THÈSE

Présentée pour obtenir le grade de

Docteur de l'Ecole Nationale Supérieure des Télécommunications de Paris

Spécialité informatique et réseaux

Rony CHAHINE

Services multi-fournisseurs et trans-réseaux : mécanismes pour un plan contrôle global

Application à un nouveau paradigme de signalisation et aux médiateurs de signalisation

Soutenue le 29 Novembre 2006 devant le Jury composé de :

Tatiana Kovacikova	Professeur à université de Zilina	Rapporteur
Francine Krief	Professeur à l'ENSEIRB Bordeaux	Rapporteur
Isabelle Demeure	Professeur à l'université de Versailles	Examinateur
Ahmed Bouabdallah	Docteur à l'ENST Bretagne	Examinateur
Bruno Chatras	Expert senior à France Télécom R&D Vice-Président du comité ETSI TISPAN	Examinateur
Claude Rigault	Directeur d'étude à l'ENST Paris	Directeur de thèse

A mes parents,

Remerciements

Je remercie L'ENST Paris et Corebridge qui sont à l'origine de cette thèse, et toutes les personnes qui ont rendu cette thèse possible par leurs aides et leurs contributions.

Mes premiers remerciements s'adressent à Claude Rigault, mon directeur de thèse infatigable et porte-conseil qui m'a encadré avec enthousiasme, auprès de qui j'ai acquis beaucoup tant au niveau scientifique qu'en matière de pédagogie. Pour cela, je lui suis très reconnaissant.

Je suis profondément reconnaissant aux Professeurs Tatiana Kovacikova et Francine Krief d'avoir accepté d'être rapporteurs de thèse. Je tiens à remercier également Professeur Isabelle Demeure, Docteur Ahmed Bouabdallah et M. Bruno Chatras pour avoir accepté d'être membres du jury.

Je remercie M. Emad Saïd de m'avoir accueilli au sein de son équipe technique chez Corebridge durant les années de ma thèse. Je remercie également mes collègues chez Corebridge pour les moments inoubliables qu'on a passés ensemble. Mes remerciements particuliers s'adressent à mon collègue Hong Son Do pour avoir participé au développement du médiateur de signalisation.

Je remercie également mes collègues du département Informatique et Réseaux pour la bonne ambiance régnant au département.

Finalement, je garde une place toute particulière pour ma famille et mes parents à qui je dédie cette thèse. Je les remercie pour leur soutien de tous les instants et les encouragements qui m'ont été très précieux durant les années de ma thèse.

Contents

List of Figures

Publications

Rony Chahine, Claude Rigault, “The Generic Context Sharing Protocol GCSP. Application to signalling in a cross-network and multi-provider environment”, MWCN 2006, Santiago, Chile, August 2006.

Rony Chahine, Claude Rigault, “Cooperative computing in the control plane. Application to NGN services and control”, MAN 2005, Ho Chi Min city, Vietnam, April 2005.

Rony Chahine, Claude Rigault, “Mécanismes coopératifs de plan contrôle global pour des services de communications multi-fournisseurs et trans-réseaux”, DNAC 2004, Paris, France, November 2004.

Rony Chahine, Claude Rigault, “New signalling mechanisms for multi-provider and cross-network services”, Softcom 2004, Split, Croatia, October 2004.

RÉSUMÉ DE LA THÈSE

1.1 Introduction

Le plan contrôle est un sujet rarement adressé. Ceci revient à l'importance mise sur la communication de données par début de la recherché sur les réseaux qui traitait la plupart du temps les "réseaux informatiques". Aujourd'hui cependant, la "convergence" est devenue un sujet fondamental et la recherche sur les réseaux s'est étendue au sujet plus général de la "communication de multimédia".

La communication multimédia exige l'implémentation d'un "plan contrôle" qui devient alors un sujet de recherche complet dans le domaine des réseaux. Les concepts du plan contrôle, dans un cadre restreint, étaient bien connus par les ingénieurs de télécommunications dans le cas particulier de transmission de voix. Cependant ces concepts doivent être prolongés et généralisés pour adresser correctement le nouveau domaine de communication de multimédia. La généralisation des concepts de plan contrôle et des mécanismes de communication est le but de cette thèse.

Dans la première partie, notre but est donc de reconsidérer les idées traditionnelles du plan contrôle afin de les placer dans un nouveau cadre théorique plus général se conformant aux conditions de la nouvelle communication multimédia généralisée et des services. Cette restructuration des concepts du plan contrôle n'a jamais été essayée auparavant dans l'approche systématique que nous employons dans notre travail.

Un avantage important du nouveau cadre théorique que nous donnons aux concepts du plan contrôle est l'identification des problèmes de recherches associées aux activités du plan contrôle et à la signalisation.

La deuxième partie de notre travail est donc consacrée à l'identification de points importants associés à la signalisation et qui restent à être adressé par les efforts de la recherche. Plusieurs de ces points dérivent de la nature spéciale du logiciel du plan contrôle, une nature spéciale que nous identifions en tant qu' "informatique coopérative". L'identification des catégories des problèmes de recherches directement liés à la nature de l'informatique coopérative des activités du plan contrôle met en relief l'importance fondamentale et la nature de la signalisation ainsi que les divers inconvénients des protocoles de signalisation et des systèmes actuels. Un de ces inconvénients est l'impossibilité de réaliser des services multi-fournisseurs et trans-réseaux qui est une limitation grave à la communication multimédia et aux services.

Dans une troisième partie de notre travail nous adressons donc spécifiquement le problème de la signalisation pour proposer un changement radical de paradigme de la manière dont la signalisation est considérée jusqu'ici. Cette nouvelle vision de la signalisation est prévue pour casser les barrières et les limitations actuelles des méthodes de signalisation et en particulier briserait la difficulté de réalisation de services de multi-fournisseurs et trans-réseaux.

Dans les parties suivantes de notre travail nous appliquons ces nouvelles idées à la signalisation pour proposer un nouveau protocole de signalisation appelé GCSP (Generic Context Sharing Protocol) et nous décrivons en détails l'organisation du contexte générique aussi bien que le protocole de signalisation lui même. Ce travail est effectué comme une application au cadre théorique général que nous avons identifié montrant ainsi à cette occasion l'efficacité et l'opérabilité de ce cadre théorique.

Finalement, comme nous n'avons pas voulu que nos idées théoriques restent sans application pratique, nous avons mis en oeuvre notre nouveau protocole de signalisation GCSP dans une application industrielle dans le contexte de services de couplage téléphonie informatique (CTI) basés sur SIP et de systèmes de téléphonie sur IP. La dernière partie de notre travail est ainsi consacrée à la description des réalisations industrielles de nos idées.

1.2 Clarification des concepts du plan contrôle : structuration des fonctionnalités du plan contrôle

Dans le nouveau cadre théorique que nous allons présenter, nous utilisons la notion de paradigme de communication. Un paradigme de communication est une façon d'organiser une communication. A l'heure actuelle 5 paradigmes de communication sont connus dont 2 synchrones et 3 asynchrones:

- En communication asynchrone, il n'y a pas blocage de l'émetteur en attente d'une réponse. Il y a 3 paradigmes asynchrones :
  - Message Passing
  - Message Queuing
  - Publication-Souscription
- En communication synchrone l'émetteur est bloqué en attendant une réponse du récepteur. On distingue les paradigmes :
  - Requête-réponse
  - Conversationnel

Notre thèse porte sur les paradigmes synchrones et en particulier sur le paradigme conversationnel.

1.2.1 Le paradigme conversationnel

Dans le paradigme conversationnel un environnement de communication est établi avant tout dialogue. Une fois établi, l'environnement reste ouvert tant qu'il n'y a pas de demande de fermeture explicite. Donc cet environnement est mémorisé durant toute la session de communication. Et dans ce sens il est persistent.

Au contraire dans le paradigme requête-réponse la session ne dure que le temps de la requête et de sa réponse. Rien n'est mémorisé pour des requêtes ultérieures.

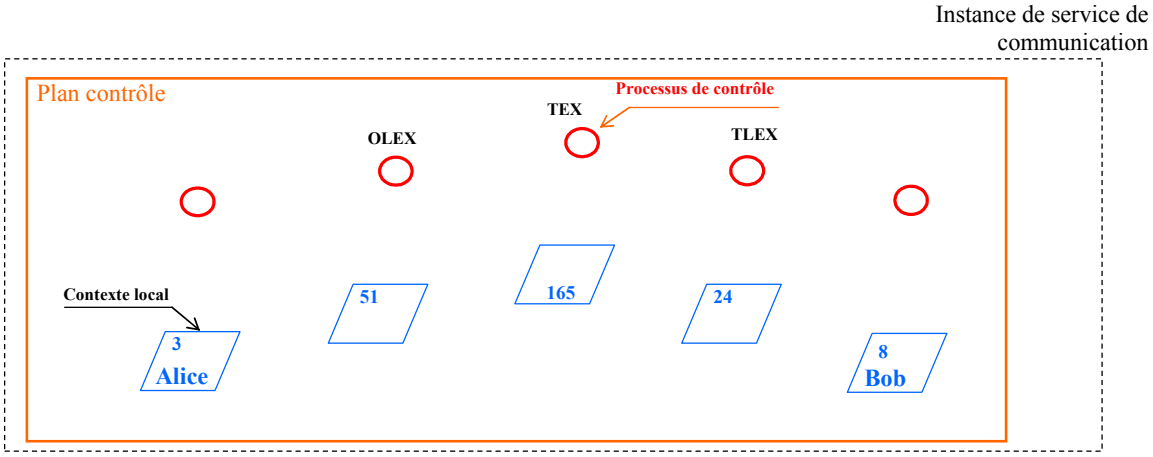
Qu'est ce que nous entendons par un environnement de communication ? D'abord c'est de la mémoire : chacun des participants de l'instance de communication ouvre une page mémoire. Nous appelons cette page mémoire un « contexte local ». En plus de

l'affectation de mémoire, il peut être nécessaire d'affecter des ressources comme des circuits ou des bandes passantes.

1.2.2 Le plan contrôle

Un service qui utilise le paradigme conversationnel doit donc utiliser une application particulière dont le rôle est de mettre en place cet environnement de communication. On appelle cette application particulière, « ***l'application de contrôle*** ». Le contrôle permet donc de mettre en place, de modifier éventuellement puis de relâcher un environnement de communication. Nous pouvons déduire que le contrôle est spécifique du paradigme conversationnel car on ne met pas en place d'environnement de communication dans les autres paradigmes.

[Figure 1] montre plusieurs partenaires impliqués dans une même instance d'un service qui utilise le paradigme conversationnel. Chez chaque partenaire il y a un processus qui exécute des fonctions de contrôle représenté par un rond rouge. Chacun des ces processus a ouvert pour cette instance un contexte local. L'ensemble des processus de contrôle forme le plan contrôle.



[Figure 1] Processus de contrôle et plan contrôle

1.2.3 Contexte global et contextes locaux

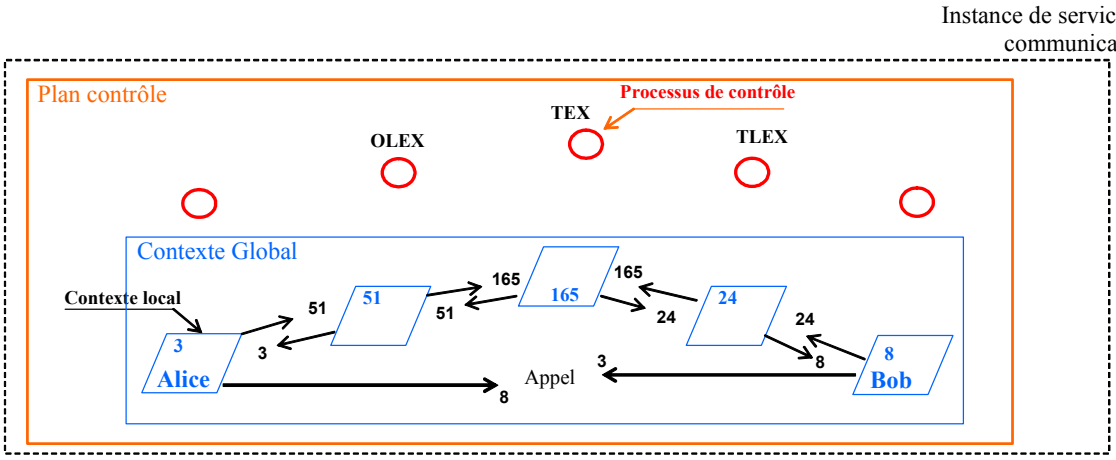
Nous expliquons maintenant un concept fondamental du plan contrôle qui est le concept d'Association. Les participants de l'instance de communication sont généralement multi-tâches. Pour qu'on processus puisse communiquer avec un autre il faut qu'il identifie la tâche à laquelle il s'adresse chez le partenaire. Comme il y a un contexte local par tâche, un processus qui communique doit donc indiquer la référence du contexte local de son

partenaire à laquelle il s'adresse. Nous appelons cette inter-référenciation l'Association. ***Une instance globale de communication est donc caractérisée par des références chez chacun des partenaires***.

La communication de la [Figure 2] est la communication 3 pour Alice, la communication 51 pour son central appelé OLEX (Originating Local EXchange), la communication 165 au central de transit, etc.

Maintenant tous les contextes locaux sont donc chaînés dans une structure de liste pour former un contexte global. Ceci ressemble à la gestion mémoire réalisée par un OS.

En conformité avec les normes du RNIS large bande et de l'IMT 2000 nous définissons qu'un Appel est l'association particulière des points d'extrémités. Dans l'exemple de la [Figure 2], c'est l'association entre Alice et Bob qui associe le contexte 3 d'Alice au contexte 8 de Bob. ***L'Appel est donc une notion de bout en bout***.



[Figure 2] Contexte global et contextes locaux

1.2.4 Domaines de contrôle

De plus il faut remarquer que dans le contrôle il y a 4 grands types d'activités [Figure 25], se sont :

- Les fonctions accès : responsable du login de l'utilisateur, de son profil et de sa traduction nom/adresse
- Les fonctions d'intelligence : qui permettent de substituer au traitement par défaut des commutateurs d'autres traitements

- Les fonctions d'appels : pour réaliser les associations de bout en bout. Il faut noter qu'aujourd'hui dans le réseau téléphonique l'appel est implicite il est nécessaire de le rendre explicite dans les réseaux multimédia pour négocier les services support. Nous rappelons que l'appel est une fonction de bout en bout. La signalisation SIP par exemple est une signalisation d'appel

- Les fonctions de connexion : qui mettent en place les services support. La signalisation ISUP par exemple est une signalisation de connexion. La connexion est une fonction de proche en proche

Ces 4 grands types d'activité déterminent 4 domaines fonctionnels du plan contrôle. Il y a des protocoles de signalisation différents pour chacun de ces domaines que nous appelons aussi domaines de signalisation.

1.2.5 Le modèle SIMPSON étendue

Il est possible de structurer ces 4 grands domaines fonctionnels du plan contrôle selon un modèle que nous appelons modèle SIMPSON étendu [Figure 30]**Error! Reference source not found.**. Dans ce modèle nous retrouvons les domaines d'accès, d'appel, de connexion et d'intelligence. On regroupe appel et connexion dans une rubrique commune que nous appelons transport pour obtenir ainsi 3 composantes horizontales :

- L'accès

- Le transport

- L'intelligence

Selon le modèle SIMPSON, chacune de ces composantes horizontales peut être décomposée verticalement :

- On trouve un niveau client tel qu'une interface graphique pour invoquer un service

- On trouve un niveau fournisseur où le service est définit par un chaînage de composants

- On trouve un niveau composant qui exécute les composants

- On trouve un niveau session pour les services d'appel

- On trouve aussi un niveau support pour les services de connexion
- Finalement on trouve un niveau media pour les interactions entre codeurs

1.3 Problèmes de recherches associées à la signalisation

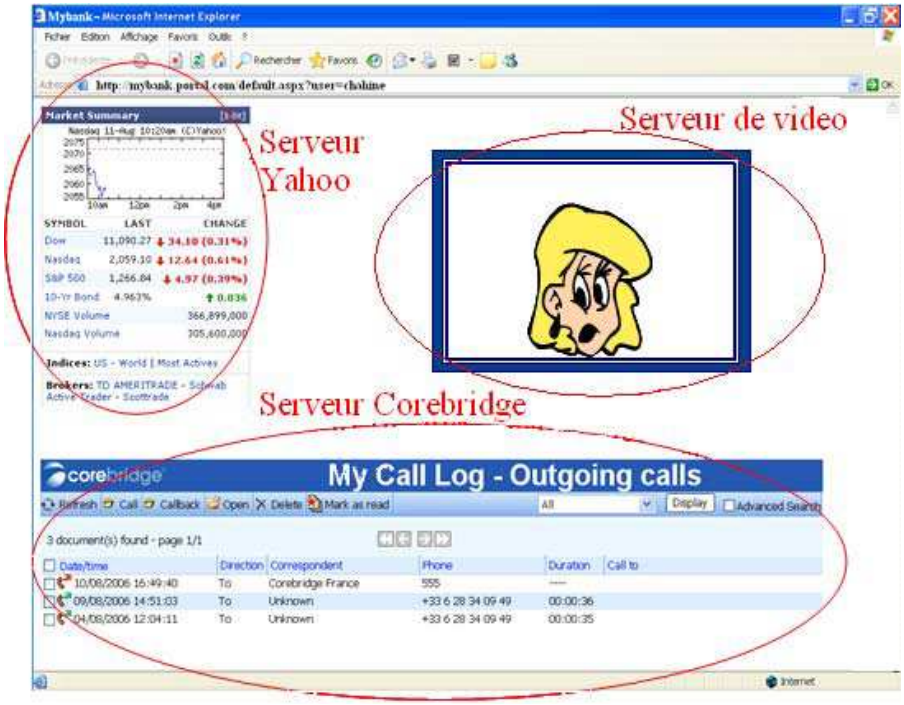
Après ces précisions sur le plan contrôle nous abordons maintenant le difficile problème des services en mode conversationnel dont les composants sont exécutés par des fournisseurs différents et qui sont localisés dans des réseaux différents. Nous appelons ces services des services multi-fournisseurs et trans-réseaux.

1.3.1 Services multi-fournisseurs

Pour illustrer les services multi-fournisseurs nous allons vais considérer l'exemple où Bob est un employé dans un centre d'appel d'une banque. Bob dispose d'un service de portail de banque qui est formé par 3 composants [Figure 3] :

- Un composant de marché
- Un composant de visiophonie
- Un composant de journaux d'appels

Ces 3 composants qui proviennent de fournisseurs différents sont intégrés dans une seule interface qui offre un service plus riche. De manière générale, un service multi-fournisseurs est définit par un graphe de composants exécutés par des fournisseurs différents.



[Figure 3] Service de portail de banque

1.3.2 Services trans-réseaux

Nous expliquons maintenant l'aspect trans-réseaux. Supposons maintenant que Bob peut aussi recevoir des appels téléphoniques des clients. Si le client Alice appelle Bob sur à son bureau, Bob dispose d'une remontée de fiche sur son PC en même temps que son téléphone sonne [Figure 32]. Cette fiche contient les informations de la banque sur Alice.

Supposons que Bob souhaite disposer du même service de remontée de fiche quand il se déplace. Dans ce cas quand Alice appelle Bob à son bureau, l'appel est transféré vers le mobile de Bob et la fiche est également transférée vers son PDA. Dans cet exemple, le service qui était disponible dans le bureau de Bob est étendu sur de nouveaux réseaux. Nous appelons un tel service, un service trans-réseaux.

Il y a eu des études des services trans-réseaux très limités telles que PINT et SPIRITS. Ces études ne sont pas généralisables. Il y a aussi les études du groupe Parlay OSA basées sur une architecture centralisée. Notre contribution propose une nouvelle approche beaucoup plus générale.

1.3.3 Médiation inter-protocoles difficile

L'une des difficultés des services trans-réseaux est le besoin de conversion entre protocoles de signalisation des réseaux auxquels les composants appartiennent. Si on a N types de réseaux le nombre de passerelles croit comme le carré de N. Ceci rend difficile et coûteux la réalisation des services trans-réseaux.

1.3.4 Médiation inter domaines très difficile

Même dans un seul réseau il peut y avoir des besoins de signalisation entre domaines. Actuellement chaque domaine possède ses propres protocoles de signalisation sans aucune unité. Si nous pouvions avoir une signalisation unique pour tous les domaines, le problème de la médiation inter-domaines serait considérablement simplifié.

1.3.5 Recherches actuelles dans le plan contrôle

L'unification des signalisations a été abordée par différentes propositions telles que NSIS ou Next Step In Signaling et IMS ou IP Multimedia Services. Pour le moment NSIS n'a pas encore défini de protocoles pour la signalisation applicative. Nous verrons que nos propositions pourraient être une contribution à la signalisation applicative de NSIS. l'IMS ne traite les services qu'en SIP et fait appel à des médiateurs SIP vers autres signalisations pour traiter le problèmes des signalisations trans-réseaux. Il nous semble nécessaire d'avoir une solution plus générale.

1.4 Proposition d'un paradigme de signalisation basé données

1.4.1 La nature spéciale du logiciel du plan contrôle

En plus de la difficulté d'accomplir les services multi-fournisseurs et trans-réseaux, s'ajoute la complexité intrinsèque du plan contrôle. Cette complexité du plan contrôle est due à la nature spéciale de son logiciel que nous allons maintenant décrire. L'étude de cette complexité va nous permettre de proposer de meilleurs mécanismes pour la signalisation.

1.4.1.1 Les étapes dans l'informatique

La première étape de l'informatique était l'informatique centralisée. Ensuite l'informatique distribuée a été introduite pour assurer le fonctionnement des minis ordinateurs connectés par un réseau. La solution retenue pour l'informatique distribuée est l'architecture client-serveur. Il faut noter que cette architecture est très proche de l'informatique centralisée du fait que le serveur reste le maître unique qui centralise l'intelligence du système. La prochaine étape dans l'informatique est l'informatique coopérative. Dans ce type d'informatique tous les partenaires sont égaux. Il existe des applications d'informatique coopérative. L'exemple le plus important pour nous est le traitement de connexion des centraux téléphoniques.

Pour le moment il n'y a pas de théorie généralement acceptée pour l'informatique coopérative. Par conséquent les applications d'informatique coopérative sont toujours aujourd'hui des solutions AD HOC. Nous allons voir que le plan contrôle est un problème d'informatique coopérative.

1.4.1.2 L'origine de la complexité excessive du plan contrôle

En effet le plan contrôle est un exemple d'application coopérative puisque :

- Tous les commutateurs sont égaux
- Il y a une hiérarchie de routage mais pas de hiérarchie de traitement

- Il n'y a pas de plate-forme centralisée qui dicte aux commutateurs ce qu'ils doivent faire pour établir une connexion

- Chaque commutateur coopère avec son commutateur voisin pour accomplir un service global

Puisqu'il n'existe pas de théorie générale pour l'informatique coopérative, le logiciel du plan contrôle a été conçu comme une solution AD HOC. C'est cela l'explication de la complexité excessive du logiciel du plan contrôle.

1.4.1.3 Les points fondamentaux de l'informatique coopérative

Nous avons identifié quelques points fondamentaux qui doivent être résolus afin de trouver une théorie pour l'informatique coopérative :

Problème 1 : Pour coopérer les partenaires ont besoin de partager leurs informations. Dans le plan contrôle le partage d'information est la signalisation. Pour cette raison, la signalisation est un sujet fondamental pour la théorie de l'informatique coopérative

Problème 2 : A un moment donné qui prend la décision du fait que tout le monde est égal. La solution du réseau téléphonique ne peut pas être considérée comme générale. Du fait de la nature lien par lien des connexions le réseau téléphonique utilise une politique round robin. D'abord Alice prend une décision, puis le commutateur originant, puis transit puis terminant et enfin Bob.

Problème 3 : Pour coopérer, il est nécessaire de connaître le modèle comportemental de son partenaire. C'est la raison des modèles d'appel dans l'IN et le CTI.

Problème 4 : Pour coopérer un partenaire a besoin de faire confiance à son partenaire. C'est le problème de la sécurité dans les réseaux.

Il y a sûrement d'autres points qu'il faudra résoudre pour arriver à une théorie générale de l'informatique coopérative. Mais déjà chacun de ces points est un sujet de recherche à part.

1.4.2 Changement de paradigme pour la signalisation

Nous allons maintenant montrer que les méthodes actuelles utilisées par la signalisation ne sont pas les meilleures et ne sont pas optimales pour les besoins du plan contrôle. Nous allons donc proposer un meilleur mécanisme pour la signalisation.

1.4.2.1 Trois mécanismes fondamentaux pour le partage d'informations

Nous redisons à nouveau que la signalisation c'est le partage d'informations entre les processus du plan contrôle. Pour partager les informations entre processus distants il y a 3 mécanismes fondamentaux :

- D'abord il y a le **mécanisme orienté variables**, en anglais data-based. dans ce mécanisme chaque processus de contrôle peut directement lire ou modifier les variables dans le contexte local de ses partenaires On dit que le contexte local est public. Ce mécanisme a plusieurs avantages :
    - Simple et générique
    - Peu de commandes : GET/SET/NOTIFY
    - Un seul SET suffit pour modifier plusieurs variables
    - Permet une communication prédictive car la lecture préalable du contexte du partenaire permet de connaître exactement son état
    - C'est plus facile de faire évoluer le protocole car il suffit de rajouter des variables dans le contexte au lieu d'ajouter une commande
    - Cela permet d'intégrer plus facilement des nouveaux composants hétérogènes pour réaliser de nouveaux services multi-fournisseurs et trans-réseaux

  L'inconvénient d'un tel mécanisme est sa faible abstraction du fait qu'un processus doit connaître la structure de données de son partenaire

- Ensuite il y a le **mécanisme orienté commandes**. Dans ce mécanisme les contexte locaux sont privés leurs variables sont modifiées indirectement par des commandes. L'avantage de ce mécanisme est le haut niveau d'abstraction du fait qu'un processus ne doit pas connaître la structure des variables de son partenaire.

  L'inconvénient de ce mécanisme est:
    - Que l'évolution du protocole nécessite l'ajout de nouvelles commandes on abouti ainsi a des signalisations qui évoluent en nids d'oiseaux. Au début il n'y a que quelques brindilles ou commandes, au bout de quelques années on a un énorme nid de commandes très nombreuses

- La communication basée commande n'est pas prédictive, l'état du partenaire n'est pas réellement connu. Il faut attendre des confirmations pour être sûr de l'effet réel de la commande
- L'évolution du protocole est coûteuse car toute modification nécessite l'évolution de toute la pile et éventuellement la modification de tous les partenaires
- Ceci rend donc plus difficile et coûteux la réalisation de services multi-fournisseurs et trans-réseaux

- Enfin il y a le **mécanisme orienté objets**. Il est semblable au mécanisme orienté commandes du fait que le variables locales sont privées. Au lieu de transmettre des commandes à distance, on invoque des fonctions distantes. Ce type de communication est basé sur le client-serveur, il est asymétrique et suppose une répartition non équilibrée des fonctions entre le client et le serveur et n'est donc pas adapté au paradigme conversationnel.

1.4.2.2 Les domaines d'applications

Nous avons vu que l'approche data-based bénéficie de plusieurs avantages par rapport au command-based. Il serait logique alors qu'elle soit la solution pour tous type d'informatique.

Or dans le cas de l'informatique centralisée la station centrale a beaucoup de partenaires. les gens trouvent que le mécanisme data based ne convient pas car la station centrale doit faire beaucoup de Get/Set dans chaque partenaire. Par exemple dans SNMP la NMS doit faire plusieurs Get/Set pour modifier plusieurs variables dans chacun des différents agents. Ceci demande une grosse puissance de calcul au niveau de la NMS et pose un problème de trafic sur le réseau. La solution serait d'envoyer une commande qui génère plusieurs Get/Set au niveau de l'agent. *Donc l'approche commandes est la solution pour le centralisé*.

Dans le cas de l'informatique coopérative on n'a ni le défaut de la congestion ni la défaut de la puissance de calcul car le nombre de partenaires est réduit. *Donc le data based est la solution pour l'informatique coopérative.*

1.4.2.3 Le paradoxe : le monde est à l'envers

Nous nous rendons donc compte d'un grand paradoxe. La gestion qui est une activité centralisée devrait utiliser des solutions command based. Or l'un des principaux protocoles de gestion, SNMP, est un protocole data based.

Au contraire la signalisation dans le plan contrôle, qui est une activité coopérative, devrait utiliser le mécanisme data based. Or au jour d'aujourd'hui tous les protocoles de signalisation sont command based.

Nous voyons que le monde marche à l'envers chacun utilise le mauvais mécanisme. Les gens de la gestion se sont aperçu du problème et ont commencé à proposer des solutions command based. Par contre les gens du contrôle n'ont pas encore identifié ce problème et continuent avec le mécanisme command based.

1.4.2.4 Un nouveau paradigme de signalisation

Devant ce paradoxe il nous apparaît clairement qu'il faut proposer un mécanisme data-based pour la signalisation. Nous proposons donc un mécanisme data-based pour la signalisation, où toute la signalisation peut être faite avec 3 commandes : Get/Set/Notify.

La signalisation devient alors l'écriture et la lecture d'une instance de données dans un contexte distant. Par exemple dans la [Figure 36], le téléphone d'Alice au lieu d'envoyer une commande Setup au central originant, il va positionner la variable « MakeCall » à 1 et la variable « abonné appelé » à 01 23 45 67 89 dans le contexte local du central.

1.5 Structuration du contexte générique

Un processus de contrôle qui communique avec un autre processus partenaire à l'aide d'un protocole « data based » a besoin de connaître la structure de données du contexte de son partenaire. Pour cela nous proposons une organisation générique des données du contexte local que nous appellerons Contexte Générique.

Le contexte générique est une structure de donnée que tous les processus de contrôle comprennent. C'est une mémoire partagée qui a une structure similaire à une MIB mais orientée objet comme dans CIM (Common Information Model). Pour prendre en compte les services multi-fournisseurs et trans-réseaux nous avons utilisé le modèle SIMPSON étendu pour structurer le Contexte Générique.

En effet nous aonvs fait une structuration UML du CG basée sur le modèle SIMPSON étendu. On a un objet GenericContext général qui contient un ou plusieurs objets qui correspondent aux domaines du modèle SIMPSON étendu. Par exemple on trouve dans le schéma UML de la [Figure 39] l'ensemble des domaines du modèle SIMPSON. On retrouve ainsi les 3 composantes horizontales et les 6 composantes verticales du modèle SIMPSON étendu.

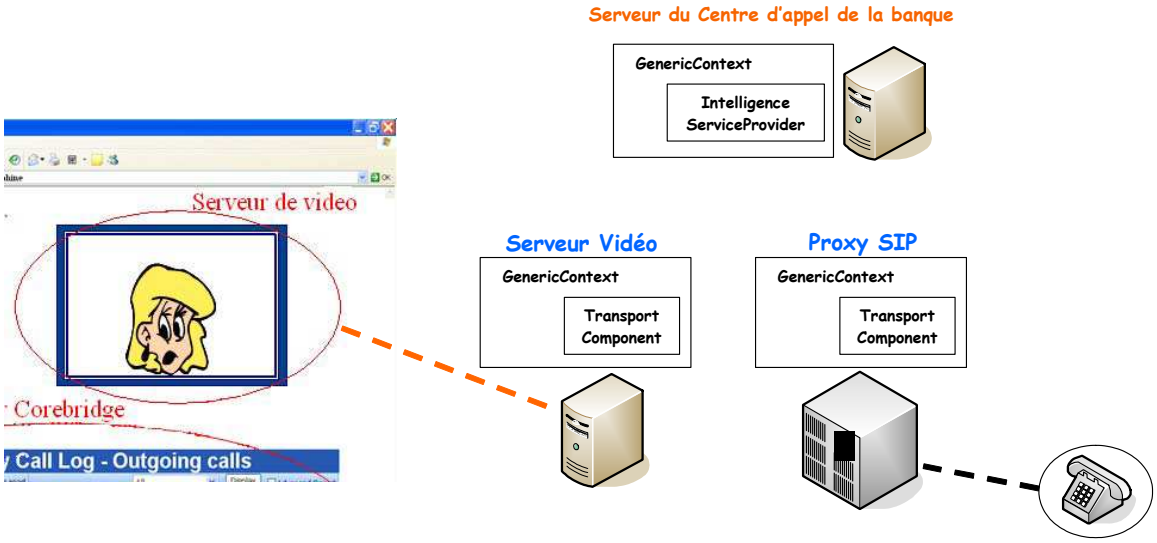
Toutes les classes qui correspondent aux différents domaines du modèle SIMPSON héritent de la classe Service. La classe Service donne une modélisation générique du service. On peut voir dans la [Figure 44] que cette classe comprend :

- L'Association : responsable de l'association de 2 contextes génériques
- La Facturation : pour pouvoir facturer le service à tous les niveaux.
- La Sécurité : assurent une identification, authentification et cryptage des données.
- Un modèle comportemental et un état courant.
- Des triggers associés au modèle comportemental qui permettent de déclencher des évènements quand ils sont armés.

Nous donnons maintenant un exemple d'instanciation du Contexte Générique [Figure 4]. Pour cela nous reprenons l'exemple de Bob qui travaille dans un centre d'appel d'une

banque. Bob dispose d'un service d'intelligence qui est le service de visiophony où la vidéo est affichée dans son portail de banque et la voix est effectuée par son téléphone SIP. Ce service est formé de 2 composants qui sont un composant vidéo fournis par le serveur vidéo et un composant voix fournis par le Proxy SIP. L'instanciation des contextes génériques pour ce service est la suivante :

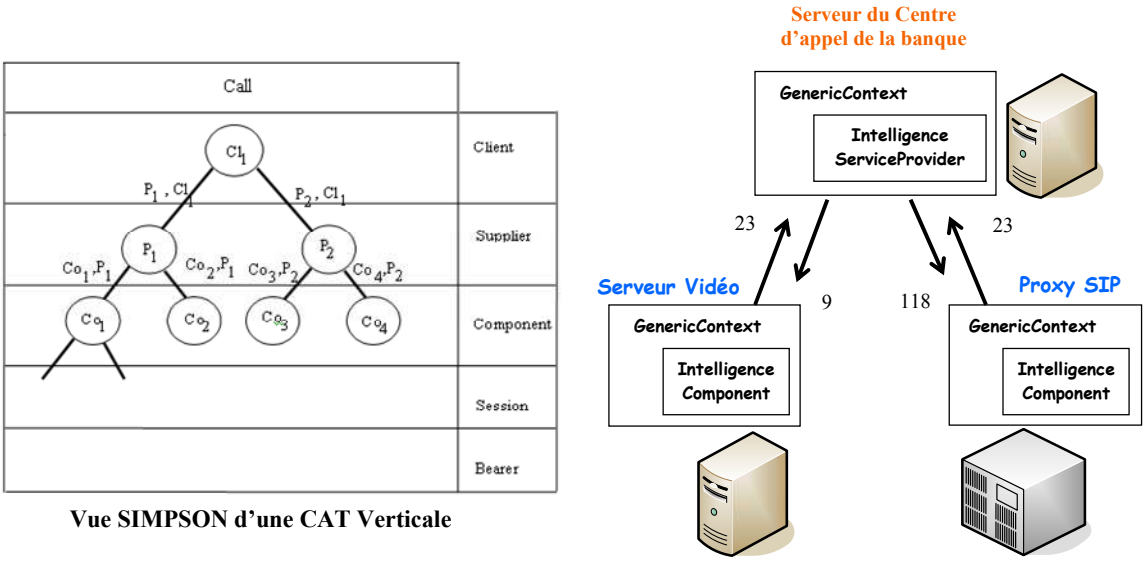
- Au niveau du serveur CTI nous avons implémenté l'objet Intelligence Service Provider.
- Au niveau du serveur vidéo nous avons implémenté l'objet Transport Component.
- Au niveau du Proxy SIP nous avons implémenté l'objet Transport Component.



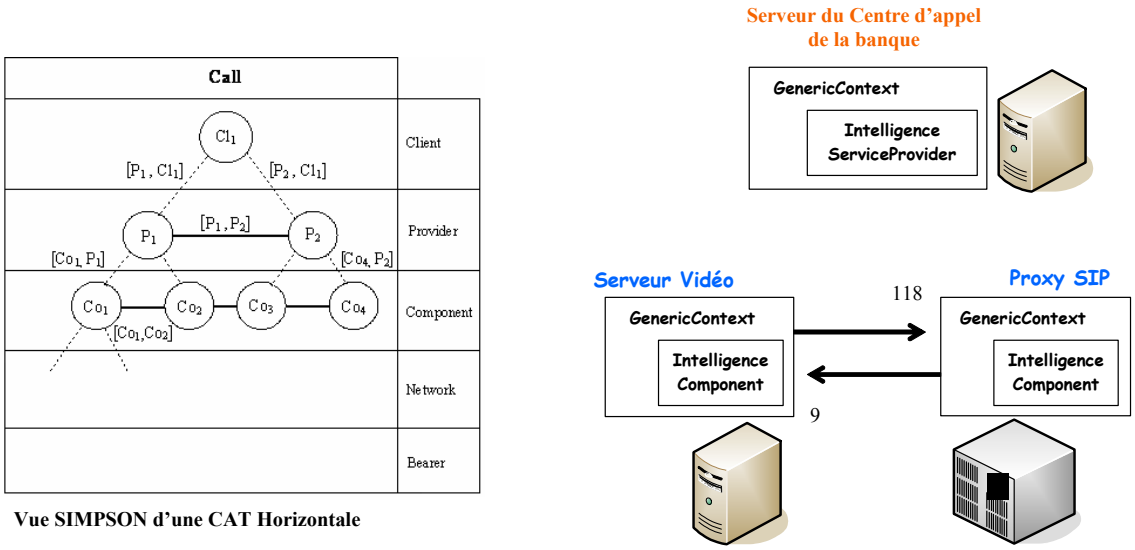
[Figure 4] Exemple d'instanciation du Contexte Générique

Nous expliquons maintenant la partie association du service générique. L'association des contextes locaux d'une même instance de service forment une liste chaînée. Nous appelons cette liste chaînée une Control Allocation Table. L'association de la CAT est semblable à l'association que fait une FAT avec les secteurs d'un disque. Dans notre exemple précédent de la [Figure 4], l'association est faite entre le serveur CTI et le proxy comme indiqué dans la [Figure 5]. Nous remarquons que cette association est hiérarchique. Les associations sont entre niveau du modèle SIMPSON. Nous appelons alors cette CAT une CAT verticale.

Pour pouvoir gagner en performance et mémoire on envisage une association directe entre le proxy SIP et le serveur de video [Figure 6]. Cette association horizontale forme la CAT Horizontale. L'association avec le niveau supérieur est alors relachée.



[Figure 5] Vue SIMPSON d'une CAT Verticale : exemple du serveur du centre d'appel



[Figure 6] Vue SIMPSON d'une CAT Horizontale : exemple du serveur du centre d'appel

1.6 Le protocole GCSP (Generic Context Sharing Protocol)

Après avoir définit le contexte générique nous proposons maintenant un nouveau protocole de signalisation que nous appelons Generic Context Sharing Protocol (GCSP). GCSP utilise le paradigme « data based ». Ce même protocole est valable pour tous les domaines de signalisation car se sont les variables modifiées qui caractérisent le domaine et non pas le protocole.

GCSP permet de lire ou de modifier des parties d'un contexte générique :

- Il est facile à manipuler car il utilise l'encodage texte comme HTTP et SIP.
- il a très peu de commandes qui sont génériques : ce sont les commandes Get/Set/Notify.
- Il respecte les contraintes de la nature coopérative du plan contrôle car il est data based.
- Il est efficace car utilise le concept de transactions de signalisation afin d'optimiser la bande passante et le temps de traitement.
- Il est flexible car au lieu d'ajouter de nouvelles commandes, il suffit d'ajouter des variables au contexte générique sans réécrire la pile protocolaire.
- Il est sécurisé.

Une trame GCSP est formée d'un header et d'un body comme indiqué dans la [Figure 48]. Le header contient des informations relatives à la requête, l'association des contextes, le n° de séquence et le body contient les instances de données du CG échangées.

Pour pouvoir communiquer avec le protocole GCSP, les processus de contrôle ont un identifiant unique qui permet de les distinguer parmi d'autres processus. Cet identifiant est l'URI GCSP qui est structuré de la façon suivante :

GCSP-URI = "gcsp:" [userinfo] <hostport>

userinfo = <user> [":" password] "@"

hostport = <host> [":" port]

Les requêtes GCSP sont les suivantes :

- Get: permet de lire une variable ou un ensemble de variables.
- Set : permet de modifier une ou plusieurs variables.
- Notify: permet de remonter des notifications.
- Open et Close Contexte : permettent respectivement d'ouvrir et de fermer explicitement un Contexte Générique.
- Describe : permet de décrire un Contexte Générique avec l'aide d'un fichier XML.
- Lookup : permet de rechercher un médiateur de signalisation GCSP/x.

Les requêtes Get et Set sont acquittés par des réponses alors que la notification n'est pas acquittée.

Afin d'améliorer les performances du protocole, GCSP implémente les transactions qui sont similaires aux transactions TCAP comme le montre la [Figure 7]. Une requête de transaction GCSP permet d'envoyer plusieurs requêtes GCSP en même temps. Cette requête de transaction contient un numéro qui permet à un processus de contrôle de retrouver le contexte de la transaction parmi d'autres. Relativement aux requêtes de transactions, nous avons les réponses de transactions. Une réponse de transaction encapsule plusieurs réponses GCSP.

Requête de transaction

Transaction_Req <Transaction_id>
<blank line>
GCSP request frame 1
<blank line>
GCSP request frame 2
<blank line>
GCSP request frame n

Réponse de transaction

Transaction_Resp <Transaction_id>
<blank line>
GCSP response frame 1
<blank line>
GCSP response frame 2
<blank line>
GCSP response frame n

[Figure 7] Transactions GCSP

Le corps du message GCSP contient la partie du contexte générique concerné par la requête ou la réponse. Pour pouvoir transmettre la partie du Contexte Générique sur le réseau, nous avons défini un nouveau langage de description similaire à SDP [Figure 51]. Il a l'avantage sur SDP qu'il permet de sérialiser des objets qui en encapsulent d'autres. En plus, on peut envoyer plus que 26 attributs dans le corps du message car on n'est pas limité au nombre des lettres. Nous avons fait des comparaisons de plusieurs exemples, [Figure 52] et [Figure 53], et il s'est avéré que notre sérialisation est 50% plus économique que XML.

1.7 Les médiateurs de signalisation : Exemple d'une application de CTI

Après avoir présenté toutes les contributions de nos travaux, nous expliquons dans ce paragraphe comment nous les avons validé par des applications pratiques. J'applique ces contributions à des services trans-réseaux entre le réseau téléphonique et le réseau de données de l'entreprise.

1.7.1 Architecture classique de TAPI

En entreprise le Couplage Téléphonie Informatique ou CTI permet de réaliser des services trans-réseaux impliquant le réseau téléphonique et le réseau informatique de l'entreprise.

La [Figure 8] décrit l'architecture du CTI qui comprend :

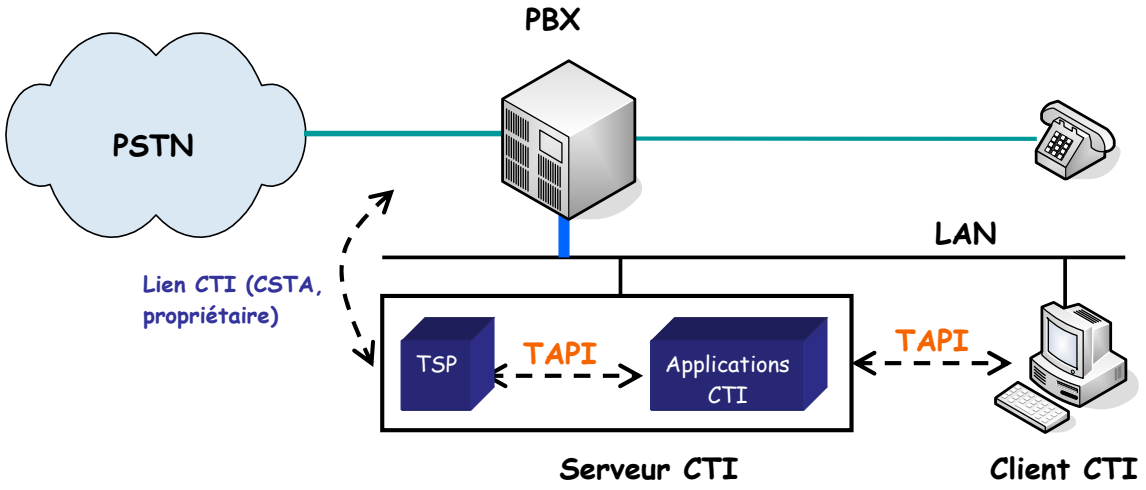
- Un PBX.
- Un serveur CTI qui centralise les applications de services.
- Un logiciel client sur le PC de l'utilisateur qu'on appelle phone bar qui permet d'invoquer les services.

Pour permettre à une application de dialoguer avec le PBX, le constructeur du PBX fourni un logiciel qu'on appelle TSP ou Telephony Service Provider. Ce logiciel peut être installé sur le serveur CTI ou sur une machine à part. Dans notre exemple le TSP offre des API TAPI aux applications du serveur CTI et d'un autre côté dialogue avec le PBX avec le protocole standard CSTA ou avec d'autres protocoles propriétaires. Ce TSP est relié au PBX grâce au lien CTI à travers le LAN de l'entreprise. Dans notre exemple aussi, la phone bar communique avec le serveur CTI grâce à l'API TAPI. Il existe pour les API d'autres solutions comme JTAPI mais TAPI est la solution utilisée par Corebridge.

Cette architecture a plusieurs inconvénients :

- Coût élevé du lien CTI : les prix des licences s'élèvent à 5000 euros pour 25 postes utilisateurs.

- Difficulté d'intégration de nouveaux types de PBX tels que des iPBX SIP.
- La difficulté d'accès au service quand le PC de l'utilisateur est à l'extérieur de l'entreprise.



