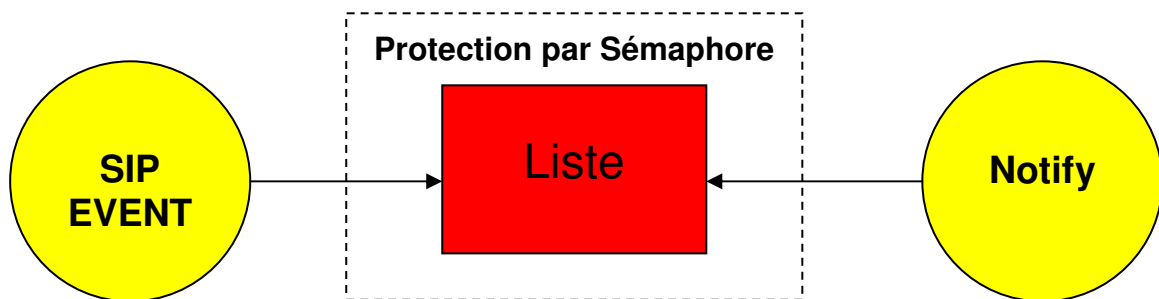


Figure 21 : Protection de la liste par un sémaphore

Il se peut que les deux tâches aient besoin d'accéder à la liste (ressource) en même temps donc il faut protéger l'accès à la liste par un sémaphore (figure 22).

