(12) DEMANDE INTERNATIONALE PUBLIÉE EN VERTU DU TRAITÉ DE COOPÉRATION EN MATIÈRE DE BREVETS (PCT)

(19) Organisation Mondiale de la Propriété Intellectuelle
Bureau international

(43) Date de la publication internationale
9 décembre 2004 (09.12.2004)

PCT

(10) Numéro de publication internationale
WO 2004/107714 A1

(51) Classification internationale des brevets[7] : H04L 29/08

(21) Numéro de la demande internationale :
PCT/FR2004/001310

(22) Date de dépôt international : 26 mai 2004 (26.05.2004)

(25) Langue de dépôt : français

(26) Langue de publication : français

(30) Données relatives à la priorité :
03.06368 26 mai 2003 (26.05.2003) FR
04.02878 19 mars 2004 (19.03.2004) FR

(71) Déposant *(pour tous les États désignés sauf US)* : **WAVECOM** [FR/FR]; Immeuble Bord de Seine I, 3, Esplanade du Foncet, F-92442 Issy-les-Moulineaux Cedex (FR).

(72) Inventeur; et

(75) Inventeur/Déposant *(pour US seulement)* : **MOULY, Michel** [FR/FR]; 4, rue Elisée Reclus, F-91120 Palaiseau (FR).

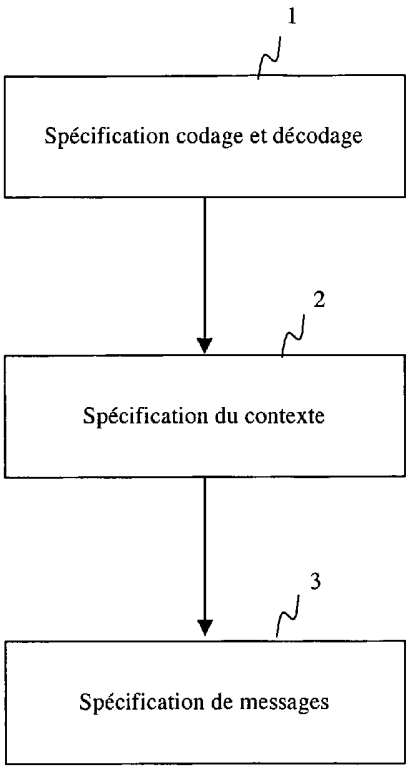
(74) Mandataire : **VIDON, P.**; Cabinet Patrice Vidon, 16B, rue Jouanet, BP 90333, F-35703 Rennes Cédex 7 (FR).

(81) États désignés *(sauf indication contraire, pour tout titre de protection nationale disponible)* : AE, AG, AL, AM, AT, AU, AZ, BA, BB, BG, BR, BW, BY, BZ, CA, CH, CN, CO, CR, CU, CZ, DE, DK, DM, DZ, EC, EE, EG, ES, FI, GB, GD, GE, GH, GM, HR, HU, ID, IL, IN, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MA, MD, MG, MK, MN, MW, MX, MZ, NA, NI, NO, NZ, OM, PG, PH, PL, PT, RO, RU, SC, SD, SE, SG, SK, SL, SY, TJ, TM, TN, TR, TT, TZ, UA, UG, US, UZ, VC, VN, YU, ZA, ZM, ZW.

[Suite sur la page suivante]

(54) **Title:** SYSTEM FOR SPECIFYING AND IMPLEMENTING A COMMUNICATION AND TRANSMISSION PROTOCOL

(54) **Titre :** SYSTEME DE SPECIFICATION ET DE MISE EN OEUVRE DE PROTOCOLE DE COMMUNICATION ET DE TRANSMISSION,



1 ... ENCODING AND DECODING SPECIFICATION
2 ... CONTEXT SPECIFICATION
3 ... MESSAGE SPECIFICATION

(57) Abstract: The invention relates to a system for specifying and implementing a protocol for managing communication and transmission between at least one transmitter and at least one receiver. The invention comprises the following steps consisting in: determining a common context (2); identifying protocol messages (3); specifying representations of stored and/or transmitted data from the description of a common context and the description of the messages; specifying protocol implementation instructions; and implementing said protocol in at least one of the transmitters and at least one of the receivers.