[Figure 8] Architecture classique TAPI du CTI

1.7.2 La nouvelle architecture GCSP du CTI

Pour corriger les défauts de l'architecture CTI que nous venons de présenter, nous proposons une nouvelle architecture CTI basée sur GCSP [Figure 9]. Pour cela nous avons :

- Remplacé TAPI par GCSP.
- Remplacé le TSP par un médiateur de signalisation SIP/GCSP.

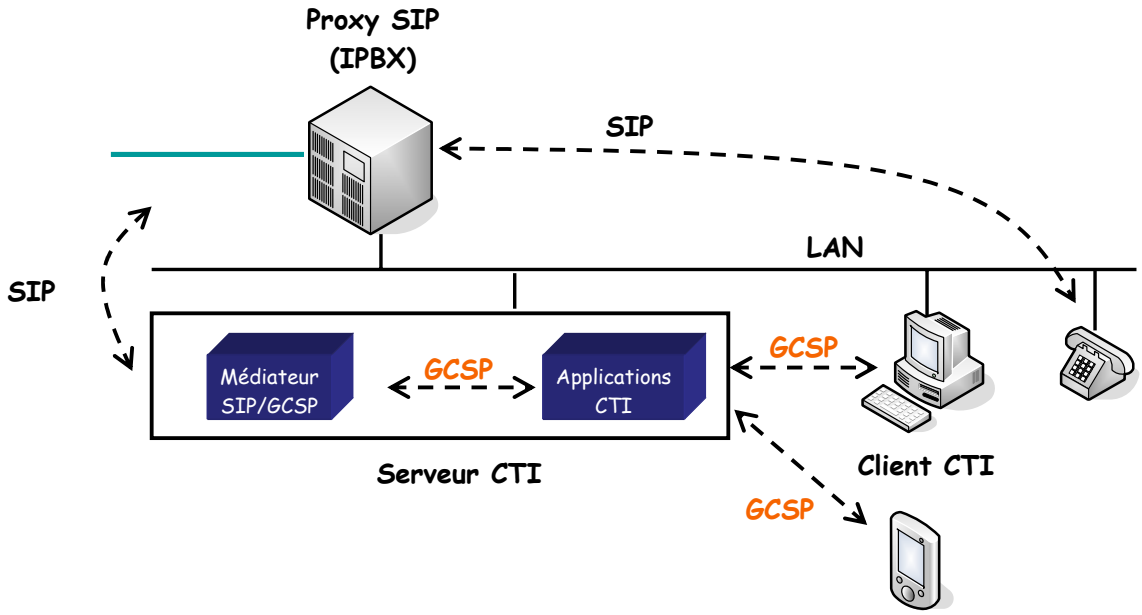
Cette nouvelle architecture a été validée sur un iPBX de la société 3COM et sur la suite d'applications Corebridge.

Les avantages de cette architecture sont les suivants :

1. Plus besoins de licences CTI.
2. Interconnexion facile avec de nouveaux PBX.

3. Possibilité d'accéder aux services depuis des PC extérieurs à l'entreprise (port 80, plus besoin de VPN).

4. Du fait que la pile protocolaire de GCSP est légère nous pouvons maintenant accéder aux services depuis des plate-formes pauvres en ressources comme des PDA et des SmartPhones.



[Figure 9] La nouvelle architecture GCSP du CTI

1.7.3 Implémentation du Contexte Générique

La [Figure 67] montre le schéma UML du CG que nous avons implémenté au niveau de la phone bar, du serveur CTI et du médiateur de signalisation. On a l'objet principale GenericContext qui contient une des trois parties relatifs à :

- Accès : responsable du login de l'utilisateur.
- Transport : responsable de la partie Call Control comme les appels bipartites et multipartites.

- Service: responsable de la notification des évènements téléphoniques nécessaires pour effectuer des services Intelligents tels que la journalisation d'appel et la supervision.

Les objets de ces 3 parties héritent de la classe Service. Ces 3 parties définissent ainsi 3 sessions : Accès, Transport et Intelligence.

1.7.4 Proposition d'une pile GCSP sur UDP

La pile GCSP que nous avons implémenté utilise la couche UDP comme couche transport comme le montre la [Figure 54]. Cette pile est formée par :

- Sous-couche Application : API pour les applications, Message Factory, contrôle de Dialog.
- Sous-couche Sequence : séquencement de messages et gestion des retransmissions.
- Sous-couche Transaction : encapsulation de plusieurs messages dans une transaction.
- Sous-couche Transport : envoie et réception de messages du réseau (UDP).

Nous envisageons dans l'avenir implémenter GCSP sur TCP pour pouvoir traverser les Firewalls ainsi que de pouvoir chiffrer les données transmises avec SSL. Comme les applications Corebridge tournaient sur la plate-forme Windows, nous avons utilisé le langage de développement C# pour développer les piles protocolaire GCSP. Nous aurons pu envisager une implémentation avec JAVA pour avoir une meilleur portabilité sur d'autres plate-formes que Windows mais ça nous aurait requis l'installation de la JRE.

1.8 Conclusion et travaux futurs

Pour conclure, les contributions de nos travaux portent sur :

1. Une approche théorique innovante sur les concepts du plan contrôle.
2. Un approfondissement de la nature coopérative du logiciel du plan contrôle et des conséquences qui en découlent.
3. Ensuite nous avons montré que les méthodes actuelles des signalisations basées commandes pour l'informatique coopérative ne sont pas optimales, pour cela nous avons proposé un changement de paradigme pour la signalisation.
4. Pour mettre en œuvre cette signalisation nous avons proposé une méthode de structuration des données du contexte local.
5. Et puis finalement nous avons proposé le protocole GCSP qui implémente le nouveau paradigme de signalisation.
6. Enfin pour démontrer la validité de notre approche nous l'avons mise en œuvre dans un produit industriel. Pour cela :
    - Nous avons proposé une nouvelle architecture CTI en entreprise qui résout plusieurs contraintes et techniques et financières dont souffrait l'ancienne architecture CTI.
    - Ensuite nous avons défini des médiateurs de signalisation pour s'interfacer avec les signalisations existantes.

Compte tenu du temps nous n'avons pas pu aller plus loin dans nos travaux. Mais le travail déjà réalisé ouvre de nombreuses perspectives intéressantes. Il faudrait maintenant appliquer GCSP à d'autres domaines du modèle SIMPSON que celui de l'IntelligenceComponent, TransportComponent et AccessComponent. Il faudrait aussi proposer GCSP comme protocole de signalisation application dans le cadre du projet NSIS en effet GCSP serait un bon protocole dans ce cadre là. Enfin on pourrait aussi étudier d'autres couches de transport pour GCSP.

INTRODUCTION

The control plane is a rarely addressed subject. This comes from the emphasis put on data communication by early network research which was mostly dealing with "computer networks". Today however, "convergence" has become a fundamental topic and network research has extended its scope to the more general subject of "multimedia communication"

Multimedia communication requires the building of a "control plane" which becomes then a full research subject in the network area. Control plane concepts, in a reduced framework, were familiar to telecommunication engineers in the particular case of voice communication. However these concepts have to be extended and generalized to properly address the new multimedia communication field. The generalization of control plane concepts and mechanisms to more complete forms of communication is the purpose of this PhD work.

In a first part, our purpose is therefore to reconsider the traditional telecommunication control plane ideas in order to place them in a more general theoretical frame fitting the requirements of the new generalized multimedia communication and services. This re-foundation of control plane concepts has never been attempted before in the systematic approach that we use in our work.

An important benefit of the new theoretical frame that we give to the control plane concepts is the identification of research problems associated to control plane activities and to signalling. The second part of our work is therefore dedicated to the identification

of important topics associated to signalling and that remain to be addressed by research efforts. Many of these topics derive from the special nature of control plane software, a special nature that we identify as “Cooperative computing”. The identification of categories of research problems directly linked to the cooperative computing nature of control plane activities gives new insight and emphasis to the fundamental importance and nature of signalling and to the various drawbacks of present day signalling protocols and systems. One of these drawbacks is the impossibility of achieving cross-network and multi-provider services which is a severe limitation to multimedia communication and services.

In a third part of our work we therefore address specifically the problem of signalling to propose a radical shift, a change of paradigm to the way signalling has been approached up to now. This new conception of signalling is intended to break the barriers and limitations of present day signalling methods and in particular would break the cross-network and multi-provider services difficulty.

In the following parts of our work we apply these new ideas on signalling to propose a new signalling protocol called GCSP (Generic Context Sharing Protocol) and we describe in details the Generic Context organization as well as the signalling protocol itself. This work is done as an application of the general theoretical framework that we have identified previously showing at this occasion the efficiency and operability of this theoretical framework.

Finally, as we did not want our theoretical ideas to remain without practical application, our new GCSP signalling protocol has been put to use in a practical industrial application in the case of implementation of Computer Telephony Integration (CTI) services over SIP and IP based telephony systems. The last part of our work is thus dedicated to the description of these industrial implementations of our ideas.

PART 1 - A CLARIFICATION OF THE CONTROL PLANE CONCEPTS: STRUCTURING THE CONTROL PLANE FUNCTIONALITIES

For a long time, engineers were convinced that there was only one way to communicate: a destination identification would be indicated, a communication environment would be established and end-points would exchange information. We know today that this vision was overly simplistic and that there are other ways to communicate by means of networks. In this chapter we introduce and describe the characteristics of the various communication paradigms used by people or machines to communicate. We will see that one of these paradigms: the "conversational paradigm" is very different from the other paradigms and requires special functions that are grouped together in "a control plane". We will concentrate on the explanation of the role of the control plane, on its main concepts and on its very special nature, and we will introduce formal definitions for the control plane and for signalling.

It turns out that control plane software may be considered as one of the major software projects ever achieved by programmers. Typical cost of control plane software for the present day digital switches range in several thousand man×years. What are the functions of control plane software and what makes it so complicated? How could we structure the

control plane activities? These are the questions that we address in this first part of our thesis.

1.1 Communication paradigms

There are several possible ways to communicate or several possible models that may be used for a communication service. These ways to communicate may be called "Communication paradigms" because they really correspond to various "thought patterns" about the way a communication service may take place. It is however necessary to introduce first some preliminary concepts and characteristics that we will use for the description of these various communication paradigms.

1.1.1 Preliminary concepts

1.1.1.1 Communication service. Instance

Generally speaking, a service is a coordinated set of functions that a system brings to people or to software applications (usually at a price). Therefore a communication service is a coordinated set of functions that allow network endpoints to exchange information.

A service instance is a single execution of a service for some particular actors or customers. Therefore two partners exchange information by means of a communication service instance or, for short, by means of a "communication".

A service instance is therefore characterized by the identity of the partners. When network end points are multitask machines, a partner is a task. For example, when two windows of a same web browser on the same PC are opened to different websites, two instances of the same communication service are initiated.

1.1.1.2 Media

A media is a type of information that people exchange and grasp by specific means. Data, text, voice, video are medias. They are different because people grasp them by different means.

The purpose of a communication service is to allow people or machines to exchange media. It happens that engineers were clever enough to find a unique way to represent all the medias as string of bytes. However this is very misleading as it would suggest that a voice byte is the same thing as a data byte and that communication services would merely be "byte moving" services. This is not the case at all. First, different medias may have

different requirements for the way their bytes are moved, and then Communication services are services that move bytes "*according to a communication paradigm*".

1.1.1.3 Session

The session concept is again a very misleading concept. The OSI reference model concept of session is not generally used.

At this time the most accepted meaning, and it will be the meaning that we will use, is that "a session takes place between partners when media are exchanged by these partners" and the word "session" will be used as a shortcut for "communication service instance"

1.1.1.4 Stateful and Stateless machines

Services may be modelled by automata or machines. For our purpose we can classify machines into two fundamentally different kinds.

First we have "combinatorial automata". For a given input they give a unique output and this output is a function of only the input:

$$Output=F(Input)$$

The combinatorial automata does not have memory, it does not remember what happened to it in the past. A good example of a combinatorial automaton is a punching ball. The input is the punch, the output is the back and forth motion of the ball. Always the same punch, always the same back and forth motion.

On the contrary, for the other class of automata, called "sequential automata" the output is not only a function of the input; it is also a function of the history already experienced by the machine:

$$Output=F(Input,History)$$

This means that the sequential automaton has a memory to remember its previous experiences. The number of different histories that a machine may experience is infinite. However many different histories may be equivalent as far as the result for an input is concerned. In this view we summarize the different histories by the concept of "State". A state is the aggregation of different histories that give the same results for inputs. While the number of different possible histories is usually infinite, the number of states may be

finite and in this case the sequential automaton is called a “Finite State Machine” or FSM.

As combinatorial automata do not have memory, they do not have history and therefore do not have states. We call them “Stateless machines”. On the contrary Finite State Machines have a memory and remember their past experience (within a service instance). We call them “Stateful machines”

The concept of State is a fundamental concept in control plane research and is the subject of many discussions. It is a fundamental engineering choice to decide for a stateful or a stateless solution. As an example the well known Session Initiation Protocol (SIP) for Telephony over IP makes use of relay functions for the SIP messages called SIP proxies. A SIP proxy can operate in either a stateful or stateless mode for each new request. When stateless, a proxy acts as a simple forwarding element. It forwards each request downstream to a single element based on the request. It simply forwards every response it receives upstream. A stateless proxy discards information about a message once the message has been forwarded. On the contrary, a stateful proxy remembers information (specifically, transaction state) about each incoming request and any requests it sends as a result of processing the incoming request. It uses this information to affect the processing of future messages associated with that request.

1.1.1.5 Asynchronous and synchronous communication

Communication between partners may be done synchronously or asynchronously. Synchronous communication means that the partners must be “on” together, and react immediately to each others messages. Technically, we talk of synchronous communication paradigms when the emitter of a message cannot proceed further in its communication activities while he is waiting for an answer. The Internet Remote Procedure Call is an example of synchronous communication.

On the contrary, asynchronous communication does not require the partners to be “on” at the same time. Technically we talk of asynchronous communication paradigms when the emitter of a message may carry on its activities while he is waiting for the answer. The so-called Message Oriented Middleware (MOM) [1] are examples of asynchronous communication.

1.1.2 Description of the various communication paradigms

According to their nature, communication services have different ways to make the communication between partners happen. We make this point by stating that a communication service uses one of several communication paradigms. After the explanation of our preliminary concepts we are now ready to describe these various communication paradigms.

At the present day, five different ways to communicate have been identified.[1]. There are two communication paradigms used in synchronous communication (request-reply, conversational) and three communication paradigms used in asynchronous communication (Message passing, Message queuing, Publication-Subscription). We give a fast description of these paradigms knowing that in the scope of this thesis, our attention will be mainly focused on the synchronous paradigms.

Communication involves partners. Always, one partner initiates or originates the communication. Following an old tradition, we call it the "A" partner, and also following a more recent tradition we extend the "A" attribute to "Alice" regardless of the nature of the partner as a human or a machine.

Depending on the service there may be one or several destination partners or terminating partners. In the case when there is only one terminating partner we call it the "B" partner or by the more recent tradition "Bob".

1.1.2.1 Asynchronous communication paradigms

As already explained, in the asynchronous communications, partners are not blocked if they don't receive immediate answers to their sending's. There are 3 different ways to communicate asynchronously.

The Message Passing paradigm

This paradigm is based on a unidirectional transfer. When Alice has something to say to Bob, she says it without a prior checking on the availability of Bob. If Bob is not available, too bad, Alice message is lost! This is a very primitive or simple of communicating but for some services this is the only possible way. Monitoring applications are a good example of applications that could take advantage of the Message passing paradigm, where the monitored partner would send regularly its parameters to a central measurement point. If the Central Point is not on line, then who cares?

Other eligible applications for the message passing paradigm is the driving of very remote machines, where the round trip delay would not be compatible with the expected latency of the communication service.

The Message Queuing paradigm

In the message queuing model, when Alice communicates with Bob there is an intermediate organization Z that hosts a memory space dedicated to Bob. This memory space is organized as a queue for messages. When Alice wants to communicate asynchronously with Bob, she drops her message in Bob's queue at Z. When Bob is available and willing to receive messages he picks up his messages from is queue at Z. The mail is the emblematic service that uses the message queuing paradigm.

The Publication-Subscription paradigm

In the Publication-Subscription model, Alice has many destination partners. We call them the "Subscribers" of Alice. When Alice decides to communicate, she wants her message to be made available to all her subscribers; we say that she "publishes" her message for all her subscribers. Here again there is an intermediate organization Z that hosts a news board for the subscribers of Alice. When Alice publishes a message, she posts it on her board at Z. Her subscribers may then consult this board at their own timing and read the messages. The emblematic service that uses the Publication-Subscription paradigm is the web. A web site is a publication board.

1.1.2.2 Synchronous communication paradigms

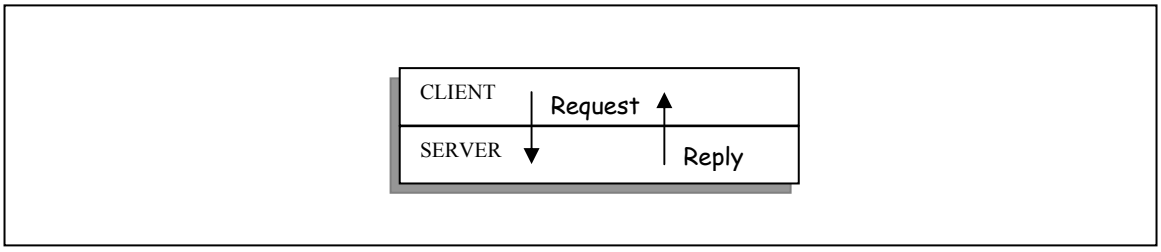
We have shortly explained the asynchronous communications paradigms for completeness. However they are not of any major importance for the purpose of this work. This work is mainly concerned with the synchronous communication paradigms that require Bob to be there and to show some awareness of the intentions of Alice!

There are two ways to communicate synchronously, one is stateless the other one is stateful.

The Request-Reply paradigm

The Request-Reply paradigm is based on an interactive, bidirectional communication limited to a single exchange of one message in each side. The message emitted by Alice to the destination Bob is called a "request". The communication instance lasts only during the time that is necessary for Bob to build up and send the reply to the request and does not go beyond.

There is no memorization of the exchange, no persistence of any resource, no persistent context. It is a connectionless mode of operation according to OSI terminology and it is a stateless paradigm because no memory is kept after the answer to the first message has been sent. In the general case of "request-reply" communication, both parties may originate the request and the other party replies with the answer. The communication is synchronous; after the emitter sends a request, it suspends its communication activities while waiting for the answer.



[Figure 10] "Client-server" as a particular case of "request-reply"

However this general case of "request-reply" is rarely used. A particular case of "request-reply" communication called the "client-server" communication paradigm is mostly used [Figure 10].

It is a single mono-directional type of "request-reply" where one party only issues request and the other party only issues answers: A client always originates the communication session. A server is "always on" and only sends answers to a client. A server never originates a request to a client. A client only pulls information from a server; a server never pushes information to a client. It should be noted that the Internet is a network that has been optimized for the client server communication paradigm, although some other types of communication may take place over it, in a non-optimized manner.

The conversational paradigm

The second synchronous communication paradigm is the "conversational" communication. It is also based on an interactive, bidirectional communication however it is not limited to a single exchange of one message in each side. Many messages may take place in both sides, constituting a "dialog". This is the definition usually given. However we find that another definition is more significant: We prefer, in this work to use the following definition of the conversational communication paradigm:

Definition: "In the conversational paradigm, a communication environment is explicitly set-up before the users start exchanging media and this environment remains established even in the absence of user activity until an explicit command is issued to release it"

From this definition, we derive that the communication environment is persistent, meaning that it remains set-up as long as an explicit release is not issued. Therefore, this environment is memorized for the duration of the communication session. It is a connection-oriented mode of operation according to OSI terminology and it is a stateful paradigm. The emblematic example of service requiring a conversational communication paradigm is the Plain Old Telephone Service POTS where resources are reserved in all participating switches and are freed when one of the participants hangs up.

From the definition, we also derive that any service or application that use the conversational communication paradigm must use the services of another application dedicated to the setting up and the releasing of the communication environment. This joined application is called "control application". The control application therefore takes care of putting in place, modifying and releasing the communication environment.

1.2 Control plane concepts

We have now described the various communication paradigms known at this time. We have seen that the conversational paradigm is very particular because it requires special functions called “Control functions” to setup modify and release the communication environment. Control is therefore specific to the conversational paradigm, in opposition to management which applies to all communication paradigms (A deeper analysis of the differences between control and management will be given further). We proceed by giving formal definitions for the concepts of control, control plane and signalling, and by deriving from these definitions some important concepts that are specific to the conversational communication paradigm.

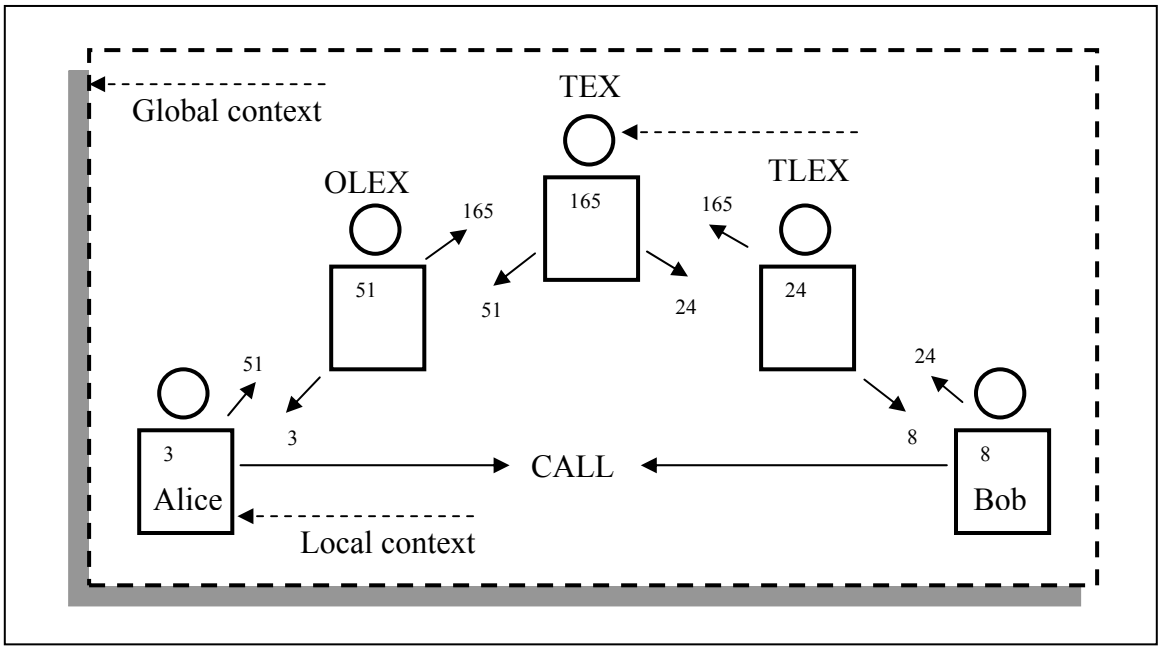
1.2.1 Formal definition of Control

A service using the conversational communication paradigm should invoke special functions dedicated to setting-up and releasing the communication environment. We define as **"Control functions"** the functions executed by all the partners of a conversational communication instance to **set-up, modify**, and finally **release** the communication environment for this communication instance. This communication environment is persistent for the whole duration of the conversational service session; therefore **control has a “session by session” significance.**

The limitation of control activities to the session duration makes a fundamental difference between control and management activities. Management, in general, is the adjustment of service parameters. The effects of control finish with the session while the effects of management persist beyond the sessions. Control acts on **Session Instance Data (SID)**, while management acts on **Service Support Data (SSD)**. Every type of service has to be managed regardless of the communication paradigm they are using. On the other hand only services requiring a conversational communication paradigm need control functions.

1.2.2 Local and global contexts

A communication service instance using the conversational paradigm is a global process involving many partners and therefore the control process is a global process. However it is executed in a distributed way since each partner has a control process running.



[Figure 11] Control processes, local and global context, associations

We call **"Local Control Process"** a running instance of a control software at one of the partners of the service instance which collaborates in a multitask environment with the other local control processes of the other partners to set-up, modify and release a global conversational service instance.

During a service session, a local control process opens a memory page to store its data and this memory page persists during the entire service session until it is freed by an explicit release command. We call "**Local context"** the memory page opened by a given participating control process in which it stores its state and Session-Instance-Data. However, the local contexts of each participating process put together are to be considered as a global context for the service session.

We call **"Global Context"**, the union of all local contexts that give a global view of the session. The global context is the memory associated to the global control process.

The circles on [Figure 11] show these local control processes. Each local control process, in its turn, opens a local context shown by a rectangle. The control process uses the local context to store its state and it's Session-Instance-Data for the session duration. When the communication session is terminated, an explicit release is issued and all the memory pages are freed, deleting in the same time all the Session Instance Data. On [Figure 11] we also represent a conversational communication session between machine (or human) Alice "A" and machine (or human) Bob "B". We have control processes running in Alice, at the Originating Local Exchange (OLEX), at the Transit Exchange (TEX) and at the Terminating Local Exchange (TLEX).

We call **"Control plane"** the connected set of all processes or entities executing control functions either in the terminals or within the network.

It is important to note at this point that humans are single-task systems while machines are multitask systems. Networks dedicated to human users (like the telephone system) don't need to specifically address a task in the (human) end-points as there is only one task running in the endpoint. On the contrary, computer networks dedicated to (multitask) machines must provide means to specifically address a given task within a (machine) endpoint. Indeed, machines being multitasks usually have several simultaneous sessions of conversational communication open at the same time and therefore have several local context open simultaneously, belonging to different communication sessions.

1.2.3 The association requirement

We have seen that the "Global Context" is the union of all local contexts that give a global view of the session. This union is organized as a "**link list**": each local context must have a pointer to the local context of the partner for this global service instance.

*We define that local control processes are **"associated"** if they can mutually address each other among multiple control instances within multitask machines*.

These associations are achieved by the cross-referencing of contexts: each participating control process must maintain a table of the context references of the other processes with which it communicates. By the association mechanism, each local context has a pointer to the others as shown on [Figure 11]. Because the global view of the communication session, i.e. the complete information about it, is spread in all the local contexts, the global context is therefore made of a link list of associated local contexts in the same manner as sectors of a disk are linked together to form a file. Local contexts association mechanisms are similar to memory and file management in an operating system. However since local contexts are distributed, the difficulties raised by the "Association"

is of the same order of the difficulties met in Distributed File Sharing and Distributed Memory Sharing (DSM) systems. In the SUN Network File System (NFS), the remote filesystem "association" is done by a MOUNT command. A UNIX client can access to a shared filesystem on a remote server by specifying the remote host name, pathname of a directory in the remote filesystem and the local name with which it is to be mounted [2]. The remote host name, pathname and local name constitute a binding reference or a pointer to the distributed filesystem sub-tree. The NFS filesystem "association" creates a set of binding references that allow a UNIX client to use shared filesystem sub-trees. On the other hand Distributed Shared Memory (DSM) is an abstraction used for sharing data between computers that do not share physical memory [2]. Thus local contexts association looks more alike to the one in Linda [3] or JavaSpaces [4] because local contexts are not stored in a physical memory but persist only during the service session and are erased from memory at service session end.

Today there are many protocols that allow such persistent cross-referencing like the TCP protocol, the dialogs and transactions identifications in TCAP [5], and the CORBA [6] associations. The disadvantage of these protocols is that they are specific to some particular networks and do not allow a cross-network operations.

1.2.4 What is a Call?

An association of special interest is the end points association. For example, on [Figure 11] Alice knows her conversation 3 is the conversation 8 of Bob and Bob knows his conversation 8 is the conversation 3 of Alice. Several research groups have found convenient to name *"*call*"* this particular association. According to this definition, the ***"Call"*** *is an association graph between network end-points*. This definition has very useful consequences and has been adopted by several ITU-T recommendations for B-ISDN and for IMT2000 [7]. More generally, to include multi-party calls we define the "***Call***" as ***an association (cross-referencing) graph of the local contexts of network end-points participating in a same conversational service instance.***

An important consequence of this definition is that the "Call" has an end to end significance. End-to-end call services include, in addition to the fundamental association service, presentation functions and bearer negotiation functions.

Another important and surprising consequence of this definition is that a Public Switched Telephone Network (PSTN) does not process calls. The calls (cross referencing of end-point local contexts) do exist over the PSTN, but they are done by the human end users: The small conversation: "Hello, I am Alice, I would like to talk to Bob…Hi Alice, Bob speaking!" is actually a protocol by which Alice and Bob associate their references. It is a

call protocol. It is not done by the network but by the end users. We will see that a Public Switched Telephone Network (PSTN) instead of processing calls assigns connections (setup bearer services) to calls.

If Alice and Bob are machines they have to use a similar call protocol to associate their local contexts. In a GSM network, when a VLR calls an HLR over the SCCP network in a connectionless mode, the call protocol used is TCAP. TCAP exchanges originating and terminating dialog identifications and therefore is a call protocol.

1.2.5 Communication environment and Connections

If the PSTN does not process calls what does it do? The PSTN actually establishes connections. While the call is an association between network end-points, ***a "Connection" is an assignment of resources to a given call***. While the Call function is an end-to-end process, ***the connection function is a link-by-link process***.

Resources may be of various natures. They might be physical 64 Kbit/S circuits like in the PSTN, a route reservation like in the Connection Oriented Packet Switching like X25 networks, they might be a bandwidth reservation like in the Integrated Service (INTSERV) QoS mechanism or a traffic aggregation reservation like in the Multi Protocol Label Switching (MPLS) networks.

Connexion services are also called "Bearer services" and the resources are also called "Bearer facilities". What technicians and many standards like the Intelligent Networks (IN) [8] standards usually name a "Call Control Function" is indeed a "Bearer Control Function".

It is very interesting at this point to note that the PSTN network has a strict link by link way of working. Contrary to a widespread belief there is no end to end function in the telephone network and in particular there is no "Look ahead function" and "Pass Along" messages are rarely used if used at all. We now understand this point because we now know that the telephone network only does connections and connections are strictly link by link processes.

At this point we can now discuss the nature of the famous "communication environment" concept that we have used to define the conversational communication paradigm. What is a "communication environment?

First it is memory: every service using the conversational communication paradigm must make a call, i.e. open local context in every participating entity and therefore in the end-

point entities. Therefore the communication environment is at least made up with all the local contexts.

Then, in addition to memory, the communication environment may also be made of resources. This is the case when connections are required. It derives that we may have connectionless calls, in this case the communication environment is made up of ***memory only*** and we may have connection-oriented calls, in which case the communication environment is made up of ***memory and resources***.

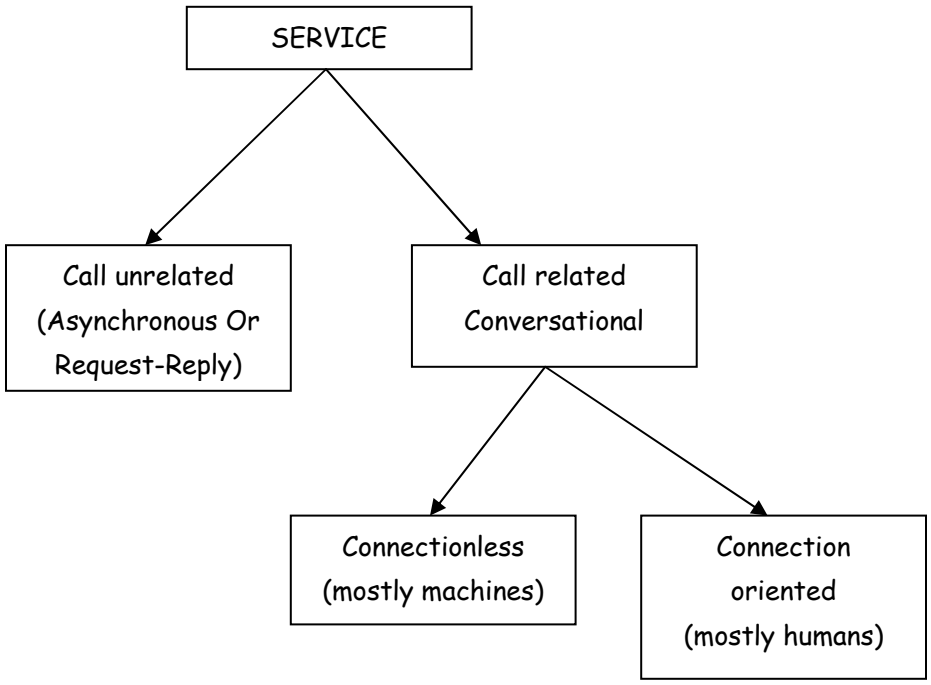
1.2.6 Service, Call, Connection

We can summarize the preceding considerations by pointing out three different concepts:

- Service: A service may use one among the five communication paradigms.
- Call: the concept of call is associated to the conversational communication paradigm. If the service does not use the conversational communication paradigm, there is no call, the service is “call unrelated”

  If the service uses the conversational communication paradigm, there is a call and the service is “call related”
- Connection: A connection is an assignment of resources to a call. There are no connections without calls and therefore Connections only make sense for the conversational communication paradigm.

  However, we may have calls without connections if no resources other than memory are needed, which is always the case for TCAP calls over the Class O or Class 1 SCCP network used for the communication between the GSM control entities or the Intelligent network control entities.



[Figure 12] The three concepts of Service, Call, and Connections

It should be noted at this point that most of the communication between machines does not need connections (except for large file transfers). On the contrary communication between humans usually requires connections to insure the necessary QoS.

1.2.7 Why PSTN did not need a "call function" and why multimedia networks do need a "call function"?

The PSTN establishes connections and assigns them to a call (established subsequently by the human-end users. We will see that this awkward order: connection first, call second is quite unfortunate and prevents a complete unbundling of service and calls). This strange sequencing is made possible because the PSTN does not allow any choice for the connection characteristics. Like Ford Customers could only get black cars PSTN users can only get 64Kbit/S connections. In the PSTN there is generally one default connection service and this default connection service is not negotiable. (ISDN departs slightly from this view because ISDN terminals may request different bearer attributes)

Because the Bearer service is non negotiable, the bearer facility may be setup prior to the call and the call may follow.

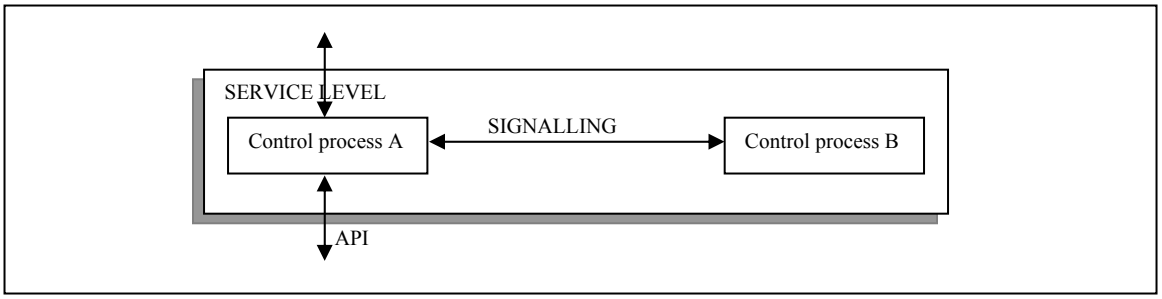
Now lets consider a multimedia network. ***Multimedia communication cannot assume a default bearer service*** (connection characteristics). It is up to the customer to decide if he wants to see his girl friend in black and white or in colour and to assume the corresponding cost if he chooses the colour! Therefore multimedia networks must necessarily include end-to-end call control functions and protocols because multimedia functionalities have to be negotiated and agreed before the corresponding bearer facilities may be setup. Because of this, Multimedia networks now necessarily include end-to-end call services which, in addition to the fundamental association service, provide presentation functions and bearer negotiation functions. In particular, the famous "***look ahead***" function by which the availability of the terminating party is tested prior to any attempt of connection establishment is now ***compulsory for multimedia communication***. We remember that this function was an impossible function in the PSTN.

Examples of call protocols used for Multimedia networks are the IP telephony "Session Initiation Protocol" SIP [9], the H323 [10] protocol suite where the call protocol is the H225-Q931 protocol, and also the "Bearer Independent Call Control" BICC [11] used in B-ISDN.

In multimedia networks a Call protocol operates first. Once the call is accepted, the agreed bearer service has to be setup by Bearer Control.

1.2.8 Signalling

A conversational service instance requires the sharing of a global context built-up by the association of local contexts. Local contexts set-up and association is done by control functions that need to communicate in a distributed environment. Indeed the control plane is a distributed application in which control functions may belong to different logical or physical machines. In addition there is no centralized point, all control plane entities are equal. To exchange instance-data in a distributed environment, control processes communicate by means of signalling. *We call **"signalling"** the exchange of instance-data between associated local contexts of partner control processes that cooperates together to set-up, modify and release a same conversational service instance* [Figure 13].



[Figure 13] Signalling and control processes

Since signalling is an exchange of messages in the control plane between control processes in order to set-up, modify and release a conversational service instance, thus it is a service that allows the distribution of control software.

1.2.9 Management

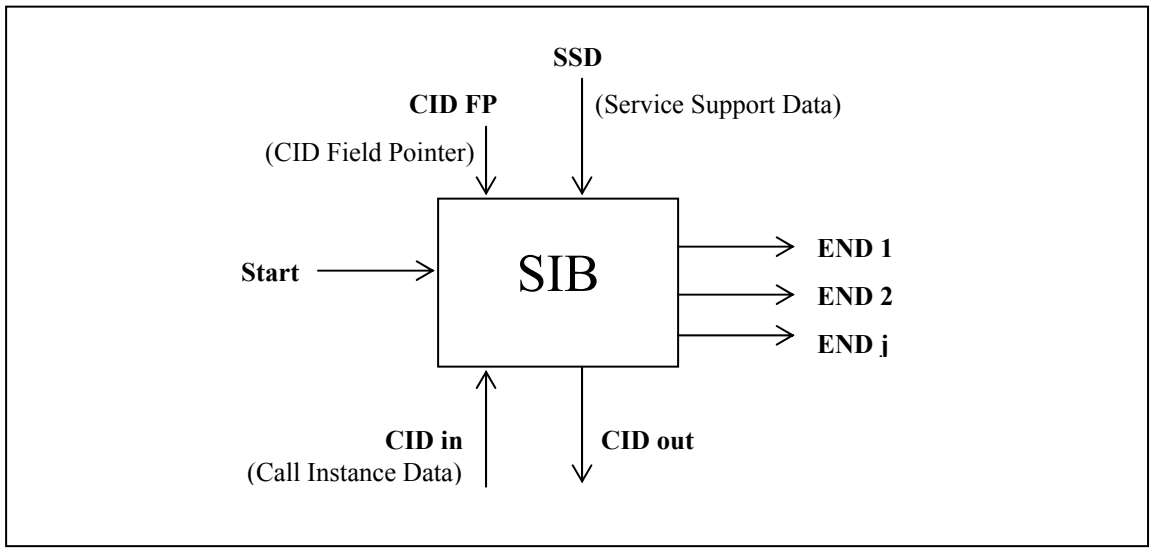
It is important at this point to underline that control should not be confused with management.

Control is the setup of the communication environment. Only services requiring a conversational communication paradigm need control functions: control is specific of the conversational paradigm. There no control in the other communication paradigms. The effects of control finish with the session: control activities are limited to the session duration, they have a session by session significance. Control acts on **Session Instance Data (SID)** contained in local contexts erased at session termination.

On the other hand, management is the adjustment of service parameters. Every type of service has to be managed regardless of the communication paradigm they are using and management applies to all communication paradigms. The effects of management persist beyond the sessions. Management acts on **Service Support Data (SSD)** contained in persistent Management Information Bases MIBs.

A good example of the difference between Control and Management is provided by the Intelligent Network [8] specification of the Service Independent Building Blocks SIBs. This specification clearly separates the control data called "Call Instance Data (CID)" and the management data called Service Support Data (SSD) [Figure 14].

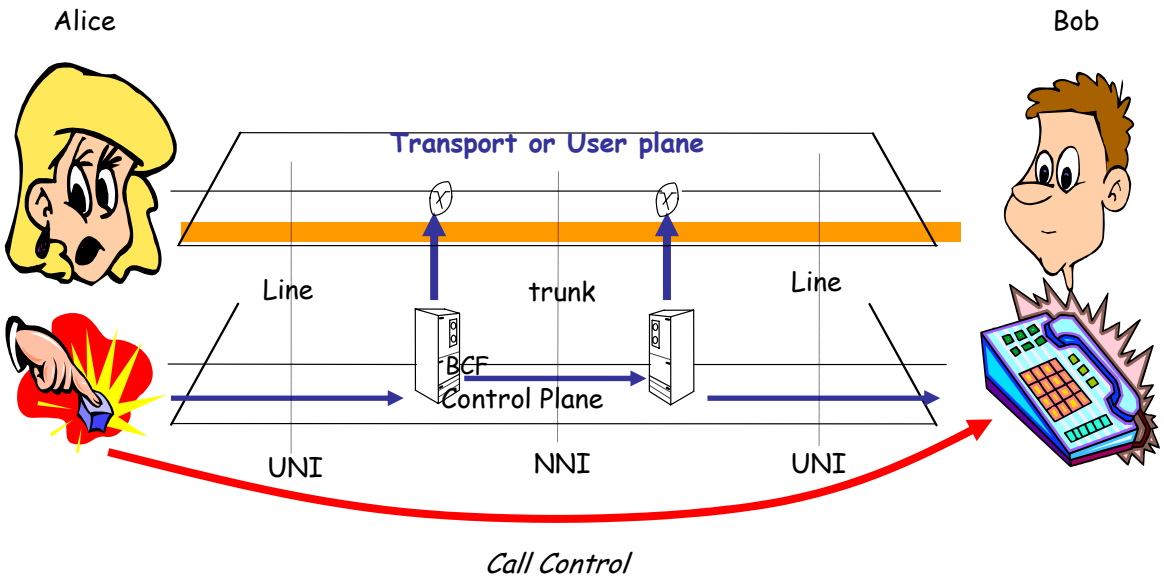
Control and Management are actually very different activities. While Control is best achieved as a cooperative process and involves very small file transfers, Management is best achieved as a centralized process and involve large file transfers. Mechanisms for control and management are indeed fundamentally different in nature.



[Figure 14] Management (SSD) data and Control (CID) Data in the intelligent Network

1.2.10 Control plane and user plane

Standardization bodies use the concept of plane. A plane is a set of *communicating* entities linked by a specific network. Standardization bodies agree that telecommunication infrastructures involve several planes, each of them dedicated to specific tasks in the provision of a global communication service. A plane being also a network, the telecommunication infrastructures therefore require several networks, one per plane.



[Figure 15] Control plane and User plane

Each plane works out-band from the others and may therefore have its own ciphering algorithms and keys. The entities in each plane work in a cooperative manner, either with other entities in the same plane or with entities in the other planes with which they are supposed to have open and standardized interfaces

We don't intend to describe in this work all the various planes required. For example, this work being dedicated to Control, we will not address issues related to the Management plane. We will concentrate in this work on the control plane and the user plane.

In the **User plane called also the Transport plane** we have the transmission and switching (or routing) capabilities for the users' media (transmission links, switches or routers).

A telecommunication infrastructure adapted to services using the conversational paradigm should include a control plane. Indeed partners using this type of conversational communication need control processes and network partners (central offices, call agents, proxies) make use of control processes. All these control processes are linked together in a control plane.

1.3 Structuring the control plane activities

We have mentioned that control plane programs belong to the largest software projects ever attempted. What makes these programs so complicated, what are the functions they have to provide? We try to classify the control plane activities into control plane domains.

1.3.1 Control plane domains

All the proposals for a control plane architecture from the PSTN, the GSM architecture, and the future proposals including UMTS, Mobile IP, TINA [12] and OSA [13][14][15] require the same categories of activities for the setup of a communication environment. These categories of activities are the access functions, the Intelligence functions, The call functions and the Bearer functions [16][17]. Because we find these functions in all the network technologies, we call them **"control plane invariants"**:

a) Access functions

The access functions are dived into two categories: *originating access* functions and *terminating access* functions:

- ***Originating access*** functions takes place when the user logs into the network. They are responsible for identifying and authenticating the user, updating its subscriber location, and downloading the user services and profile. They include Virtual Home Environment (VHE) services, location dependent services, and mail services

  In the fixed network the originating access function is executed in the calling subscriber central office at "off-hook" time (pre-selection) to fetch the calling subscriber profile (features, directory number, classes of service…). Localization and authentication is not useful since the subscriber is physically attached to this central office. For the same reason, the subscriber profile data base remains local to the switch and there is no access signalling in the fixed network.

  On the contrary, in the mobile network, the originating access session is initiated by the visited MSC (Mobile Switching Centre) and its VLR (Visitor Location Register) and continued by the HLR (Home location Register) in order to authenticate the mobile subscriber and to return the subscriber profile to the VLR

and to the MSC. Since the HLR and the VLR are distant machines, Access Signalling is required in the mobile network.

A user profile contains all the information specific to the user: class of service, subscribed services, identifications, and pointers to his various mail (voice mails, emails, MMS…). In the IP Multimedia Services architecture the user profile becomes a "Virtual Home Environment" concept.

In GPRS, the originating access function is called the "GPRS Attach procedure". In a SIP based Telephony over IP network, the origination access function is accomplished by means of the SIP REGISTER method.

Terminating access functions are services required for the contact function (Name address Translation, Presence services, calling party record presentation…). The basic function is the name/address translation. A telephone number is not an address, it is a name. An address is routable, a name is not routable.

In the PSTN a subscriber has two different identifications: a Directory Number which is a name and a Line Equipment Number which is an address. A central office does not ring a Directory number; it rings a Line Equipment Number. Within a central office, there is total independence between Directory Numbers and Line Equipment numbers. Therefore a database lookup has to be performed at every incoming call to convert the Directory Number (Name) into a Line Equipment Number (address). This particular Name/Address translation is called "Terminating Translation" and is a local function of the terminating switch in the fixed network.

On the contrary, in the mobile network the terminating access function is executed by the MSC receiving the communication and consists in contacting the HLR in order to convert the called party number MSISDN (Mobile System ISDN Number) into an MSRN (Mobile System Roaming Number) that allows to route the call within the PLMN (Public land Mobile Network)

In IP networks, the DNS function that translates the domain names to IP addresses may be viewed as a terminating access function.