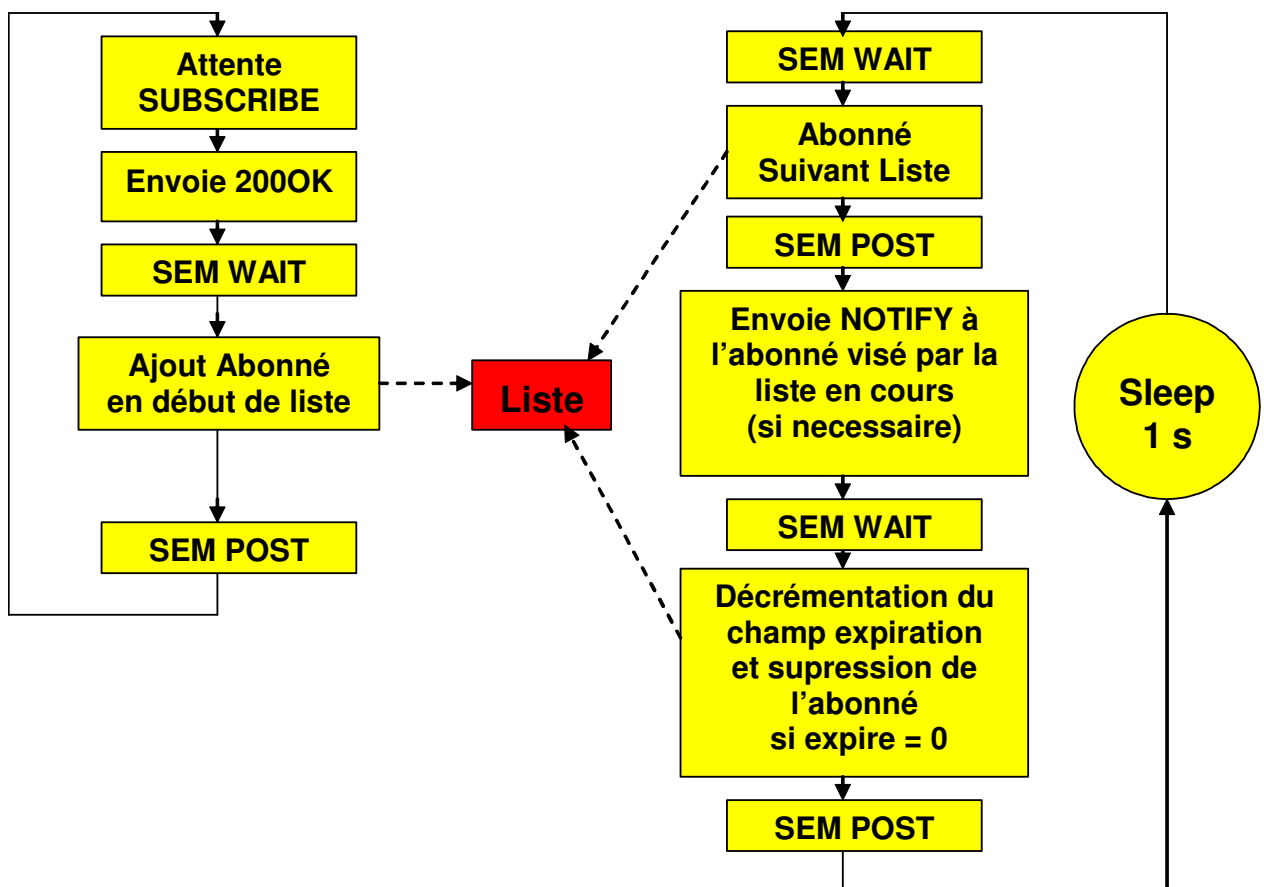


Figure 22 : comportement des tâches SIPEVENT et NOTIFY

d) Envoi de requêtes du client

Les requêtes entre l'ordinateur de l'utilisateur (client) et la carte ARM (serveur) se fait grâce à l'envoi de « SUBSCRIBE » défini avec les bibliothèques osip et eXosip.

Identiquement à la création de «MESSAGE», la création d'un SUBSCRIBE se fait en plusieurs étapes :

- Déclaration d'une structure de type *osip_message_t* qui va être renseignée par les différents paramètres et envoyée au serveur.

```
osip_message_t *message;
```

- Construction du Notify avant l'envoi au serveur. La fonction utilisée vient de la bibliothèque eXosip, elle retourne '0' si il n'y a pas d'erreurs.

```
eXosip_subscribe_build_initial_request (&message, "MESSAGE",  
adresse_cible, adresse_host, NULL);
```

- Renseignement de la structure avec les paramètres tels que l'expiration, message de la suscription ainsi que le type de données. On utilise ici trois fonctions de la bibliothèque osip.

```
osip_message_set_expires (message, "120");  
osip_message_set_body (message, buf, strlen (buf));  
osip_message_set_content_type (message, "text/plain");
```

- Envoi du subscribe une fois qu'il est renseigné grâce à une fonction de la bibliothèque eXosip mais avant il faut bloquer les ressources de la pile sip grâce à une fonction ayant le comportement d'un sémaphore. Une fois l'émission du message effectué, on débloque les ressources grâce à une autre fonction. La fonction d'envoi du message renvoi '0' si il n'y a pas d'erreurs.

```
eXosip_lock ();  
eXosip_eXosip_subscribe_send_initial_request(message);  
eXosip_unlock ();
```