(57) Abrégé : L'invention concerne une spécification et une mise en oeuvre d'un protocole de gestion de communication et de transmission entre au moins un émetteur et au moins un récepteur, comprenant les étapes suivantes : - détermination d'un contexte commun (2) ; - identification des messages de protocole (3) ; - spécification des représentations de données stockées et/ou transmises à partir de la description d'un contexte commun et de la description des messages ; - spécification d'instructions d'implémentation du protocole ; et - mise en oeuvre dudit protocole dans au moins un des émetteurs et au moins un des émetteurs.

WO 2004/107714 A1

(84) États désignés *(sauf indication contraire, pour tout titre de protection régionale disponible)* **:** ARIPO (BW, GH, GM, KE, LS, MW, MZ, NA, SD, SL, SZ, TZ, UG, ZM, ZW), eurasien (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), européen (AT, BE, BG, CH, CY, CZ, DE, DK, EE, ES, FI, FR, GB, GR, HU, IE, IT, LU, MC, NL, PL, PT, RO, SE, SI, SK, TR), OAPI (BF, BJ, CF, CG, CI, CM, GA, GN, GQ, GW, ML, MR, NE, SN, TD, TG).

Publiée :

— *avec rapport de recherche internationale*

— *avant l'expiration du délai prévu pour la modification des revendications, sera republiée si des modifications sont reçues*

En ce qui concerne les codes à deux lettres et autres abréviations, se référer aux "Notes explicatives relatives aux codes et abréviations" figurant au début de chaque numéro ordinaire de la Gazette du PCT.

SYSTÈME DE SPÉCIFICATION ET DE MISE EN OEUVRE DE PROTOCOLE DE COMMUNICATION ET DE TRANSMISSION

La présente invention se rapporte au domaine des protocoles de communications.

Plus précisément, l'invention concerne la génération automatique de spécification de protocoles de communication entre un ou plusieurs émetteurs et un ou plusieurs récepteurs.

Le développement de protocoles de signalisation dans le domaine des télécommunications comporte une phase de spécifications, une phase de développement des logiciels dans les entités distantes qui communiquent en appliquant ces protocoles, suivi souvent d'une phase de test.

Dans beaucoup de cas, les entités communicantes ne sont pas développées par les mêmes équipes ou entreprises (par exemple l'infrastructure fixe et les mobiles dans le cas d'un réseau de télécommunications pour mobiles). Dans ces cas, les spécifications ne sont pas, en général, rédigées par les équipes de développement, mais par des groupes composés de personnes venant de différentes entreprises, typiquement parmi celles qui vont développer les équipements, ou encore les acheter pour les mettre en œuvre.

On observe en pratique un décalage important entre les méthodes et approches des groupes de spécification, comparées avec celles des équipes de développement. Au pire, la spécification d'un protocole est rédigée en langage naturel, n'est pas complète, contient des incohérences, et demande pour être comprise de nombreuses informations dispersées dans d'autres documents.

A l'opposé, un logiciel une fois réalisé est une instance formelle, complète et logiquement cohérente de la spécification du protocole, et ce tout en pouvant être erroné, au sens où il ne permet pas un interfonctionnement tel que prévu par le groupe de spécification avec une entité distante paire développée indépendamment.

Ce décalage, outre les risques de réalisation erronée, de problèmes de multiplicité de représentation et de problèmes d'évolution qu'il entraîne, est la source d'un important travail d'analyse et d'exégèse, et plus généralement d'efforts de compréhension de la part des équipes de développement.

Ce travail de reconstruction n'apparaît pas intrinsèquement nécessaire, et est dupliqué par chaque équipe de développement. Cela est une source de coûts inutiles, et d'inefficacité de la chaîne prise dans son ensemble.

Une autre source de surcoûts vient de ce que les spécifications de protocoles intéressant un même objet (par exemple, un portable GSM) ne sont pas rédigées par un seul groupe. La variété des approches rédactionnelles suivies par ces différents groupes limite fortement la possibilité d'unifier l'approche de développement. Cela est une source d'inefficacité du développement lui-même, des tests, ainsi que du volume de code généré (il est difficile de mettre en commun ce qui intrinsèquement peut l'être, simplement parce que spécifié différemment).

Il apparaît donc utile d'étudier comment réduire ce travail intermédiaire, par l'introduction de méthodes formelles et unificatrices. De telles méthodes peuvent être dans un premier temps appliquées à une équipe de développement, puis, dans la mesure du possible aux spécifications elles-mêmes.