It should be noted that Terminating access depends from the Originating access through the localization function and therefore both functions have to remain grouped as **"Access services"**

b) Intelligence Functions

Intelligence functions are often referred to as "service functions", which is not very proper since everything in networks provide a service! Actually people call them "service" functions as a short cut for "value added services" where the "intelligence function is a service not normally provided by the network.

To grasp the concept of "intelligence functions" we have to consider that the telephone service is a connection executed "link by link" by telephone switches as a sequence of "elementary connection actions" In a first approach, we can classify these "elementary connection actions" in the following way:

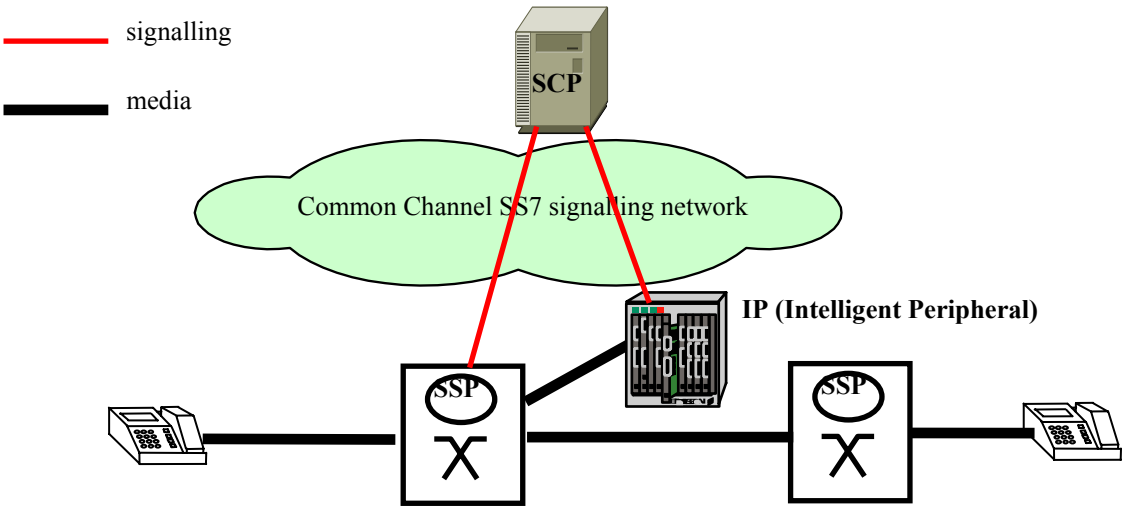
- Connect the user to a subscriber receiver
- Connect the user to a Tone and Announcement Circuit
- Send a prompt (Dial tone)
- Send a Vocal announcement to the user
- Receive digits keyed by the user
- Route a call (translate dialled digits into a route index indicating the trunk group where the call should be connected)
- Connect an inlet of the Switching Fabric to an outlet (belonging to a given trunk group)
- Ring a user
- Supervise events on a subscriber line
- Disconnect an established connection
- Release assigned resources
- Establish a charging scheme (Charged party, Charge index, Initiation time) for a call
- Establish a charging ticket for a call.

There would be many different possible ways to sequence these "elementary connection actions". It happens that, from the origin of telephone times, telecommunication operators have agreed on a common way to sequence these "elementary connection actions" to achieve the Basic Call Process. We call this particular sequence the POTS service (Plain Old Telephone Service). Therefore the POTS sequence is the default service programmed in all the telephone switches.

However, as we have underlined it, many other sequences would have been possible giving telephone services different from the POTS. By definition, we say that a ***network is an "Intelligent Network" if it is possible to substitute to the default connection sequence (POTS) of a switch, an alternate sequence programmed in an external service platform called a Service Control Point SCP [Figure 16].***

Therefore, an "Intelligent network" is a network where external computers (service platforms) may program at their convenience a sequence of "elementary connection actions" and an "Intelligent Network Service" is an ***alternate bearer control***.

It derives from this definition that an Intelligent network service is not any kind of service! It is a service that can only be executed by the network, it cannot be executed by a terminal outside the network since it is a sequence of connection actions that only a switch can do and based on information that only the network has. Terminals outside the network do not have this information and cannot do these connection actions. A good example of Intelligent network service is the 800 number calls (free calls) where the call charging is assigned to the called party, when the default sequence normally assigns the call charging to the calling party.

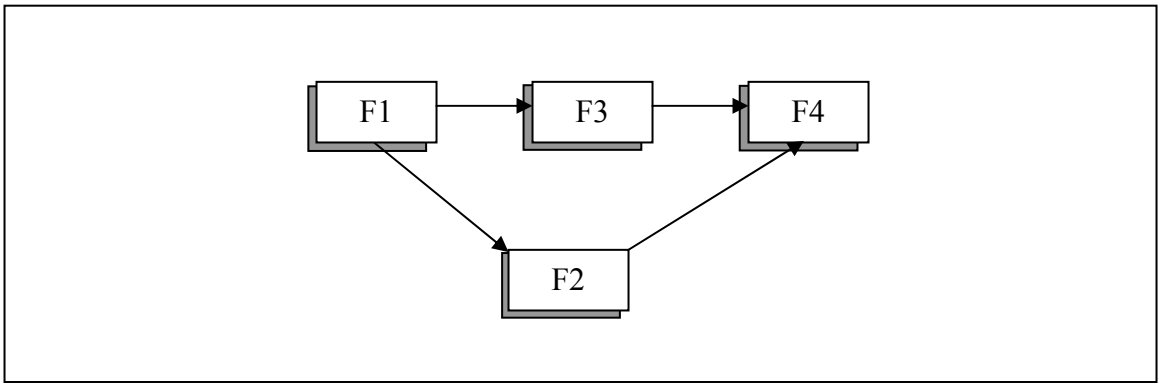


[Figure 16] Switches and service platforms in the intelligent network

It derives also from this definition that a service that could be done entirely in a terminal without resorting to any special connection processing or to special functions that can be found only within the network would not be an Intelligent Network Service.

The Intelligent Network technology therefore does not address any type of service, ***it addresses services that may only be achieved by a substitutive sequence of network functions.***

IN services are new combinations of "elementary connection actions" called also "service features" [Figure 17]. Service features may be viewed as service components and IN services are designed as a graph of such components [18].



[Figure 17] Intelligent Network service graph

c) Call functions

The call function establishes, supervises and releases the temporary associations between network end points, whether they are done in the connection oriented mode or in the connectionless mode. The call has an end-to-end significance, it allows to associate end-points (network terminals) through the referencing mechanism. This association persists even during the absence of communication activities. A typical call function is provided by the TCAP protocol; all TCAP messages contain an OTID (Originating Transaction ID) and a DTID (Destination Transaction ID) which combined together form a unique call reference.

We have already mentioned that the PSTN does not operate call functions. To be more precise, in the PSTN the call is implicit. The PSTN achieves a connection service for a call that will be established afterwards by the end-users!

We have also seen that this state of affairs will stop with multi-media networks: multimedia networks must necessarily include end-to-end call control functions and protocols because multimedia functionalities have to be negotiated and agreed before the corresponding bearer facilities may be setup.

Because of this, Multimedia networks necessarily include end-to-end call services which, in addition to the fundamental association service, provide presentation functions and bearer negotiation functions.

In a SIP based Telephony over IP network, the SIP protocol itself is a call protocol (and not a connection protocol).

d) Bearer functions

The Bearer or Connection function provides the required quality of service. A connection is a reservation of resources (circuit, bandwidth, codec, scheduling priority, traffic aggregate, CPU/MEM …) for a call.

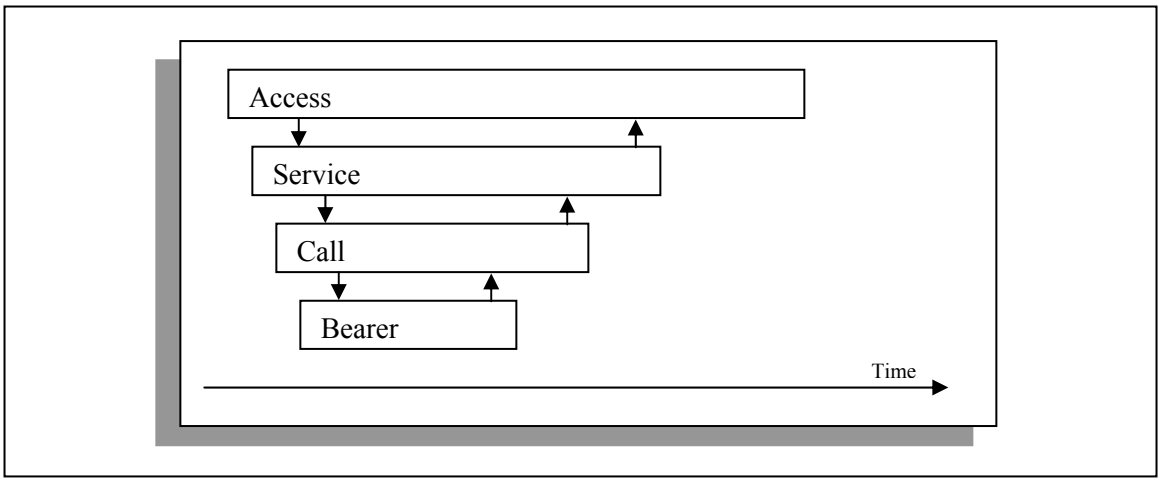
In a SIP based Telephony over IP network, the Bearer Capabilities are described by the Session Description Protocol (SDP).

1.3.2 Precedence principle

Invariant functions should not be invoked in any order, they should follow a global sequencing scheme [16] shown on [Figure 18].

We must note however that the PSTN does not abide by this precedence principle which explains why there is no complete service/call/connection independence in the PSTN

On the contrary, because Service/Call and Call/Connection is a requirement of Multimedia networks, these new networks will be conformant to the precedence principle.



[Figure 18] Global sequencing of a communication service

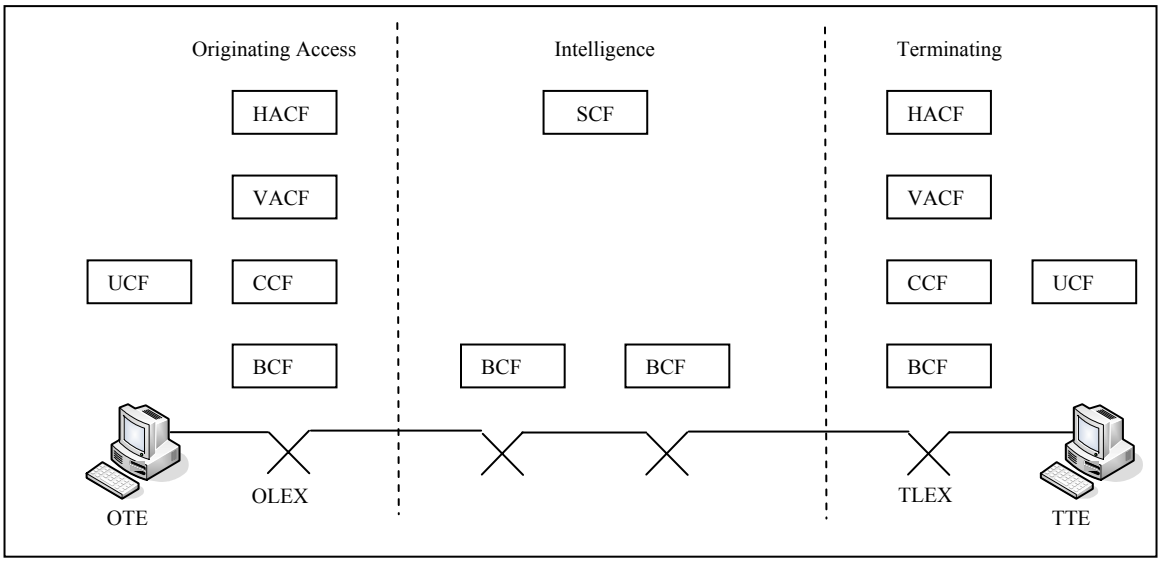
According to this principle, a user has first to log into the network and initiate access functions. After completion of the access functions, the subscriber may be satisfied with the default “Basic Call process” or invoke an Intelligent Network service.

In any case a Call function will be invoked and will also be used to negotiate the bearer services.

Then the Bearer service with its appropriate QoS will be established.

1.3.3 A generic functional architecture

In this paragraph we give an overview of the functional architecture implementing the global sequencing scheme and the invariant functions separation. The functional architecture for the control and service plane is derived by assigning a separate entity for each invariant function. A detailed description of this functional architecture may be found in [16][17][19].



[Figure 19] Global control plane functional architecture

The architecture is shown in [Figure 19]. It entails the following elements:

The User Client Function (UCF), located in the client terminal, provides the user with a service browsing environment and an interaction with his access, connectivity and service providers.

The Visitor Access Control Function (VACF) serves the UCF to which it provides a graphical interface server and a secondary database. The VACF is a partner for the Access function.

The Home Access Control Function (HACF) is the primary database for originating and terminating functions. It stores user profiles and locations. The HACF is the main partner for the Access function.

The Service Control Function (SCF) is an Intelligent Network service execution function within an Intelligent Network Service Platform. The SCF is a partner for the Intelligence function.

The Call Control Function (CCF) relays call signalling messages transmitted end to end between network end-points. In a SIP based Telephony over IP network, the CCF corresponds to the SIP proxies. The CCF is a partner for the Intelligence function. It should be noted that the CCF here does not correspond to the entity called CCF in Intelligent Network specifications.

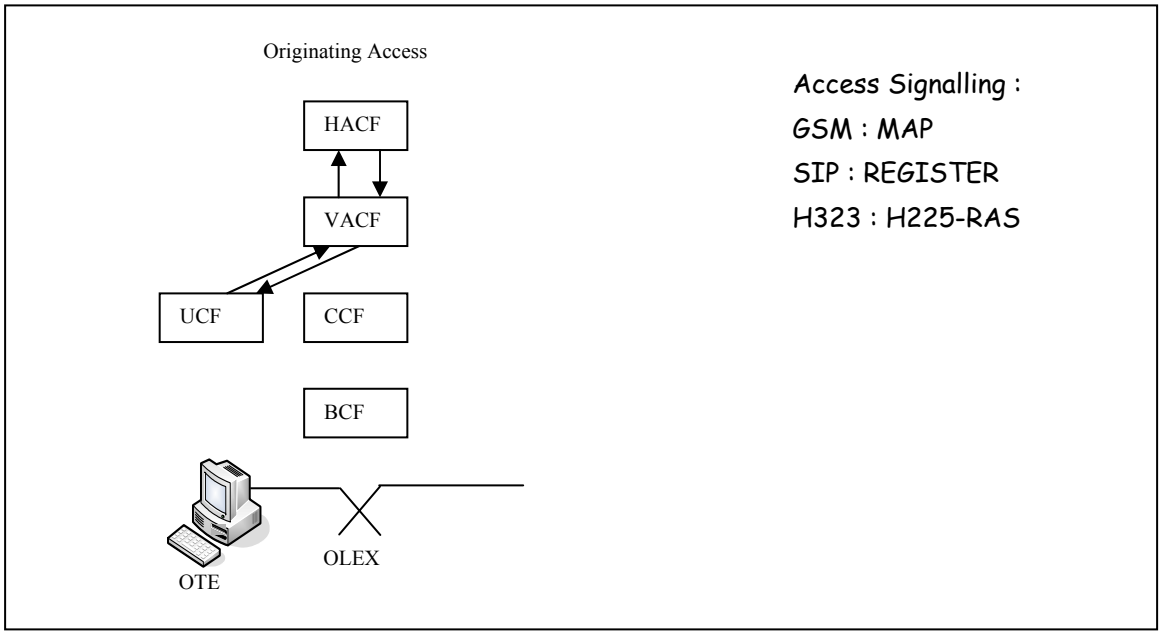
The Bearer Control Function (BCF) routes the call and reserves, link by link the bearer facilities required on each link between the network nodes of the route. A connection oriented call set up involves many BCFs, one for every switch involved in the setup route. The BCF is a partner for the connection function.

The control entities of the global control plane described above cooperate together by means of signalling to put in place, modify or release a conversational communication instance. These control functions are spread across the 4 invariant control plane domains (Access, Service, Call and Bearer). We will see that they may also be operated by different providers.

We now give a brief description of the dynamic operation of the architecture.

1.3.3.1 Originating Access

The originating access session is responsible of logging the user terminal to the network, of the authentication of the user, of updating his location and of downloading his profile. We describe the originating access function on the general case [Figure 20] and we will instantiate the general case on the example of a mobile GSM network.



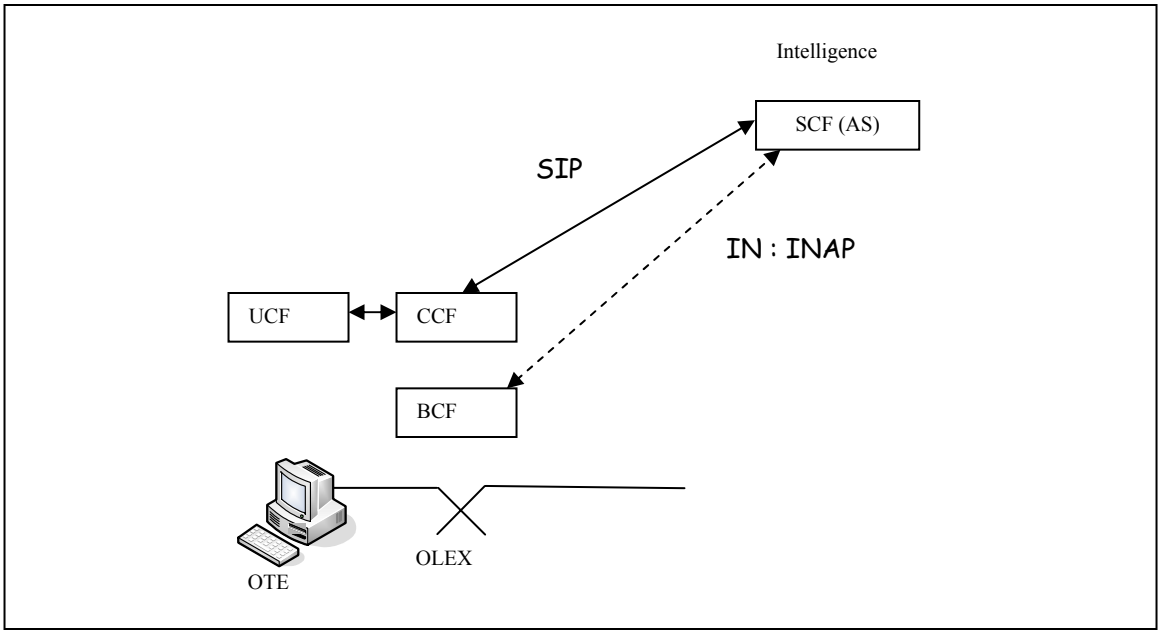
[Figure 20] Originating access signalling

When a user logs on a terminal, the UCF reads a login file from a Subscriber Identity Module (SIM) card. Essentially, the login file contains the user's identity, the identification of its telecommunication services supplier and its authentication information. The UCF transmit by means of signalling (called Mobile Application Part MAP in the case of GSM) this information to the VACF (VLR in the case of GSM) which in its turn contacts the HLR to update the user location and download the authentication data. Once authentication is achieved a new HLR interrogation downloads the user profile into the VACF.

1.3.3.2 Intelligence

Intelligent Network services are executed by a Service platform hosting the SCF. An Intelligent Network service is defined by a graph of service components. They enrich the network basic services and allow to build richer services out of the elementary connection

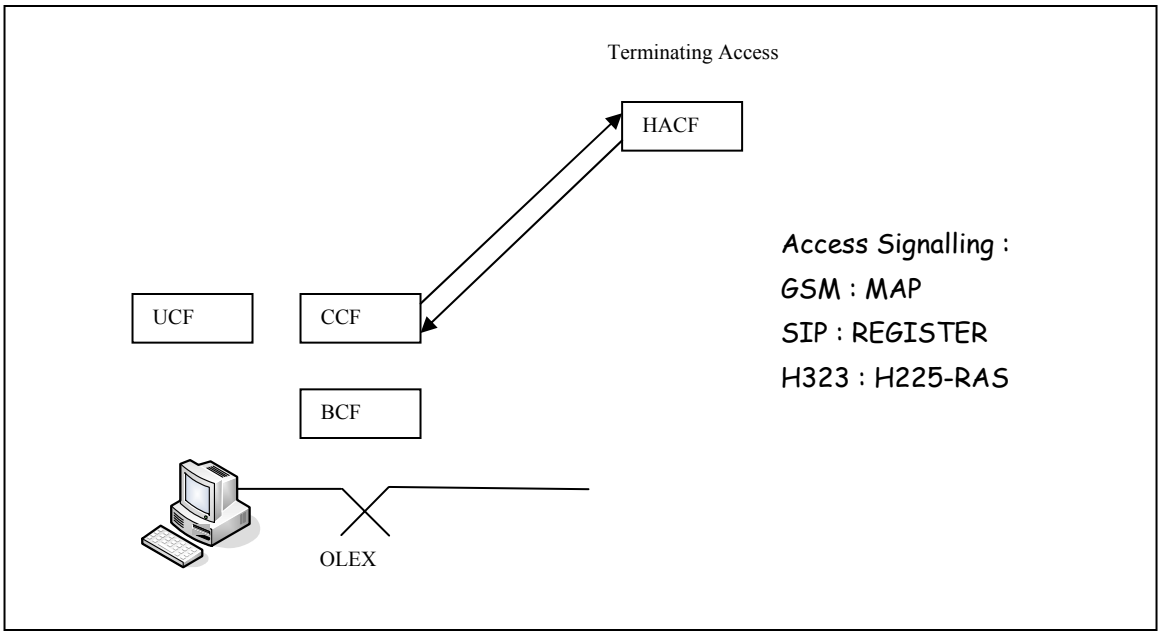
actions of the Bearer Control function. In the Intelligent network technology of today, the precedence principle is not respected, connection takes place first while the call is still implicit and intelligent network services are triggered from the BCF (called CCF in ITU-T IN specifications). In future multimedia networks the precedence principle should be respected and services triggered directly from the user before a call is made. In SIP based networks this will be done by the intermediate of the CCF (SIP proxy) by forwarding the SIP messages to the SCF called, in this case, Application Server AS.



[Figure 21] Intelligence signalling

1.3.3.3 Terminating Access

Some intelligent services may require establishing a call, in this case the SCF communicates by means of signalling with the CCF. The terminating access session provides a name/address translation. At terminating access, the CCF interrogates the HACF of the called party's telecommunication operator. (HLR in a GSM network) [Figure 22]. The HACF performs the name/address translation (MSISDN to MSRN in a GSM network) and returns to the requesting entity a routable address for the called party.

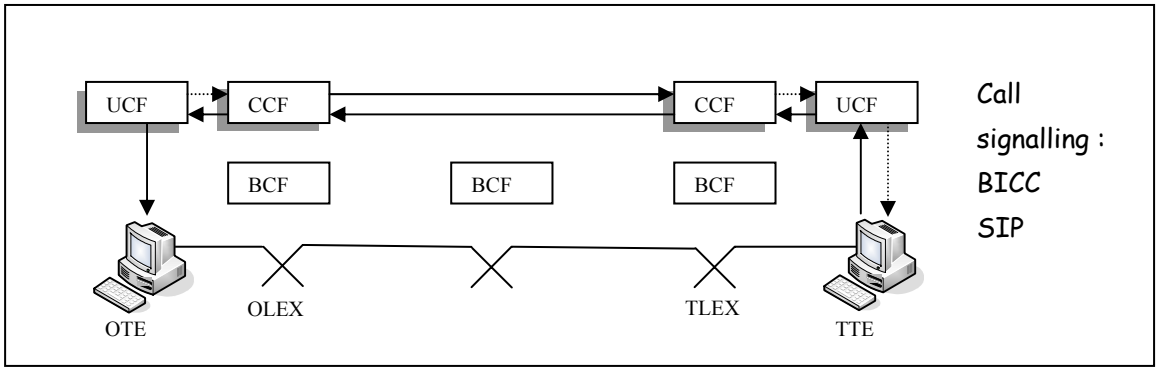


[Figure 22] Terminating Access signalling

1.3.3.4 Call

Call Control Functions establish end-to-end associations and are required in multimedia networks to negotiate bearer services. Calls have an end to end significance but call signalling may be relayed by proxies called here CCF (Call Control function)

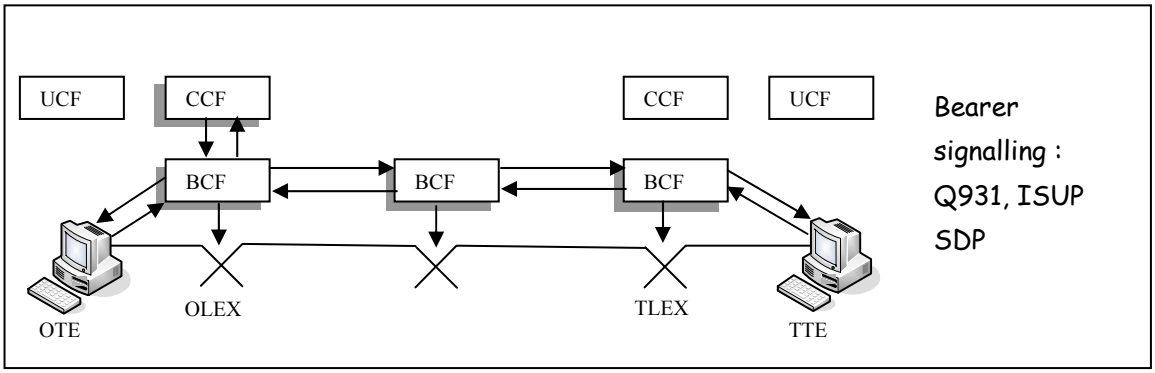
Call signalling has been defined only in the case of Multimedia Networks and include the Bearer Independent Call Control Protocol defined for Broadband ISDN networks (ATM networks), the Session Initiation Protocol for Multimedia over IP networks and also the H225-Q931 signalling of the H323 protocol stack.



[Figure 23] Call Signalling

1.3.3.5 Bearer

In many cases the Call requires a reservation of resources. It is always the case when end-points are humans because the human brain is so intolerant to delay that some kind of QoS mechanism have to be set up which ends up to make resource reservations. It is also true in some cases of data communication.



[Figure 24] Bearer Signalling

Bearer Control Functions are responsible for setting up, link by link (or hop-by-hop), the resources that will be used in the media transfer. Example of resources may be PCM time slots (circuits), or a QoS path in RSVP [20]. Bearer control is triggered by the CCF (where the Bearer capabilities have been negotiated). Examples of Bearer Control signalling are Q931, ISUP [21], SDP [22].

1.3.4 Signalling domains and legacy protocols

The general decomposition of control plane activities into control plane domains leads to the identification of the following “Signalling Domains”:

- The “Access Signalling Domain” for the signalling between access functions.
- The “Intelligence Signalling Domain” for the signalling between Intelligence functions.
- The “Call Signalling Domain” for signalling between Call Control functions.
- The Bearer signalling domain for signalling between Bearer Control functions.

In [Figure 25] we give examples of legacy protocols and map them into the four signalling domains.

Domains	Legacy Protocols
Access	MAP [23], V 5.2, Register (SIP), H225-RAS (H323)
Intelligence	INAP [24] and CAP (CAMEL), SIP (IMS)
Call	BICC, H323 (H225-Q931), SIP
Connection	H245, Q931, ISUP, SDP, RSVP

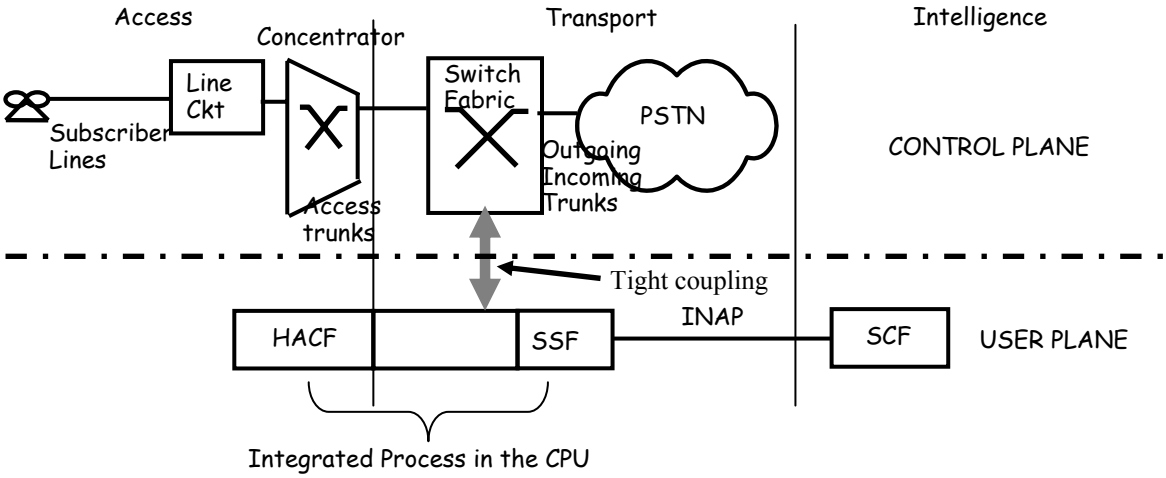
[Figure 25] Signalling domains and legacy protocols

1.3.5 The end of the integrated switch model

Telecommunication networks are based on the Telephone switch.

In the user plane this equipment has the following structure: Subscriber lines are terminated by line circuits contained in line concentrators. Line Concentrators connect lines to access trunks with a concentration ratio depending on line traffic (in a typical case 150 access trunks serve 1000 lines). A switching fabric connects any access trunk to an outgoing trunk going to a distant exchange or an incoming trunk or coming from a distant exchange.

In the control plane a (duplicated) computer called Central Processing Unit CPU executes the Access (HACF) the Connection (BCF) and the Intelligence service Switching (SSF) functions.



[Figure 26] Structure of the integrated switch

However these functions are not executed as separate processes, they are integrated in a huge single process that technicians call the “Call Processing program”, leading to what we call the “integrated switch model”.

It would be desirable, for many reasons that will later, to separate the functions of each control plane domain into separate processes, eventually distributed in separate platforms. Unfortunately this is not feasible with the synchronous Time Division Digital switching

technology because this technology requires a tight coupling between the CPU and the Switching Fabric.

This state of affairs changes with the advent of the asynchronous Voice over IP Technology and in particular with the soft-switch technology. In this new technology the switching fabric is replaced by the spread out routers of the internet network ant the tight coupling requirement between control units and switching fabric disappears to be replaced by a loose coupling between media-gateway controllers and media gateways

It becomes then feasible to separate the functions of the various control plane domains into separate processes running nearly independently … eventually in different machines.

1.3.6 Unbundling control domains

As the separation of the functions of the various control plane domains becomes feasible, it becomes legitimate to raise the question of "who does what in switching?" and it appears clearly that the so called "Call Processing" function was actually an integrated processing (by a master) of activities of a different nature that could be advantageously processed by peer cooperative systems without subordination relations. Several research groups have attempted to identify the control activities that could be eligible for a separate processing in different cooperative platforms, eventually belonging to distinct business partners, and propose unbundled architectures. Therefore, the unbundling concept is not limited to the access function. Indeed, the unbundling concept may be extended to many more activities that we are now going to identify.

1.3.6.1 Horizontal unbundling

A first proposal from the TINA research effort consists in separating Access services, Transport services and Intelligent Network services. We have already described these services:

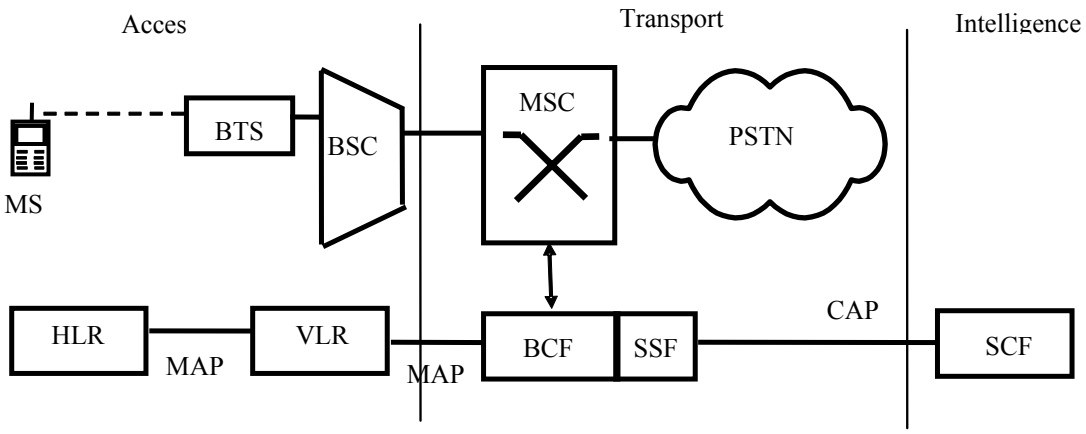
Access includes the originating and terminating access functions that we have described

Transport includes the Call Control functions and the Bearer control functions that we have described

Intelligence includes the Intelligence functions that we have described

[Figure 27] shows how a mobile telephone network may be partitioned into these 3 domains. Different stakeholders may operate these three functional domains, in an unbundled manner, provided that they work in a cooperative manner.

We define as horizontal unbundling this first unbundling scheme.



[Figure 27] Mapping of a GSM network on the various functional domains

1.3.6.2 Vertical unbundling

We have introduced the concept of plane and defined the user plane and the control plane. Standardization bodies also agree that multimedia network, often called Next Generation Network (NGN) will be made of several planes, each of them dedicated to specific tasks in the provision of a global communication service. A plane being also a network, the NGN will include several networks, one per plane. Each plane having its own terminating entities may therefore have its own ciphering algorithms and keys. The generally agreed NGN model (see [Figure 28]), that we will name **"NGN plane model"**, considers three different planes: the Service plane, the Call Control plane and the Transport plane.

In the **Service plane** we have the service provision platforms operated by Service providers. In the **Control plane** (or session plane), we have the entities in charge of the end-to-end Call setup protocols and the entities in charge of the link-by-link setup of bearer services. In the **Transport plane** we have the transmission and switching (or routing) capabilities for the users' media. This NGN plane model is to be considered as a cooperative model. The entities in each plane work in a cooperative manner, either with other entities in the same plane or with entities in the other planes with which they are supposed to have open and standardized interfaces.

A very interesting aspect of this model is that each plane may be unbundled, i.e. operated by different stakeholders. We may therefore have Call Control Service Operators acting in the Control plane and Connectivity providers acting in the transport plane. This is made possible by the soft-switch architecture and already today we find, in many places, Call (or Session) services operators using Media Gateway Controllers to provide Call

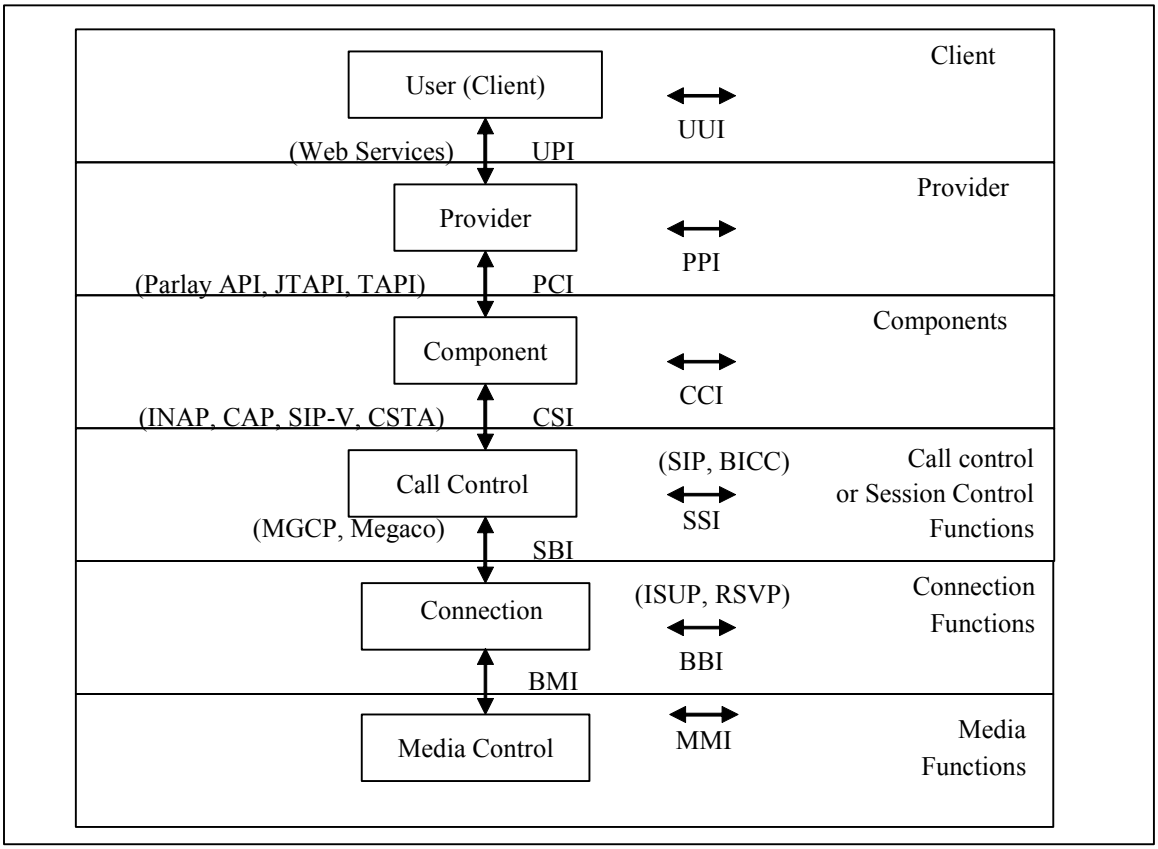
services over the media gateways located in the transport plane. Media gateways may actually be embarked in new IP telephone sets, now becoming available at a normal telephone set price, and compatible with the standard Media Gateway Control protocols MGCP or MEGACO [25]. Some other Call Control operators like the "Skype" company [26] use proprietary downloadable interfaces at the expense however of using a PC as a terminal. This new separation between Control operators and transport operators confirms in the NGN the end of the "Integrated of the telephone switch" that is currently taking place.

NGN plane model	ETSI Model	SIMPSON model
Service plane and service network	Service control	Client
		Provider
		Components
Control plane and control network	Call Control services	Call Control services
	Bearer control	Bearer control
	Media Control	Media Control
Transport plane and transport network	Transport services	
	Transport Control	
	Transport flows	

[Figure 28] Various models for service provision in the NGN

While agreeing with the three planes of the NGN plane model, the ETSI [27] has further distinguished, within each plane, sub-services of a different nature that do not justify however a different communication network. The control plane includes Call Control Activities, Bearer (Connection) Control activities and Media control activities these last ones being dedicated to the communication between the participants various media coders and decoders. All these control plane sub-functions may however communicate through the same "control network". In the same way, the ETSI model distinguishes in the transport plane the transport services and the transport control functions from the transport user flows, although they all communicate via the transport network. Another research group [28] has further refined the service plane sub-functions. In this model published under the name of **"SIMPSON model"** [29], where Simpson stands for "Signalling Model for Programmable Services over Networks" (see [Figure 29]), "**Multi-provider Services"** are taken into account. We define as "multi-provider service" a service built as a graph of independent service components of different nature and executed by different component providers. As an example we may consider a car manufacturer Virtual Private Network (VPN) service. In addition to VPN functions that may be supplied by an intelligent network service operator, financial components supplied by a banking company and inventory management components supplied by a specialised provider may be included. Such a service integrating service components

from different suppliers in a single user interface is a "multi-provider service". In order to take multi-provider services into account the service plane must include client sub-services, provider (integrator) sub-services and component sub-services. The Simpson model includes these sub-services in the NGN service plane as well as the ETSI sub-services of call control, bearer control and media control for the control plane and therefore the Simpson model is an unbundling model for the Service and the control planes.



[Figure 29] SIMPSON Model

The various Simpson levels constitute another unbundling scheme that we call a vertical unbundling. It should be noted however that for every horizontal service function that we have underlined in the horizontal unbundling scheme (Access, Transport, and Intelligence) we have a vertical Simpson column.

1.3.6.3 Signalling and APIs

The SIMPSON model is a powerful model to identify and characterize functions, interfaces and mechanisms in the control and service planes. It is also a powerful

modelling technique for many different service architectures. We will use this model here to identify two different types of interactions between control and service plane entities with the example of Parlay [30] services over a Soft-switch architecture [31]. Interacting entities may belong to the same service level and they operate in the peer-to-peer mode. We call **"horizontal signalling"** this type of horizontal communication within a single Simpson level. On the contrary, they may appear in adjacent service levels. We call **"vertical signalling"** this type of vertical communication. Vertical signalling protocols are frequently referenced as **APIs** (Application Programming Interfaces). On the [Figure 29], the SIMPSON model shows the various vertical and horizontal signalling protocols required in the control and service planes architectures, identified by generic acronyms. In parenthesis we have given some examples of legacy protocols at the various levels.

As APIs we find that the User to Provider Interface UPI may be implemented by Web services. At the provider and component levels, the Parlay group proposes a new service architecture differing from the Intelligent Network IN architecture by a supplementary level of service customization. At the provider level a service integrates some component abstractions in a Parlay service platform. Some of these component abstractions are Call Control components. An example of the Provider to Component interface PCI is the Parlay API. By this API, a service provider invokes Call Control components in a Parlay gateway such as an Ericsson Jambala platform [32] belonging to a Call service operator. At the Call service (or Session service) level and Bearer service level the softswitch architecture implements a separation between call control services (performed by the Media Gateway Controllers MGCs) and bearer control functions (performed by the Media Gateways MGs). On one hand, IP connectivity operators (at the bearer level) provide customers with MGs and take care of the IP forwarding functions. On the other hand, Call Control operators (at the network level) operating MGCs outsource the Call Control functions or the IP-Centrex functions formerly performed by PABXs. An example of Component to Session Interface CSI API could be the intelligent network INAP or CAP set of operation [33], by which a Parlay gateway may invoke the services of a Service Switching Point SSP control unit within a Soft-switch MGC.

Finally the Call control functions of the MGC request Bearer Services from Media Gateways MG by means of the Session to Bearer Interface SBI API. The MGCP or MEGACO protocols are examples of such SBI APIs. As horizontal signalling protocols or peer to peer protocols we find in the client level the User to User Interface UUI type of signalling between Clients. SMS should be considered as medias and therefore do not enter in this category. User to user signalling, has been frequently mentioned, but has not been implemented so far.

We find in the provider level the Provider to Provider Interface PPI type of signalling between servers. Here again we cannot say that examples of such signalling protocols exist at the present time. We find in the component level the CCI type of signalling between component providers. An example of CCI signalling is the MAP signalling between different mobile networks. Another example of CCI signalling is the SCP-to-SCP signalling of future IN capability sets, or the SCF to SDF signalling of IN-CS1. We find at the Call service operator level the SSI Session to Session Interface types of signalling. The Session Initiation Protocol SIP signalling or the Bearer Independent Call Control BICC signalling are examples of SSI Call signalling protocols. Peer-to-peer services implement proprietary SSI Call signalling protocols to associate their users for file transfers. Finally, in the bearer level, we find the Bearer to Bearer Interface BBI type of signalling. Examples of BBI signalling abound due to fact that the Plain Old Telephone Service POTS is indeed a Bearer control service. All Circuit Associated Signalling CAS protocols are actually BBI types of signalling protocols. The most important BBI signalling protocol at this point is the ISDN User Part ISUP signalling protocol widely used between telephone exchanges. We may remark at this point that the main horizontal signalling protocols, implemented so far, belong to the lower levels of the SIMPSON level. This comes from the very centralized way in which services have been designed up to now. There is now a very sharp contrast between the cooperative computing of nowadays telephone exchanges and the centralized computing presently used in the service layers. An important direction for research is certainly to make the service layers more cooperative, which will require the development of horizontal signalling protocols in the upper layers of the SIMPSON model.

1.3.6.4 Two dimensional unbundling or “Extended SIMPSON model”

Unbundling is a 2-dimension problem as shown on the table of [Figure 30]. Each place in this table is actually an independent business opportunity. The condition for this however is that all parties shown on the table agree to work together in a cooperative manner. Benefits from this cooperation would be twofold:

a) A richer service offer

b) A controllable service complexity, each partner having a limited set of functions to develop.

Access	Transport	Intelligence	
Access Client	**Transport client**	**Intelligence Client**	**Client**
Access Services provider	**Transport services provider**	**Intelligent network service provider**	**Provider**
Access component operator	**Transport Components**	**Intelligent network components**	**Components**
Access session services	**Call and bearer control services**	**SSP, IVS**	**Session**
Access transport network	**Transport network**		**Bearer**
Access media control	**Media Transport control**	**IVS media control**	**Media**

[Figure 30] Horizontal and vertical unbundling dimensions of the extended SIMPSON Model

1.3.6.5 Extended SIMSPON view of a general communication service

We describe now how a general communication service, of any kind, may be mapped on the extended SIMPSON model [Figure 31]:

a) Access functions:

At login, the Access Client invokes an Access Provider service. In most of the cases, the service invoked is a Registration (or a login) service at the Provider level. This Registration service is done by the composition of service components from the Component level such as Authentication. As an example, in the GPRS network, a GPRS Attach service (Registration) requires an authentication component in addition to a localisation (location update) and a mail component. The mail component is the SMS and emails received when a mobile subscriber switches on his mobile phone.

As explained in [paragraph 1.3.1], the Name/Address Translation component is required in the terminating access phase. As an example, in the GSM network the Telephone number MSISDN (Mobile Station ISDN Number) is a name, it is not routable. The HLR has to be interrogated to return a MSRN (Mobile Station Roaming Number) which constitute a routable address.