VI. Phase de test

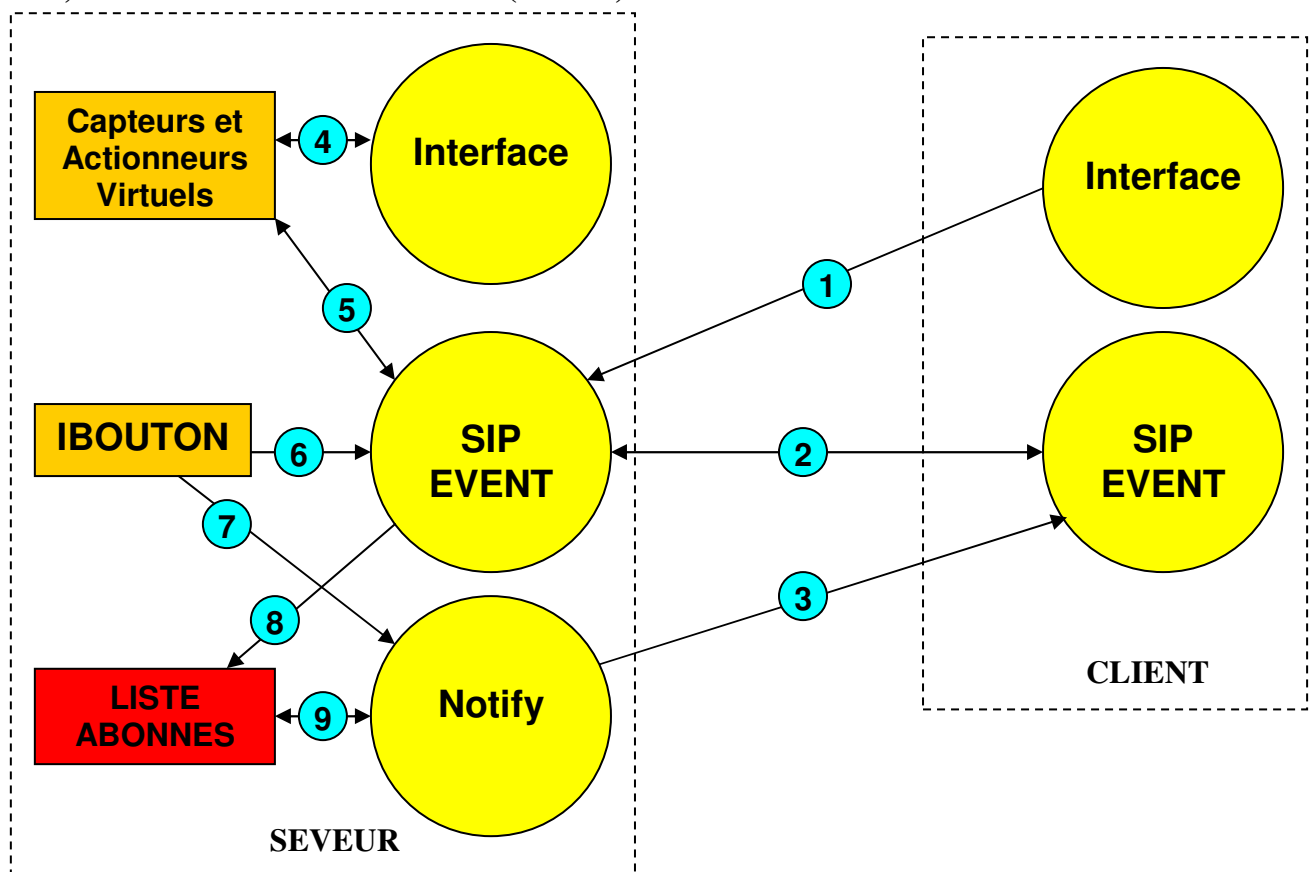
a) Programme test client

Ce programme permet de récupérer des informations de capteurs ou d'actionneurs ou de fixer la valeur d'un actionneurs. Ces capteurs et actionneurs (arrosage automatique, fermeture des volets automatiques,...) sont virtuels et sont situés sur la carte arm qui stocke leur valeur dans un tableau. Pour réaliser ces fonctions, le programme est constitué de deux tâches. La première tâche est une interface utilisateur permettant d'envoyer des messages ou des subscribes, la deuxième tâche permet de récupérer les réponses du serveur ou les notifications. Le capteur de température est toujours disponible et l'on peut s'inscrire à un service de température.

b) Programme test serveur

Le programme de test serveur permet de répondre aux messages GET ou SET et d'envoyer ou de changer la valeur des capteurs et actionneurs qu'elle détient. Une interface est aussi disponible pour que l'utilisateur puisse fixer la valeur des capteurs et actionneurs pour vérifier si la transaction client serveur fonctionne correctement.

c) Interaction entre les tâches (thread)