On connaît dans l'état de la technique quelques méthodes publiées allant dans ce sens, notamment : ASN.1, XML, CSN.1 et SDT. Ces différentes méthodes et les langages formels associés ne correspondent pas tous aux mêmes parties de la spécification d'un protocole. Il est donc nécessaire de rentrer un peu plus dans le détail de ce qu'est un protocole et sa spécification.

Un protocole est essentiellement un langage permettant à deux machines distinctes et distantes d'échanger des informations en utilisant *in fine* un médium de transmission. En pratique ce langage est organisé en différentes parties plus ou moins indépendantes, chacune étant considérée elle-même comme un protocole. Les protocoles sont alors arrangés en couches, chaque protocole à l'exception du « plus bas » utilisant non pas directement le médium de transmission mais un service de transmission fourni par le protocole « directement inférieur ».

A l'instar des langages humains, les protocoles peuvent être analysés du point de vue du vocabulaire ou lexique (liste de types de messages, listes de parties de message), de l'encodage (phonèmes et morphèmes pour les langages humains, formes d'onde et encodage numérique pour les protocoles), de la syntaxe (règles de grammaire), de la sémantique (sens des messages et des parties de message), des règles de procédure (règles régissant les dialogues entre les entités dialoguant ; c'est cette partie qu'évoque le mot « protocole »), et enfin de la praxis (effet des messages).

Certains de ces aspects sont bien déterminés, et apparaissent en clair dans la majorité des spécifications (le lexique, l'encodage, les règles de procédures). D'autres sont souvent mal perçus et, de ce fait, spécifiés sans méthode (sémantique et praxis).

Dans une majorité de spécifications, on peut identifier deux parties distinctes : le lexique et l'encodage d'une part, et la spécification des « procédures ». Le lexique apparaît sous la forme de listes des messages et de parties de messages (« élément d'information »), chacun avec la grammaire et l'encodage associé. Les règles de procédure sont spécifiées comme une liste de procédures plus ou moins indépendantes, décrites comme les actions entraînées par la réception de messages venant de l'entité paire ou de commandes venant d'autres logiciels dans la même machine. Cela couvre le plus souvent aussi bien les règles de procédure (actions et réactions visibles, et donc testables, dans le flot d'information circulant entre les entités paires) que la « praxis » plus générale (actions internes non nécessairement visibles). La spécification de la sémantique est dispersée dans la description des éléments lexicaux, dans la description des procédures, et souvent dans d'autres documents.

Les méthodes formelles qu'on peut trouver dans les publications se concentrent sur la formalisation de l'encodage et la syntaxe d'une part (ASN.1, CSN.1, XML), et sur les procédures vues comme des séquences temporelles d'action (SDT, SDL).

ASN.1 (Abstract Notation 1) est un langage formel spécifié par l'ISO (série X.680 et X.690) permettant de décrire des structures de données en distinguant une description abstraite (i.e., pouvant s'appliquer à différentes méthodes d'encodage) et des méthodes d'encodage de ces structures de données. Il est appliqué couramment à la description de la structure des messages d'un protocole, chaque message étant considéré comme une structure de donnée. Parce que formelle, une spécification écrite en ASN.1 peut être traduite automatiquement en un logiciel d'encodage et de décodage de messages.

ASN.1 a été récemment étendu pour permettre la description formelle de méthodes d'encodage, dans le but d'appliquer ASN.1 à des spécifications soit non écrite originellement en ASN.1, soit pour lesquelles les méthodes générales d'encodage spécifiées par l'ISO ne sont pas satisfaisantes pour une raison ou une autre.

XML est un langage formel dont les buts sont proches de ceux d'ASN.1 en ce qu'il permet de décrire des structures de données et donc des messages quand on les limite à une structure de données. Comme ASN.1, XML permet la génération automatique de logiciels d'encodage et décodage.