These Registration services require an association during all the registration period. For example, in the mobile network, there is a session between the MS (Mobile Station) and the VLR which identifies his location. This location is updated when the user moves. The session ends when the MS is switched off or when the user makes a handover which

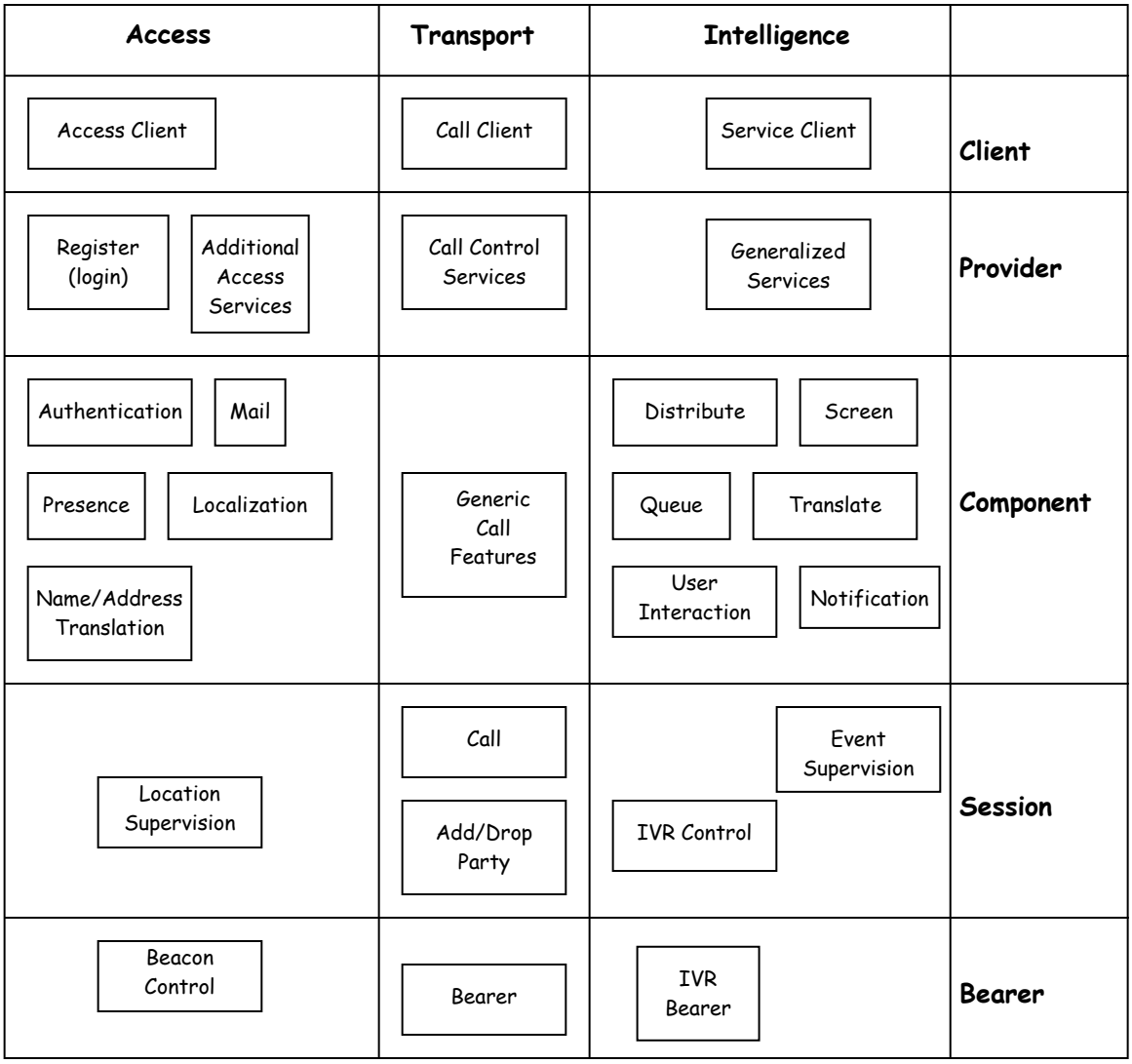
moves him under the control of a different VLR. In addition, there is a session between the MS and the HLR which describes the MS state (on/off). When the MS changes from a state to another, trigger points may be fired to execute a new service (for example deliver SMS when the MS is switched on).

b) Transport functions:

A call is initiated by means of the call client. On the Provider level, we find the Call Control Services. These are sophisticated call setup logics, including multiple calls and multi-party calls. These sophisticated call setup logics are made from combinations of call features components at the component level. Finally, components from the Transport level require session functions. In the PSTN or the IMS, these functions are done by the CCF and the S-CSCF respectively. Finally, bearer functions are required to setup a bearer. While the session has an end-to-end significance, a bearer service is built link by link or hop-by-hop.

c) Intelligence functions:

Generalized Services are invoked from the Service Client. These Generalized services are made at the Provider level by a combination of intelligence components executed at the component level. An Example of intelligence components may be a Least Cost Routing (LCR) component which allows a CTI user to customize the routing of his calls depending on the day of the week and the time. An other example of intelligence component is the Automatic Call Distribution (ACD) component which allows to make call centre services. The User Interaction component allows the sending of voice notifications or music to a user and to receive DTMF indications from the user. A Notification component offers line notification for a user. Some components, such as the notification and the user interaction, require a session. The session allows for example to deliver event notifications for a notification service while the IVR control makes a session between the voice server and the switching function.



[Figure 31] Extended SIMPSON view of a general communication service

PART 2 - RESEARCH PROBLEMS ASSOCIATED TO SIGNALLING

In the first part of this work, we have outlined the fundamental structure of control plane activities in order to place them in a general theoretical frame fitting the requirements of the new generalized multimedia communication and services.

Now, in this second part, we analyze the difficulties of implementing the control plane functions, leading us to the identification of research problems associated to control plane activities and to signalling. Many of these topics derive from the special nature of control plane software, a special nature that we identify as “Cooperative computing”. Our analysis of the cooperative computing nature of control plane activities will give new insight on the fundamental importance of signalling and to the various drawbacks of present day signalling protocols and systems. One of these drawbacks is the difficulty to build easily and at lower cost conversational services that could be extended over heterogeneous networks (Cross network services) or services that have components from different service providers (multi-provider services). Thus we outline the constraints for a new control plane proposal that will improve the building of richer and cheaper services.

2.1 The cooperative computing nature of the control plane software

So far control plane activities have required huge amounts of programming effort, certainly classifying them among the largest programs ever developed. The programming of the classical call control of present day's digital telephone exchanges has required thousands of man-years of programming effort. *The origin of this difficulty may be traced into the special cooperative nature of control software.* To analyse this point let's review the main characteristics of the various computing organizations.

2.1.1 Cooperative computing versus Centralized and distributed computing

For the purpose of our work we consider that computer science proposes three different solutions to the problem of driving processes: centralized computing, distributed computing and cooperative computing.

Centralized Computing was the original state of the art of computer science. A very powerful mainframe would master all the processes in a company. All terminals, machines, tools, would be intelligence-less slaves executing orders of the central Master computer.

Later on, new companies pushed forward a new type of computer science called **Distributed Computing**. In distributed computing, many smaller computers, called "minis", work together, specializing on given types of tasks and providing some amount of department or activity independence. This new computing organization required communication, and therefore networks, between the computers. The general solution developed by computer science for distributed computing is the "Client-Server" architecture, based on the "request and reply" communication paradigm. However, the client server architecture should be considered more like an adaptation of the former centralized scheme to the distribution problem than like a radically new solution. The client is mostly concerned by customisation and interface problems and the essential service data and service logic are located in the server central position.

A radically new solution to the distribution of intelligence on many smaller computers would be a new kind of computer science called **"cooperative computing"**. In

cooperative computing, there is no central position, all the computers are equal and no one is in a *permanent* position to give orders to the others.

While many different efforts are taking place towards the development of a theoretical solution for cooperative computing, (grid computing, peer to peer processing, agents…), no generally accepted theoretical base has been yet proposed

Nevertheless some examples of working cooperative processes, successfully developed, do exist. The main one, for our concern, is the "call control" of telephone switches. Indeed control functions work in a cooperative manner. In the Public Switched Telephone Network (PSTN) switches cooperate for setting up a communication between two or more subscribers. There is a routing hierarchy, there is no control hierarchy. all switches are equal, there is no centralized platform controlling the setup of a call or its release. Each switch works on a peer-to-peer basis to achieve a global service. It is necessarily so for performance reasons: Today a public switch control unit has a processing capacity of the order of 10^6 Busy Hour Call Attempts (BHCA). This would be insufficient to establish all the communications on a country scale. Centralized or distributed computing (client/server), based on a central point, would not provide a working solution for a big and a fast network. The control plane therefore requires a different kind of computing with a distribution of the computing power on different machines without any central processing point. This is the "cooperative processing" organization.

2.1.2 Requirements of cooperative computing

The explanation of the tremendous complexity of control plane software may be traced into the special cooperative nature of control software. It is because of this special cooperative nature of control activities, and of the lack of a general theoretical base for this new type of computing, that "call control" was developed as an ad hoc solution through a huge effort and many trial and errors. So far, the main efforts attempted by the computer science research community towards the handling of control activities are directed towards some adaptations of the client-server architectures. This is the case of the, now generally accepted, Session Initiation Protocol (SIP). A remarkable exception to this statement is the "active network" research and its connections to agent programming. However this type of solution still remains in a very early stage. We advocate at this point that the special cooperative nature of control activities for services requiring a conversational paradigm justifies a serious attention to fundamental research in cooperative computing. For this purpose it is possible to identify some key subjects for research in cooperative computing:

- Cooperative computing requires information sharing: information sharing between control and service plane partners is called “signalling”. It derives that Signalling research is not merely a research problem for telephony; it is a fundamental computer science research problem for cooperative computing. Signalling is certainly one of the foundations of cooperative computing. We propose a new signalling paradigm based on information sharing where signalling operations consist of reading and writing instance-data in a remote context of a partner control process [PART 3] and a new local context generic structure that all control processes understand. We call such local context a Generic Context [PART 4].

- Cooperative computing requires the set-up of Associations between the partners: each process involved in a service session has pointers to his partner processes. All pointers, put together, form an association tree that gives a global view of the service session. We call such association tree a Control Allocation Table (CAT). A CAT links together all local contexts of control processes involved in a same conversational service instance. CAT mechanisms are detailed in [PART 4].

- Cooperative computing requires policies for the distribution of decision authorities: it is not because every participating entity is equal that decisions should not be taken. For a given problem, at a given moment, who takes the decision for the whole cooperation? This is a general and very difficult problem (also experienced in other areas than computer science: setting up decision mechanisms for cooperating countries for example is a very important and unresolved problem at this time). This point also shows that the ad hoc solution of the telephone networks is not a general solution. Telephone networks use a round robin policy: First Alice takes a decision, then the originating exchange, then the transit exchange, then the terminating exchange, then Bob. It is clear that this policy, well adapted to the link-by-link operating mode of the connection process cannot be generalized to any kind of cooperative computing problem.

- Cooperative computing requires behaviour models for the partners: how to take a decision if the behaviour of the partners is not known (i.e. they are not predictable)? Each partner should have a behaviour model of the partners with whom it has working relations. This is the reason for the so important "Basic Call State Model" BSCM in intelligent network technology, and for the various call models in Computer Telephony Integration CTI, as well as the connection modelling in MEGACO signalling. This consideration is an argument in favour of stateful proxies in the new VoIP architectures rather than stateless proxies. Behaviour models are taken into account in the design of the Generic Context [PART 4].

- Cooperative computing requires confidence in the partners: this obvious consideration is not an easier research problem than the preceding ones. In their monolithic model, the historical telephone operators would identify the "trusted domain" of their own closed network and the "un-trusted domain" of the third party service suppliers. The soft-switch architecture and its derivatives like the NGN IMS architecture are bound to terminate the integrated model of the telephone switch and to raise the confidence problem of cooperation among external partners. Authentication is required to ascertain the identity of the partner. Ciphering is required to avoid eaves dropping or information substitution. Security issues are taken into account in the design of the Generic Context [PART 4] and the design of the new signalling protocol [PART 5].

2.2 Limitation of current concepts

Our analysis of the cooperative computing nature of control plane activities has given a new insight on the fundamental role of signalling. We now examine how cross-network and multi-provider services could be designed with the current signalling mechanisms and we discuss the limitations of these current signalling mechanisms for such generalized services.

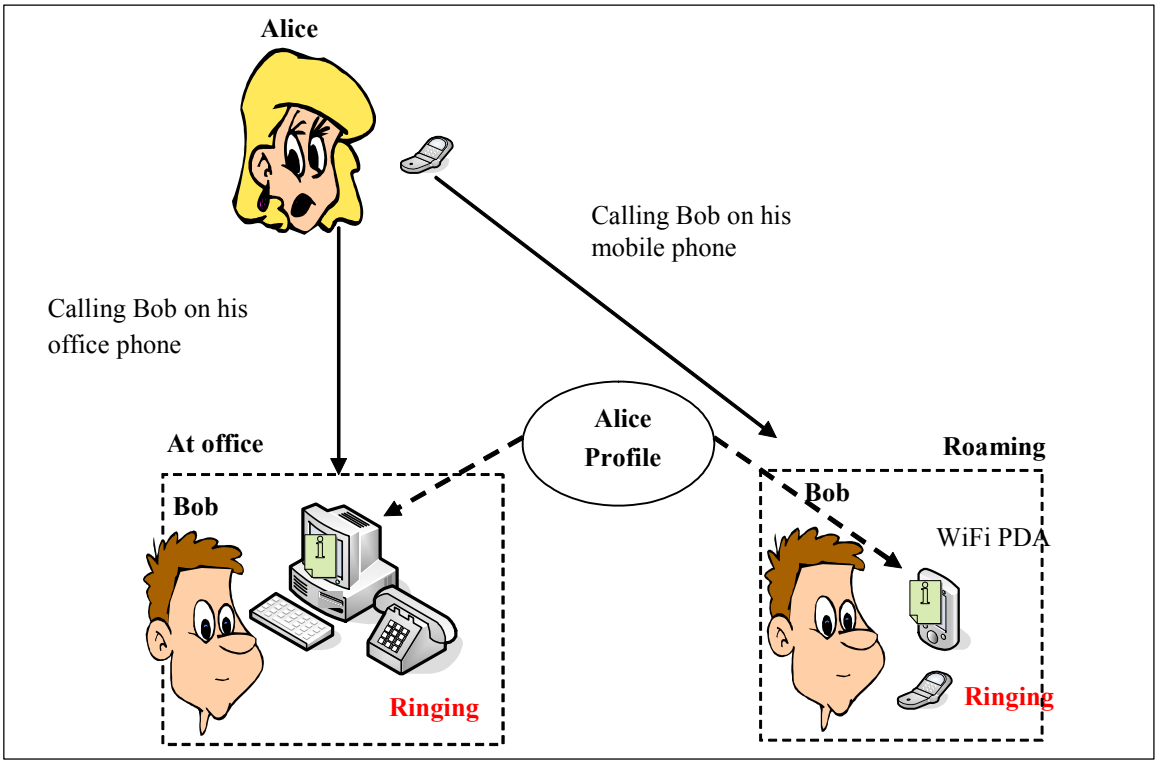
2.2.1 Difficult multi-provider and cross-network services

We have defined that a "multi-provider service" is a service built as a graph of independent service components of different nature and executed by different component providers. This is an extension of the Intelligent Network (IN) service idea where IN services are new combinations of the "elementary connection actions" (or service features) constituting the default basic connection control function of the switches. Service features may be viewed as service components and IN services are designed as a graph of such components [8]. We may generalize this view to a new idea of combining network functions or components with other components to provide a richer service. According to this generalization, a "multi-provider service" may be designed as a composition of multiple service components hosted by different service providers over the network.

We have already considered the example of a car manufacturer Virtual Private Network (VPN) service. In addition to VPN functions that may be supplied by an intelligent network service operator, financial components supplied by a banking company and inventory management components supplied by a specialised provider have been included. Such a service integrating service components from different suppliers in a single user interface is a "multi-provider service". The SIMPSON model has given us a representation of this generalized unbundling of services client sub-services, provider (integrator) sub-services and component sub-services.

To analyse now the cross-network services, let's consider another example where Bob is an employee in a bank call centre. He has a banking portal that offers, among other features, a currency display, a market summary and an auction service. Those three service components may be hosted by different service providers over the network. Using web services or other types of middleware, the different service components are invoked

to build a new richer multi-provider service by integrating them into a single customized service, with a single user interface. Suppose, in addition that Bob receives customers calls. If customer Alice calls Bob, Bob gets Alice bank profile as a pop-up screen while his phone rings [Figure 32]. Now imagine that Bob wants to carry on receiving company calls while he is away from his office. If Alice calls Bob again the call will be forwarded to Bob mobile phone and the banking information about Alice is sent to his PDA. In this example, the service that was available in Bob office is now extended across different networks. We call such a service a ***"cross-network"*** service. Cross-network services are considered in PINT [34] and SPIRITS [35] for a limited set of services and in a more general, but very centralized manner by Parlay and OSA.



[Figure 32] A cross-network service example: the customer popup service

Today, signalling paths are missing both for cross network services and for multiple providers inter-working in a single service. Partial solutions do exist: Web Services or other types of Middleware achieve some multi-provider services. These solutions however do not apply to multiple or heterogeneous networks and therefore at present time cross-network and multi-provider services don't really exist.

To solve the multi-provider and cross-network difficulties, our work looks for more cooperative solutions were multiple service platforms would work together. We propose a new signalling protocol that we call the *Generic Context Sharing Protocol* (GCSP)

[PART 5]. GCSP is used by control processes that belongs to different or a same SIMPSON domain to communicate.

2.2.2 Expensive protocol interoperation. Lack of mediation with legacy signalling protocols

Cross-network services require signalling gateways to do the translation from a signalling protocol to another. If we have *N* different networks, each one with a different signalling protocol, the number of gateways needed to connect the N networks is of the order of N^2. This shows that the signalling network interconnection requires expensive development costs in the $O(N^2)$ range. In addition, the design of a signalling gateway itself is a difficult task. Indeed, a signalling gateway is designed according to the state machines of the two signalling protocols. The complexity comes from the behaviour models heterogeneity of the two signalling protocols that the gateway is translating from and to.

We will propose in [PART 6] a new signalling mediation mechanism which allows different signalling mediators to cooperate together in order to find the adequate signalling mediator for the protocol translation. In order to reduce the signalling gateways number, thus their expensive cost, this proposal consists in using the multi-purpose GCSP signalling protocol defined in this work as an intermediate and in converting all signalling protocols to GCSP and then to perform mediation to legacy protocols from the single GCSP. This means that the number of required types of gateways will grow like $O(N)$, instead of $O(N^2)$, leading to lower mediation cost between legacy protocols.

2.2.3 Lack of inter-domain protocol unity

The reduction of signalling protocols interoperability cost and signalling gateways numbers requires that the new control plane should assure an inter-domains protocol unity. We mean by inter-domains protocol unity the possibility for service components belonging to control plane domains (Access, Provider, Transport, Session, …) to communicate without the need of signalling translation. This inter-domain protocol unity would reduce considerably the number of signalling gateways making service components inter-working easier.

To assure inter-domains protocol unity, control processes of the new control plane should communicate with a signalling protocol common to all SIMPSON domains. Since domains are different, control processes which belong to different domains have different local context instance-data structure.

For example a control process of a “MakeCall component” (Transport Component of the unbundling model) could have in its local context as instance-data {startTime, callingParty, calledParty, maxDuration} and a control process of a “login component” (Access Component of the unbundling model) could have in its local context as instance-data: {startTime, subscriberId, location, securityParams}. This example shows that instance-data of different domains are different and that inter-domain protocol unity, requires that a control process in one control domain should be able to understand the instance data structure of a partner process in a different domain. To make this possible, local contexts should have a generic instance-data structure that all control processes involved in a same service instance can understand.

Our work also proposes a generic instance-data structure for the local context that we call the *Generic Context* (GC) [PART 4].

2.3 Specific signalling requirements

After reviewing the cooperative nature of control plane software and the limitations of the current signalling protocols we proceed with an analysis of the specific performance requirements of signalling and of the control plane mechanisms such as the signalling transactions that insure this performance.

2.3.1 Performance

Because it impacts the call set-up delay, performance is a crucial factor that should be taken into consideration, when designing a new signalling protocol. Signalling transactions are the key mechanism that insures an appropriate performance for signalling protocols.

In the PSTN, these delays correspond to ISUP/MTP processing delays and SS7 queuing/propagation delays respectively [36]. Call setup delay, known as post-dialling delay or post-selection delay [37], is defined as the interval between entering the last dialled digit and receiving ringback tone. Another, related, measure is the time between entering the last dialled digit and when the called party phone starts to ring. E.721 [37] recommends average answer signal delays of 0.75 s for local, 1.5 s for toll and 2.0 s for international connections, with 1.5 s, 3.0 s, and 5.0 s as 95% values.

The PSTN network is a robust network with tightly engineered QOS, particularly with regard to call setup delay. This is due to the common channel signalling (SS7) that has prompted many performance studies [38], and a series of ITU recommendations which specify performance targets for signalling transport [39], call processing [40] in addition the signalling network engineering [41] and switch design needed to ensure call setup delay performance.

In the IP based telephony networks, the session establishment, the packets routing and the resource reservation are handled by three different protocols: session establishment is handled by SIP or H.323, the routing protocol may be handled by BGP [42], and the resource reservation may be handled by a protocol such as RSVP. This has a direct impact on extending the call setup delay due to the multiple interactions that can have the three different protocols with each others. Study [43] shows that the TCP connection setup associated with H.323 calls substantially increases call setup delay (over a path with errors that makes TCP to retransmit), even after tuning TCP implementations for

more rapid retransmission. TCP can also increase the call setup delay because of its three way handshaking and the retransmission of the initial SYN packet with retransmit delay of 6 and 24 seconds [44], which was an important issue that made SIP go over UDP [45]. Another factor that could increase the call setup delay is the DNS address-name resolution. In fact the address-name resolution may involve querying multiple DNSs that could be located over distant and heterogeneous IP domains.

The current cross-network services implementation increases the call set-up delay because of the multiple signalling gateways needed to assure the service mobility from one network to another. A phone call from one network to another different network is in reality two phone calls, one from the originating network to the signalling gateway and the other is from the gateway to the terminating network. This will on one side increase the call set-up delay and on the other cause a bottle neck that could be responsible for traffic congestion, a high retransmission rate and bandwidth saturation.

2.3.2 Signalling transactions (Multiple thread control)

To respect the performance requirement of signalling, a signalling protocol should be the least verbose as possible and should use the mechanism of “signalling transactions”. It has been shown in study [16] that signalling transactions are essential to a signalling protocol and increase its performance. The “Signalling Transactions” mechanism is particularly used in communications between control processes that generate a high signalling traffic and need a protocol performance optimisation. The transaction mechanism is achieved by the TCAP in the IN network. In the IN network, two Signalling Points (SPs) can have multiple simultaneous “dialogs”.

The Signalling transactions mechanism is similar to the “Process and Threads” mechanism of operating systems. A thread is a basic unit of CPU utilization and consists of a program counter, a register set and a stack space. A process contains multiple threads that share code section, data section and operating system resources such as open files and signals.

The work of a multitask machine could be organized as a multiple process control with one thread per process or as a multiple thread control (one process with multiple threads)

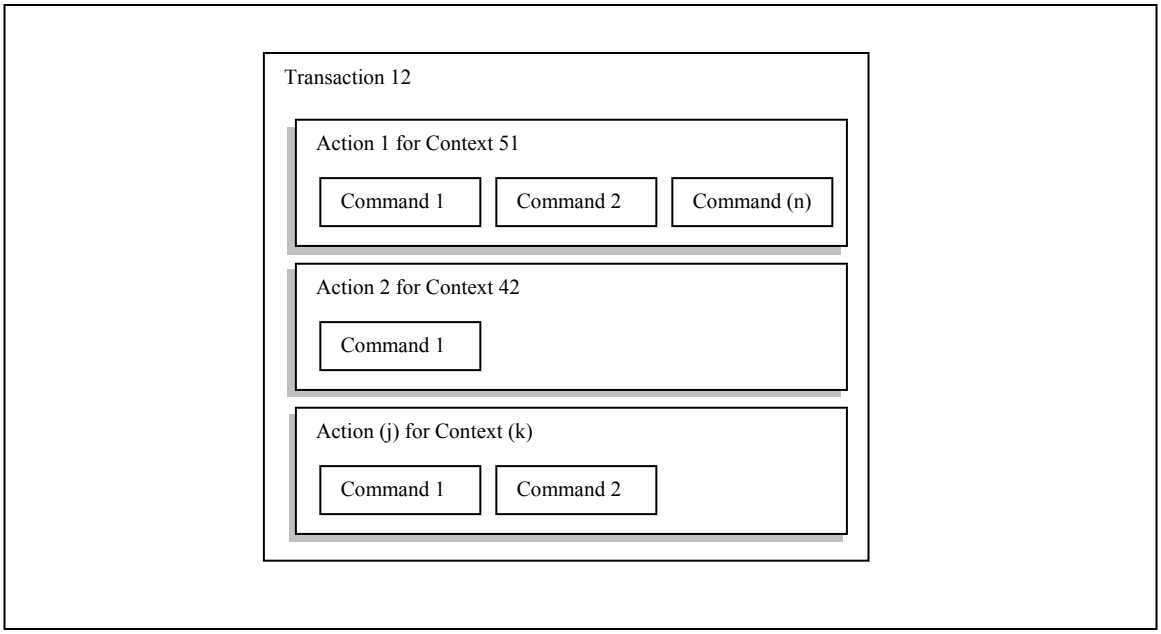
In the multiple process implementation, each process has its own memory and file resources. One multi-threaded process uses fewer resources than multiple redundant processes, including memory, open files and CPU scheduling.

Multiple process control takes a lot of overhead in context switching while multiple thread control greatly reduces the context switching overhead, increasing tremendously machine performance. For example in operating systems network daemons initially written as multiple processes have been rewritten in kernel threads to increase greatly the performance of network server functions [46].

For performance, multithread control should be implemented in signalling machines. A signalling process is called a "signalling transaction" or "transaction" for short and a signalling thread (actually a signalling call) is called a "sub-transaction". In signalling, transactions encapsulate a set of sub-transactions. Each sub-transaction has a unique identifier and describes a stateful communication formed by a set of commands exchanged between two control processes. All the commands exchanged between two control processes that refer to the same communication, have the same sub-transaction ID

The emblematic protocol that implements multithread control in signalling machines is the Transaction CAPability protocol TCAP of SS7 Common Channel signalling. A signalling process is called a "signalling transaction" or "transaction" for short and a signalling thread or sub-transaction (actually a signalling call) is called a "signalling dialog" or "dialog" for short, and the "dialog" commands are called "components". A "dialog" is known to be stateful, it is set-up by a TC_BEGIN and explicitly released by a TC_END. Like all stateful communications, a "dialog" has a unique identifier that identifies it during the "dialog" session. To optimize the protocol performance, multiple "dialogs" can be encapsulated into a single TCAP transaction. A transaction has also a unique ID and is set-up by a TR_BEGIN command and explicitly released by a TR_END. On the other hand, MEGACO also has defined its own description of transaction; a sub-transaction is called an "action", it targets one Context (or local context) and contains one or several "commands", and several "actions" are encapsulated into a single transaction [Figure 33].

The famous SIP protocol doesn't support "signalling transactions" in the meaning of multiple thread control that we just explained. However, by an unfortunate choice of vocabulary, the SIP protocol also uses the word transaction with an other meaning, to represent the complete exchange occurring after one request until its final response. To distinguish "SIP transaction" from "Signalling transaction" we will always state the prefix "SIP" in front of the word "transaction". In fact a SIP transaction is one request and its responses, "SIP transactions" have been implemented in the SIP protocol stack to provide reliable messages transmission. Indeed since the transport layer of SIP relies on UDP, it was necessary to design the SIP transaction layer which is responsible for retransmission, timeouts management and messages filtering.



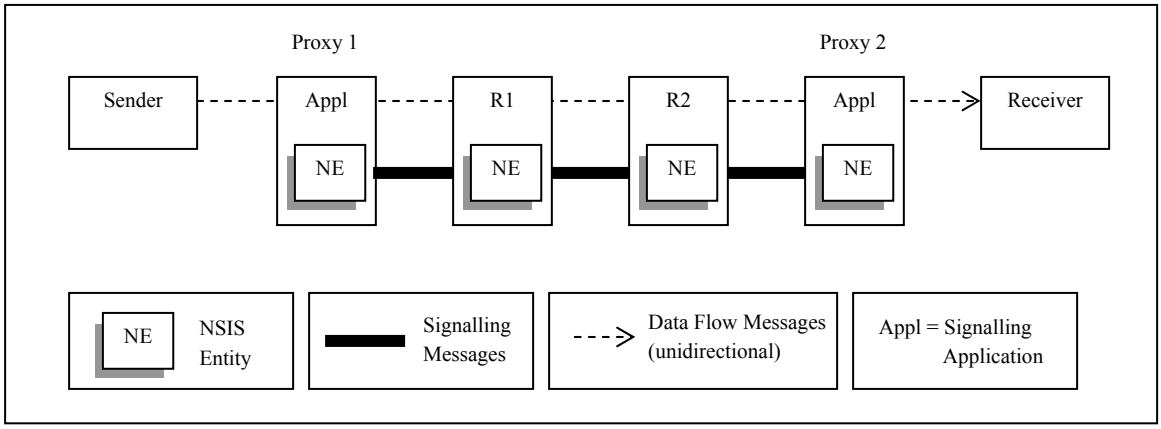
[Figure 33] MEGACO Transaction

2.4 Current research efforts

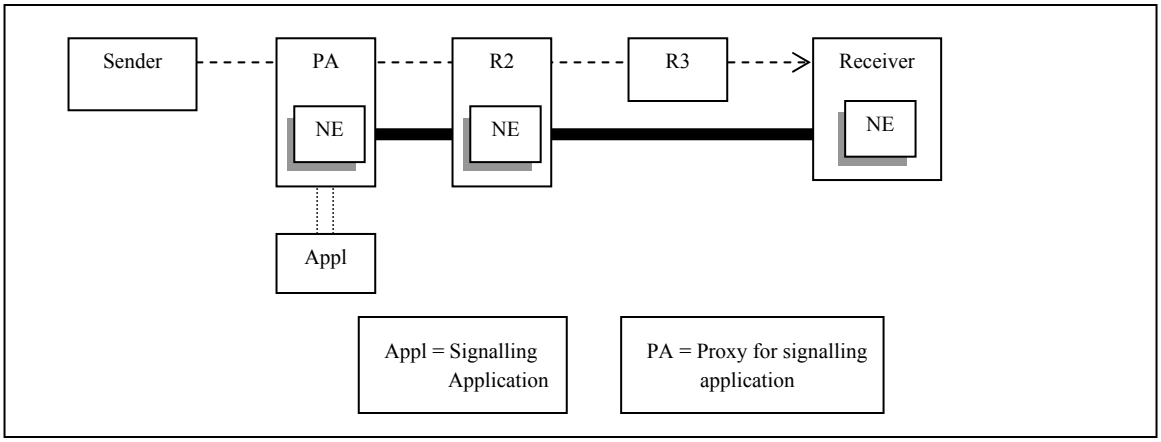
After our careful review of the special role of signalling for cooperative software, of the limitations of the current signalling methods and of the specific requirements of signalling we give a short analysis and a brief description of the main characteristics, concepts and limitations of the current research efforts in the area of signalling.

2.4.1 NSIS

The IETF Next Steps in Signalling (NSIS) working group defines requirements for signalling across different network environments [47], such as across administrative and/or technology domains, and an architectural framework [48] for protocols for signalling information about a data flow along its path in the network. The framework defines a Transport Layer and a Signalling Layer. The transport layer is a robust layer that will assure the transport of application signalling in a similar manner as the SS7 network provides a transport layer for ISUP, MAP and INAP signalling protocols. The NSIS Transport Layer separates the control plane from the user plane thanks to NSIS Entities (NEs) (comparable to SS7 SPs) which are responsible for the routing of signalling messages. In addition path-coupled [Figure 34] and path-decoupled [Figure 35] signalling is possible through the NEs network and messages delivery and reliability is assured by RSVP.



[Figure 34] "On path" NSIS proxy



[Figure 35] "Off path" NSIS Proxy

The NSIS Transport Layer aims to provide congestion management and reliability mechanisms however the approach is still narrow and the NSIS working group does not provide yet details about those mechanisms. In addition the NSIS working group has to study deeper the interaction between the Transport Layer (NTLP) and the Signalling Layer (NSLP). Indeed, study [49] shows that message overlapping between the call signalling and the resource reservation signalling is not an easy issue and it is necessary and useful to separate the call and resource reservation signalling in Internet telephony. Performance can be affected by the call set-up delay that may be increased because of the overheads generated by RSVP on the transport layer and the NSIS router and Entities (NEs).

While most of the efforts are concentrated on the NSIS Transport Layer, there is still a lot to be done on the signalling applications layer. The NSIS framework provides a quick sketch about QoS signalling applications, however signalling applications for conversational services are not yet considered. While multi-provider services are easier to achieve in the Internet network, the NSIS signalling layer does not study the cross-network services problem and how to interact with heterogeneous networks.

2.4.2 IMS

The IP Multimedia Subsystem (IMS) is a control plane associated with a unified NGN packet based transport plane. It introduces multimedia session control in the packet-switched domain and at the same time brings circuit-switched functionality in the packet-switched domain [50]. The IMS aims to standardize an access-independent IP-based architecture that inter-work with existing voice and data networks for both fixed (PSTN, ISDN, Internet) and mobile users (GSM, CDMA). The IMS architecture makes it

possible to establish peer-to-peer IP communications with all type of clients with the required quality of services. In addition to session control the IMS architecture also addresses functionalities that are necessary for complete service delivery like registration, security, charging, bearer control and roaming.

The signalling protocol selected for the IMS core network is the SIP protocol. Signalling is handled by Call Session Control Functions (CSCF) (Proxy-CSCF, Interrogating-CSCF and Serving-CSCF). Performance and reliability issues are under study however the robustness of the SS7 network is not yet reached. To provide network services, the IMS core network defines a set of Application Servers (AS) that are similar to a SCF in the IN network. The interface between a CSCF and an AS is the IMS Service Control (ICS) and the selected protocol is also SIP. To provide cross-network services the IMS has recourse to gateways. The IMS defines three gateways (AS servers), the SIP AS, the OSA Service Capability Server (OSA SCS) that enables services providers to offer OSA or Parlay services, and the IM-SSF (IM-Service Switching Function) to interface with Camel services environment. The S-CSCF, like a CCF in the PSTN, has trigger points and a call model. When a trigger is matched, an event notification is sent to the AS that handles the execution script of the service. However since the protocol selected for the ISC interface is SIP, interaction between the S-CSCF and the IM-SSF is not easy because the communication between the two nodes should be stateful. In the IN network, an association is assured by TCAP for each dialog between the CCF and the SCF. This association is important because during the service execution session, the CCF has to communicate with the SCF eventually to inform it about the service end when one the participants hang-up. In addition the communication on the ISC interface is not performing and optimized because SIP does not handles transactions like TCAP to send multiple dialogs through a single transaction. Finally multi-provider services are easy to achieve since all the IMS core network is IP based as well as the Internet network, thus the integration of heterogeneous components of a service is easier and cheaper to achieve.

The IMS aims at being a global control plane and multi-provider services become feasible since all the IMS core network is IP based. However this feasibility assumes that service components are reachable only via the IP network. IMS does not solve the cross network problem for services. Our proposals intend to solve this difficulty and therefore complement the IMS architecture to give it a more general scope.

PART 3 - THE PROPOSAL OF A DATA BASED SIGNALLING PARADIGM

We have now laid down a theoretical framework structuring the control plane activities and analysed constraints for the implementation of signalling activities. In this third part of our work we will find out that the current signalling methods are not optimal and are mostly adapted to heterogeneous networks, therefore not well adapted to the requirements of a global control plane.

To propose a better suited signalling mechanism for multi-provider and cross-network services which can be implemented on all signalling domains, we review in this chapter the various signalling mechanisms that could be used by control processes to communicate and we detail their characteristics with regard to the cooperative computing requirements of the control plane.

We may remark today that signalling is generally understood as the invocation of remote operations or exchange of notifications between local processes of a global control process**.** These operations are network specific and the commands for their invocation are *stateful*, which means that the effect of a command will depend on the state of the remote entity, making signalling conversion between different state machines in signalling gateways, at least a very difficult problem, and probably an impossible problem in the general case. We advocate that the excessive complexity of the control and service plane may be solved by theoretical improvements in cooperative computing and therefore in signalling protocol theory.

In this objective, and after studying the various candidate mechanisms that may be used for signalling, we come to the conclusion that the command based approach is not the most appropriate, thus we propose a change of paradigm in signalling toward a data based approach with a more general definition by which *"signalling"* is the *"writing or reading of information by a local control process in remote parts of the global context"*. This change of paradigm will allow us to improve the creation of multi-provider and cross-network services.

3.1 The various solutions for signalling

The various mechanisms for sharing information between two distributed processes have been classified [51] into three different categories: the ***command based approach***, the ***data based approach*** and the ***object oriented approach***. We underline the characteristics of these different approaches and we analyse their domain of application in the perspective of centralized computing, distributed computing or cooperative computing. This analysis will lead us to propose the best adapted approach to satisfy all the signalling and control plane constraints that we have identified earlier.

3.1.1 The command based approach

3.1.1.1 Description

In the ***command based*** approach, processes involved in a service session use a predefined set of commands to share their information. In addition, the local context of a remote process is ***private***, it cannot be modified directly by a command. It is private in a term that only the owner process has the direct rights of reading and modifying its local context. When a process receives a command it performs the requested actions and modifies its local context. In the command based approach, local contexts have identical command semantics.

3.1.1.2 Characteristics

The characteristics of the command based approach can be resumed as follows:

High level of abstraction. The commands used by processes are defined at a high level of abstraction and have well defined semantics, thus a process does not need to be aware of the remote context data structure and data-instances. To enrich the communication between processes, more commands have to be added to the existing set of commands.

Efficient with respect to bandwidth. To achieve an action a control process sends a single command to a remote process, thus command based communication may be efficient with respect to bandwidth.

Blind communication. In the command based approach, when a process sends a command it can not be sure of the response, we characterize the command based

communication as a “blind communication”. Indeed a command based communication requires that the control process which sent the command waits for acknowledgments and responses to know if the command was successfully executed in order to take further actions. Such behaviour is due to the local context privacy which prevents a process from knowing in advance the state of a remote process.

Expensive protocol evolution. Protocol evolution is expensive in a command based paradigm. As the signalling protocol evolves, more commands are added to the protocol stack which makes it compulsory to upgrade all the protocol stacks on all concerned machines; more control problems must be resolved during the design phase, there is less freedom during the operational phase. Dynamic upgrade remains a hard solution. It appears that a command based signalling protocol is built as a bird builds its nest. At design time we have a reduced set of commands and as the protocol evolutes in time, more commands are added to it.

3.1.1.3 Domains of application

The command based approach is better suited to centralized processing because a single interaction may be sufficient to obtain the desired effect rather than multiple GET/SET interactions.

Management applications for example are intrinsically centralised processing application and therefore would require preferably the command based solutions. For example a Network Management Station (NMS) has to manage a wide set of Agents, and make multiple GET and SET on the different agents of the star architecture. This is a very heavy mechanism. It is then advised that the NMS sends commands to the Agents instead of reading their instance-data then setting them using Get/Set mechanism like SNMP [52] does. Indeed in this centralized configuration, the management station has to multiply in an excessive way the Get/Set commands on the various agents. It is shown in [51] chapter 9 that for centralized processing, the command base mechanism would be more efficient with respect to bandwidth, memory and processing power since a single control interaction may be sufficient to obtain the desired effect instead of multiple Get/Set. For this reason many authors favour now a command based approach for centralized management.

3.1.2 The data based approach

3.1.2.1 Description

In contrast with the command based approach, in the ***data based*** mechanism, also called variable oriented approach, cooperating processes in a same service session can read each-other local context and CIDs and thus modify them directly. The local context of a control process is ***public*** for trusted partners that have rights, according to a trust and security policies, to read and modify instance-data of this context. To make this possible, local context should have a specific data structure that all processes can understand. A solution is to organize local context attributes following a tree structure, like an SNMP MIB, or an object oriented structure like the OSI MIB [53]. This tree structure could be stored in a memory page or on a disk. A Management Information Base (MIB) of an SNMP agent is a good example of such data structure. Instead of communicating with a wide set of commands, processes use simple commands like GET and SET to read or modify instance data of a remote process. This communication mechanism is possible if all communicating processes, share a same data structure.

3.1.2.2 Characteristics

The characteristics of the data based approach can be summarized as follows:

Better performance for simultaneous actions. Study [54] shows that with a programming language, it is easier to achieve simultaneous actions by modifying with one request many variables values then using specific commands. This can be generalized on the data based and command based mechanisms. In a data based mechanism a single Set order can modify several variables of a remote context. This is one of the reasons that made SNMP designers use the data based paradigm.

Predictive communication. In the data based mechanism a process can read the state and the behaviour model of a remote process (stored in its local context) and therefore may predict the behaviour of the peer process when modifying its context. In the data based mechanism there is no insecurity that the command could not be appropriate; which is not the case with the command based mechanism.

Cheap and easy protocol evolution. On the contrary of the command based paradigm, It has been shown in [51] chapter 1 that the data based paradigm allows the initiation of the operational phase before the design of all control application has been concluded. This gives more freedom to enrich the local contexts data structure during the operational phase without modifying the existing orders (Get/Set/Notify).

Low level of abstraction. In the data based approach, control functions are defined with a low level of abstraction, thus a control process needs to understand the remote context data structure. To enrich the communication between control processes, there is no need to add more commands, it is enough to enrich the data structure of the local contexts.

Less efficient with respect to bandwidth. Such a data based mechanism has been used by management protocols like SNMP, ILMI, CMIP [55] and NetConf [56]. However, because of the unequal management functionality distribution these protocols are implemented in a centralized architecture and, as we have already mentioned it is shown in [51] chapter 9 that in a centralized configuration this data based approach may be inefficient with respect to bandwidth, CPU, time and memory. Indeed the management station has to multiply in an excessive way the Get/Set commands on the various agents of the star architecture and many authors now favour a command based approach for centralized management.

3.1.2.3 Domains of application

While the data based mechanism has been widely used in network management we have shown that it is not the most suitable mechanism for this type of centralized computing applications. In Centralized computing the inter-process communications are one to many leading to bottle necks and requiring huge processing power in the central point.

However, the data based mechanism is better suited for cooperative computing applications because in cooperative computing the communication relations are one to one and not one to many. There is no central point and no multiplication of GET/SET commands. Since a control application is a cooperative computing application ***we derive that the data based approach is then the advisable choice for signalling***.

3.1.3 The object oriented approach

3.1.3.1 Description

Finally in the ***object-oriented*** mechanism, processes communicate by invoking remote objects located on various machines using the client-server architecture. Examples of this mechanism are web services, CORBA and RMI. Like in the command based approach, the local context of a remote control process is ***private***, it cannot be modified directly. In addition, the local context of the client and the server for a single service session are asymmetrical, they do not have the same data structure.

3.1.3.2 Characteristics

The characteristics of the object oriented approach can be summarized as follows:

Asymmetrical and state-less communication. This object oriented mechanism works well for "distributed processing" but may not be efficient for "cooperative processing". As partners communicate in client-server architecture, the server can not address the client if the client did not initiate the communication; therefore the communication between the client and the server is not bidirectional. A server cannot "push" a data value in a client context.

Blocking or "synchronous" communication. when a client invokes a remote method on the server, the control process is blocked until the results has been sent by the server. This prevents from receiving notifications while the client is waiting for the results of the remote method invocation.

Unequal repartition of the control functions. The object oriented approach, centralises the main control functions on the server. This unequal repartition of the control functions is not compatible with the control plane requirement that all control entities should be equal.

Expensive protocol upgrade. To upgrade the protocol, more functions and classes should be added to the server, this would require also to upgrade all the clients which communicate with the server in order to handle the new upgraded functions.

3.1.3.3 Domains of application

The object oriented paradigm is used by state-less applications that rely on a client-server paradigm and are not adequate for control applications which require association and a stateful communication. The object oriented approach is now popular thanks to the web services. Web services brought an easy mechanism to build multi-provider applications with the constraint of having all components of the application coming from a same network, eventually the IP network.

3.2 Our proposal of a paradigm shift for signalling

3.2.1 The proposal

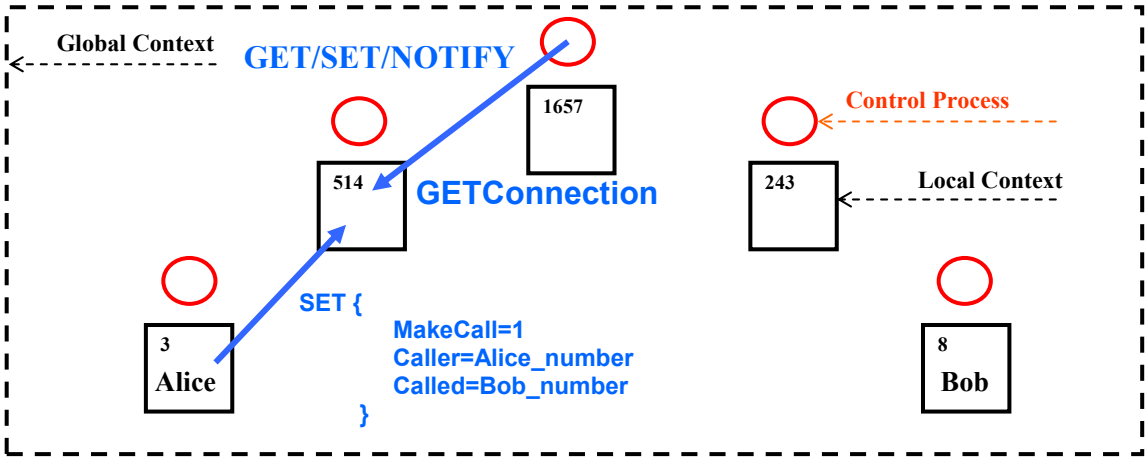
Up to now, the command based mechanisms have been the only method used in signalling. Signalling protocols like SIP, MAP, ISUP and H323 are all based on the command based mechanism.

At this point, we have to realize a very strange paradox: signalling that pertains to a cooperative computing environment is traditionally done by means of a command based mechanism, good for centralized computing. In the meanwhile, management that pertains to centralized computing has an emblematic protocol, the SNMP protocol that uses the data based mechanism good for cooperative computing! The world is upside-down! It should be the over way around: signalling (cooperative computing) should be done with the data based solution and management (centralized computing) should be done with the command based solution! It appears that we are in a situation similar to the pre-Galileo time when people had a paradigm where the sun was turning around the earth!