5. MESSAGE SIP ou SUSCRIBE SIP

6. 200 OK REPONSE SIP

7. NOTIFY SIP

8. Ecriture ou Lecture par Inteface

9. Ecriture ou Lecture par SIP EVENT

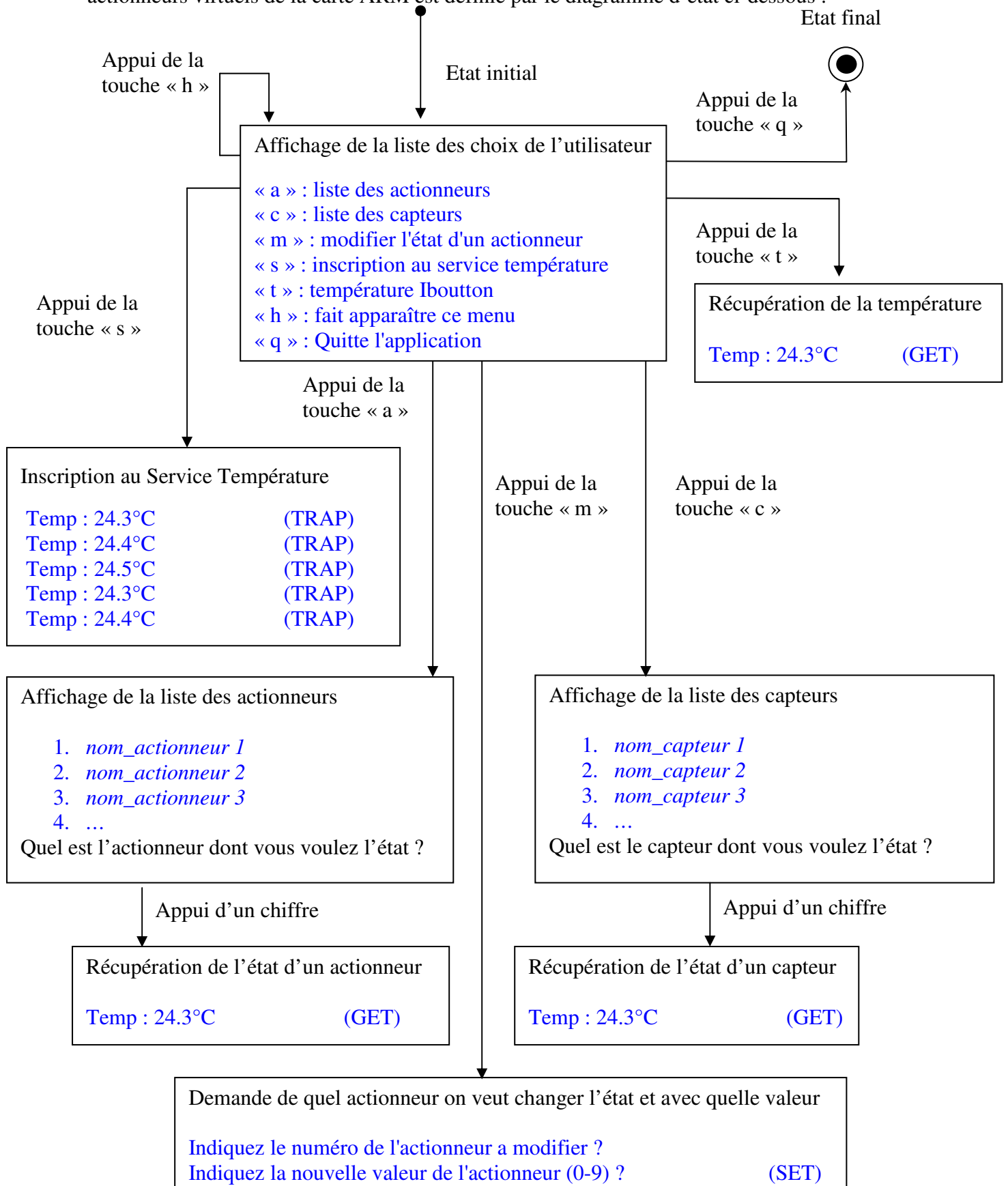
1. Lecture Température par SIP EVENT

2. Lecture Température par Notify

3. Ajout d'abonné par SIP EVENT

4. Lecture ou Suppression d'élément de la liste par Notify

L'interface du programme client qui permet de faire des requêtes SIP (MESSAGE ou SUBSCRIBE) pour récupérer la valeur de l'IBOUTON ou d'interagir avec les capteurs et actionneurs virtuels de la carte ARM est définie par le diagramme d'état ci-dessous :



Chaque action de l'utilisateur correspond à une demande d'informations ou d'activations d'actionneurs ou de capteurs.

Voici quelques exemples d'actionneurs :



Arrosage automatique



Volet automatique



Chauffage

Voici quelques exemples de capteurs :