CSN.1 est un langage formel développé par l'inventeur, et utilisé dans quelques spécifications de protocoles de l'ETSI. Ce langage permet de décrire la grammaire de chaînes de bits (et s'apparente beaucoup plus aux langages de description de grammaire formelle comme la BNF, que des langages de description de données), et peut être utilisé pour décrire la structure en tant que chaîne de bits des messages de la majorité des protocoles de télécommunications. Il se distingue dans ces buts de deux manières de l'ASN.1 ou de XML : il peut être utilisé a posteriori, pour décrire des messages dont l'encodage est décrit de manière quelconque dans une spécification, et il ne cherche pas à aborder même partiellement la sémantique liée à la structure. Comme dans le cas des langages précédents, une spécification en CSN.1 peut être utilisée pour la génération automatique de l'encodage et du décodage de messages. Par rapport à ASN.1 et XML, CSN.1 présente l'avantage de s'appliquer à tout protocole, et donc de

permettre une réalisation unifiée de la partie correspondante de la réalisation du protocole. En contrepartie, CSN.1 ne décrit pas les messages en tant que structures de données, et cette liaison ne peut donc pas être générée automatiquement.

Dans un tout autre domaine, SDT ou SDL (ainsi que d'autres méthodes apparentées) s'adressent à la formalisation des procédures vues comme séquences temporelles d'action. Ces méthodes sont centrées chacune autour d'un langage formel décrivant des événements (réception de messages ou de commandes internes) et des séquences d'action initiées par ces événements (envoi de messages ou de réponses internes, mémorisation de données, transition d'états, ...). Ces méthodes mettent l'accent principalement sur la vision des entités des protocoles comme des automates d'états, un état étant un résumé des aspects du contexte (données mémorisées) de chacune des entités paires qui interviennent dans le choix de l'action à entreprendre suite à chaque événement. Ainsi, ces méthodes permettent de formaliser principalement les règles de procédure et, dans une certaine mesure, la praxis, et enfin, indirectement, la sémantique des messages.

Un inconvénient de cette technique de l'art antérieur est que les méthodes publiées jusqu'à présent souffrent de nombreuses limitations. En premier lieu elles ne permettent qu'une formalisation très partielle des spécifications de protocoles, principalement l'encodage des messages et parties de message.

Une autre limitation très importante, particulièrement visible dans le cas de l'ASN.1, est qu'elles ne permettent pas en général d'intégrer dans une même approche des protocoles qui ne sont pas spécifiés initialement suivant ladite méthode. CSN.1 est ici une exception.

Quant à SDL et apparentées, ces méthodes ne capturent que la notion d'automate ; or les protocoles récents, à l'exemple de ceux faisant partie des spécifications de l'UMTS (de l'anglais « Universal Mobile Telecommunications System » ou « Système de télécommunication Mobile Universel » en français), sont des automates très simples, la complexité étant plutôt dans les structures de données et les algorithmes d'encodage.

Un autre inconvénient des spécifications en langage formel selon l'état de l'art ne donnant pas de forme de signal est qu'elles conduisent généralement à des formes de signal inefficaces.

L'invention selon ses différents aspects a notamment pour objectif de pallier ces inconvénients de l'art antérieur.

Plus précisément, un objectif de l'invention est d'optimiser la spécification de protocole de communication et sa mise en œuvre concrète.

Un autre objectif de l'invention est de permettre un développement rapide et efficace de piles de protocoles de communication, notamment dans le domaine des communications mobiles.

L'invention a également pour objectif de permettre la génération automatique d'une partie substantielle des logiciels nécessaires à la mise en œuvre du protocole.

L'invention a aussi pour objectif de limiter l'intervention humaine dans les différentes phases de mise en oeuvre d'un protocole de communication dans un émetteur et/ou un récepteur, afin notamment dans réduire la durée et de limiter les erreurs humaines pouvant entraîner un fonctionnement de l'émetteur et/ou de du récepteur non optimal.

Dans ce but, l'invention propose un système de spécification et de mise en œuvre d'un protocole de gestion de communication et de transmission entre au moins un émetteur et au moins un récepteur, comprenant:

- des moyens de détermination d'un contexte commun comprenant des moyens de description d'un ensemble de données gérées par le ou les émetteurs et/ou leou les récepteurs ;
- des moyens d'identification des messages de protocole susceptibles d'être échangés entre le ou les émetteurs et le ou les récepteurs, l'identification étant indépendante de la ou des représentations utilisées pour les messages;
- des moyens de définition de représentations de données stockées et/ou transmises par le ou les émetteurs et/ou le ou les récepteurs à partir de

la détermination d'un contexte commun et de l'identification des messages, la définition de représentations permettant de déterminer des instructions d'implémentation du protocole dans au moins un des émetteurs et au moins un des récepteurs ; et

- des moyens de mise en œuvre du protocole dans au moins un des émetteurs et au moins un des récepteurs à partir des instructions d'implémentation, de façon à ce que le ou les émetteurs soient aptes à communiquer avec le ou les récepteurs.