We therefore propose a necessary change of paradigm for signalling, where the multiple, domain and network dependant, command based signalling protocols with their closed and specific commands would be replaced by a single data based signalling protocol valid for all signalling domains and for all networks.

According to this new paradigm, we define signalling as the writing (or reading) of information by a control process in (or from) a remote local context of the same global context [Figure 36]***. All signalling may then be performed with the unique following requests:***

- OPEN/CLOSE context
- GET/SET variable, value
- NOTIFY variable, new value



[Figure 36] New signalling paradigm

3.2.2 Advantages of the data based approach for signalling in the cooperative computing environment of the control plane

We have already seen, in the previous chapter, that in the general case, a data based information sharing mechanism has the following advantages:

- Better performance for simultaneous actions.
- Predictive communication.
- Cheap and easy protocol evolution.
- Low level of abstraction.

In the cooperative computing situation of the control plane, a single point only addresses a few entities and does not suffer from the bottleneck and increased bandwidth inconveniences of data based mechanisms for centralized architectures. The disadvantages of the data based mechanism being minimal in the cooperative situation, we can then profit, with this new paradigm, of the great advantages of the simpler and more general data based approach.

In the specific case of signalling, we can add the following advantages:

A multi-provider and cross-network service is designed by the association of heterogeneous components from different service providers. Since new heterogeneous components will continue to hit the market, future control protocols and signalling mechanisms should be carefully designed to allow these new components to cooperate

with existing ones, and to allow the initiation of the operational phase before the design of all control application has been concluded. The data based mechanism is well suited for the design of such protocols because of the cheap and easy protocol evolution characteristic, will allow building a more generic interface between heterogeneous components and will provide an easier service implementation and easier cooperation between network and service providers.

Also, in a cooperative computing environment, processes have a behaviour model and a current state. Signalling information will vary with the current state of the destination process, even if the requested service is the same. Such behaviour is also known as context-sensitivity as described in [57] and [58], it enables a software system to adaptively take different actions in different contexts. For example if the bank call centre administrator sends a fire alarm to the phone of all the employee, depending on the phone current state, the system will send a text message and a beep sound to idle phones and a voice message to off hook phones. Data based mechanism is better suited for "context-sensitivity" in control plane applications because it allows to read, with a Get request, the current state of the remote partner process and adjust to this current state by sending adapted information.

We conclude from these arguments that a command based mechanism is the advisable choice for centralized computing applications while a data based approach is a better choice for a cooperative computing application: In non centralized control plane applications data based communication is not a handicap, it offers a generic and simple interface making multi-provider and cross-network services as well as context-sensitive services easy to implement.

3.3 Our approach to define a new signalling protocol

To achieve a paradigm shift for signalling which relies on a data based mechanism we define a generic data structure that we call the ***Generic Context*** (GC) for the Call Instance Data (CID) of the cooperating processes. In a data based approach, a control process needs to understand the data structure of a remote context thus it is necessary to have a generic structure of the local context data that all control processes understand. In order to cover all the signalling domains, we will use the 2-dimensional unbundling and the SIMPSON model to implement this generic structure on all the signalling domains. This will allow an easy implementation of multi-provider and cross-network services.

In addition we define a new signalling protocol, the ***Generic Context Sharing Protocol*** (GCSP), where signalling is achieved by reading or modifying data instances in remote contexts with Get/Set/Notify commands under trust and security restrictions.

The GCSP protocol associated to the generic structure of a GC will assure an inter-domains protocol unity; one protocol for all domains.

Finally to assure mediation with legacy protocols, we define a set of signalling mediators. Signalling mediators translates from GCSP to an existing signalling protocol. However the design of the signalling mediators is slightly different from actual mediation in terms that we propose a cooperative network of signalling mediators that can cooperate to translate a signalling protocol, reducing considerably the number of mediators in the network.

PART 4 - STRUCTURING THE GENERIC CONTEXT

We now come to the definition of a generic data structure for the Call Instance Data (CID) of all the cooperating control processes local contexts. We have found the definition of this data structure necessary for the implementation of our paradigm shift for signalling consisting in using a data based mechanism. We call this generic data structure the ***Generic Context*** (GC).

To facilitate the creation of multi-provider and cross-network services, we implement this generic data structure on all signalling domains. Since a control process needs to understand the data structure of its partner process, the Generic Context, if implemented on all signalling domains, will allow control processes which belong to different domains to cooperate easily. The Generic Context concept is essential for providing an inter-domains protocol unity; control processes of different signalling domains will use a same signalling protocol based on simple Get/Set/Notify commands to read or modify instance-data of a Generic Context.

To describe the Generic Context structure, we start by describing the main characteristics of the Generic Context and its main UML schemas and then we introduce a new association concept that we call the Control Allocation Table (CAT). The CAT describes the association between two or more Generic Contexts involved in a same conversational service instance which gives a global view for the service session. Finally we give a

detailed description of the Generic Context data structure and implementations on some SIMPSON domains.

4.1 The Generic Context overview

4.1.1 The Generic Context as a Distributed Shared Memory (DSM)

In this section, we define the Generic Context (GC) as a Distributed Shared Memory (DSM) used for sharing data between control processes that do not share physical memory. According to chapter 18 of [2], instance-data in a DSM can be structured following a contiguous bytes (**byte-oriented**), a language-level objects (**object-oriented**) or immutable data items (**immutable data**). We propose an object-oriented structure for the Generic Context instance-data. While the byte-oriented approach offers a low level of abstraction and makes it difficult to read and organize the instance-data schema, the immutable data approach, also known by Tuples (Linda, TSpaces, JavaSpaces …), does not offer an organized hierarchy of the instance-data which makes it difficult to read and enrich the Generic Context schema. In addition using Tuples is not easy to make the difference between two variables that have the same name but belong to different categories. For example a MakeCall variable in the Generic Call Control object is different from the one in Multi-Party Call Control. On the other hand, the object-oriented approach offers an organized structure of the objects and takes profit from the object-oriented paradigm advantages such as inheritance and objects protection. With the object-oriented approach it is possible to access a variable of an object by specifying its path (object1.object2.variable9) which makes it easy to distinguish between two variables that belong to different objects. In addition, the object-oriented approach allows reading and modifying a whole object instance-data with one command which is optimal to the protocol performance.

The data structure of the GC is designed according to an object-oriented approach, as done in the Common Information Model (CIM) defined by ITU. However two control applications that use GCSP communicate with a data based mechanism; there is no remote method invocation like in an object oriented communication. Communicating processes use Get/Set/Notify commands to exchange data as in SNMP. While SNMP MIB has a hierarchical tree view of all managed objects; the simplicity of a MIB prohibits defining more complex data and expressing relations between data elements. The GC offers a richer syntax for representing control information and relationships between control objects. An object-oriented approach allows for a greater flexibility in the design of the GC. It also allows for better reusability as the schema becomes bigger and richer. MIBs on the other hand do not allow the same degree of reusability since they

do not support inheritance and so allow the addition of duplicated schema entries as models grow up to support more vendors and more device/application types.

One of the differences with the MIB concept of Management is that a GC is opened at the session initiation and erased from memory at the end of the session. It is not stored in a database and persists only during the communication session. To read or modify Call Instance Data in a remote GC, a process should be trusted and should have enough access rights. This issue is taken into account in the GC schema structure, thus we have defined a security object that represents authentication, access rights and encryption instance data.

4.1.2 The Generic Context main class diagram

To structure and describe the GC classes and their relationships we used the extended SIMPSON model [Figure 37] and the Unified Modelling Language (UML).

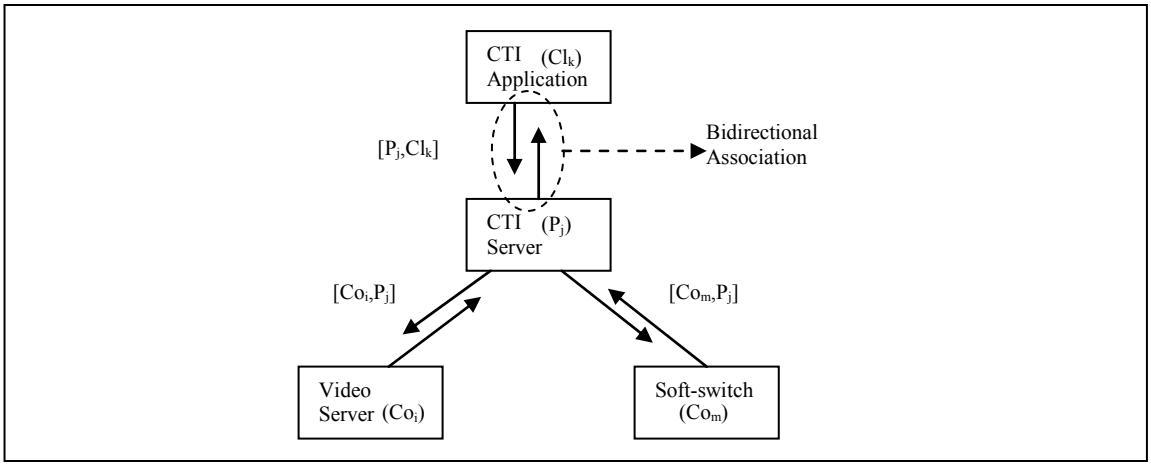
Access	Transport	Intelligence	
Access Client	Transport Client	Intelligence Client	**Client**
Access Service Provider	Transport Service Provider	Intelligence Service Provider	**Service Provider**
Access Component	Transport Component	Intelligence Component	**Component**
Access Session	Transport Session	Intelligence Session	**Session**
Access Bearer	Transport Bearer	Intelligence Bearer	**Bearer**
Access Media	Transport Media	Intelligence Media	**Media**

[Figure 37] The extended SIMPSON model

[Figure 39] and [Figure 40] describe the UML model of the Generic Context schema. Call Instance Data in the Generic Context are organized by the GenericContext main object and a set of objects relative to each service type defined according to the extended SIMPSON model. However, during a service session, not all objects are instantiated for a participating process but only objects relative to the service type which the process functions represent.

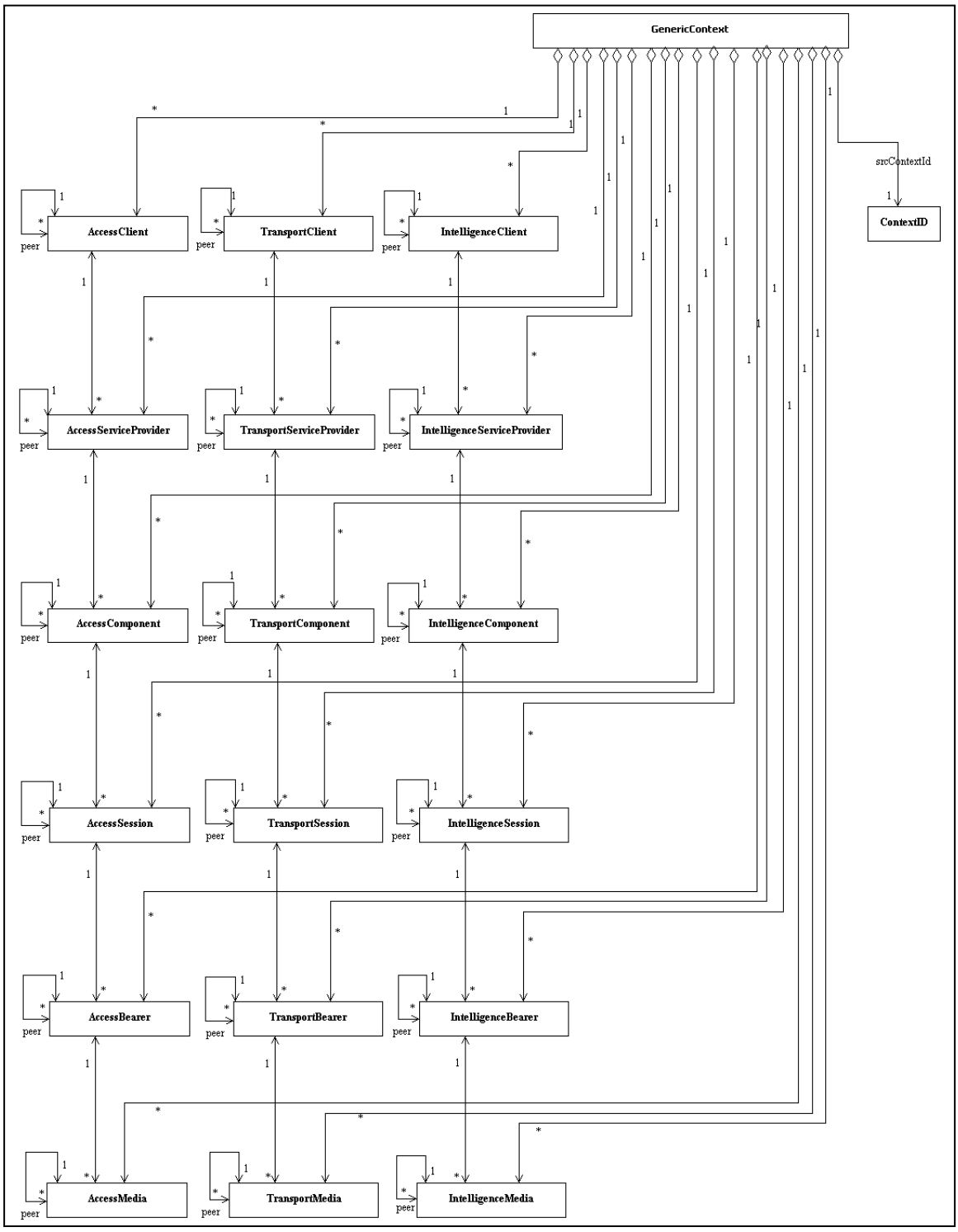
We give as an example a video conference service [Figure 38], between Alice and Bob, started from Alice Computer Telephony Integration (CTI) application. Alice looks for Bob's name in the address book of her CTI application, and then clicks on the video conference button to start a video session with Bob. The CTI application (client) sends a request to the CTI server (service provider) to start the video conference session. The CTI server sends Alice and Bob phone numbers and IP addresses respectively to a soft-switch and to a video server to start voice and video sessions. According to the SIMPSON model, Alice CTI application is an Intelligence Client application thus in its Generic Context the IntelligenceClient object is instantiated (as well as the GenericContext object) while the rest of objects remains empty. In the CTI server, the IntelligenceServiceProvider is concerned; it holds the Call Instance Data required by the

service provider process while in the soft-switch and video server the IntelligenceComponent parts are used.

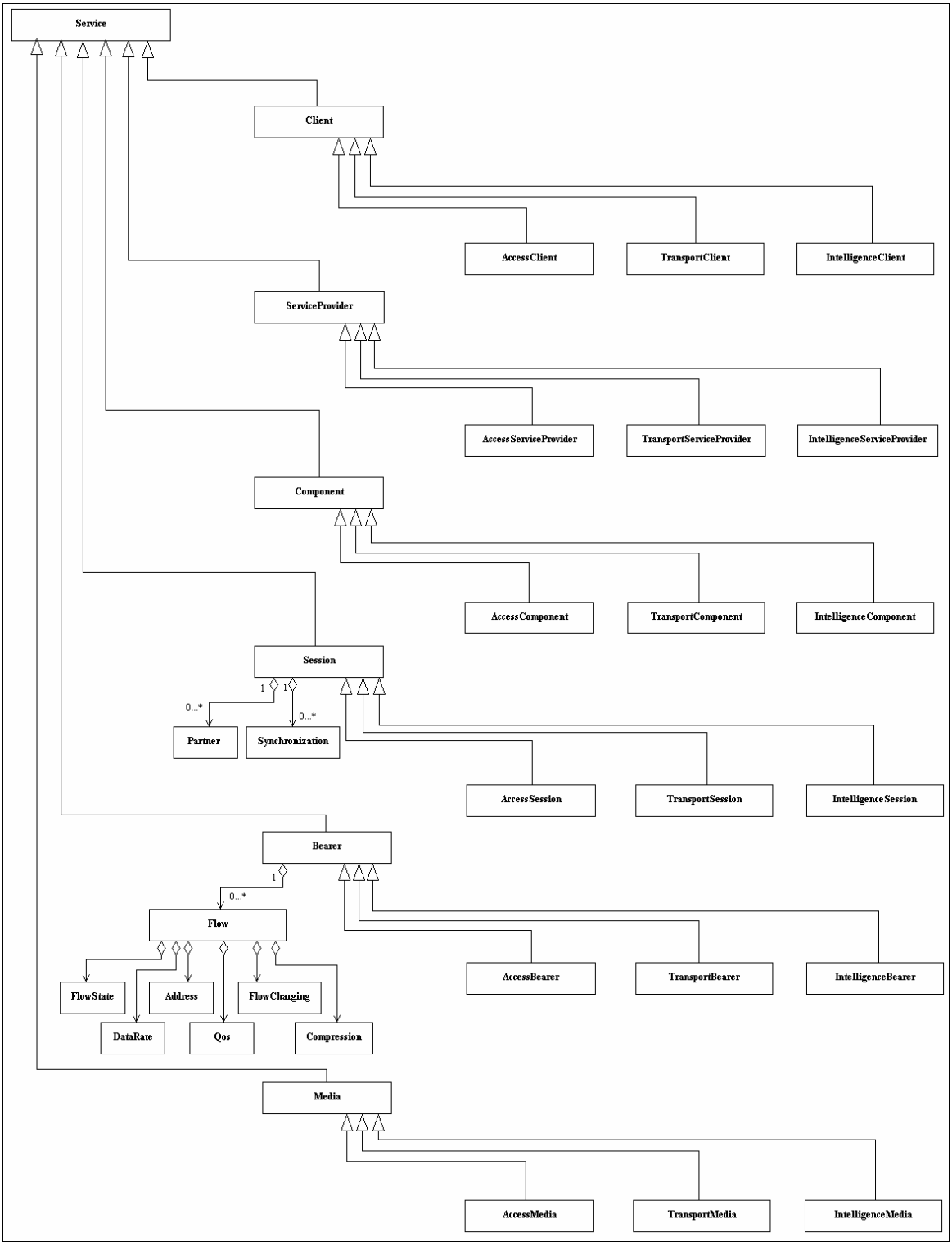


[Figure 38] Video conference association example

All objects in a same SIMPSON level have joint attributes. [Figure 40] shows for instance that on the Bearer level, AccessBearer, TransportBearer and IntelligenceBearer inherit attributes from the Bearer object which in its turn inherits from Service. We will detail the Service object in [paragraph 4.3.1]. However as with SNMP and CIM, each vendor can enrich the Generic Context schema by adding his own objects. New added objects must inherit from one of the main Generic Context objects. The object-oriented approach copes with issues of reusability, complexity and extensibility. As the GC gets extended it will not become incomprehensible. Moreover, depending on what part of control the application software is interested in, it will only need to understand the Generic Context model that has to do with its specific area of interest.



[Figure 39] Application of the generalized unbundling to the Generic Context schema



[Figure 40] Generic Context inheritance structure

4.2 Generic Contexts association

We now consider the association mechanism between local contexts involved in a same service instance. ***We propose a new association mechanism*** that we call a ***Control Allocation Table*** (CAT) and an association object for the Generic Context. The CAT describes the association between Generic Contexts of a service instance and gives a global view of the service session.

In a conversational communication paradigm, control processes need to mutually address each other. A process should have a reference to another process with which it has working relations. This reference, or association, allows a process to send requests and notifications, at any time, to a partner process during a service session. The GC is designed to enable such association between partner processes. In its generic data structure, a process holds an Association object that binds it to a remote process. *In the GC, the association is done at the application level* which allows to design services independently from the transport level (in HTTP the association is done by the TCP protocol).

In our video conference service session example, control applications need to mutually address each other; the soft-switch has a reference to the CTI server in order to inform it of the voice session termination when a party hangs up. In [Figure 38], the CTI server has a pointer to the video server to inform it of the service termination. Association is taken into account in the Generic Context schema. An object at level N of the SIMPSON model can have many vertical associations that bind it to objects from a lower level (N-1) and one vertical association that binds it to an object of the (N+1) level. In our example the IntelligenceServiceProvider object of the CTI server is associated to the IntelligenceClient object of Alice client application and to two IntelligenceComponent objects of the soft-switch and the video server. The association between objects is bidirectional; the CTI server should be able to send requests and receive notifications from the video server. Each local context is identified by its local reference: the soft-switch context is referenced Co_m, the CTI server context is referenced P_j, and the client context is referenced Cl_k… In each context, couples of local references and remote references are maintained and constitute binding references. When a party hangs up his telephone, the soft-switch is notified by the network of the call termination and gets the charging ticket. It finds in its context the binding reference [Co_m, P_j] for the Service Provider and sends him together with this reference the charging value. With the binding reference [Co_m, P_j]**,** the service provider can find the proper context P_j and locate in it the binding reference [P_j,Cl_k,]**,** that identifies the client context Cl_k. In its turn the Service Provider can send together with this binding reference [P_j,Cl_k,] the call cost to the client

local context Cl_k. After the termination of the call, all contexts in implicated nodes are freed.

In legacy networks the association is performed by the Transaction Capability TCAP protocol. In our more general example of [Figure 38] the soft-switch can notify the CTI server that a party has ended his call by going on hook. In the reverse direction the CTI server can notify the soft-switch that a party has ended his call on his browser. This is allowed by the binding references. The soft-switch context has a binding reference [Co_m, P_j] to the CTI server context and vice versa the CTI server context has a binding reference [Co_m, P_j] to soft-switch context. Such a bi-directional persistent binding does not exist in the pure client server mode of communication of the web services.

Let's take an example where the soft-switch and the CTI server would communicate by web services (a pure client-server mechanism) rather than by a persistent ORB like CORBA and where the web services are located in the soft-switch. Then the soft-switch is in the server position and the CTI server is in the client position. If a party ends the call on his CTI application, the CTI server, warned by the client application, can notify the soft-switch with the web services. But if a party goes on hook, the soft-switch cannot notify the CTI server as a server does not maintain a reference pointing to the client that has previously addressed him: A Server is not able to push information into the client (Service Provider).

To solve this problem the CTI server should be able to send events to the soft-switch and the soft-switch to push events into the CTI server. The solution, for an efficient system, is to maintain an association between the two nodes that persists during the full session duration. Such a persistent association is achieved by the concept of binding reference. Peculiar implementations of the binding reference may be the simple TCP socket, TCAP transaction and dialog identifications, CORBA associations. In the web services case, we have to implement web services on both sides (the soft-switch and the CTI server) and to maintain manually the binding references because the web services mechanism is based on the state-less client-server paradigm. This is not an acceptable solution because it is expensive to implement and is not performing because of the additional processing time required for the binding references management and especially because of the XML files length exchanged on the network.

4.2.1 The Control Allocation Table

We present a new scheme for the binding mechanism: Processes involved in a same global service session have to associate their local contexts to build a Global Context for sharing instance data. In the same manner as a File Allocation Table (FAT) links together

sectors of a same disk to build a file, ***a Control Allocation Table (CAT) links together the various local contexts in different platforms to build a global context.***

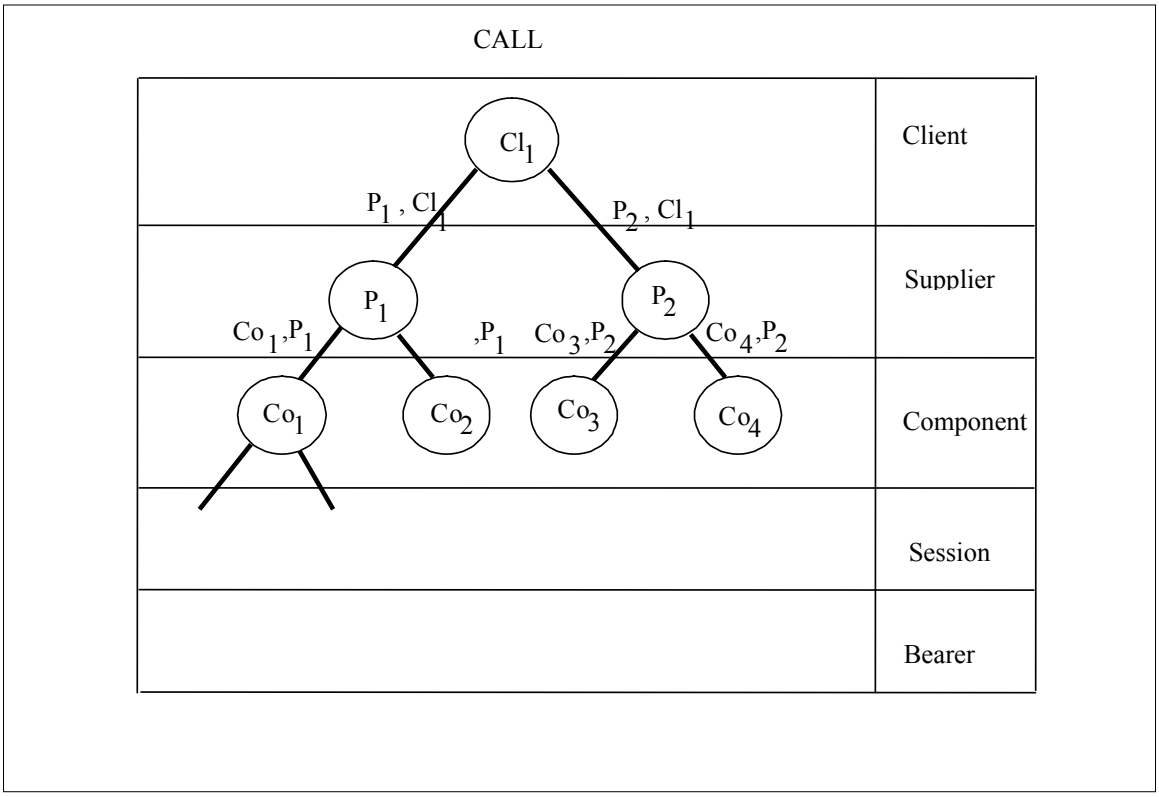
When a process needs to share information (instance data) with a partner process, it gets the binding reference of its peer context from the CAT. The CAT is a binding reference graph, distributed on all the associated contexts [Figure 41]. The CAT is a data structure which persists during the whole session duration; it is created as the contexts are opened and is erased at session termination.

The Vertical Control Allocation Table

The implementation of the CAT graph is achieved when exchanging the initial signalling messages (like the TC-Begin in the Intelligent Network). A difficulty appears when the communicating entities belong to different or a same network with no common signalling protocols.

One way to overcome this difficulty is to relay the exchange of signalling information by means of vertical signalling with an upper level entity in the SIMPSON model. The association graph becomes then a tree structure as shown on [Figure 41]. ***We call such a tree structure a "Vertical Control Allocation Table" or V-CAT which associates local contexts from different SIMPSON levels***. On [Figure 41], local contexts on the level N are represented by a circle and the indexes are the session identifiers that allow building up the binding references between local contexts from the Nth level to N-1 or N+1 level.

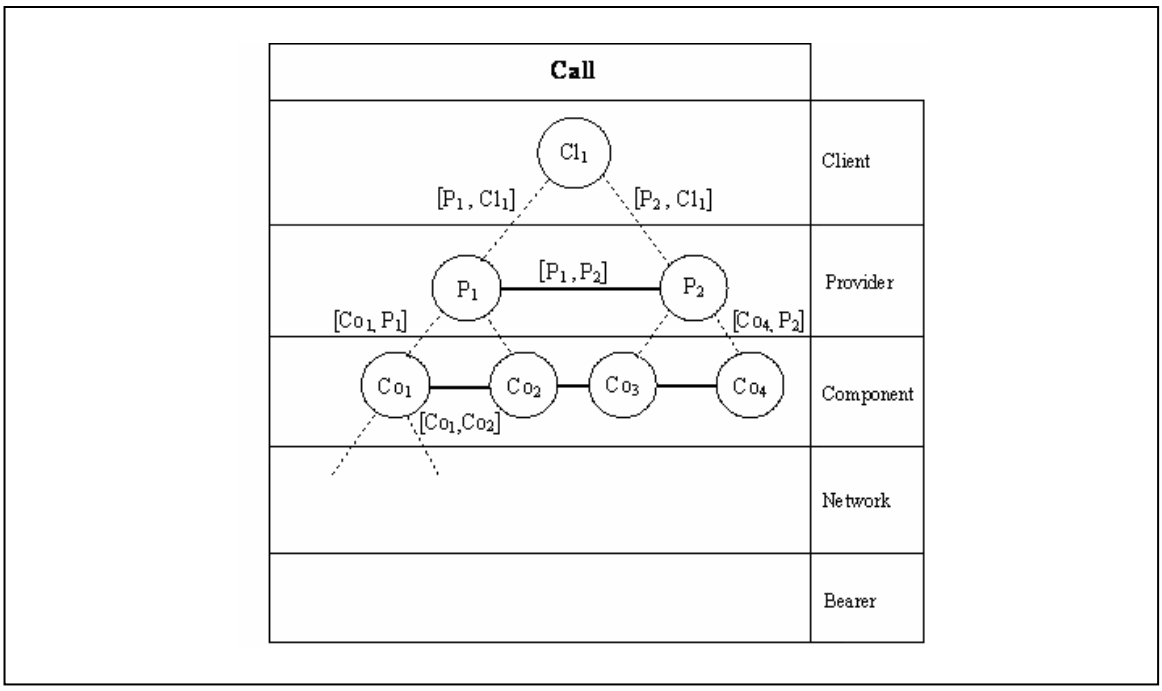
In our previous CTI example [Figure 38] the Video server and the Soft-switch belong to a same SIMPSON level (Component level) and a same network, the IP network, but have no common signalling protocol. Thus the CTI server which is on an upper level (Provider level) relays the exchange of signalling information. It holds an association reference to the Video server local context and another reference to the Soft-switch local context.



[Figure 41] SIMPSON view of a Vertical CAT

The Horizontal Control Allocation Table

This V-CAT structure has the ability to handle multiple heterogeneous networks and multiple heterogeneous entities. However the systematic relaying by central entities at an upper level may introduce some signalling latency. It seems preferable for performance to achieve direct interconnection, even between heterogeneous networks. ***We call "Horizontal CAT" (H-CAT) an association graph with direct signalling links within a same horizontal SIMPSON level*** [Figure 42]. This raises the problem of incompatible signalling protocols and call models in different networks.



[Figure 42] SIMPSON view of a Horizontal CAT

A new approach to enable horizontal signalling between heterogeneous networks or heterogeneous entities would be to use a generic structure for the local context (the Generic Context) and a unique inter-domain protocol (the GCSP Protocol). The horizontal signalling can then be achieved in a 2-phase mechanism [Figure 43]. The different stages of the 2-phase mechanism are as follows:

Vertical Binding. In the first phase, Generic Contexts of different entities are bound together following a V-CAT.

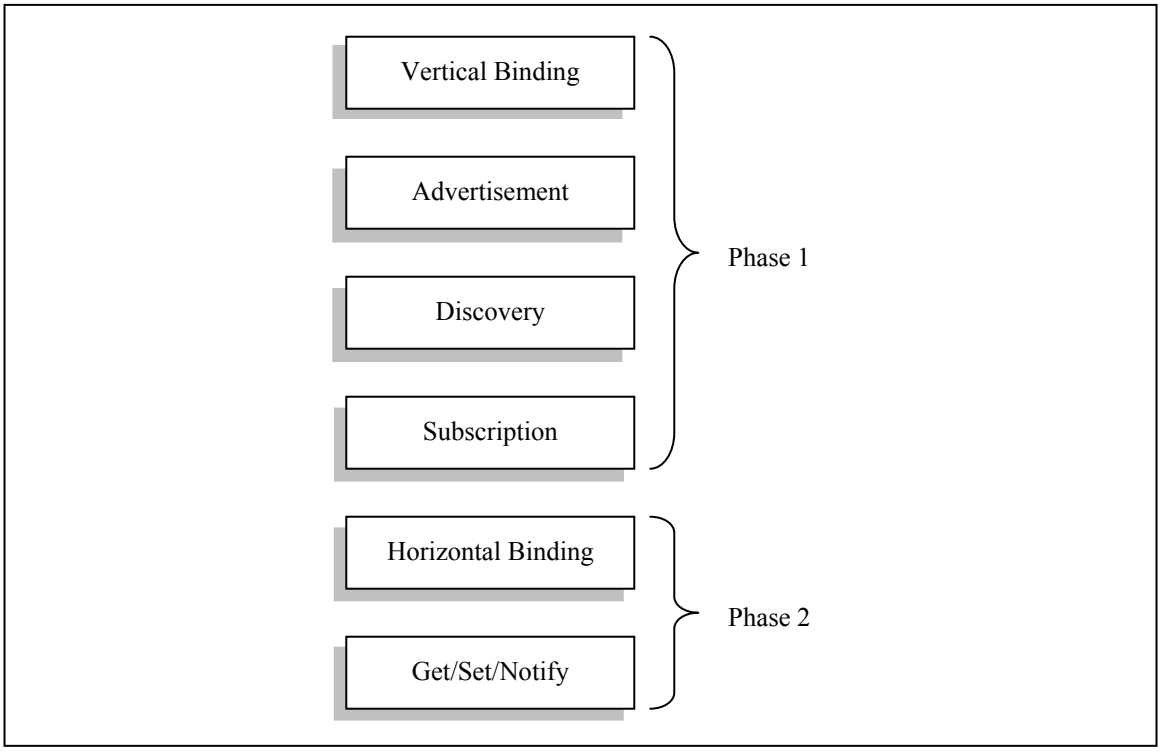
Advertisement. Entities on level N advertise their capabilities to the upper node (N+1) to which they are bound. Such capabilities can be for example, the type of media they handle (video, voice …), the possible actions they offer (start video session, select stream type, end video session, make call, hold, un-hold, transfer, conference …).

Discovery. After advertising their capabilities, entities of level N discover their neighbours capabilities on the same level via the upper entity.

Subscription. To communicate with a peer entity, an entity has to subscribe to the peer entity services according to trust and security restrictions.

Horizontal Binding. An entity retrieves the address of a peer entity from the upper level entity to which it is bound. It then creates a new binding reference directly to the peer entity.

Get/Set/Notify. Entities can now communicate directly without by the upper entity.



[Figure 43] Horizontal CAT build up in heterogeneous networks

In our previous CTI example [Figure 38] the video server and the soft-switch Generic Contexts can be bound together by means of a Horizontal CAT. The signalling messages are then exchanged directly between control processes of the video server and the soft-switch without the need of the CTI server relay. Thus the binding references with the CTI server can be freed. When the Horizontal CAT is built up, the soft-switch can communicate directly with the video server, using Get/Set/Notify commands, to inform it for example of the service termination when a party hangs up.

4.3 Generic Context detailed description

We now focus on a detailed description of the Generic Context schema. We have used the SIMPSON model and the Unified Modelling Language (UML) to structure and describe the Generic Context classes and their relationships. We start by describing the general structure of the Generic context schema. We then detail how the association between control processes is implemented and we proceed with other object classes of the Generic Context and show how it can be enriched with new vendor specific parts.

4.3.1 The Service UML class diagram

As shown in [Figure 44], the Service class is inherited by all level objects of the SIMPSON model. We detail hereafter the different parts of the Service class and their role.

Association class

The association is the key point for achieving conversational services; we have designed it at the application level to be completely independent from transport protocols. The Association class is formed by a pair of source and destination references and addresses (sourceAddress, destinationAddress, sourceReference, destinationReference). The contextID attribute of a Reference class is a unique identifier that allows distinguishing between many Generic Contexts running on a same host machine. On the other side the Address class allows to localize a host machine on the network, it contains one private address and many public addresses. On a host machine, an association object is unique like a TCP socket.

Security class

Authentication and encryption are handled by the Security class. In a same service instance we can have multiple levels of authentication and encryption. For example in our CTI service example, Alice has to give a login and password in order to have access to CTI application functions. After the login phase, the communication can be encrypted between the client and the service provider. On the other hand the CTI server may also need to authenticate with the soft-switch and to use an encrypted communication.

Charging class

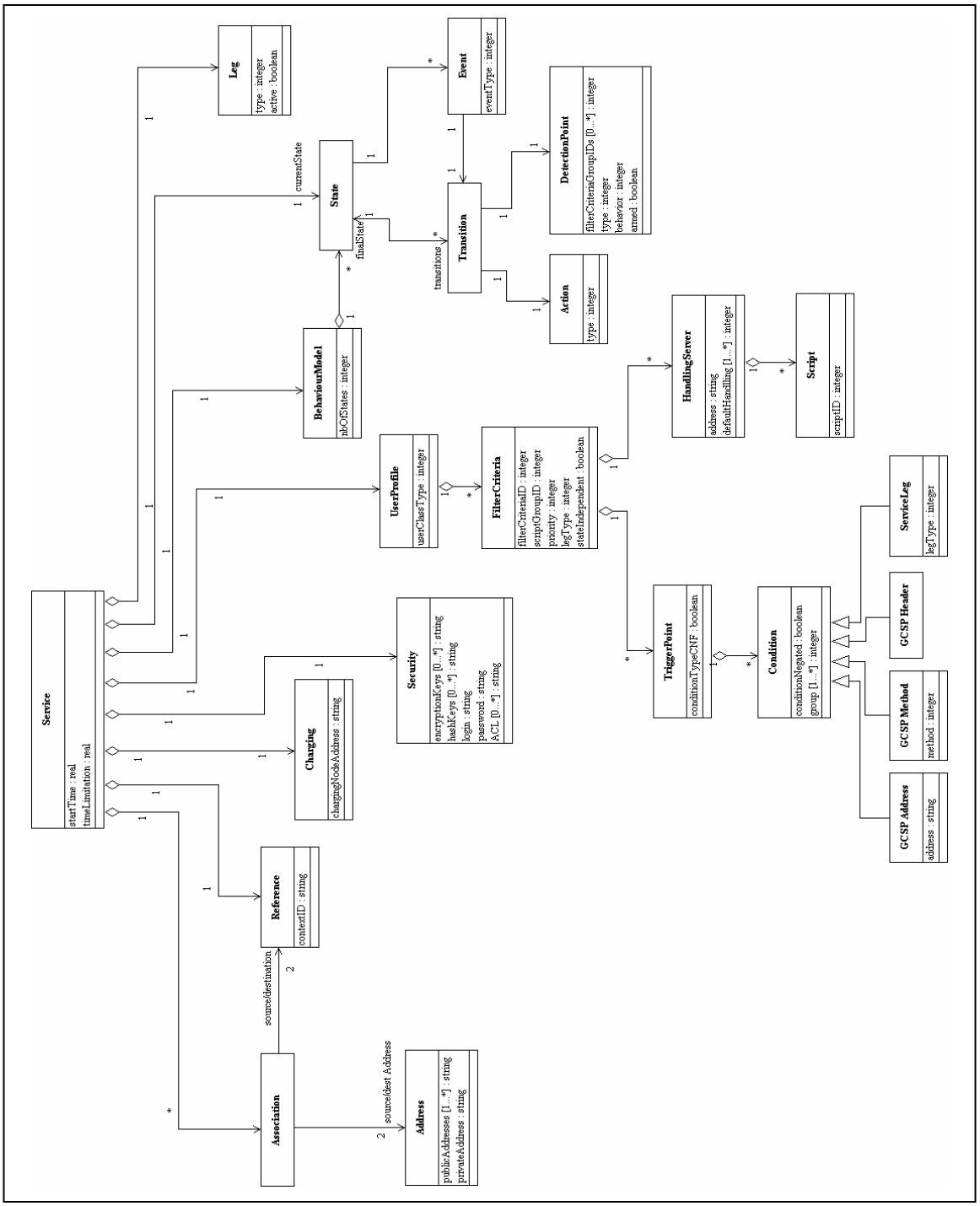
A service can be charged on many levels of the SIMPSON model. For example in a Blackberry [59] service, a user receives in real time his emails on a special PDA designed by Blackberry. The PDA is always connected to a Blackberry server by a TCP connection over GPRS. When the user receives an email, the Blackberry server is notified by the email server and sends over the permanent TCP connection the email message to the destination user. This service is charged at different levels; first the user has to have an annual license to connect to the Blackberry server (charging is achieved by the Blackberry server which correspond to the service provider level of the SIMPSON level) then he should pay a monthly GPRS subscription and there will be a charge for the data volume of sent and received emails (achieved by SGSN and GGSN GPRS nodes which correspond to the component level of the SIMPSON model).

BehaviourModel and State classes

Each service has a behaviour model described by a finite state machine and transitions between those states. In order to predict a remote process behaviour when modifying its Generic Context, control software needs to know in which current state is the remote process and what is his behaviour model. Thus we define in the Service class a currentState and a BehaviourModel parts. The BehaviourModel class contains one or many states. A state is linked to other states by means of transitions. On the reception of an $Event_i$, control software changes its state from $State_j$ to $State_k$ after accomplishing an $Action_m$. A transition between two states is unique; it is defined by an initial state, a final state, an event and an action. In addition, the Transition class has a DetectionPoint, like in the Intelligent Network Basic Call State Model (BCSM) [60], which, if armed, contains a filter criteria ID for further specialized script execution. The “behaviour” attribute in the DetectionPoint class indicates whether the call should be suspended, interrupted or resumed. Detection points can be found in all behaviour models at all SIMPSON levels. An example of detection point at a Service Provider level is when Alice logs to her CTI application, the CTI server notifies her by a popup window with a list of her missed calls. On the other hand, when a GPRS user starts his mobile phone, a detection point is triggered in the Service GPRS Support Node (SGSN) (Component level of the SIMPSON model) which sends him back his waiting messages (SMS, MMS). Finally in the POTS service, detection points are located in the Service Switching Function (SSF) of a local exchange (Session level).

UserProfile class

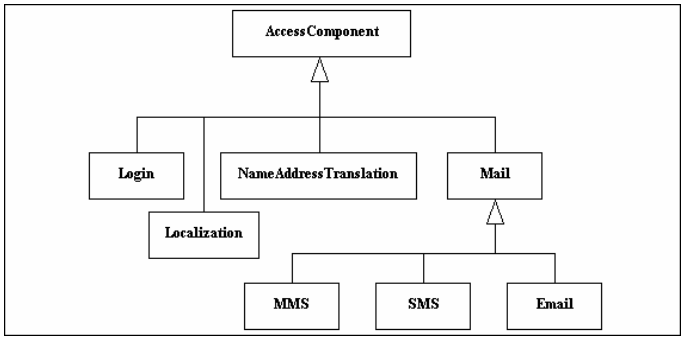
The UserProfile class describes the user category (VIP, normal…). It contains also filter criteria of the user that executes the service. A filter criterion is a mask which, if it matches certain conditions, notifies a specialized server to execute one or many scripts. A filter criterion has an ID and a priority ID in order to organize the matching priorities of many filter criteria for a single detection point. A filter criterion is state independent if the "stateIndependent" attribute is set to true. In this case, when the filter criterion is matched the conditions are verified regardless from the service state and behaviour model. A filter criterion has zero or many trigger points. A trigger point, as defined in IMS architecture [61], has a set of conditions combined with Oring and Anding statements. A condition may take place on a GCSP address, a GCSP method, a GCSP header or a service leg type (active/passive, originating/terminating). When a trigger point is matched a notification is sent to the handling server with the script ID to execute. If the handling server is not found, the HandlingServer class contains the default handling script IDs that can be executed locally.



[Figure 44] Service UML class model

4.3.2 The Access Component UML class diagram

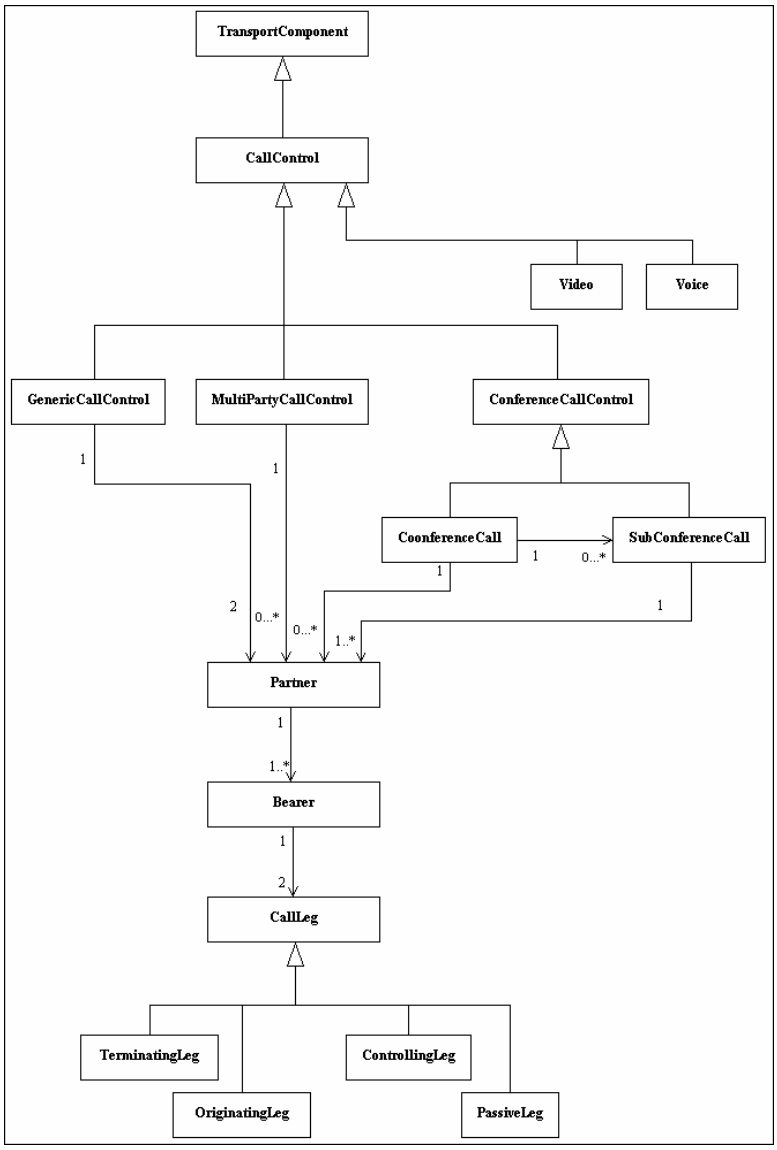
In [Figure 45] we propose access service components that inherit from the AccessComponent class. Originating access services like Login and Mail are generally invoked when the user connects to the network. In GPRS a login is performed by a GPRS_ATTACH. On the other hand, Localization and Name Address Translation are terminating access services. Vendors can enrich the Generic Context schema by adding their specific parts and inheriting from existing objects. The object-oriented approach allows flexibility in the design from which vendors can take advantage to cover any control area with a new proprietary model.