Thermomètre électronique

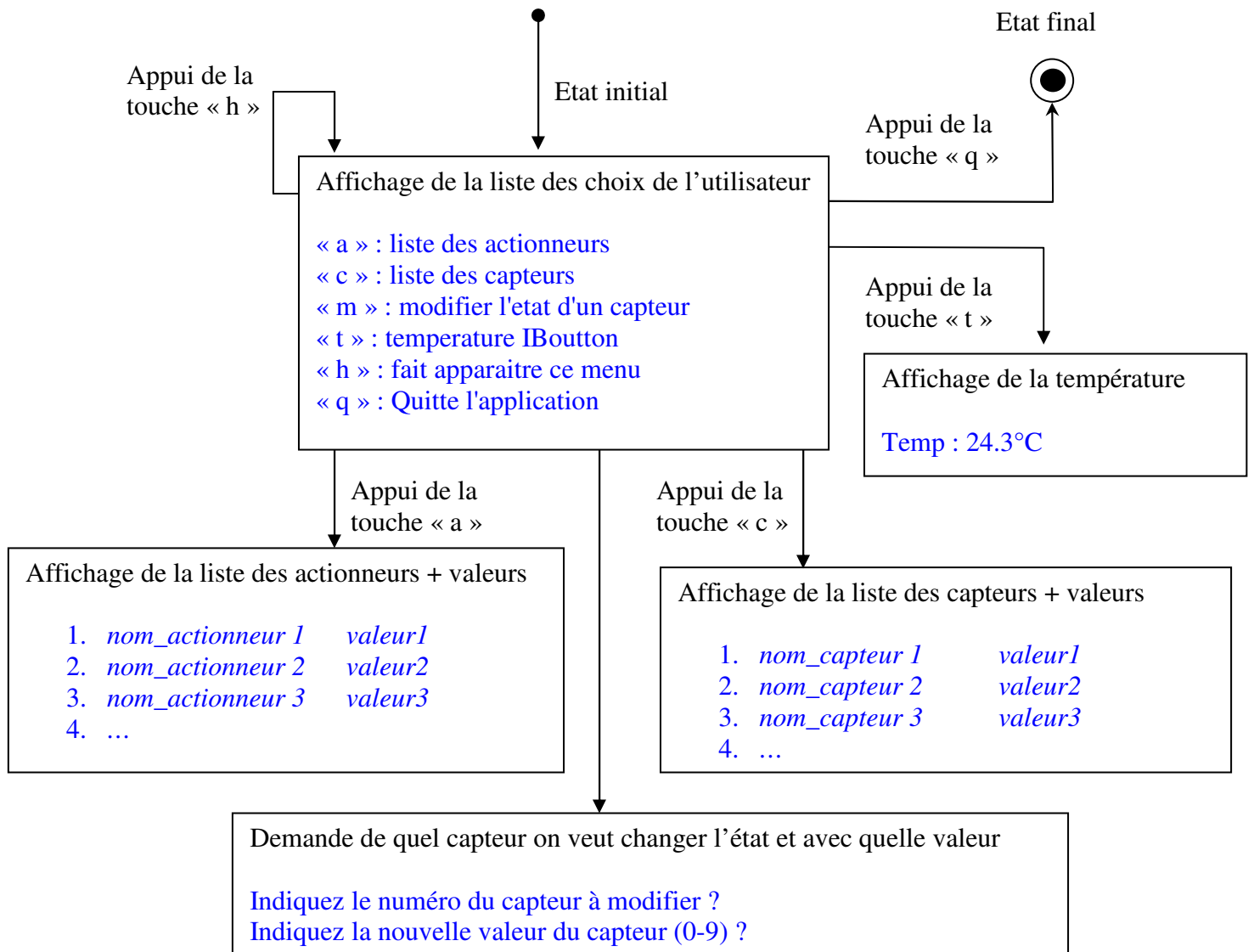


Pluviomètre électronique



Hygromètre électronique

L'interface du programme serveur permettant de modifier l'état des capteurs virtuels et de visualiser l'état des actionneurs virtuels est définie par le diagramme d'état ci-dessous :



d) Historique des tests

- Test d'envoi d'un message sip avec send_im.c
- Test des fonctionnalités de SUBSCRIBE / NOTIFY avec un PC et la cible
- Création d'un programme permettant de gérer plusieurs SUBSCRIBE simultanément
- Création d'une interface pour gérer des capteurs et des actionneurs virtuels sur la cible
- Intégration des fonctions GET et SET sur l'hôte avec une interface
- Problème d'accès par nom de machine résolu
- Programme final : 2 PC peuvent faire des GET et SET sur la cible et s'inscrire au service de température

e) Evolution du programme

Liste de bugs non corrigés :

- Lorsque l'hôte coupe sa connexion alors qu'il est abonné à un service, le programme sur la cible s'arrête car il n'arrive plus à joindre son correspondant.
- La programme hôte ne vérifie pas si l'utilisateur saisit des numéros de capteur ou d'actionneurs qui n'existent pas sur la cible
- La mise à jour des subscribes n'est pas gérée correctement lorsqu'un abonné qui n'a pas atteint son temps d'expiration demande une nouvelle inscription au lieu de mettre à jour le champ expire dans la liste on ajoute un nouveau maillon donc l'utilisateur reçoit momentanément deux fois plus de notify car il figure deux fois dans la liste.

VII. Conclusion

La version finale du programme met en évidence l'utilisation du protocole SIP dans le contexte de la domotique. En effet, il permet la gestion à distance de plusieurs capteurs et actionneurs grâce aux fonctionnalités GET & SET ainsi que l'abonnement au service de température. Bien que la documentation des bibliothèques Osip et Exosip soit très réduite, leur utilisation reste relativement simple et facilite le développement.

VIII. Références

Documentation Bibliothèques Osip et Exosip :

- <http://www.gnu.org/software/osip/>
- <http://www.antisip.com/documentation/eXosip2/>

Site du groupe de travail HomeSIP :

- <http://www.enseirb.fr/cosynux/HomeSIP>