On note qu'un protocole de gestion de communication s'entend ici au sens strict et ne comprend notamment pas les protocoles de synchronisation de bases de données qui ici ne sont pas considérés comme des protocoles de gestion de communication.

Selon une caractéristique particulière, le système est remarquable en ce que les moyens de détermination d'un contexte commun comprennent eux-mêmes des moyens de description d'une structure du contexte, la description étant indépendante de la ou des représentations utilisées pour le stockage et/ou la transmission des données.

La notion d'indépendance signifie ici qu'on peut changer et/ou modifier la représentation de stockage sans invalider la représentation abstraite.

Selon une caractéristique particulière, le système est remarquable en ce que les moyens de détermination d'un contexte commun comprennent en outre des moyens d'identification et de description des types d'objets permettant d'analyser la structure du contexte, l'identification et la description des types d'objets étant indépendantes de la ou des représentations utilisées pour le stockage et/ou la transmission.

Selon une caractéristique particulière, le système est remarquable en ce que tout ou partie des données gérées sont les données que le protocole vise à maintenir cohérent dans au moins un émetteur parmi le ou les émetteurs et au moins un récepteur parmi le ou les récepteurs.

La notion de données cohérentes signifie ici que ces données ont une identité sémantique mais pas nécessairement une identité de représentation. Par exemple, une fréquence exprimée en MHz peut avoir la même signification qu'une fréquence exprimée en Hz alors que la représentation est différente.

Selon une caractéristique particulière, le système est remarquable en ce qu'au moins une partie de l'identification des messages est effectuée en termes d'actions et/ou de ses effets sur le contexte commun.

Selon une caractéristique particulière, le système est remarquable en ce qu'il comprend des moyens de liaison entre :

- une description de la forme de signal effectivement transmis ; et
- l'identification des messages de protocole.

Selon une caractéristique particulière, le système est remarquable en ce que la forme du signal appartient au groupe comprenant :

- les suites de données binaires ;
- les suites d'éléments pris dans un alphabet prédéterminé; et
- les suites de formes d'ondes modulées.

Ainsi, le signal peut être associé à des types de symboles très divers.

Selon une caractéristique particulière, le système est remarquable en ce que la liaison se fait dans un langage de spécification de données comprenant des fonctions permettant la description des données sous forme d'attributs d'objets abstraits et de fonctions s'appliquant sur des représentations concrètes.

Selon une caractéristique particulière, le système est remarquable en ce que la spécification de la forme de signal se fait en langage CSN1 enrichi de fonctions permettant la description des données sous forme d'attributs d'objets abstraits et de fonctions s'appliquant sur des représentations concrètes.

Selon une caractéristique particulière, le système est remarquable en ce qu'il comprend des moyens d'utilisation des données de contexte dans une description formelle d'un encodage et/ou d'un décodage.

Selon une caractéristique particulière, le système est remarquable en ce que les moyens de détermination d'un contexte commun, les moyens

d'identification des messages de protocole et les moyens de définition de représentations de données mettent chacun en oeuvre un langage formel permettant une détermination automatique du logiciel de mise en œuvre du protocole.

Selon une caractéristique particulière, le système est remarquable en ce qu'il comprend des moyens de détermination automatique du logiciel de mise en œuvre du protocole.

Selon une caractéristique particulière, le système est remarquable en ce qu'il comprend des moyens de détermination automatique de tests de dispositifs mettant en œuvre le protocole.

Selon une caractéristique particulière, le système est remarquable en ce que les moyens de détermination d'un contexte commun comprennent des moyens de traduction d'un mode de représentation d'un protocole en la description d'un ensemble de données.

Selon une caractéristique particulière, le système est remarquable en ce que les moyens d'identification des messages de protocoles comprennent des moyens de traduction d'un mode de représentation de protocoles.

Cette caractéristique permet notamment le développement d'une description dans le formalisme de l'invention à partir d'une description existante, issue par exemple d'un comité de normalisation.