[Figure 45] Access Component UML class diagram

4.3.3 The Transport UML class diagram

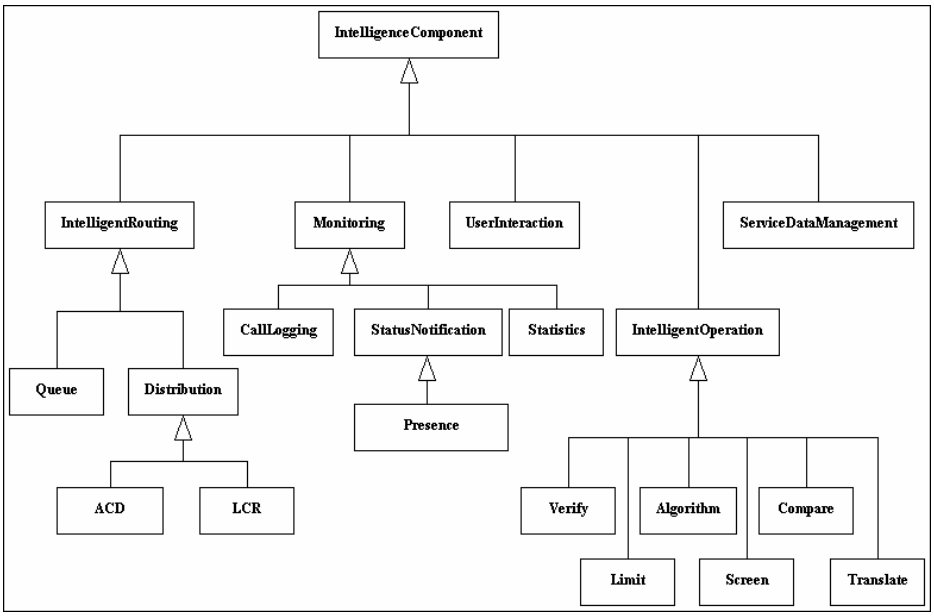
In this section we propose a call control model for multimedia communications. In [Figure 46], a CallControl object is a transport component that covers voice and video sessions. To define transport call control classes we have used the Parlay standardization work on call control [62]. A two party call is represented by a GenericCallControl instance object, while calls between multiple parties are represented by MultipartyCallControl and ConferenceCallControl objects. On the session level we define the Partner object class that represents a party involved in a call. It is possible that a CallControl object has no partners at service initialization and that the partners are added progressively. The Call (association between Partners) has an end to end significance while the Bearer has a link by link or hop by hop significance. A partner requires from 1 to many bearers (links) to be setup and the establishment of each bearer requires a connection model with two legs.



[Figure 46] Transport UML class diagram

4.3.4 The Intelligence Component UML class diagram

A service using transport and access components can be enriched by "intelligence" components (Intelligent network services or Application Servers functions). We follow a method similar to the IN CS1 Service Independent building Blocks (SIBs) [60] to propose intelligent network components using our experience in Computer telephony Integration. [Figure 47] gives a description of these service components and their relationships.



[Figure 47] Intelligence Component UML class diagram

PART 5 – THE GCSP PROTOCOL (GENERIC CONTEXT SHARING PROTOCOL)

In the previous chapter we have proposed a generic structure for the local context instance-data which can be understood by any control process that belongs to any signalling domain. In order to read or modify a Generic Context instance-data, we propose a new data based signalling protocol, the Generic Context Sharing Protocol (GCSP). GCSP is a text based protocol which offers to control processes a simple and reduced set of commands (Get/Set/Notify) to manipulate Generic Context objects. This signalling protocol insures an inter-domains protocol unity and thus reduces the number of signalling gateways needed for translation between different signalling domains.

We now describe and detail the GCSP protocol. We start by an overview of the GCSP protocol and discuss its benefits and advantages, then we proceed with a detailed description of the GCSP commands and frames.

5.1 Objectives

GCSP signalling protocol is used to share, modify and exchange generic contexts parts over the network between two remote control processes. GCSP being data based oriented is more adapted to cooperative computing and thus better suited for control plane software than command based signalling protocols. The Generic Context Sharing Protocol is:

- Easy to use: GCSP is a text based protocol and uses simple Get and Set commands to read or modify generic context data to trigger remote operations as in the SNMP paradigm. Control processes can also exchange Notify commands.

- Data based: as we early stated, control software need information sharing. With GCSP a control process can read the current state of a partner process and its behaviour model, before modifying its generic context, and thus can predict its future state. Even though the generic context has an object oriented structure, GCSP does not use the same mechanisms as used by Object Oriented Middlewares. There is no client server architecture and no remote method invocation with GCSP. Processes using GCSP work on a cooperative basis, without any centralized point, and modify locally a remote generic context (or specific part of it) after downloading it. Downloading the object is an option in some Object Oriented Middleware like RMI. In general the object is not downloaded and only remote method invocation is possible.

- Efficient: To respect the performance requirement of signalling GCSP should be the least verbose as possible and should allow the use of signalling transactions. Following a Get command a part of the remote generic context is downloaded. When the remote generic context is downloaded we can perform several modifications before uploading it on the remote host. This is equivalent to transactions in MEGACO and TCAP which are essential to the protocol performances [16]. Renowned mechanisms may also be used to increase performance. As in an SNMP MIB, generic contexts objects names can be replaced by numbers to reduce the size of GCSP messages.

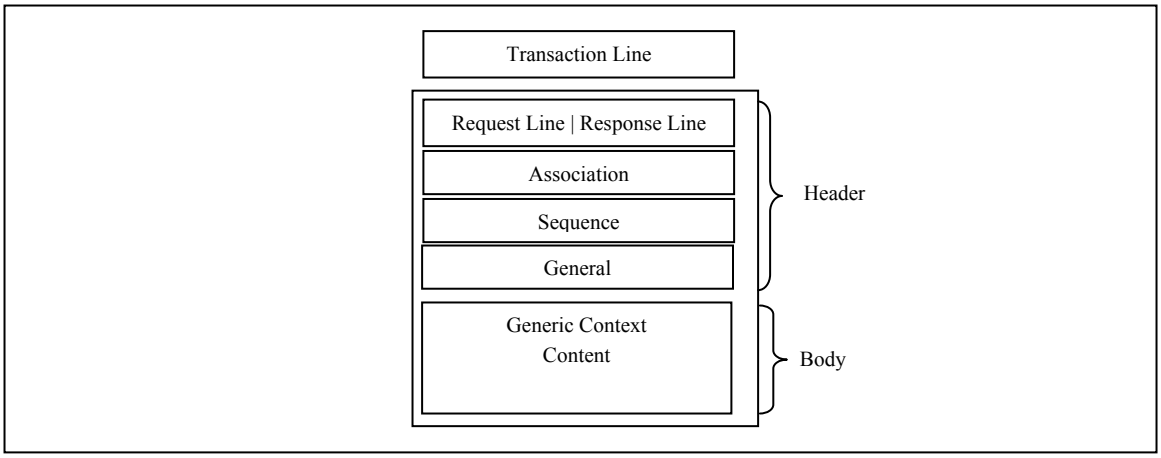
- Flexible: being a data based protocol it is easier to enrich the protocol stack. Instead of adding new commands, as it would be necessary with a command

based protocol, a simple update of the generic context schema is enough to enrich the protocol.

- Secured: security is taken into account in the Generic Context design. A security object handles the user identification, authentication and communication encryption. GCSP can also be encrypted with an SSL layer.

5.2 Overview of the Generic Context Sharing Protocol mechanisms

GCSP is a text based protocol like HTTP and SIP. A GCSP frame consists of a header and a body as shown in [Figure 48]. GCSP is an application level protocol and is independent from the transport layer. In our implementation of the GCSP stack we have proposed to run GCSP over UDP, thus we have taken into account the UDP port number in the GCSP URI discussed in the paragraph that follows. However for security reasons and NAT traversal, GCSP may be transported by TCP.



[Figure 48] GCSP Frame

5.2.1 GCSP URI

To identify a GCSP communications resource (node) on the network, we propose a GCSP URI which has a general form as follows:

GCSP-URI = “gcsp:” [userinfo] <hostport>

userinfo = <user> [“:” password] “@”

hostport = <host> [“:” port]

The “gcsp:” scheme follows the guidelines in [63]. It uses a form similar to the mailto URL, allowing the specification of GCSP request-header fields and the GCSP message-

body. These tokens, and some of the tokens in their expansions, have the following meanings:

- **user**: The identifier of a particular resource at the host being addressed. The term "host" in this context frequently refers to a domain. The "userinfo" of a URI consists of this user field, the password field, and the "@" sign following them. The "userinfo" part of a URI is optional and may be absent when the destination host does not have a notion of users or when the host itself is the resource being identified. If the "@" sign is present in a GCSP URI, the user field must not be empty.

- **password**: A password associated with the user. While the GCSP URI syntax allows this field to be present, its use is not recommended, because the passing of authentication information in clear text (such as URIs) has proven to be a security risk in almost every case where it has been used.

- **host**: The host providing the GCSP resource. The host part contains either a fully-qualified domain name or numeric IPv4 or IPv6 address. Using the fully-qualified domain name form is recommended whenever possible.

- **port**: The port number where the request is to be sent.

An example of a GCSP URI is: gcsp:rony.chahine@192.168.1.9:3030 where:

- User = rony.chahine

- Host = 192.168.1.9

- Port = 3030

5.2.2 GCSP Header

Header lines provide information about the request or the response, or about the object sent in the message body. The header lines are in the usual text header format, which is one line per header, of the form "header-name: value", ending with CRLF. It is the same format used for email and news postings, defined in RFC 822, section 3. We give hereafter a detailed description about the different sections of the header.

The Request Line

The “request line” header is the first line in a GCSP frame; it describes the type of request a process can invoke. Generally, following a request, a response is expected which includes a response code like with HTTP. A GCSP “command” header line is as follows:

command [path:object1.object9.object3] [lock|unlock] [tcp] GCSP/<version>

- The “command” parameter may be of the following commands: Get, Set, Notify, Open-Context, Close-Context and Lookup.

- The “path” keyword is not mandatory; it indicates the path of the object attributes that are expressed in the body. If the “path” keyword is missing it means that the body describes the root object which is the GenericContext object.

- To synchronize and resolve the concurrency on an object, a lock may be set when reading this object before modifying it with a Set command. A control process reads the object instance-data with a Get command and includes the “lock” keyword in the “command” header line to prohibit other processes from modifying it. When the object is uploaded back with a Set command, the “unlock” keyword must be included in the command header line to unlock the object. Any object locked is automatically unlocked after a timeout which prevents dead-locks problems from occurring.

- The “tcp” keyword is optional and indicates that GCSP is running over TCP. If the “tcp” keyword is omitted it means that default protocol, which is UDP, is used.

- Finally the “version” indicates the GCSP frame version.

Hereafter we give a detailed description of the GCSP commands:

- **Get:** a control process can query a remote generic context data using a Get command. The command should indicate the full path of the targeted part. If the query targets the root object (GenericContext), the path is then not necessary and can be omitted. For example, if a process wants to query the Generic Call Control part of the remote generic context it sends the following command:

  Get path:GenericContext.GenericCallControl lock GCSP/1.0

- Following a Get command, a response is expected. The response indicates its status response. If it is 200 OK, the body of the response contains the queried generic context data:

GCSP/1.0 200 OK

- **Set:** after downloading and modifying a remote generic context part a control process can upload it with a Set command. The command must indicate the full path of the targeted part of the generic context as well as the protocol version. The message body encloses the generic context data that should be modified in the remote generic context.

  Set path:GenericContext.GenericCallControl unlock GCSP/1.0

  An object can be set directly without the need to download it first. Only the target variables of the object are filled the other can be omitted from the object. Further details about setting an object will be provided in [PART 6].

- **Notify:** GCSP is a state-full protocol. During a session, control processes can send notifications using a Notify command. Subscription to notification is done via a Set command. For example a Detection Point is armed by a Set command and when a filter criterion matches, the control process sends a notification to the concerned partner process. The Notify header line indicates the object raising the notification and the message body contains the notification data. For example the notification below is sent to an application server after a filter criteria match. The GCSP body contains data relative to the FilterCriteria object which is the script ID to execute and the filter criterion priority:

  Notify path:GenericContext.AccessComponent.UserProfile.FilterCriteria GCSP/1.0

- **Open-Context:** to start a conversation with a remote party, a control process sends an Open-Context command with the Association section filled except the Dest-Context line. The remote process opens a new local context and answers back with a 200 OK response and put the Src-Context in the Dest-Context line and fill the Src-Context with the reference of the new generic context that has been created.

  To reduce the number of exchanged GCSP messages on the network, an implicit open context can be sent into a Set command. In this case the Set request contains an empty Dest-Context. When the remote process receives the Set request it create a new Generic Context and answers back with a GCSP response which encloses the new context ID. This procedure is not possible with a Get command; a Generic Context must exist before sending the Get command.

- **Close-Context:** a conversation is ended with a Close-Context command. Generic contexts involved in the conversational service are closed and freed from memory. After a process receives a Close-Context command, it answers back with a 200 OK (if the generic context is closed) with an embedded Close-Context command to notify the remote process that there is no data to send so latter can close its generic context. After the remote process closes its generic context, it sends a 200 Ok confirmation to the partner process.

- **Describe:** to have richer services, constructors may add their own objects to the generic context schema like with SNMP MIB. This situation leads to heterogeneous generic contexts. In order to know the structure of a new object in a remote generic context, a control process sends a Describe command with the full path of the required object. For example:

  Describe path:GenericContext.IntelligenceProvider.CTI GCSP/1.0

  The answer, if 200 OK, contains in the body an XML file describing the CTI object in the following way:

```
<?xml version="1.0" encoding="UTF-8"?>
   <name>object_name</name>
   <inheritance name=<value> path =<value>/>
   <abbreviation>abb_number</abbreviation>
   <attributes>
       <name=<value> type=<type> abbreviation=<a_number>/>
       <name=<value> type=<type> abbreviation=<a_number>/>
                .
                .
   </attributes>
```

  The <name> tag indicates the name of the object we asked for its description. If the object inherits from another object, the <inheritance> tag holds the name of the inherited object and the path indicates the full path of the object in the form baseObject.subObject1.subObject2... All the attributes are in the <attributes> tag. An attribute has a name, a type (integer, string, boolean, date, array…) and an abbreviation. The abbreviation allows querying the object by its number instead of its name which increases the protocol performance. For example the abbreviation of the Context object is "1" and GenericCallControl is "6":

  Get GenericContext.GenericCallControl GCSP/1.0

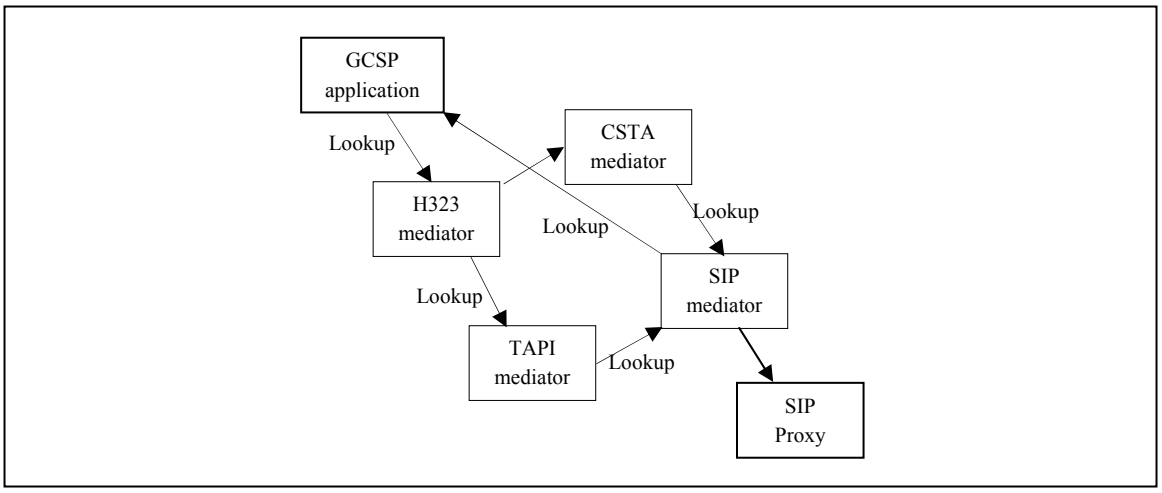
  Would become:

  Get 1.6 GCSP/1.0

- **Lookup:** control processes that implement GCSP can communicate with other control processes that use different signalling protocols via signalling mediators. A signalling mediator is a gateway that translates from GCSP to another signalling protocol. However GCSP mediators can work on a cooperative basis to find the adequate mediator that translates to the required signalling protocol. The mechanism of selecting a signalling mediator among others will be detailed it in a further study. Let us take an example of a GCSP process that wants to communicate with a SIP proxy. The GCSP control application has to find a SIP mediator. The GCSP process sends a Lookup command to any mediator it can reach, for example the H323 mediator, with its address in the source address and the address of the H323 mediator in the destination address. The Lookup command indicates the signalling protocol we are looking for as well as the address of the SIP proxy we want to reach to select the adequate SIP mediator if there are more than one that could be reached:

  Lookup SIP sip:proxy@enst.fr GCSP/1.0
  From: chahine@enst.fr
  To: H323_9287@mediators.gcsp.net
  Src-Context: 63
  Dest-Context:

  The H323 mediator forwards the Lookup query through the signalling mediators' network and the SIP address mediator is sent back to the source process. [Figure 49] shows the Lookup query mechanism.



[Figure 49] GCSP mediator lookup mechanism

When the SIP mediator, which translates from GCSP to SIP, is reached, it answers back to the GCSP application with its address in the GCSP frame header:

GCSP/1.0 200 OK
From: sip_90234@mediators.gcsp.net
To: chahine@enst.fr
Src-Context: 7589
Dest-Context: 63

The SIP mediator acts as a state-full proxy, it has a generic context (ID=7589) bound to the generic context of the GCSP application (ID=63) with an association. All messages sent from the GCSP application to the SIP proxy are forwarded by the SIP mediator which translates from GCSP to SIP and from SIP to GCSP.

The Response Line

The "response line" header is the first line in a GCSP frame; it describes the type of response a process may receive. Generally, following a request, a response is expected which includes a response code like with HTTP. A GCSP "response line" header is as follows:

GCSP/<version> <status-code> <reason-phrase>

- The "version" parameter indicates the GCSP frame version.
- GCSP "status-code" is returned by the server to the GCSP client to determine the outcome of a request. We designate by "client" the control process sending a request and by "server" the control process receiving the request. The first digit of the status-code defines the class of response. The last two digits do not have any categorization role. There are 5 values for the first digit:
    - 1xx: Informational - Request received, continuing process
    - 2xx: Success - The action was successfully received, understood, and accepted
    - 3xx: Redirection - Further action must be taken in order to complete the request
    - 4xx: Client Error - The request contains bad syntax or cannot be fulfilled
    - 5xx: Server Error - The server failed to fulfil an apparently valid request
- The "reason-phrase" is intended to give a short textual description of the status-code. The status-code is intended for use by automata and the reason-phrase is

intended for the human user. The client is not required to examine or display the reason-phrase.

Hereafter we give a summary table of the possible status codes and their reason phrases that may be returned in a GCSP response:

Status Code	Reason Phrase	Description
100	Continue	Tells the client that the first part of the request has been received and that it should continue with the rest of the request or ignore if the request has been fulfilled.
110	In Progress	The request is being processed and there is a delay coming from the remote party.
200	OK	The request sent by the client was successful.
301	Moved Permanently	The resource has permanently moved to a different URI.
302	Found	The requested resource has been found under a different URI but the client should continue to use the original URI.
400	Bad Request	The syntax of the request was not understood by the server.
401	Not Authorized	The request needs user authentication.
403	Forbidden	The server has refused to fulfil the request. Sent for example when the server receives a Set request on a Generic Context that is bound to another client.
404	Object Not Found	The object requested by the client was not found.
405	Attribute Not Found	The attribute requested on an object was not found.
408	Locked	The request was unsuccessful due to a conflict in the state of the object that is locked.
409	Request Timeout	The client failed to send a request in the time allowed by the server. Received when the client does not unlock a remote object before a timeout.
500	Internal Sever Error	The request was unsuccessful due to an unexpected condition encountered by the server.
501	Not Implemented	The request was unsuccessful because the server can not support the functionality needed to fulfil the request.
503	Service Unavailable	The request was unsuccessful to the server being down or overloaded.
505	GCSP Version Not Supported	The server does not support or is not allowing the GCSP protocol version specified in the request.

Association

Two generic contexts of partner processes are bound together with an association. A GCSP association is bidirectional; both processes can address mutually each other. The association section in GCSP header consists of the source (From) and destination (To) addresses (private and/or public), and the source and destination contexts IDs. Many addresses can be sent in the From and To fields separated by a space character. This kind of association is not new; it is described by a socket in TCP. However in GCSP, the

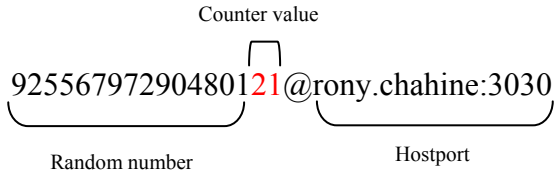
association is done at the application level which allows implementing richer services requiring a conversational communication paradigm independently from the transport protocol. In HTTP the association is done by TCP: when the server wants to respond back to a client, it retrieves the client address from the TCP socket. The From field is not mandatory in Http 1.1 and even when specified in a request it does not allow to distinguish between two service instances running on an same host machine.

[Figure 50] Association section lines in a GCSP frame header

In [Figure 50] we give an example of association between two communicating partner processes. In the From field we find the source addresses, in our case we have two addresses; a public address (chahine@enst.fr) and a private address (400854585532112@enst.fr), while in the To field we find the destination address. The Src-Context and Dest-Context are references of the source and the targeted generic context. A context reference is unique within a single machine like a TCP port. To generate a context id we propose to build as following:

<random number><counter value><@><hostport>

The counter value is incremented each time a new Generic Context is created. Here is an example of a context ID:



Sequence

The sequence number tracks how many messages in a communication between two GCSP applications. Each time a process sends a request to a same remote process, it increments the sequence number. This will allows distinguishing which response corresponds to which request. The response has the same sequence number as its request. The “sequence” header is as follows:

Sequence <sequence_number>

The sequence number is incremented which each new request sent to a same remote process.

General

The General header section is present in all type of GCSP frames. The header lines are structured as follow:

- Content-Type: gives the type of data in the body, such as application/gcsp.
- Content-Encoding: gives the type of the body data encoding, such as the UTF_8.
- Content-Length: gives the number of bytes in the body.
- Date: gives the date when sending a request or response, in the format such as "Mon, 5 September 2005 16:52:41 GMT".
- Expires: gives the date/time after which the response is considered obsolete.
- WWW-Authenticate: the WWW-Authenticate response-header is used like in an HTTP response. This response header field must be included in 401 (Unauthorized) response messages. The field value consists of at least one challenge that indicates the authentication scheme(s) and parameters applicable to the Request-URI.
- Authorization: a GCSP client that wishes to authenticate itself with a server usually, after receiving a 401 response does so by including an Authorization request-header field with the request. The Authorization field value consists of credentials containing the authentication information of the user agent for the realm of the resource being requested.

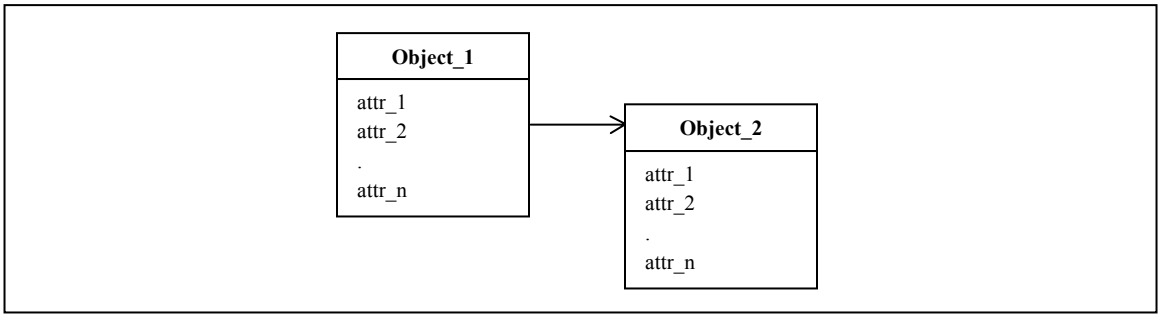
5.2.3 GCSP Body

A GCSP message may have a body of data sent after the header lines. If a GCSP message includes a body, there are usually header lines in the message that describe the body, in particular, the Content-Type and the Content-Length.

When a GCSP application sends a Get command to a peer application, the latter responds generally by 200 OK in a header line and sends the queried object in the message body. To transfer the object data over the network we had to define a new description method

that could be easy to use and be the least verbose as possible in order to be compliant with the signalling requirements. Because describing an object that encapsulates another object is not taken into account by SDP and because SDP attributes number is limited to the alphabet letters number we had to define an alternative description method that could answers to those new requirements. Thus we came up with a new description language similar to SDP.

A blank line separates the GCSP header from the message body and each line of the message body ends with a CRLF. 0 gives an example of an object that has (n) attributes and uses another object.



[Figure 51] Generic context objects schema example

Objects in [Figure 51] will be represented as follows in the GCSP message:

<HEADER LINE_1>
<HEADER LINE_n>
<BLANK LINE>
#s: <Object_1>CRLF
attr_1: <value> CRLF
attr_2: <value> CRLF
attr_n: <value> CRLF
#s: <Object_2> CRLF
attr_1: <value> CRLF
attr_2: <value> CRLF
attr_n: <value> CRLF
#e: <Object_2> CRLF
#e: <Object_1> CRLF

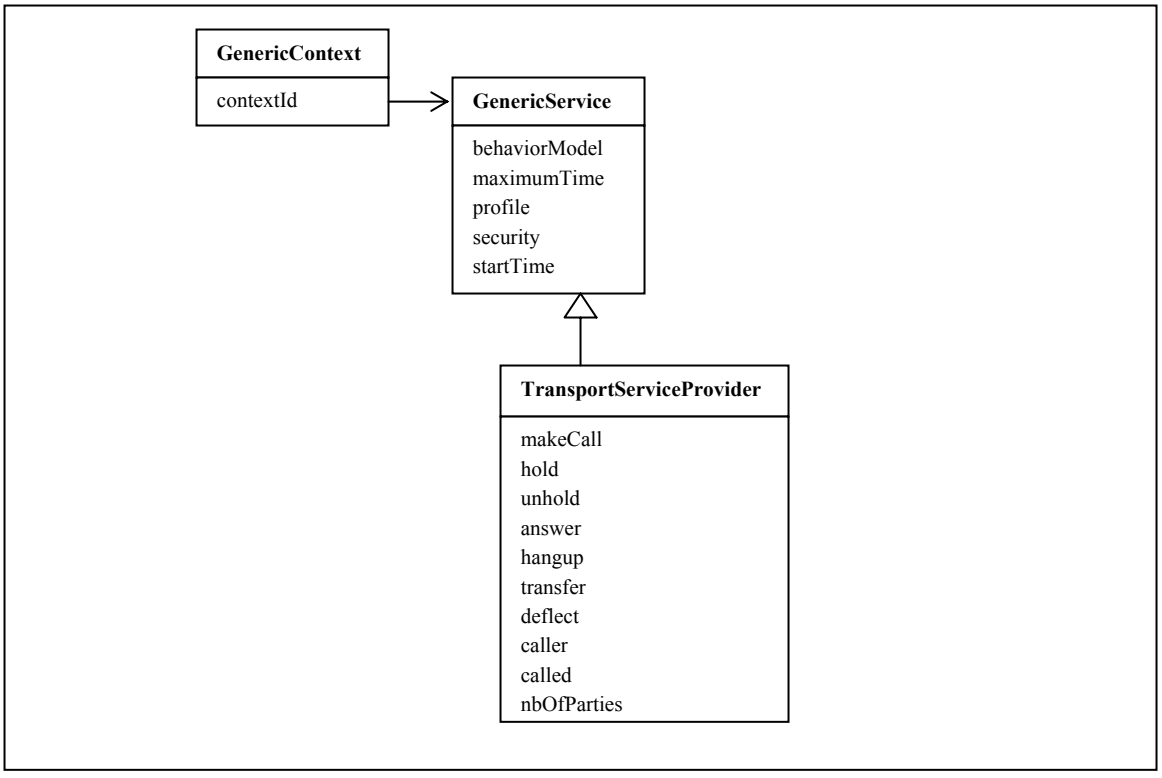
The <#s> indicates an object name start tag; it is followed by the object name and ends with a CRLF. The object attributes follows the object's name start tag; one attribute per line. An attribute line starts with the attribute full name (attr_1 for example) or the attribute's abbreviation (attribute's abbreviation is a number which is described by the GCSP Describe command). The line (#s: <Object_2> CRLF) indicates that Object_1 has

Object_2 as attribute. When all Object_1 attributes are described we close the object tag with a (#e: <Object_1> CRLF) line.

5.2.3.1 GCSP serialization vs. XML serialization

One common way to serialize a GCSP object is the usage of XML like with Netconf. However the benchmark results showed us that a GCSP serialization is approximately 50% less verbose than an XML serialization. This is crucial to enhance the protocol performance especially when it comes to high traffic between two entities like between an S-CSCF and an AS.

The following example [Figure 53] shows a serialization of the TransportServiceProvider object [Figure 52] done with GCSP and XML. In this example we show the necessary attributes used to do a make call, we do not use all the attributes of the GenericContext object as well as the GenericService object.



[Figure 52] TransportServiceProvider UML class diagram

GCSP serialization	XML serialization
229 characters	404 characters
#s: GenericContext #s: TransportServiceProvider MakeCall: 1 Hold: 0 Unhold: 0 Answer: 0 Hangup: 0 Transfer: 0 Deflect: 0 Caller: 569 Called: 0145817777 NbOfParties: 2 #e: TransportServiceProvider #e: GenericContext	<?xml version="1.0" encoding="UTF-8"?> <GenericContext> <TransportServiceProvider> <attr name="MakeCall" value="1"> <attr name="Hold" value="0"> <attr name="Unhold" value="0"> <attr name="Answer" value="0"> <attr name="Hangup" value="0"> <attr name="Transfer" value="0"> <attr name="Deflect" value="0"> <attr name="Caller" value="569"> <attr name="Called" value="0145817777"> <attr name="NbOfParties" value="2"> </TransportServiceProvider > </GenericContext>

[Figure 53] GCSP serialization Vs. XML serialization

5.2.4 Transactions

To optimize the protocol performance, the GCSP protocol allows the use of transactions like in MEGACO and TCAP transactions. Transactions are particularly useful between two entities that request high volume traffic like between a SCF and a SSF or between and AS (Application Server) and an S-CSCF in the IMS core network. To send a set of GCSP requests in one frame, a transaction request (Transaction_Request) frame is used; it encapsulates a set of GCSP request frames as follows:

Transaction_Req <transaction_id>
<blank line >
GCSP request frame1
< blank line >
GCSP request frame2
< blank line >
GCSP request frame n

- The "Transaction_Req" keyword indicates that the current frame is a transaction request and contains several GCSP request frames.

- The "transaction_id" parameter indicates the number of the transactions sent between two control processes. It is incremented with each new transaction between those two processes.

- The GCSP request frame is a GCSP frame that contains one of the GCSP commands (Get/Set …). The frame must contain a header and may contain a body.

The response to a transaction request is sent in a transaction response frame as follows:

Transaction_Resp <transaction_id>
< blank line >
GCSP response frame1
< blank line >
GCSP response frame2
< blank line >
GCSP response frame n

- The "Transaction_Resp" keyword indicates that the current frame is a transaction response and contains several GCSP response frames.

- The "transaction_id" parameter indicates the number of the transaction in response to a request transaction. A transaction response does not increment the transaction_id number.

- The GCSP response frame is a GCSP frame that contains a GCSP response. The frame must contain a header and may contain a body.

5.3 Proposal of a GCSP stack implementation over UDP

We describe in this paragraph the proposal of a GCSP stack implementation which uses UDP as a transport layer. Thus we put the light on the different layers involved in the GCSP stack and in particular we detail the application layer.

5.3.1 The transport layer

GCSP is an application signalling protocol which requires a robust and reliable transport layer as any signalling protocol would do. Since there are no easy implementations of such a reliable transport layer in the Internet network that we could put in place in an existing company LAN (required in the implementation of a new CTI architecture PART 6]), our attention was mainly focused on UDP and TCP that could be chosen to be the transport layer in the GCSP stack.

The choice between a TCP and UDP implementation depends on the application requirements.

a) UDP

For the GCSP/SIP signalling mediator implementation detailed in PART 6], we have chosen UDP to be the transport layer.

The reasons that pushed us to implement GCSP over UDP instead of TCP are the following:

1. Light protocol: the stack can be implemented on any platform such as a PDA or a Smart Phone.

2. Call setup delay: in the absence of the TCP three way handshaking, the call setup delay is reduced. In fact some of the current TCP implementations have a retransmission delay of the initial SYN packet between 6 and 24 seconds. Thus, even a single packet loss can lead to unacceptable call setup delays.

3. Less overhead and bandwidth congestion: a TCP implementation requires three messages at connection setup and 4 messages at release. These messages bring a poor

usage of the bandwidth and require that 7 messages should be sent with each GCSP command and response.

However to counterbalance the poor reliability of UDP we have designed an application-layer reliability that will be fully discussed in [paragraph 5.3.2]. Using an application-layer reliability rather than TCP has the advantage that timers can be adjusted according to the requirements of a signalling application rather than being at the mercy of a kernel protocol stack.

Reliable transmission on the application level unlike TCP, the problem with sequence number re-use is avoided by assigning each Generic Context an identifier which is unique in a host and never re-used in time.

b) TCP

When Bob uses his CTI application (phone bar) outside the company, mostly he will be running with a private IP address unknown for the CTI server. In this case, TCP can be used to tear down a permanent connection from the user application (outside the company) to the CTI server (located inside the company). However this implementation requires that the TCP connection should be maintained from the application logon till logoff.

In addition, TCP can be used if a transport-layer security protocol such as TLS is required. The GCSP frames can then be encrypted with an SSL layer.

5.3.2 The application layer

In this paragraph we detail the general and functional architecture of the application layer that involved most of our development efforts.

5.3.2.1 General architecture

The application layer of the GCSP stack has many roles from providing high level APIs to message sequencing and transaction management. In addition, since the UDP transport layer is not reliable, we implemented an application-layer reliability that handles messages retransmission.

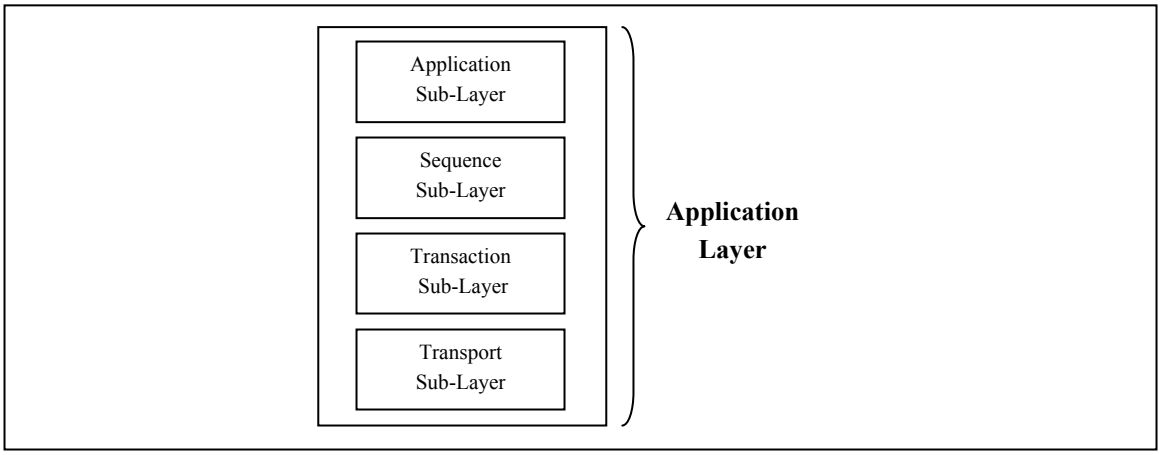
The different sub-layers of the GCSP application layer shown in [Figure 54] are the following:

- **The Application sub-layer**: it provides high level APIs and message factories which intend to bring easiness in the development of GCSP based applications. In addition the application sub-layer is responsible for creating and managing GCSP "dialogs" and generic contexts. A GCSP "dialog" consists of a GCSP request and its responses. All exchanged messages in a dialog have the same sequence number.

- **The Sequence sub-layer**: it is responsible for message sequencing and retransmission in a same dialog. The Sequence sub-layer regroups timers responsible for message timeout management.

- **The Transaction Sub-layer**: it manages GCSP messages encapsulation into transactions and handles transactions retransmission and timeouts. The transaction sub-layer is not implemented in the current version of the GCSP stack.

- **The Transport sub-layer**: is responsible for sending and receiving messages to and from the transport network (UDP in our case). It plays a fundamental role in parsing GCSP messages. In fact the transport sub-layer receives object messages from the Sequence sub-layer and encodes them into bytes before sending them into the UDP socket. And vice versa, it decodes byte messages received from the UDP socket into object messages before sending them to the Sequence sub-layer.



[Figure 54] GCSP Application Sub-Layers

5.3.2.2 Functional architecture

After presenting the general architecture of the GCSP stack, we give a detailed description of its functional architecture which includes logical entities involved in the application layer of the GCSP stack.

In [Figure 55] we show the different logical entities involved in each sub-layer. Each logical entity is a thread itself. These entities have the functional role as follows:

- **Processor**: is a server that listens on a UDP or TCP port. It receives GCSP byte messages from the network layer and put them in the Channels queue.

- **Channels**: read GCSP byte messages from the queue and parse them to check if there is any error then encode the GCSP object messages. Since parsing is a heavy task that can slow down the stack, many Channels can be run at the same time depending on the configuration of the stack. For example in our case since we had processors that can run two parallel threads at the same time (Intel dual core processor), we set the number of Channels to two. This reduces considerably the time to parse all messages in the queue. After the GCSP object messages are parsed, the Channels dropped them in the Sequence Manager queue.

- **Sender**: receives GCSP object messages from the sequence manager and encode them to bytes before sending them on the network layer. Since encoding GCSP objects to bytes messages is less heavy then parsing byte messages, thus no Channels are needed.

- **Sequence Manager**: is responsible for GCSP messages retransmission. When it receives a request from the Application Manager it starts a new Timeout Manager thread with this request as parameters and put a copy of the request in the Sender's queue. When a response to the sent request is received from the Channels, the Sequence Manager retrieves the corresponding Timeout Manager kills it and forward the response to the Application Manager thread.

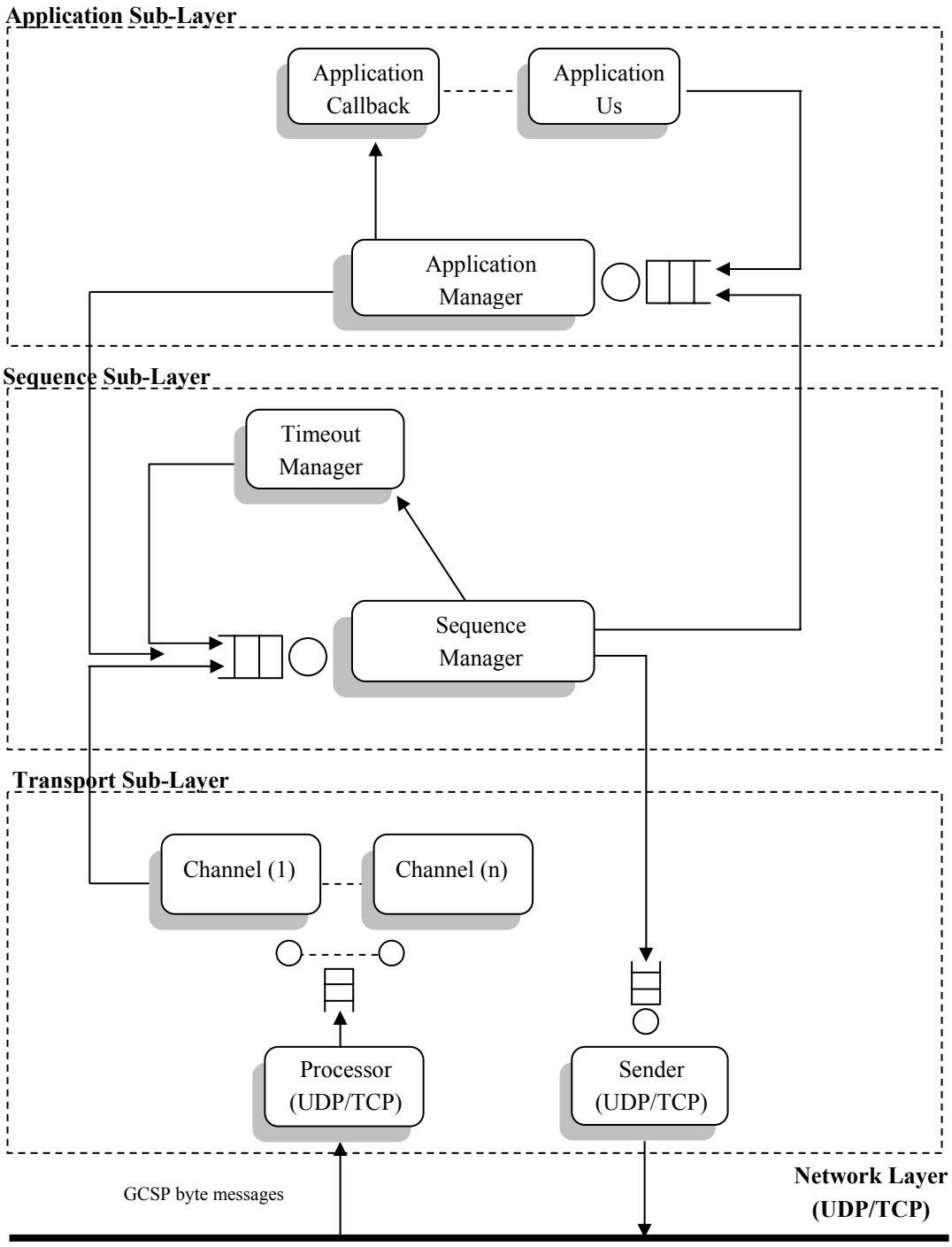
- **Timeout Manager**: there is one Timeout Manager for each GCSP request sent on the network layer. The Timeout Manager has a timer (in our case we set it to 50 ms) which if expired before receiving the GCSP response, resends the GCSP request to the Sequence Manager which on its turn send it to the Sender thread. The number of retransmission is programmable also, we set it to 2 retransmissions. When a GCSP response is received, the Timeout Manager thread is terminated by the Sequence Manager.

- **Application Manager**: is responsible for Dialogs creation, modification and release. It manages Dialogs, Generic Context IDs creation as well as sequence number incrementing. In the RAM memory, a dialog is a zone which holds a copy of the local Generic Context, the current sequence number, the From and To addresses as well as the source and destination context IDs.

- **Application Callback**: receives GCSP messages from the Application Manager and forward them to the application which has implemented this callback. In our case the callback was implemented by the CTI applications. For instance when a Make Call is received in a GCSP request, the CTI application is notified by the Callback Manager.

- **Application User**: The Application User contains all the APIs that are accessible to the developers to build easily GCSP requests. In our case the APIs include all the call control functions that will be described in [paragraph 6.2.1]. These APIs use factories that generate GCSP messages from simple input data. For instance the MakeCall(Alice, Bob) API use a factory that generates a GCSP request in which the MakeCall attribute is set to 1 and where [caller=Alice], and [called=Bob].

- **Queues**: are synchronized linked lists used by the multiple threads to pass object or byte messages. The synchronization resolves concurrency problems as well as deadlocks.