Ainsi, grâce à son universalité, l'invention permet avantageusement la réutilisation des parties communes à plusieurs protocoles et notamment l'invention offre la possibilité d'avoir un noyau de protocole que l'on enrichit de spécificités pour créer d'autres protocoles ayant chacun leur spécificité.

Selon une caractéristique particulière, le système est remarquable en ce qu'il comprend en outre des moyens de visualisation de la sémantique des messages de protocoles échangés entre le ou les émetteurs et/ou le ou les récepteurs.

Cette étape est particulièrement utile et efficace pour tester un protocole de communication et/ou ses dispositifs de mise en œuvre.

En outre, l'invention concerne un procédé de spécification et de mise en œuvre d'un protocole de gestion de communication et de transmission entre au moins un émetteur et au moins un récepteur, remarquable en ce qu'il comprend les étapes suivantes :

- détermination d'un contexte commun comprenant une description d'un ensemble de données gérées par le ou les émetteurs et/ou le ou les récepteurs ;
- identification des messages de protocole susceptibles d'être échangés entre le ou les émetteurs et le ou les récepteurs, l'identification étant indépendante de la ou des représentations utilisées pour les messages;
- définition de représentations de données stockées et/ou transmises par le ou les émetteurs et/ou le ou les récepteurs à partir de la détermination d'un contexte commun et de l'identification des messages, la définition de représentations permettant de déterminer des instructions d'implémentation du protocole dans au moins un des émetteurs et au moins un des récepteurs ; et
- mise en œuvre du protocole dans au moins un des émetteurs et au moins un des récepteurs à partir des instructions d'implémentation, de façon à ce que le ou les émetteurs soient aptes à communiquer avec le ou les récepteurs.

De plus, l'invention concerne une spécification d'un protocole de communication, caractérisée en ce qu'elle est obtenue par la mise en œuvre du procédé décrit précédemment, de spécification et de mise en œuvre d'un protocole de gestion de communication et de transmission.

L'invention concerne également un langage formel de description d'au moins un élément de protocole de communication, remarquable en ce qu'il est adapté à la mise en œuvre du procédé précédemment décrit, de spécification et de mise en œuvre d'un protocole de gestion de communication et de transmission.

L'invention concerne aussi un dispositif de communication, remarquable en ce qu'il comprend des instructions permettant l'émission et/ou la réception de

données vers ou d'un dispositif tiers un autre dispositif selon un protocole obtenu par la mise en œuvre du procédé tel que décrit précédemment.

De plus, l'invention concerne un produit programme d'ordinateur comprenant des éléments de programme, enregistrés sur un support lisible par au moins un microprocesseur, caractérisé en ce que les éléments de programme contrôlent le ou les microprocesseurs pour qu'ils effectuent les étapes suivantes adaptées à la spécification de protocole de communication et de transmission entre au moins un émetteur et au moins un récepteur:

- détermination d'un contexte commun comprenant une description d'un ensemble de données gérées par le ou les émetteurs et/ou le ou les récepteurs ;
- identification des messages de protocole susceptibles d'être échangés entre le ou les émetteurs et le ou les récepteurs, l'identification étant indépendante de la ou des représentations utilisées pour les messages;
- définition de représentations de données stockées et/ou transmises par le ou les émetteurs et/ou le ou les récepteurs à partir de la détermination d'un contexte commun et de l'identification des messages, la définition de représentations permettant de déterminer des instructions d'implémentation du protocole dans au moins un des émetteurs et au moins un des récepteurs ; et
- mise en œuvre du protocole dans au moins un des émetteurs et au moins un des récepteurs à partir des instructions d'implémentation, de façon à ce que le ou les émetteurs soient aptes à communiquer avec le ou les récepteurs.

L'invention concerne également un produit programme d'ordinateur, comprenant des séquences d'instructions adaptées à la mise en œuvre du procédé de spécification de protocole de communication entre au moins un émetteur et au moins un récepteur lorsque le programme est exécuté sur un ordinateur.

Les avantages du langage formel, de la spécification de protocole, du dispositif et des programmes d'ordinateur sont les mêmes que ceux du procédé de spécification de protocole, ils ne sont pas détaillés plus amplement.

D'autres caractéristiques et avantages de l'invention apparaîtront plus clairement à la lecture de la description suivante d'un mode de réalisation préférentiel, donné à titre de simple exemple illustratif et non limitatif, et des dessins annexés, parmi lesquels la figure 1 présente un synoptique de spécification d'un protocole conforme à l'invention selon un mode particulier de réalisation.