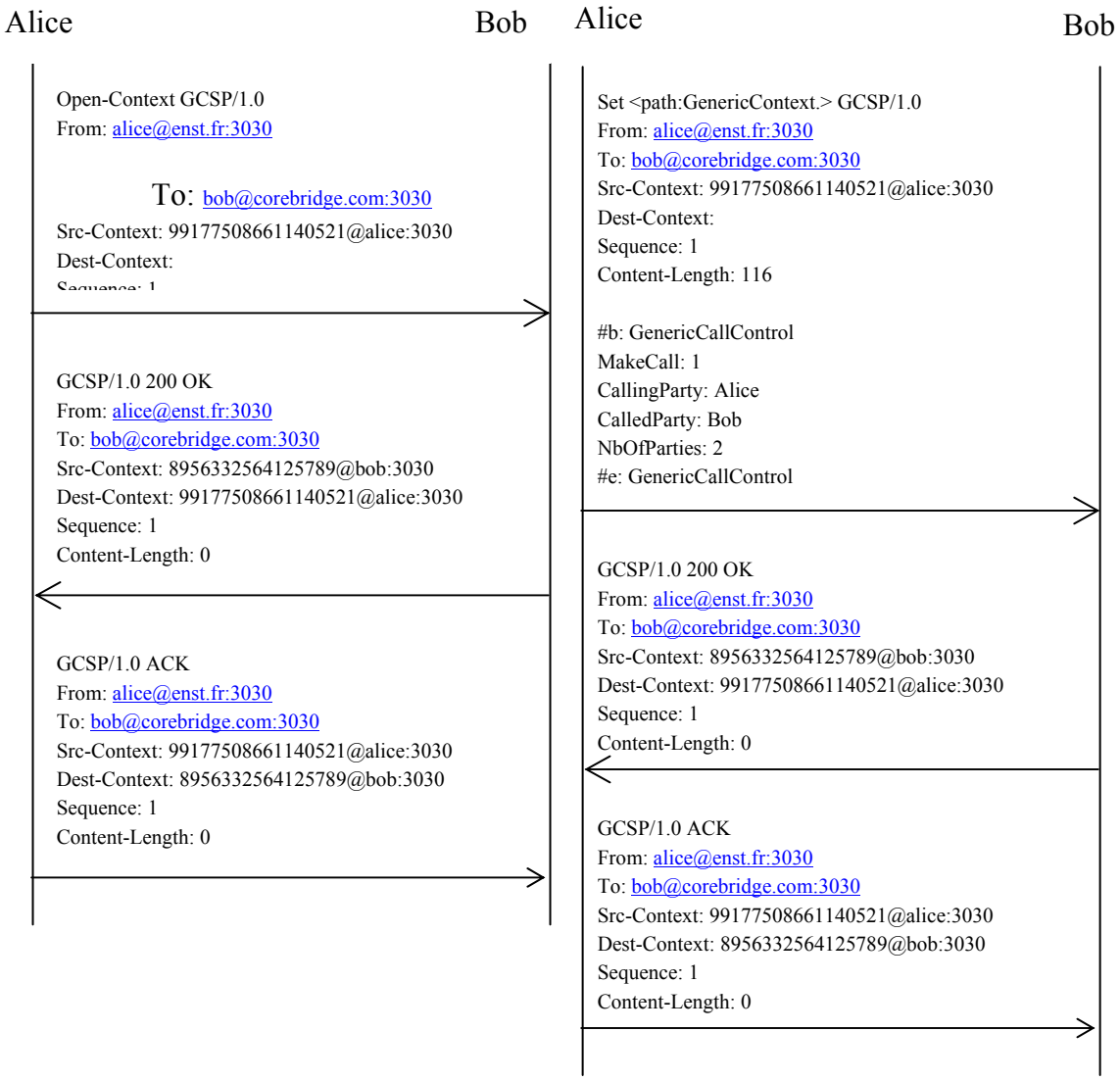


[Figure 55] Functional architecture of the GCSP application layer

5.3.2.3 The three-way handshaking mechanism

Since UDP is not a reliable transport protocol, we needed to add a three-way handshaking mechanism which ensures that a remote Generic Context is opened when two control processes start a new communication. This handshaking mechanism, similar to TCP mechanism, consists of 3 messages exchanged at the beginning of a new communication.

The three messages exchanged are {Open-Context, 200 OK, ACK} or {Set, 200 OK, ACK} like shown in [Figure 56] (in the Set request, the Dest-Context value is empty).



[Figure 56] GCSP three-way's handshaking mechanism

PART 6 - SIGNALLING MEDIATORS: EXAMPLE OF A CTI APPLICATION

In this part we validate by a practical implementation all the theoretical concepts that we have introduced in the previous parts of our work, such as the Generic Context and the GCSP protocol. We apply these theoretical ideas to cross-network services between the telephone network and the data network inside a company. In the case of business communication, cross-network services are normally achieved by the Computer Telephony Integration (CTI) technology. However the CTI architecture still presents an expensive installation cost and a difficulty in providing service mobility and in supporting new types of PBX and service technologies such as SIP based PBXs and Parlay services.

To resolve the different problems raised by the current CTI architecture, we propose a new GCSP based CTI architecture. We give therefore a brief description of the current CTI technology and we detail our proposal and implementation of the new GCSP based CTI Technology. This will allow us to outline the details of the GCSP stack implementation in the signalling mediator and the CTI applications.

6.1 The Computer Telephony Integration (CTI)

Computer Telephony Integration (CTI) is a technology seeking to improve the telephone services through the help of the users' personal computers. Later on, the definition of CTI has grown to include call routing, the integration of multiple media channels such as Web, voice, and e-mail and integration with interactive voice response (IVR) units. For many organizations, the definition of CTI continues to include such "classic" applications as "soft phone" and "screen popup". Since its inception, the deployment of CTI has been difficult for many organizations to achieve. Smaller organizations without dedicated IT staff often lack the funding necessary to hire consultants who can develop and maintain custom integrations. As CTI has matured, however, the requirement to integrate telephony applications with business applications such as customer relationship management (CRM) systems has created additional pressure for smaller enterprises to deploy CTI. Before defining a CTI strategy, organizations should consider alternative methods that do not require the dedicated IT staff or consultants traditionally associated with CTI projects.

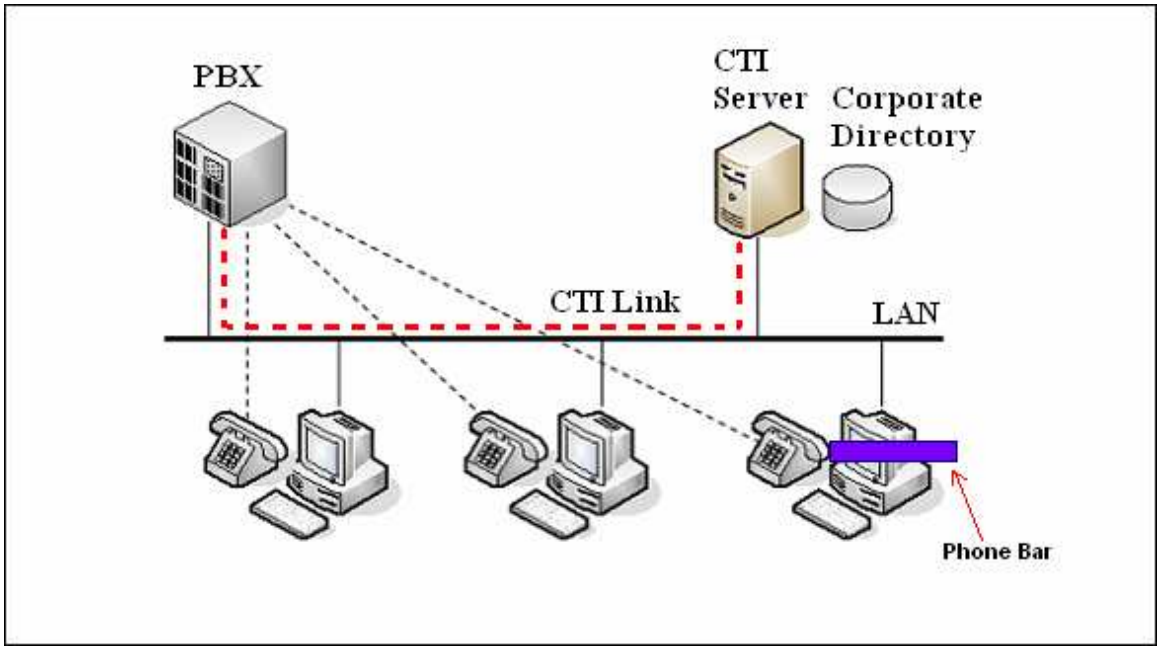
6.1.1 Basic principles of CTI

The CTI architecture is based on the CTI link which binds together the PBX and a CTI server. We have on one hand the PBX which connects all the company phones and on the other hand the CTI server which controls the employees' computers.

The CTI link protocol may be a proprietary protocol or CSTA, ASAI, Meridian Link an many others. These protocols are complemented by a set of CTI applications. These applications can be located on the CTI server or the user desktop computer. ***We call the CTI application installed on the user desktop computer a phone bar***. The phone bar is a graphical interface that allows the user to invoke CTI services located on the server. The phone bar is located at the client level in the SIMPSON view. CTI services, among many features, allow searching customers' profiles in a database, initiating and handling phone calls.

The bridging achieved by the CTI link brings many facilities to the company. It allows to control incoming and outgoing calls. On the CTI link the CTI server receives notifications from the PBX when an incoming call arrives. Such notifications contain information about the inbound call such as the caller ID. The CTI server can decide to

establish the call or to route it to an alternate number. On the other hand the CTI server receives call commands from the employees phone bar in order to place calls. On its turn the CTI server sends these commands to the PBX via the CTI link. This bridging by protocols such as CSTA between the CTI server and the PBX requires a permanent link. In general the protocols of the CTI link is based are transported over the company's LAN on TCP/IP protocols.



[Figure 57] Example of a CTI architecture in a call centre

The key elements of a CTI architecture 0 are the following:

- **The PBX**: handles incoming and outgoing phone calls.
- **The CTI server**: handles call flows from the PBX and the phone bar. Fetches customers' profiles in the Corporate Directory on incoming calls and when the user fetches for customers numbers. The CTI server holds the intelligence call routing logic such as Automatic Call Distribution (ACD) or Least Cost Routing (LCR). It tells the PBX what to do when receiving an incoming call.
- **The phone bar**: is an application installed on each employee desktop. It allows the employee to fetch customers' profiles in the Corporate Directory and initiate phone calls. In addition, when a customer calls the company, the employee receives the customer profile popup on his screen through the phone bar.

- **The CTI link**: is a permanent link between the CTI server and the PBX by means of the company's LAN. It allows sending commands to the PABX and receiving notifications from the PBX.

- **The Corporate Directory**: contains customers' information such as the address, phone numbers, company, customer ID … It allows fetching customers information or identifying an incoming call.

Each employee CTI seat is composed of a computer and a telephone. The computer and the telephone are associated together via the CTI link.

6.1.2 The CTI architecture

6.1.2.1 CTI applications development

To facilitate the development of CTI applications, many programming interfaces are available on the market. They define a set of classes and functions which makes possible the interaction between the computer domain and the telephony system. The CTI application is then independent from the telephony system and can run with different equipments from different providers. There are three major APIs available on the market:

- Microsoft TAPI

- Novell TSAPI

- Sun Microsystems JTAPI

In the following paragraphs we will focus on the TAPI architecture which is used in the Corebridge solution.

6.1.2.2 TAPI general architecture

Telephony API (TAPI) enables programmers to develop CTI applications which can control different equipments such as CISCO and 3COM PBXs and phones. The interface with the PBX is assured by a TSP (TAPI *service provider*), which is equipment dependent and does the interface between the physical equipments (phones, fax, PBX) and the TAPI applications. Telephony Service Providers (TSP) are dynamic link libraries (DLLs) that translate the commands for a specific telephony device, carrying out the low-level tasks required to communicate over telephone and IP networks. Each PBX supplier provides a TSP interface to control his equipments.

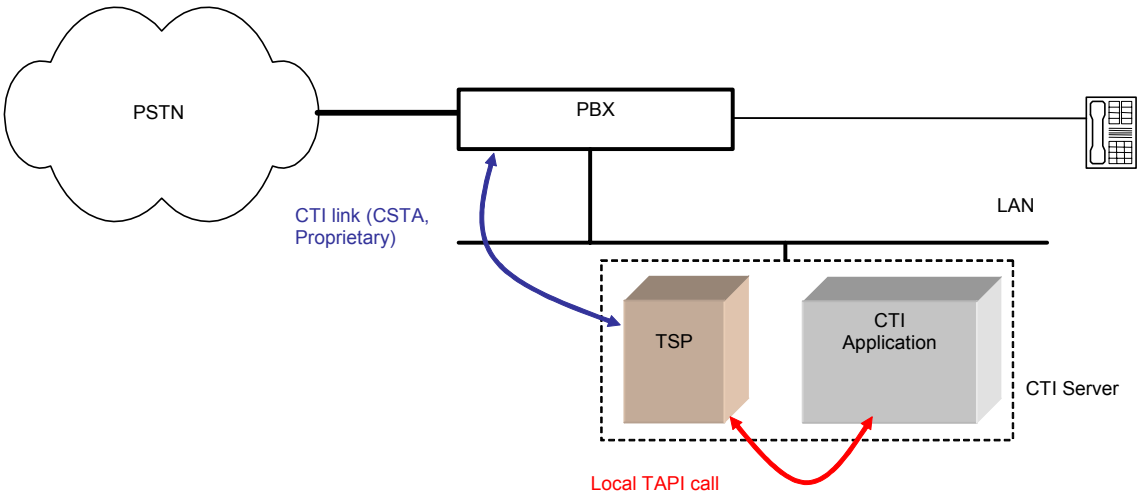
The TSP translates the TAPI commands received from a TAPI application to specific commands that could be understood by the targeted equipment. And vice versa, it translates from events received from the equipment to TAPI commands that the application understand.

The TAPI functions provided to CTI applications depend basically on the concerned equipment and its TSP. Each TSP may support all or a part of the TAPI functions. For example if a TSP doesn't support the Caller ID (CLID) function, the CTI application can not ask the equipment to provide the CLID.

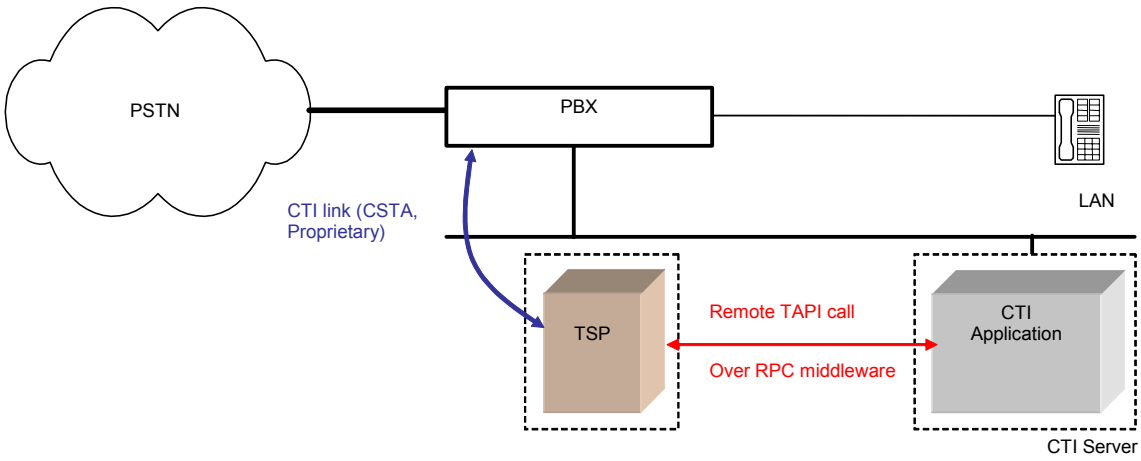
The TSP can be installed in two different architectures:

- If the TSP is installed on the same machine as the CTI application, the latter can directly use the TAPI APIs of the TSP; we say that the TSP is installed in a first party architecture [Figure 58].

- However if the TSP is located on a distant machine the CTI application can not access remotely the TAPI APIs of the TSP since TAPI is not a protocol. Instead, a Windows middleware is required to insure interoperability between the TSP and the CTI application. This middleware is mounted over RPC (Windows DCOM middleware) and allows the CTI application to make a remote call to the TSP APIs as if the TSP is installed on the same machine; we say that the TSP is installed in a 3rd party architecture [Figure 59].



[Figure 58] TAPI first party architecture



[Figure 59] TAPI third party architecture

6.1.2.3 Corebridge TAPI Architecture

The Corebridge CTI architecture is TAPI based from the phone bar to the CTI server [Figure 60]. It has been decided to rely on Microsoft TAPI DLL which is embedded in the Microsoft Windows operating system. This reduces the cost of the global CTI architecture as there is no need for additional software or hardware in desktop computers and it offers a rich set of APIs for coupling the telephony system and the enterprise applications. Corebridge CTI architecture is composed essentially from:

- A PBX: connects the enterprise telephony to the public switched telephony. The PBX communicates with the CTI services of the company over the CTI link using protocols such as CSTA, ASAI, Meridian Link …

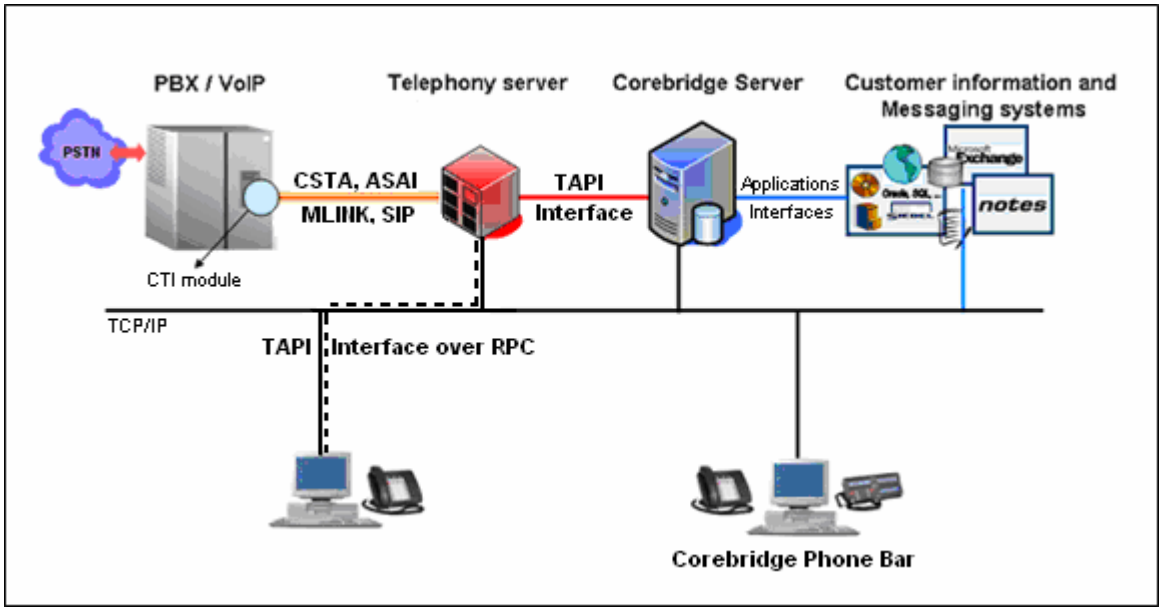
- A CTI server: This function is at the provider level in the SIPMSON model. It hosts the Corebridge CTI application which makes the bridge between the telephony and the company's data architecture. The Corebridge CTI application allows users to manage all their contacts (e-mail, voice, fax, call notifications, SMS) from their existing mail client (Lotus Notes or Microsoft Outlook) from anywhere or any desktop. All the communications like email, fax, voicemails, and missed calls are integrated in the mail client the same as the telephony call log. In addition the CTI server is responsible of the intelligent telephony routing logic such as Automatic Call Distribution (ACD) and Least Cost Routing (LCR). All the company calls can be received on a single line then the CTI server decides to which user the call should be routed according to a given algorithm.

- A telephony server: the TSP can be installed on the CTI server as well as on a standalone server. When installed on standalone server, we call it a telephony server. It interfaces the Corebridge CTI applications and the PBX. Corebridge applications have to register on the TSP in order to receive telephony events (incoming call notifications) and to interact with PBX (make a call, transfer a call …).

- The phone bar: on each computer desktop there is a Corebridge phone bar which offers to the user telephony services such as PC telephony and customer profile popup service. In the Corebridge TAPI architecture, a user makes a call directly to the PBX without passing through the Corebridge server.

  The phone bar also provides with data services such a customer lookup in the corporate directory and integration with the CRM applications. The Corebridge client interacts with the telephony system using the TAPI DLL of its desktop computer. The TAPI interface interacts with the telephony server which translates the TAPI commands to proprietary commands that the PBX understands.

- The customer information and messaging systems: it is the existing set of applications in the company such as the messaging systems, CRM applications and customers contact databases.



[Figure 60] Corebridge 3rd party architecture

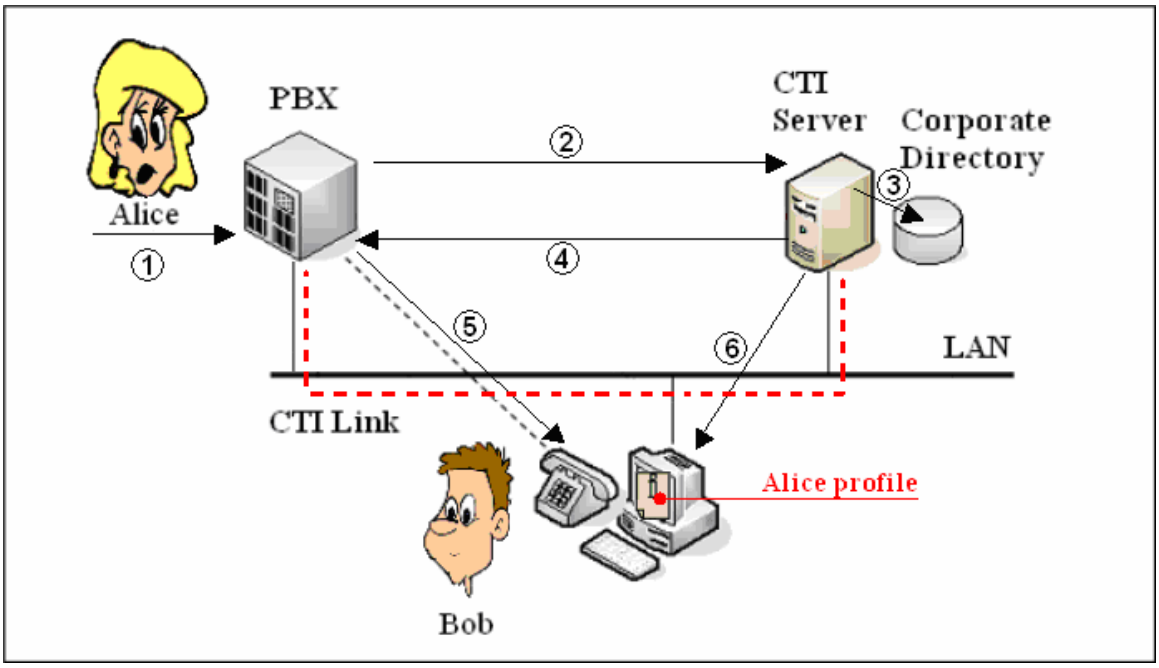
6.1.3 Detailing the customer profile popup service

After describing the CTI architecture we detail an example of CTI services and particularly the customer popup service. In this example shown in [Figure 61], customer Alice calls Bob who works in a bank call centre on his office phone. Bob phone rings and simultaneously, Bob receives Alice customer profile as popup message on his screen.

The sequencing of the messages involved in this service is the following:

1. Alice calls Bob on his phone office.
2. The PBX holds the call and sends a notification with the caller number to the CTI server.
3. The CTI server looks up in the Corporate Directory and find out that the caller is Alice, then retrieves her profile.
4. The CTI server tells the PBX to route the call to Bob extension (it could decide to route it somewhere else if there is a routing rule set on).
5. The call is send to Bob extension which starts ringing.
6. At the same time as Bob telephone rings, Alice profile pops up on Bob screen.

Note that the time interval between the telephone ringing and the customer profile popup should be less than 200ms. The less this time is, the more Bob has time to read Alice profile before answering. In addition, some delay comes from 3rd party applications and is added to total delay for the customer profile popup. These applications could be for example CRM applications. When a customer calls, the CTI server sends also a notification to this CRM application with the customer ID fetched in the Corporate Directory. The CRM application pops up an additional customer profile on Bob's screen which also needs time to popup.



[Figure 61] The customer profile popup service sequencing

6.2 The proposal of a new CTI architecture

In this section we analyse the Corebridge TAPI based CTI architecture in the light of service mobility and product extensibility to support new types of PBXs. We then propose a new CTI architecture for Corebridge applications which is more adequate in providing multi-provider and cross-network services and to the product evolution and development cost reduction.

6.2.1 Corebridge telephony architecture analysis

Corebridge telephony architecture is Microsoft TAPI based from the phone bar to the server. To make or receive phone calls the Corebridge phone bar uses Microsoft TAPI DLL which is integrated in Microsoft Windows operating system. It uses some of the TAPI functions in particular those which are related to the call control part. These functions are as follows:

1. ***Make Call***: makes a call.
2. ***Answer Call***: answers a call.
3. ***Hold***: holds a call.
4. ***Un-hold***: un-holds a call.
5. ***Hang up***: terminates a call.
6. ***Attended call transfer***: transfer a call received from B to C after C has answered.
7. ***Blind call transfer***: transfer a call received from B to C without waiting for C to answer. The call is directly transferred to C on ringing.
8. ***Deflect***: when B receives a call, the call is deflected automatically on ringing to C. The difference between Deflect and Call Divert is that with the latter, all calls are forwarded to a same destination number while with Deflect the call is routed to a particular destination number according to a routing table.
9. ***Conference***: makes a conference between (n) parties.

10. ***Monitor***: a CTI service may requests a phone line monitoring. In this case, the PBX sends all the telephony events which concern this line to the CTI application which uses this service.

11. ***Login***: when Bob starts his phone bar, it first logs in to the CTI server and logs out only when Bob shuts down his phone bar.

Corebridge phone bars can be installed according to a first party architecture or a third party architecture. The most common installation is the third party architecture where there is one CTI server and all Corebridge phone bars are connected to this CTI server using the TAPI protocol over RPC invocations.

TAPI based architecture can no longer answer to the new market demands in terms of deployment cost reduction, interaction with soft-switches, and service mobility:

- **Expensive installation cost**: the installation of a TAPI architecture requires buying CTI licenses from the PBX manufacturers in addition to the TAPI TSP (not all the PBX manufacturers provide a free TAPI TSP). In fact, to be able to communicate with the PBX, the PBX CTI link should be enabled. This requires to buy as many licenses (from the PBX manufacturer) as the total number of employees that will be making use of the CTI architecture. The total licenses cost is important and reaches approximately 5000$ for 25 users depending on the PBX type.

- **Interaction with soft-switches**. Classical PBX constructors deliver a TSP which offers a TAPI interface for CTI applications developers, however the new soft-switched, basically SIP enabled, does not deliver TSPs that are TAPI compliant. Thus TAPI applications can no longer reuse their existing code to interact with new upcoming soft-switches. This handicap is also extended to the public switched telephony; in fact, the public switched telephony has opened its doors to the 3rd party service providers through the Parlay APIs. These APIs are not TAPI compliant nor do Parlay gateway manufacturers offer TSPs which are TAPI compliant.

- **Mobility**. It is difficult to extend the call centre functionalities, such as the profile popup, on mobile platforms like Smart Phones and PDAs because from one side the TAPI DLL integrated in Windows CE and Pocket PC has limited functions and resources and on the other side the user will be facing firewall traversal problems which we discuss in the following paragraph.

- **Firewall traversal**. In fact TAPI 3rd party architecture is based on RPC (between the phone bar and the TAPI server) which makes it difficult to bypass firewalls. In general RPC TCP port is shut on the firewall and the best way to traverse it is by using port 80. This handicap prevents from using the phone bar outside the office, in other terms it would be difficult for an employee to have the screen popup on his PC at home without having a VPN connection to his company. This is due to the disabled RPC port on the company firewall, thus there is no TAPI connection between the phone bar (at home) and the CTI server (at office).

- **Strategy**. In addition to the technical issues which prevent TAPI applications market from growing, is the new Microsoft SIP strategy that consists of pushing the development efforts on SIP which will be fully integrated in Microsoft Windows Vista. In parallel, the developments will stop on TAPI and the TAPI DLL will no more be integrated in future Microsoft Windows versions.

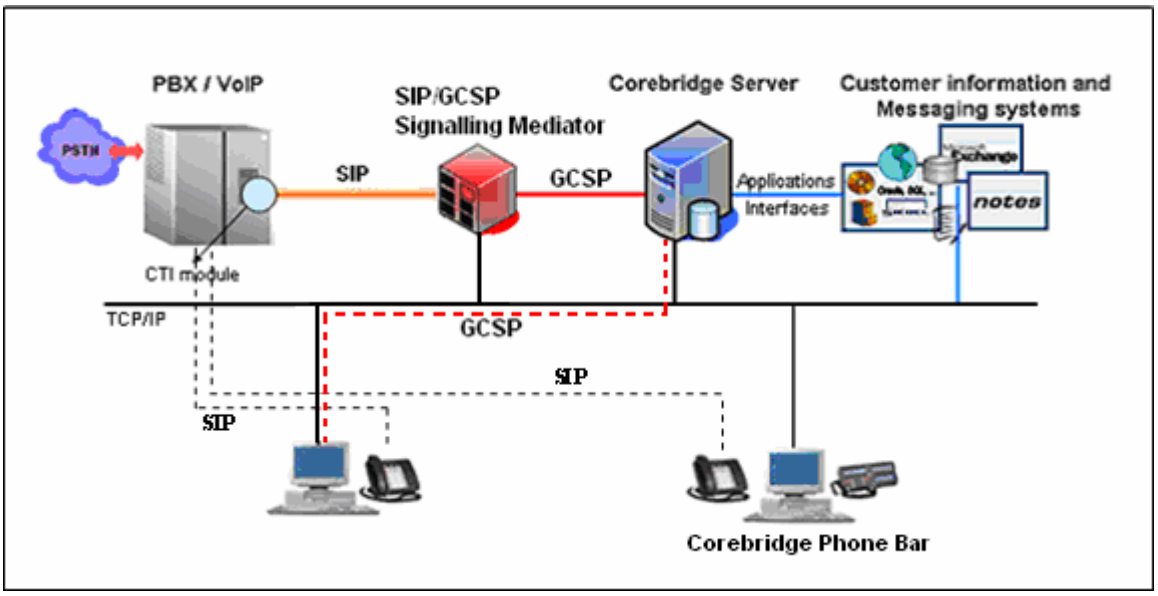
6.2.2 Our proposal of a new GCSP based CTI architecture

After analysing the technical issues and the new market trends that threaten the TAPI applications market growth, we conclude that the TAPI CTI architecture is no longer a viable solution. Thus we propose a new GCSP based architecture for the future CTI applications which will assure easy multi-provider and cross-network services at low development cost efforts. This new CTI architecture will simplify the interaction with new PBXs and new signalling protocols, it will also provide easy service mobility across heterogeneous networks. We have designed this new architecture to fit with an enterprise network and we have extended it to public switched network such as the PSTN network.

6.2.2.1 A new enterprise CTI architecture

To build this new CTI architecture, we propose to remove the TAPI link between the phone bar and the Telephony server and centralize the communications through the CTI server [Figure 62]; the phone bars are able to send and receive calls over GCSP.

The removal of the TAPI link includes also the removal of the TAPI TSP. Since the TAPI TSP is removed, a new interface for the CTI applications to communicate with the PBX is needed. Thus we propose a signalling mediator which will replace the TSP and translates from GCSP to the signalling protocol the PBX supports, in our case to SIP.



[Figure 62] A new enterprise CTI architecture based on GCSP

In the new CTI architecture depicted in [Figure 62], the TSP is replaced by a SIP/GCSP signalling mediator and all the CTI applications located on the server and the phone bars communicate using GCSP. This new architecture has many benefits on the TAPI architecture, among them:

- Cheaper installation cost since no CTI licenses are needed any more. The CTI link (SIP or Parlay) is always enabled with no additional cost.

- Easy interconnection with new types of PBXs and signalling protocols without putting in question all the existing CTI architecture; in such case a new signalling mediator is needed. This will reduce considerably the mid and long term development efforts costs especially when new signalling protocols are hitting the market like SIP and the Parlay APIs. Parlay allows us to extend the enterprise CTI architecture to the PSTN which will be fully discussed in [paragraph 6.2.2.2].

- Service mobility and firewall traversal. GCSP can be implemented over TCP and UDP. When implemented on TCP, it can run over port 80 which makes easy the firewall traversal. In this case the customer profile popup service can be extended outside the company network. Bob can receive Alice profile on his laptop anywhere outside the company without having to VPN.

- Facilitating the extension of the CTI applications on mobile platforms such as PDAs and SmartPhones. The GCSP protocol stack is light and can be

implemented on any device since it does not require an RPC middleware as TAPI does.

6.2.2.2 Extending the enterprise CTI architecture to the public switched network

Parlay APIs opens the legacy public network (PSTN) to third parties service providers. It allows creating richer services that have one leg in the PSTN network and another in the Internet network and also provides cross-network services and service mobility from and to the PSTN network.

The Parlay APIs allow us to extend the new CTI architecture and in particular the customer profile popup service that we had inside the company to the public switched network and to assure a complete service mobility (telephony and data). Indeed when Alice calls Bob on his office phone, Bob will receive Alice profile on his PDA (data mobility), and Bob mobile phone will ring instead of his office phone (telephony mobility).

In the example shown in [Figure 63], Bob leaves his office and head home. At home, he starts his PDA and connects to his WIFI network and logs in the enterprise CTI server. The connection to the CTI server is done via GCSP. After logging to the server, Bob declares through a web page that he wishes to receive his office phone calls on his mobile phone. The CTI server notifies the Parlay server which sends a command to the Parlay Gateway to monitor Bob office phone and arms a Detection Point in the Intelligent Network function of his PSTN central office (in the SSF).

When Alice calls Bob on his office phone, the call flows are as follow:

1. Alice's Originating Local Exchange (OLEX) forwards the call to Bob's Terminating Local Exchange (TLEX).

2. The Service Switching Function (SSF) in the TLEX detects an armed DP (which is related to Bob's office phone number); it suspends the call processing and fires a notification to the Public Network SCF which works as a Parlay Gateway.

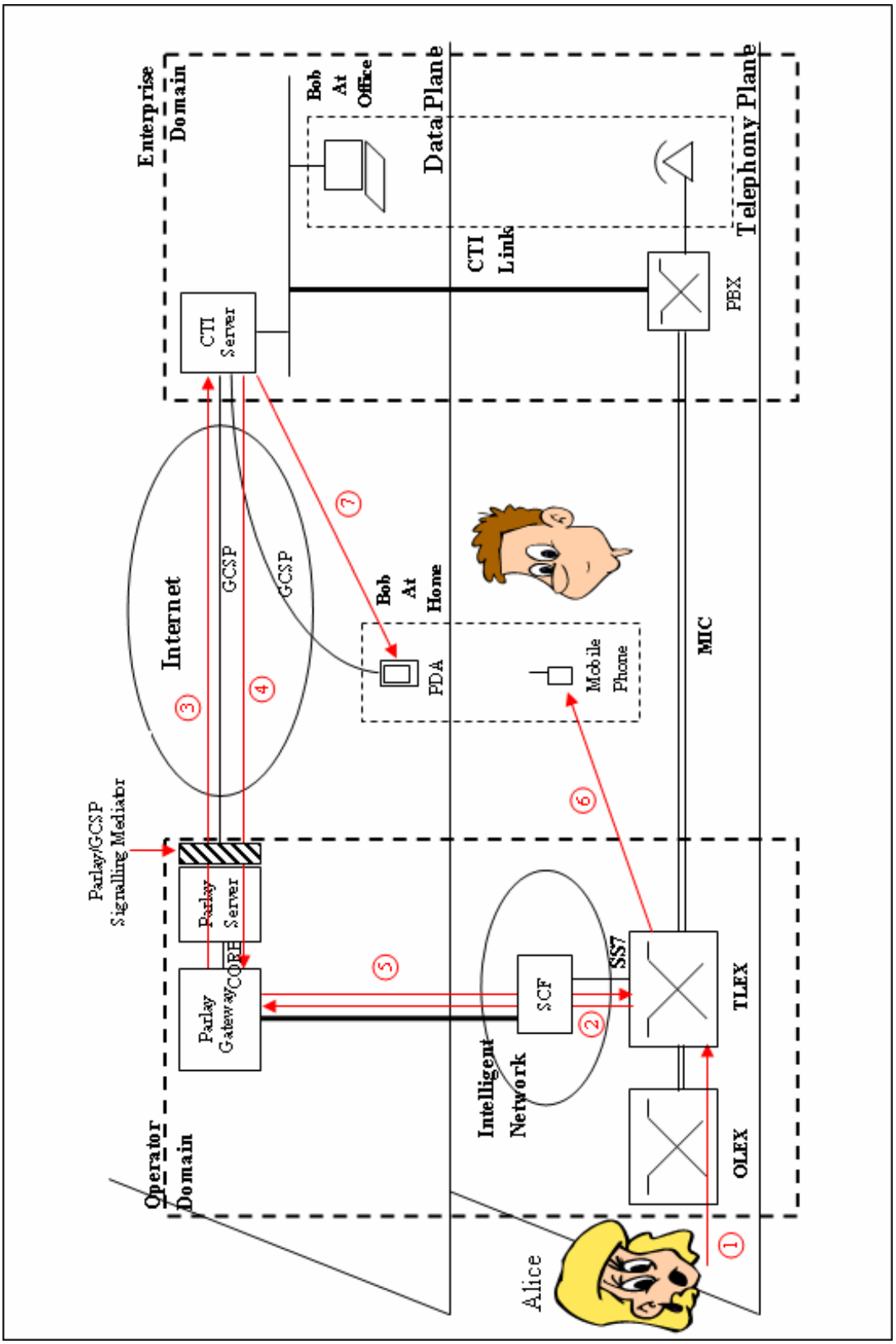
3. The Parlay Gateway notifies the Parlay Server which translates the notification from Parlay to GCSP and sends it to the CTI server located in the company.

4. The CTI server looks in the Corporate Directory for Bob's new location and number and retrieves his mobile phone number. Afterward, a GCSP command is sent to the Parlay Server then to the Parlay Gateway.

5. The Parlay Gateway acts as an SCF with the new number to route the call to. The SCF sends a connect command to the SSF function located in the switch, with Bob's mobile phone as a destination number, which resumes the suspended call processing.

6. The call is forwarded to Bob's mobile phone through the GSM network (the GSM network is not shown in our example figure).

7. In parallel, the CTI server pushes Alice's profile onto Bob's PDA.



[Figure 63] Extending the enterprise CTI architecture to the public switched network

6.2.3 Extended SIMPSON view of the Corebridge services

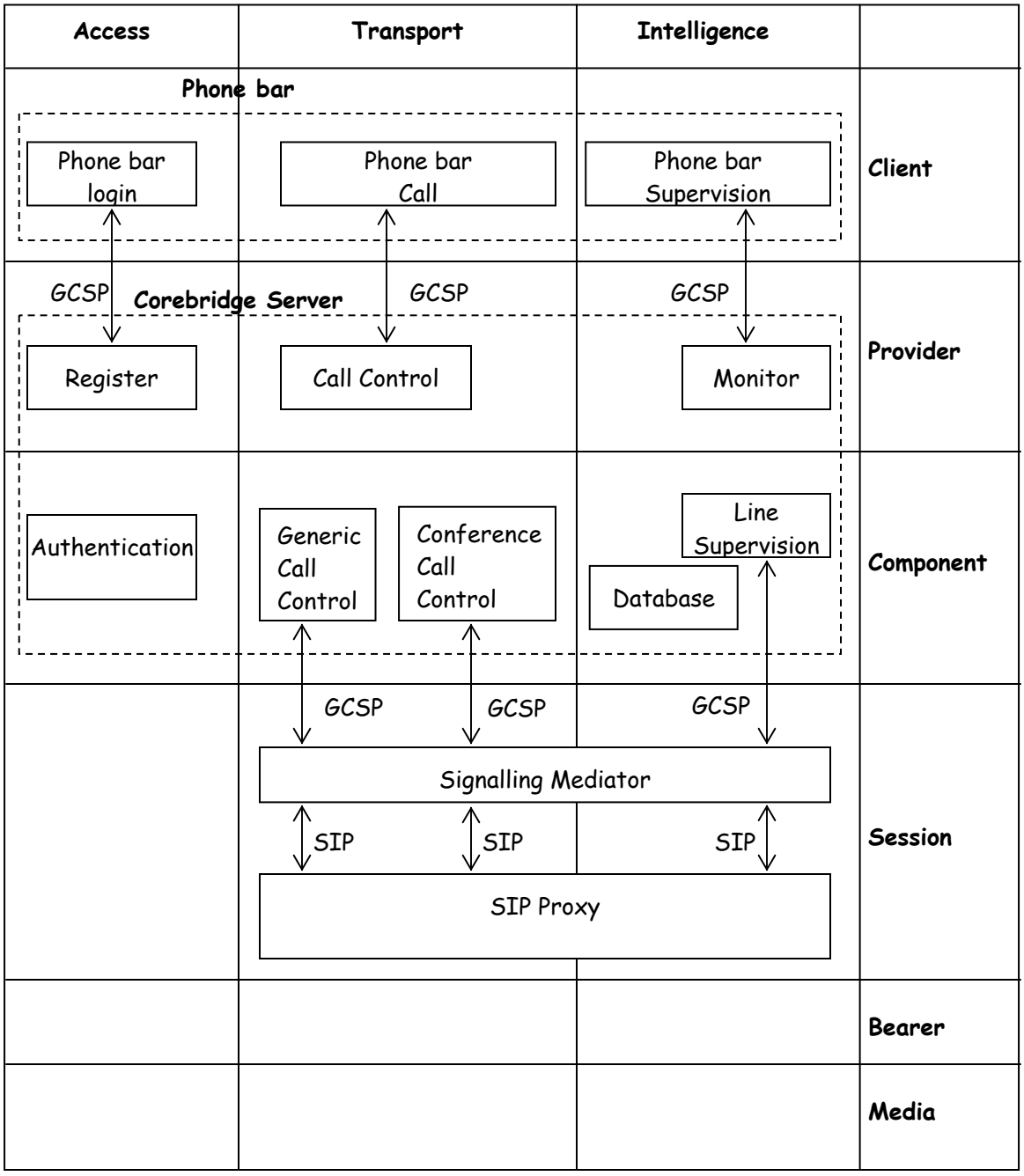
We now describe how the Corebridge services may be mapped, on the extended SIMPSON model according to the general communication service detailed in [paragraph 1.3.6.5]. In this example, we only show the services we used in our implementation of the new CTI architecture. These services are the login service, the call control service and the monitoring service.

[Figure 64] shows that the Client level consists in the Corebridge phone bar which interacts with the CTI server by means of GCSP commands.

At the Provider level we find the main service logic of the login, call control and monitor services. At this level, a service is an association of service components from the Component level. The communication between the Provider and the Component level is done inside the CTI server and is for the moment proprietary.

The Component level regroups all the necessary components for building a global service. In our case the Registration service is built of an Authentication and a Database component. On the Transport level, a Call Control service is built of Generic Call Control and Conference Call Control components. And finally a Monitor service is an association of a Line Supervision and a Database component.

Some components require a session. The Generic Call Control component requires a session during all the conversation between Alice and Bob. Another example is the Line Supervision which requires a monitoring session. This session is established by the SIP proxy for the monitoring service when it receives a Subscribe command. Finally the communication between the Component and the Session level is assured by the GCSP/SIP signalling mediator.



[Figure 64] Extended SIMPSON view of the Corebridge services

6.3 Implementation of a GCSP/SIP signalling mediator

After introducing the new CTI architecture, we put to light and detail our implementation of the GCSP stack which will be used in the phone bar, the CTI server and the GCSP/SIP signalling mediator. We will focus on the signalling mediator which translates from GCSP to SIP. It replaces the TAPI TSP in the traditional CTI architecture and allows the CTI server to interact with a SIP proxy, in particular a 3COM implementation of a SIP proxy called the VCX.

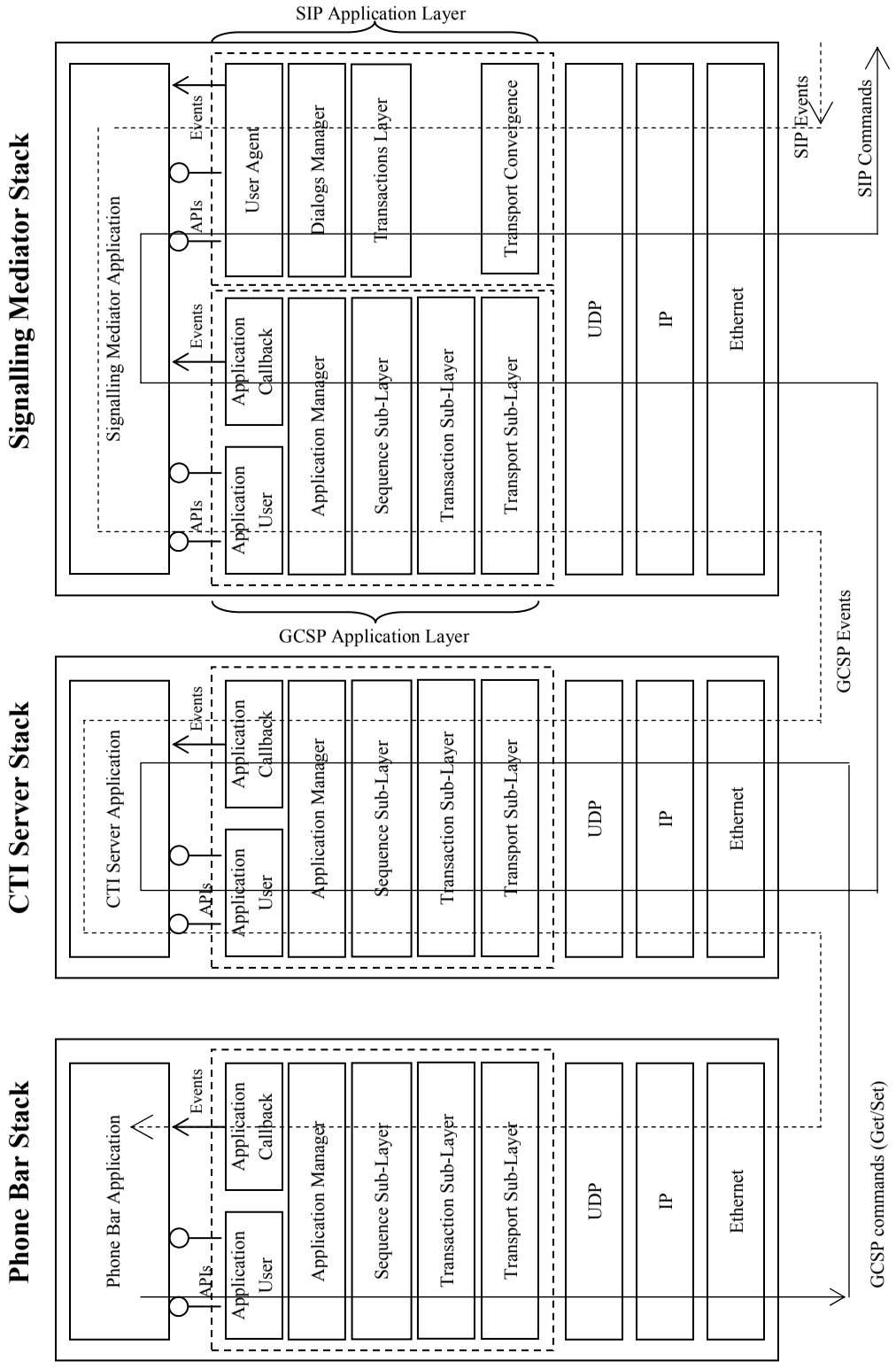
The communication protocol used between the telephones and the soft-phones from one side and the VCX (SIP Proxy) from the other side is the 3COM implementation of the SIP protocol [Figure 62]. Thus the GCSP protocol is only used between the phone bar, the CTI server and the signalling mediator.

6.3.1 The GCSP stack architecture

We have implemented the GCSP stack described in [paragraph 5.3] and we chosen UDP to be the transport layer for our first implementation of the stack. TCP will be implemented in future releases to assure complete service mobility which will allow Bob to use his phone bar outside the company without the need to VPN.

Since Corebridge applications suite is Windows based, we implemented the GCSP protocol stack in C# version 2.0. The DOT NET framework, required by C#, is already used by some of the Corebridge applications and is integrated in Windows XP (and more). An implementation with Java was considered, however this would require an additional installation of the Java JRE on the user desktop.

We now describe the GCSP stack architecture, shown in [Figure 65], which we implemented in the phone bar, the CTI server and the signalling mediator.



[Figure 65] The GCSP stack of the new CTI architecture

The GCSP stack implementation in the phone bar and the Corebridge server is approximately the same. The phone bar and CTI server applications send GCSP commands with the APIs offered by the Application User sub-layer and receive notifications from the Application Callback sub-layer.

In the signalling mediator, we find two stacks; the GCSP and the SIP stack. The Signalling Mediator Application layer plays a fundamental role in GCSP/SIP message translation. In addition, this Application layer handles the mapping between SIP Dialogs and GCSP Associations. When it receives a message from a SIP user agent, it fetches the corresponding SIP Dialog in a dynamic table and retrieves the corresponding GCSP Association. The Association allows to locate the remote GCSP context and to send the GCSP command (translated from SIP) to the GCSP end node. Finally the Application layer is responsible for message sequencing; it maps the different message sequencing of GCSP and SIP because of the differences in their behaviour models.

The association of GCSP sessions is done by the Application Manager sub-layers. The phone bar can locate the CTI server address thanks to the Association table. Each time a new association is created, it is automatically added to the Association table, and freed when the communication is ended.

6.3.2 Designing the Generic Context

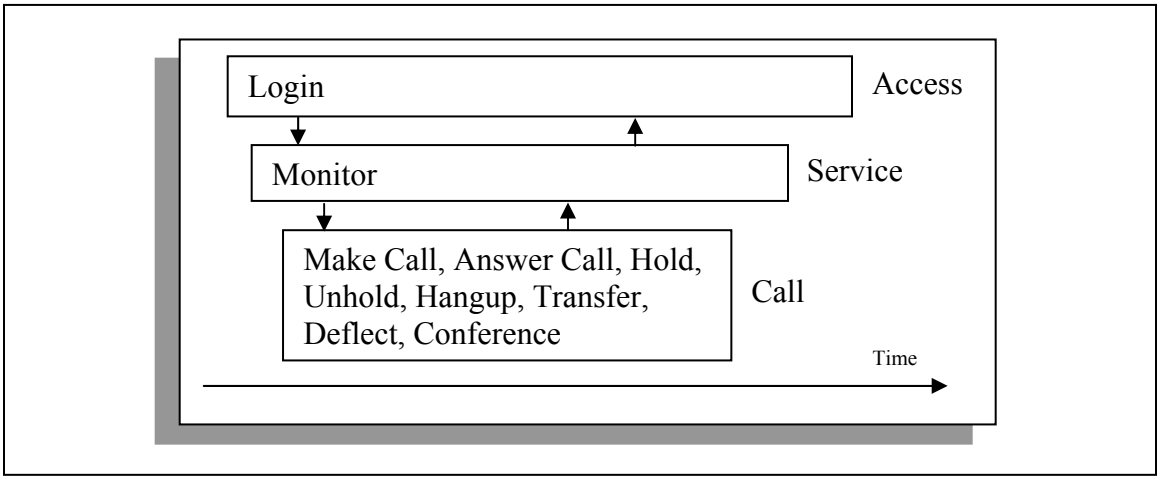
In [PART 4] we gave a description of the general schema of the Generic Context as well as detailed parts of it. To design the signalling mediator and the GCSP stack, we will therefore detail and map the call control functions that we have discussed in [paragraph 6.2.1] on the Generic Context schema.

The functions we need to implement are the following:

1. Login
2. Monitor
3. Make Call
4. Answer Call
5. Hold
6. Un-hold
7. Hang up
8. Attended call transfer
9. Blind call transfer
10. Deflect
11. Conference

We designed the new GCSP based CTI service which regroups the functions enumerated above with respect to the global sequencing of a communication service described in [paragraph 1.3.2]. Thus a CTI application requires first to login and then monitor the line it wants to supervise before making or receiving any phone call 0.

For example when Bob opens his phone bar, the application logs in to the CTI server with a login and password. Once logged in, the application requires monitoring Bob's line (0153757569) and thus receiving all the telephony events of this line. After the login and monitor phase, Bob can now receive and make phone calls from his phone bar.



[Figure 66] Global sequencing of the new GCSP based CTI service

In [Figure 67] we give a description of the Generic Context design which we implemented in the GCSP stack. This design follows the general schema given in [PART 4], however we did not implement all of the initial attributes but only the ones required for the purpose of our implementation in addition to new classes needed for the GCSP stack.

The main classes we implemented in the Generic Context are the following:

- **GenericContext**: contains a Service object which could be a UserLogin, a CallControl or a MonitorTelephonySubscriber.

- **Service**: is the main class for all services that are derived from it. In addition, the Service class contains a reference to an Association object.

- **Association**: is composed of a source and destination GCSP URI (for example: gcsp:rony.chahine@192.168.1.9:3030). In addition to the addresses, the

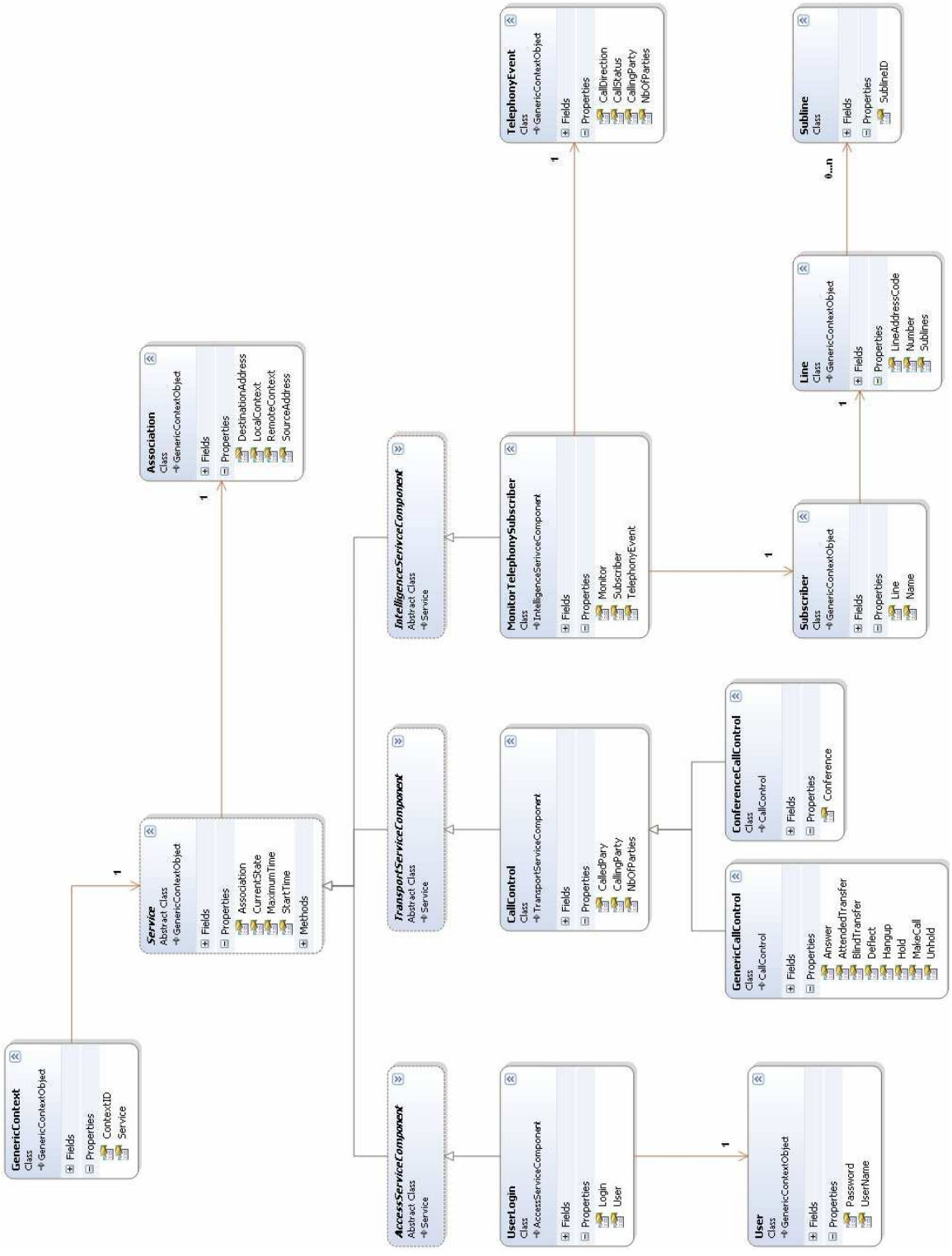
Association class contains a local and remote context IDs. To generate a unique context ID we bound together a time tick and a counter value as following: timetick_countervalue.

- **UserLogin**: contains the "login" attribute which if set to 1 means that the user wants to login and when set to "0" means that the user is logging out. The User object has two attributes; UserName and Password. We used the Windows NT credentials to authenticate a user when he opens his phone bar and logs to the CTI server. These credentials are for example: UserLogin=worldwide\rony and the Password=my_windows_password. In the CTI server, we use the Login object to login to the VCX soft-switch (SIP Proxy). In this case the UserName is a SIP URI (sip:cti.server@corebridge.com:5060) and the Password a VCX required password.

- **CallControl**: the Call Control has two main classes; the Generic Call Control and the Conference Call Control class. While the former allows initiating calls between two parties, the latter handles multi-parties conversations. In the base class Called and Calling party can be extensions or SDA phone numbers, such as 7107 or 0145817107, or SIP URIs (sip:rony.chahine@enst.fr). When a Bob wants to call Alice, a Set request is sent with the "MakeCall" attribute set to 1 and the CallingParty and CalledParty respectively set to Bob and Alice phone numbers.

- **MonitorTelephonySubscriber**: the "monitor" attribute decides if the user wants to monitor (set to 1) or unmonitor (set to 0) a line. The "name" attribute of the Subscriber object describes the owner of the line while the Line object describes the line he wants to monitor. A Line may hold multiple sub-lines, for example a 3COM SIP phone can handle up to 3 sub-lines where each sub-line is responsible for a communication between two end parties.

  The Telephony Event object contains all the Notification attributes. It is filled when a GCSP user receives a GCSP notification. This of course is only possible after the user has asked for a line monitoring. The telephony events are then sent with the Telephony Event object.



[Figure 67] The Generic Context implemented in the GCSP stack

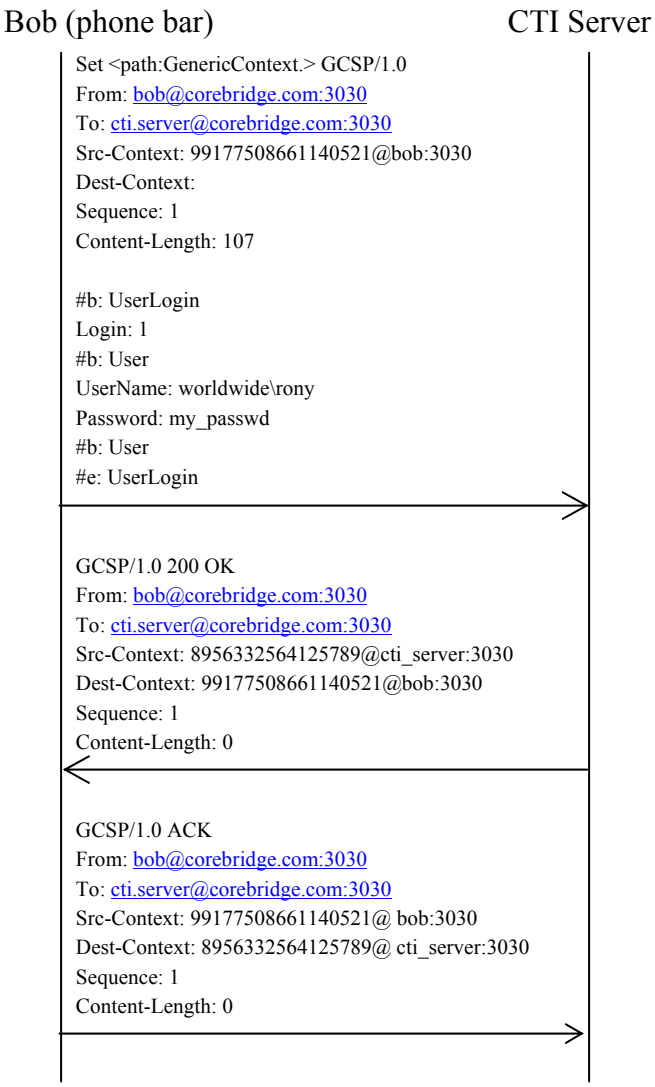
6.3.3 Basic call flows

We now give some basic call flows which involve communication sessions between the phone bar, the CTI server, the signalling mediator and the SIP proxy.

We will therefore detail the login session as well as the monitoring session, and for the call control we will only stop on the make call function. The rest of the call control functions are similar to the make call example.

6.3.3.1 Login session call flows

In the login session, when Bob opens his phone bar, it sends Bob's user name and password to the CTI server as shown in [Figure 68].

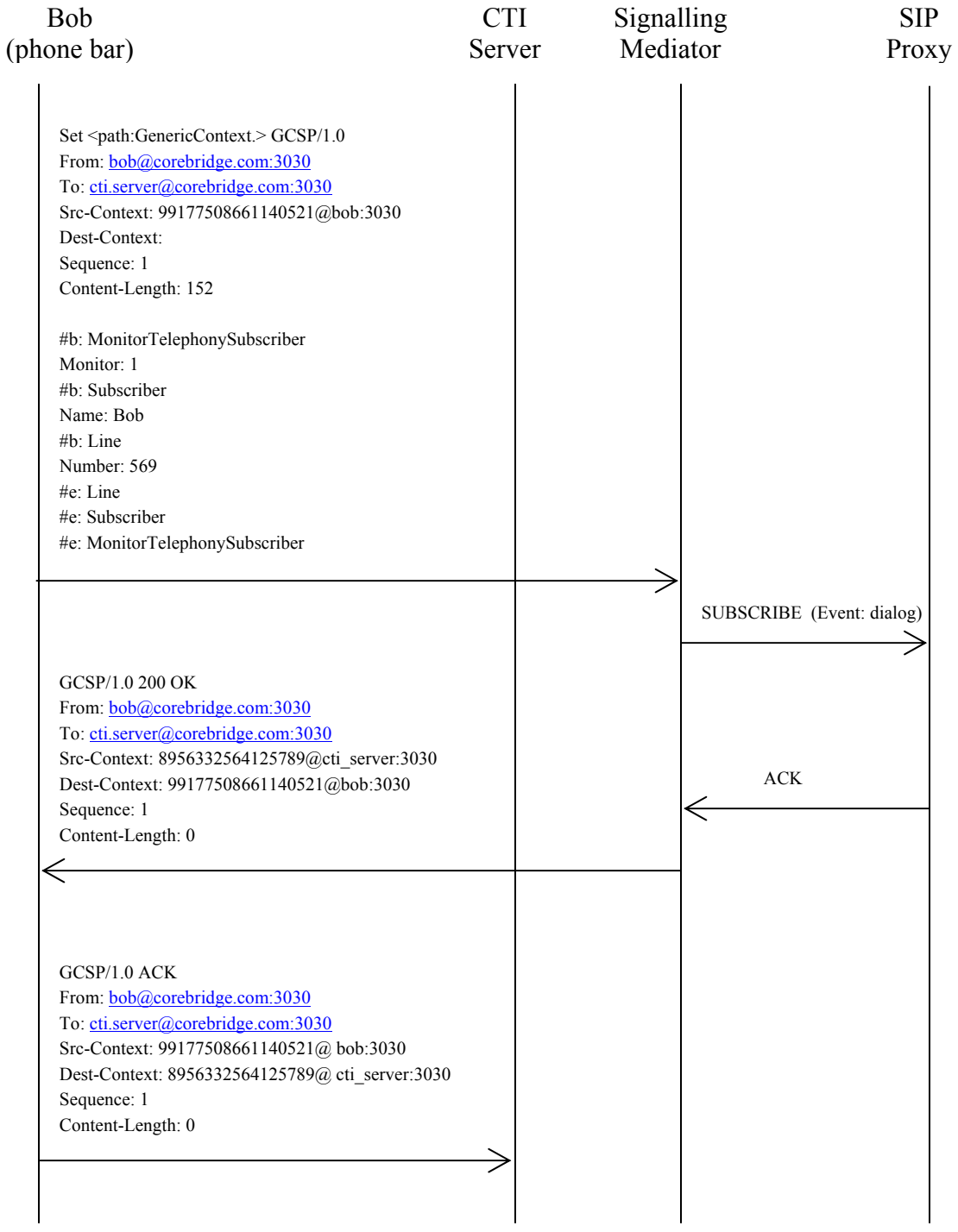


[Figure 68] Login session call flows

6.3.3.2 Monitoring session call flows

In the monitoring session [Figure 69] shown in [Figure 69], the phone bar asks the CTI server to supervise Bob's line extension (569) in order to receive all the telephony events, from the SIP proxy, which are related to this line. When a SIP telephony event is received, the signalling mediator translates it to a GCSP notification then sends it to the phone bar.

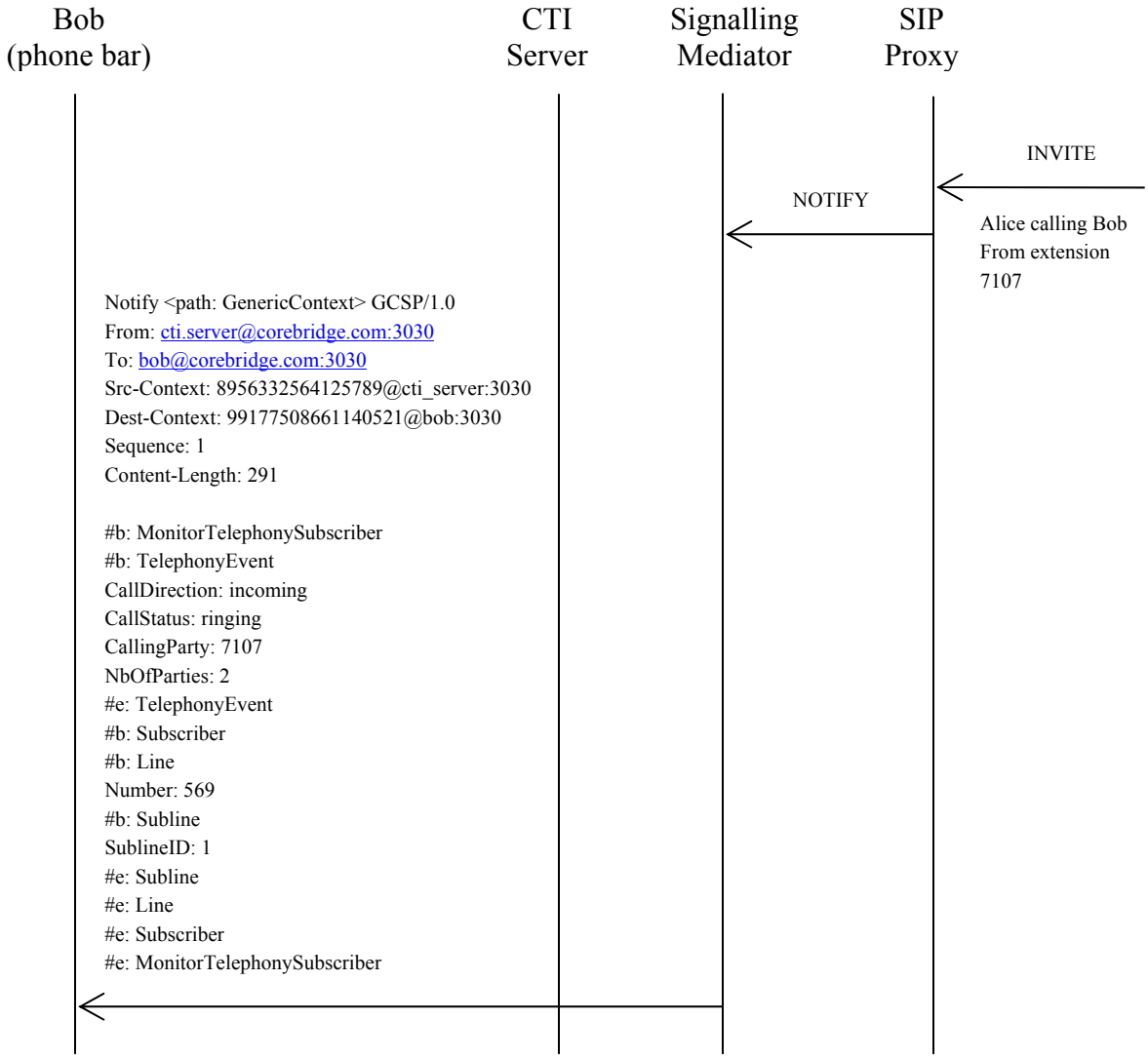
There is a GCSP association between the CTI server and the signalling mediator which is not represented in the figure below.



[Figure 69] Monitoring session call flows

In the example shown in [Figure 70], Alice calls Bob on his phone extension (569). When the SIP proxy receives the Invite command, he fires a Notify event to the signalling mediator which translates it to a GCSP notification.

The GCSP Notification message body shows a call coming from Alice (7107) to Bob on his phone extension (569) in particular on his phone sub-line 1. If Bob receives 2 simultaneous phone calls, the second Notification related to the second call will target his sub-line number 2.



[Figure 70] Event notifications call flows

6.3.3.3 Call flows of a call initiated from the phone bar

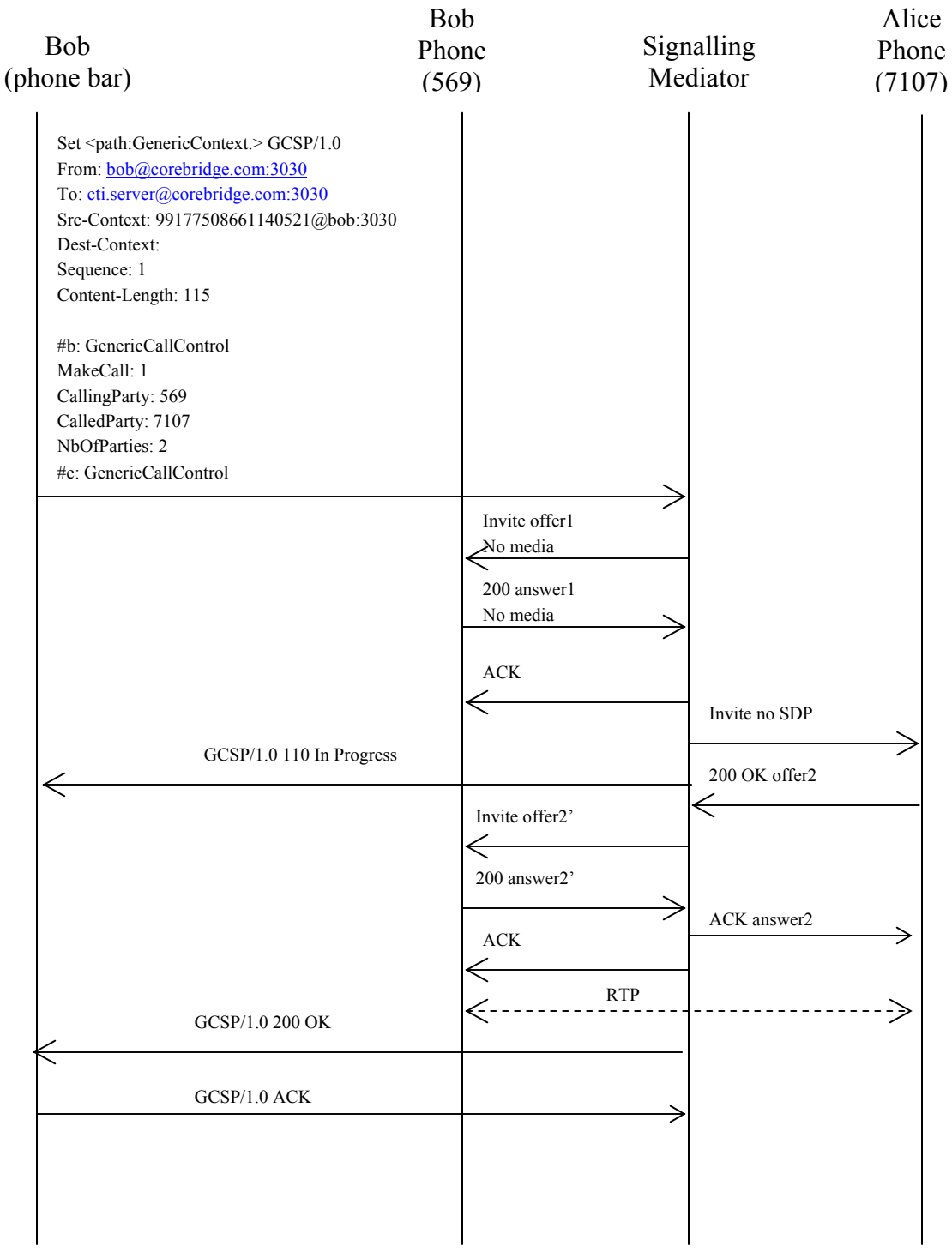
In this example, Bob calls Alice on her SIP phone and initiates the call from his phone bar application.

A GCSP Set request is sent from the phone application to the CTI server which forwards it to the signalling mediator.

The signalling mediator is a statefull B2BUA, it initiates a SIP call between Alice and Bob phone extensions according to a 3rd party call control mechanism described by [paragraph 4.4] in [64].

Because of the time delay that Bob and Alice could cause before answering their phones, the signalling mediator sends periodically “In Progress” GCSP responses to the phone bar before its timers expires and cause to re-initiate another Set request or drop the session.

The SIP proxy is not shown in [Figure 71], it stands between the signalling mediator and the SIP phones while the CTI server stands between the phone bar and the signalling mediator.



[Figure 71] Call flows of a call initiated from the phone bar

CONCLUSION AND FURTHER WORKS

We have now arrived to the end of the description of our PhD work. We will first summarize what we have achieved in this work, we will draw conclusions on its importance and novelty, in our view, and we will outline some further works that our research calls for.

Our PhD work has been dedicated to the control plane, a rarely considered research topic in the telecommunication world. Our motivation to tackle this research problem was the feasibility of multi-provider and cross-network services, a promising development theme for more complete forms of communication, but so far an impossibility with current control plane concepts. It turns out that our contributions have permitted to effectively build an example of such a cross network service.

In order to propose a global control plane well adapted to the new forms of multimedia communication and services, we have approached the current control plane concepts, their requirements and limitations with a new theoretical framework. This new approach has been complemented with an analysis of the very special "cooperative computing" nature of control plane software pointing to far-reaching consequences, in particular on a new interpretation of the role and importance of signalling. From this analysis we came to the conclusion that "data based signalling mechanisms" are better adapted than "command based signalling mechanisms" to this cooperative nature of the control plane.

This led us to the proposal of a new signalling paradigm and a new signalling protocol called GCSP.

The definition of this new signalling protocol, and its application to new cross network CTI services, along with its implementation in an industrial application have made the remaining part of our research work which has therefore been conducted according to the methodology shown in [Figure 72].

Following that methodology we have been able to make the following contributions.

a) First contribution: a new theoretical framework for control plane concepts

We have approached the new generalized multimedia communication services by the communication paradigms they use. This new approach has allowed us to replace the control plane concepts in a new theoretical framework leading to a re-foundation and new abstract definitions of these concepts. In opposition to the former ideas these new definitions are more adapted to the requirements of the new generalized multimedia communication and services. From this new approach, we have analysed the various functions involved in the control plane and we have revealed the fundamental structure between these various activities and underlined the conditions under which they may be operated independently and separately. This leads to generalized unbundling models for the control plane extremely useful to understand the current trend towards the breakout of the former "integrated switch model" that we can witness, and also to understand the advent of the soft-switch architecture and the advent of "session operators" like Skype, strictly unbundled from the connectivity operators. This re-foundation of control plane concepts has never been attempted before in the systematic approach that we use in our work. We have finally used it to outline the generic control plane architecture.

b) Second contribution: the special "cooperative computing" nature of control plane software

With this new perspective on control activities, and a new understanding of their fundamental nature, we could approach now the research problems they raise. For this we found that the control plane specificity came from the special "cooperative computing" nature of its software and from the special requirements of this type of software. One of these specific requirements is the information sharing between cooperating partners, achieved in the control plane by means of signalling. This led us to underline the very fundamental role of signalling in a cooperative computing environment and to a constructive redefinition of the concept of signalling. We then reviewed the current signalling concepts and their difficulties in achieving cross-network and multi-provider

services which is a severe limitation to innovative multimedia communication and services. We finally analysed the current research efforts such as NSIS and IMS, and showed their limitation in providing a global control plane for such multimedia communication and services.

c) Third contribution: a change of paradigm in signalling

After our re-foundation of control plane concepts and the outlining of the special control plane requirements, we came to the conclusion that the current "command based" signalling mechanisms used in the control plane are not optimal in view of the cooperative nature of control plane activities and should be replaced by "data based signalling mechanisms". We are indeed in a paradox situation where typical centralized computing applications like management use "data-based" protocols (like SNMP) when "command based" protocols would be more adapted, and typical cooperative computing applications like the control activities use "command based" signalling protocols when "data based" signalling protocols would be more adapted! It is an upside world! Today, management engineers have realized this situation for their field and are proposing to shift to "command based" protocols for management. On our side, the control side, we propose a radical shift and a change of paradigm where signalling will be done according to a "data based" mechanism rather than a command based mechanism. In this new approach, signalling becomes the sharing of information between control processes instead of sending and receiving commands. In this new data based signalling paradigm, control processes communicate by means of simple Get/Set/Notify commands to share and modify instance-data of their communication context. The data based approach of signalling is better adapted to the control of multimedia communication and services and it facilitates the achievement of cross-network and multi-provider services.

d) Fourth contribution: the definition of a Generic Context Data Structure

The underlying assumption of the data based mechanism is that partner control processes of a conversational service instance need to understand the structure of the data in the other partners local contexts. Thus, we proposed a generic structure for control process's local context that we called the "Generic Context" and we produced a formal description of it using UML modelling. The structuring tool that we have used is the extended SIMSPON unbundling model.

We also defined a new association mechanism by means of an association table, called CAT. The CAT is taken into account into the Generic Context design; it describes the local contexts association and gives a global view of the service session.

e) Fifth contribution: GCSP: a universal, cross-network and multi-domain signalling protocol

At this point we were in a position to propose an end to the nightmare of the innumerable signalling protocols (for each domain and each network). We proposed a unique data based signalling protocol called GCSP (Generic Context Sharing Protocol) that works for all the signalling domains and for all the networks. GCSP is a text based protocol used by control processes to share Generic Context instance-data by means of simple Get/Set/Notify commands. We have given a complete description of the GCSP protocol objectives and mechanisms and we have proposed an implementation of a GCSP protocol stack over UDP.

f) Sixth contribution: a new cross-network CTI GCSP based architecture

After the definition of our new GCSP signalling protocol we wanted to validate its ability to facilitate the introduction of cross-network services. The occasion was given by a request of the Corebridge Company to extend its customer profile service to public phones in addition to the company phones normally benefiting from this type of CTI service. By means of the GCSP protocol we have been able to design a new CTI architecture which provides a cheap installation cost, easy service mobility and support of new types of PBX and service technologies such as SIP based IPBXs and Parlay services. In this new GCSP based architecture, the customer profile service formerly available only to company phones are now extended to public phones, including mobile phones and PDAs illustrating by this the effectiveness of the GCSP concept in the achievement of cross-network services.

g) Seventh contribution: a network of signalling mediators

In addition to the concept of signalling mediators we have proposed to use GCSP as an intermediate Protocol reducing the “many to many” mediation problem to a “one to many” problem. According to this view, we have proposed a signalling mediation achieved by a network of signalling mediators. A signalling mediator translates from GCSP to a target signalling protocol (such as SIP or H323). We used one of these signalling mediators in the implementation of the GCSP stack in Corebridge CTI architecture.

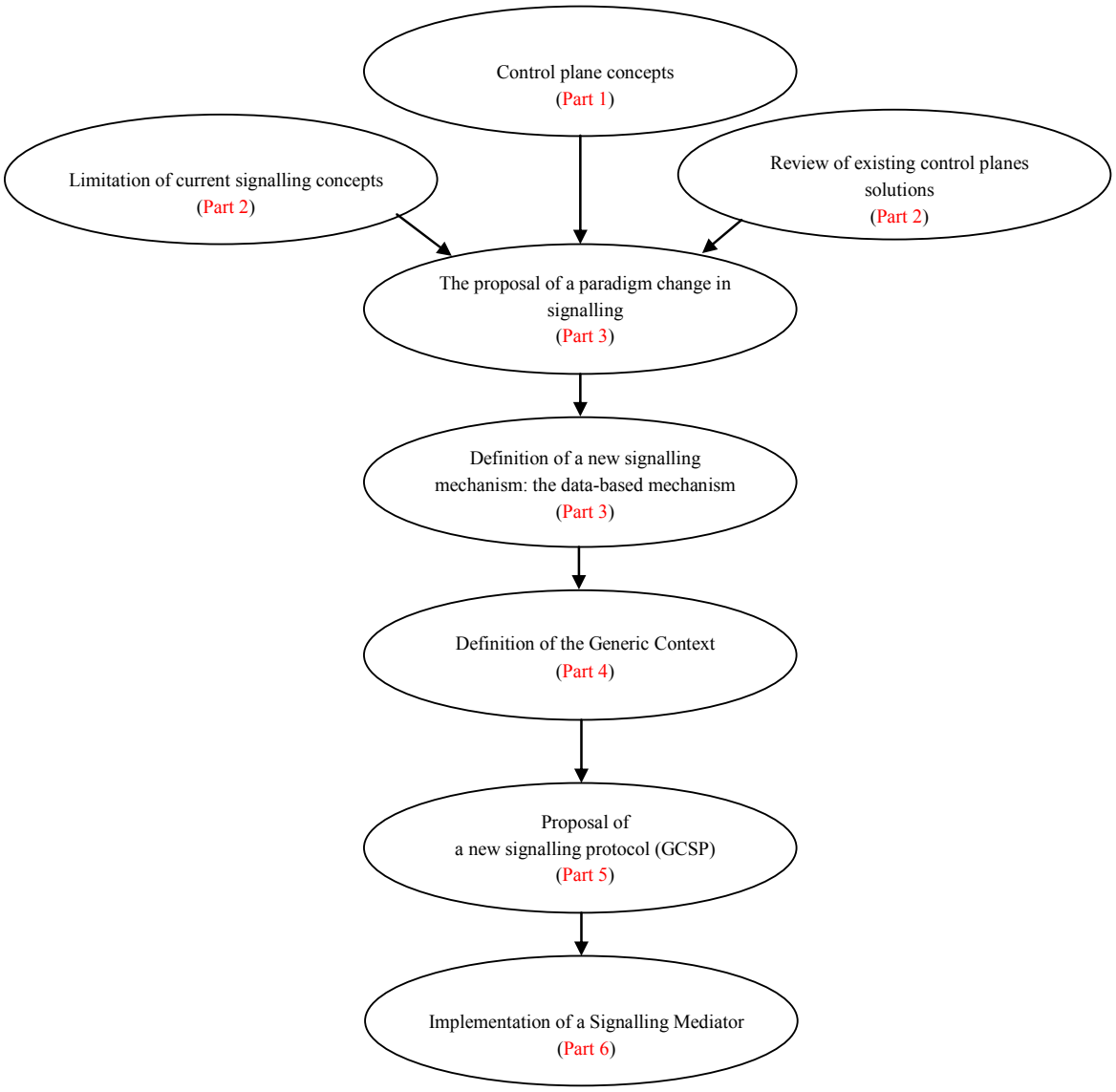
e) Implementation

Finally, we have put our theoretical ideas into practice for the Corebridge cross-network services between the telephone network and the data network. We then implemented the

GCSP protocol stack in Corebridge CTI applications and the GCSP/SIP signalling mediator.

Further works

At this point of our work we have proven the feasibility of our concepts and showed how useful they are in practical implementations such as the Corebridge implementation mostly centred in the intelligence domain. Of course a long way remains before we see the GCSP signalling protocol becoming the universal signalling system. Further works would consist in achieving GCSP implementations in all the other signalling domains: Access, and Session, and also on using GCSP on other transport networks than UDP. A transport network of particular interest would be the NSIS signalling network as defined by the IETF. This would be an interesting development which could establish GCSP as the choice protocol for the Application, signalling layer of NSIS.



[Figure 72] Research methodology

LIST OF ACRONYMS

CID	Call Instance Data
CIM	Common Information Model
CMIP	Common Management Information Protocol
CORBA	Common Object Request Broker Architecture
CSTA	Computer Supported Telephony Applications
GC	Generic Context
GCSP	Generic Context Sharing Protocol
ISUP	ISDN User Part (SS7)
MIB	Management Information Base
MEGACO	MEdia GAteway COntrol protocol
MOM	Message Oriented Middleware
NETCONF	Network Configuration
NMS	Network Management Station
NSIS	Next Step In Signaling
POTS	Plain Old Telephone Service
RMI	Remote Method Invocation
SDP	Session Description Protocol
SIMPSON	Signaling Model for Programmable Services Over Networks
SIP	Session Initiation Protocol
SNMP	Simple Network Management Protocol
SSD	Service Support Data
TAPI	Telephony Application Programming Interface
TCAP	Transaction Capability Application Part (SS7)

REFERENCES

[1] Christophe Bezault, “Les middlewares orientés messages: Parnorama de l’offre et exemples avec MQSeries”, Dunod, Paris, 2001

[2] G.Coulouris, J.Dollimore,T.Kindberg, “Distributed Systems: Concepts And Design”, fourth edition 2005, Addison-Wesley

[3] David Gelernter, “Generative Communication in Linda”, ACM Transactions on Programming Languages and Systems, Vol. 7, No. 1, January 1985.

[4] http://java.sun.com/developer/products/jini/

[5] ITU-T Q.771_Q775 : TCAP : Transaction Capability

[6] CORBA : www.corba.org

[7] ITU-T recommendation Q 1701 Signalling requirements for IMT-2000 networks

[8] Global Functional Plane for Intelligent Network CS1. ITU-T Recommendation Q1213

[9] IETF RFC 3261: “Session Initiation Protocol (SIP)“, June 2002

[10] ITU-T Recommendation H. 323

[11] ITU-T Recommendation Q1901: BICC: Bearer Independent Call Control

[12] TINA-C Consortium

[13] ETSI ES 201 915 Phase 1: "Open Service Access (OSA); Application Programming Interface (API); all parts", 02/2002

[14] ETSI ES 202 915 Phase 2: "Open Service Access (OSA); Application Programming Interface (API); all parts", 07/2002

[15] ETSI ES 203 915 Phase 3: "Open Service Access (OSA); Application Programming Interface (API); all parts", 10/2002

[16] Philippe Martins, “Architecture de contrôle et middleware pour les réseaux de prochaines générations et évaluation de performances”, PHD thesis, ENST Paris, April 2000

[17] Guillaume Buridant, “Un modèle de signalisation générique pour les réseaux IP de nouvelle génération : unification par RSVP”, PHD these, L’Université de Versailles Saint Quentin en Yvelines, 2004

[18] Global Functional Plane for Intelligent Network CS1, ITU-T Recommendation Q1213

[19] B-ISDN Signalling Capability Set 3, UIT-T

[20] IETF RFC 2205: “Resource ReSerVation Protocol (RSVP)”, Version 1 Functional Specification

[21] ITU-T Recommendation Q.764, “Signalling system No. 7 – ISDN user part signalling procedures”, 12/1999

[22] IETF RFC 2327: “SDP: Session Description Protocol”, April 1998

[23] Digital Cellular Telecommunications System (Phase 2+); Mobile Application Part (MAP) Specification Norme GSM 09.02 (v.7.5.0)

[24] Intelligent Network Application Part. Normes Q.1200 – Q.1999

[25] IETF RFC 3015: Megaco Protocol Version 1.0, Nov. 2000

[26] Skype : http://www.skype.com

[27] TISPAN: http://www.item.ntnu.no/fag/ttm4130/NGN_Workshop_Protocol.ppt

[28] Astronefs: “Network and Telecommunication Global Service Convergence: White paper”, http://www.infres.enst.fr/~rigault/white-paper.pdf

[29] C.Rigault, R.Chahine, “New signalling mechanisms for multi-provider and cross-network services”, Softcom 2004, Split, Croatia, October 2004

[30] www.parlay.org

[31] Soft-switch architecture : IETF RFC 2705: Media Gateway Control Protocol (MGCP), Version 1.0 Oct. 1999, IETF RFC 3015: Megaco Protocol Version 1.0, Nov. 2000

[32] Ericsson’s Service Capability Server Parlay Gateway http://www.ericsson.com/mobilityworld/sub/open/technologies/parlay/about/parlay_about_gs1

[33] CAMEL: ETSI recommendation TS 101 046 (V7.0.0)

[34] IETF RFC 2458: “Toward the PSTN/Internet Inter-Networking--Pre-PINT Implementations”, November 1998

[35] IETF RFC 3910: “The SPIRITS (Services in PSTN requesting Internet Services) Protocol”, October 2004

[36] ITU Recommendation E.723, "Telephone network and ISDN quality of service, network management and traffic engineering", Telecommunication Standardization Sector of ITU, Geneva, Switzerland, 1992.

[37] ITU Recommendation E.721, "Network grade of service parameters and target values for circuit-switched services in the evolving isdn,", Telecommunication Standardization Sector of ITU, Geneva, Switzerland, May 1999.

[38] V. A. Bolotin, P. J. Kuhn, C. D. Pack, and R. A. Skoog, "Common channel signalling networks: Performance, engineering, protocols and capacity management," IEEE Journal on Selected Areas in Communications,Vol. 12, pp. 377–544,Apr. 1994. Special issue.

[39] International Telecommunication Union, "Message transfer part signalling performance," RecommendationQ.706, Telecommunication Standardization Sector of ITU, Geneva, Switzerland, Mar. 1993.

[40] International Telecommunication Union, "Signalling performance in the telephone application," Recommendation Q.725, Telecommunication Standardization Sector of ITU, Geneva, Switzerland, Mar. 1993.

[41] R. A. Skoog, "Engineering common channel signalling networks for ISDN," in Twelfth International Teletraffic Congress, Vol. 2, (Torino), pp. 1–7 (2.4A,), June 1988.

[42] Y. Rekhter and T. Li, "A border gateway protocol 4 (BGP-4)," Request for Comments (Draft Standard) 1771, Internet Engineering Task Force, Mar. 1995.

[43] Tony Eyers and Henning Schulzrinne, "Predicting Internet Telephony Call Setup Delay", in IP-Telephony Workshop (IPtel), Berlin, Germany, April 2000

[44] W. R. Stevens, "TCP/IP illustrated: the implementation", Vol. 2. Reading, Massachusetts: Addison-Wesley, 1994.

[45] H. Schulzrinne, J. Rosenberg, "Signalling for Internet Telephony", February 2, 1998

[46] A. Silberschatz and P.B. Galvin, "Operating system concepts" 5th edition, Addison-Wesley, 1998

[47] IETF RFC 3726, "Requirements for Signalling Protocols", April 2004

[48] IETF RFC 4080, "Next Steps in Signalling (NSIS): Framework", June 2005

[49] Henning Schulzrinne, Jonathan Rosenberg and Jonathan Lennox, "Interaction of Call Setup and Resource Reservation Protocols in Internet Telephony", Jun. 15, 1999, Technical Report

[50] M.Poikselka, G.Mayer, H.Khartabil, A.Niemi, "the IMS: IP Multimedia concepts and services in the mobile domain", John Wiley & Sons, Ltd, 2004

[51] Aiko Pras, PHD Thesis: "Network Management Architectures", 17 February 1995

[52] IETF RFC 1157, "A Simple Network Management Protocol (SNMP)", May 1990

[53] ISO documents 9595, 9596 and ITU.700, X.711

[54] Leslie Lamport, "Verification and Specification of Concurrent Programs", 16 November 1993

[55] W. Stallings: “SNMP, SNMPv2 and CMIP – The Practical Guide to Network Management Standards”, Addison Wesley

[56] IETF draft: http://www.ietf.org/internet-drafts/draft-ietf-netconf-prot-12.txt

[57] B. Schilit, N. Adams, and R. Want, “Context-Aware Computing Applications,” Proc. IEEE Workshop Mobile Computing Systems and Applications, pp. 85-90, 1994.

[58] A.K. Dey, “Understanding and Using Context,” J. Personal and Ubiquitous Computing, vol. 5, no. 1, pp. 4-7, Feb. 2001

[59] BlackBerry, http://www.blackberry.com

[60] ITU-T Recommendation Q1214, “Ditributed Functional Plane for Intelligent Network CS-1”, 10/95

[61] 3GPP-TS 29.228 V6.8.0, “IP Multimedia Subsystem (IMS); Stage 2 (Release 6)”, 2004-12

[62] ETSI ES 201 915-4 V1.4.1, “Open Service Access (OSA); Application Programming Interface (API); Part 4: Call Control SCF”, 2003-07

[63] IETF RFC 2396, “Uniform Resource Identifiers (URI): Generic Syntax”, August 1998

[64] IETF RFC 3725, “Best Current Practices for Third Party Call Control (3pcc) in the Session Initiation Protocol (SIP)”, April 2004