En préliminaire de la description de la nouvelle méthode proposée, il est utile de faire un détour par les méthodes de programmation mettant l'accent sur les données, et de décrire la relation entre données et protocoles.

Les deux dernières décennies ont vu un développement important de la programmation de logiciel partant des données plutôt que des algorithmes. Ces méthodes consistent à décrire d'abord les structures de données mémorisées par le logiciel, en mettant en évidence les « objets » et types d'objet gérés par le logiciel. Un objet, à ce sens, peut être un objet concret, selon le sens commun, comme un écran ou un clavier, mais aussi quelque chose de plus abstrait comme une zone de l'écran ou un fichier, ou même très abstrait comme une file d'attente. Le point important de cette approche est qu'un type d'objet ne décrit pas seulement ses attributs, mais aussi les actions dont l'objet peut être l'objet.

Cette approche de la programmation a beaucoup d'avantages. Elle amène en particulier une modularité très efficace, centrée sur les types d'objet.

Jusqu'à présent, la présentation qui est faite des structures de données dans les protocoles, sous l'influence entre autres d'ASN.1 est erronée. Ce qui est décrit comme des données sont les messages et parties de message, alors que les messages sont beaucoup plus proches des actions, au sens de la programmation par les objets, que des données.

A l'instar de cette approche de la programmation, il faut chercher les données d'un protocole dans ce qui est mémorisé par chacune des entités paires,

plutôt que dans ce qui est transmis. Cette réflexion amène à mettre en avant la notion de « contexte » du protocole.

Nous définissons ici le contexte d'une entité comme la structure de donnée mémorisée par cette entité et contenant les données qui interviennent dans la mise en œuvre du protocole. Ces interventions sont de différents ordres : il peut s'agir de données gérées (i.e., modifiées) par l'effet de la réception d'un message du protocole, ou de données qui influencent les actions du protocole (données d'état, dont, mais pas seulement, l'état au sens de SDL).

Parmi les données apparaissant dans le contexte de chacune des entités, certaines sont identiques sémantiquement de part et d'autre, c'est-à-dire dans deux entités paires du protocole. Selon l'invention, la partie identique est appelée « contexte commun ». Le contexte commun peut être décrit de manière monolythique ou, au contraire, par morceaux séparés. Ces données sont les « mêmes » au sens qu'elles ont le même sens, et sont maintenues identiques, à l'exception de certains états transitoires. On peut présenter un protocole comme ayant pour rôle, entre autres, de maintenir l'égalité de ces données. Ainsi, une part importante de la fonction de n'importe quel protocole est la gestion de données répartie dans plusieurs dispositifs émetteurs et/ou récepteurs.

On présente, en relation avec la figure 1, un procédé de spécification de protocole selon l'invention. Ce procédé est préférentiellement mis en œuvre par un micro-ordinateur.

Ce procédé est divisé en étapes successives :

- une première étape 1 de spécification d'encodage et du décodage ;
- une seconde étape 2 de spécification du contexte ; et
- une troisième étape 3 de spécification de messages.

L'étape 1 de spécification de l'encodage et du décodage est effectuée en CSN.1 qui est connu en soi.

Le premier aspect formalisé de la spécification d'un protocole suivant la méthode décrite est la spécification formelle de la structure des chaînes de bit échangées. Il s'agit de la description du lexique selon sa forme, comme une liste

de structures de messages ou d'éléments de message. Cette description spécifie des ensembles de chaînes de bit acceptables, ainsi que la structure (syntaxe) de ces chaînes. Elle permet de générer automatiquement un logiciel d'encodage et de décodage, ce dernier incluant la détection de chaînes erronées (i.e., n'appartenant pas à l'ensemble des chaînes acceptables).

Cette spécification est faite en CSN.1 mais en variante, tout autre langage formel permettant de décrire la syntaxe de toute chaîne de bit est applicable.

L'étape 2 consiste à décrire formellement la structure des données formant le contexte de chacune des entités paires. Cette description mettra en avant la partie commune des deux contextes, c'est-à-dire les données gérées comme une base de donnée répartie.

Cette description formelle peut utiliser n'importe quel langage formel de description de structure de donnée, comme ASN.1, des méthodes de description telles qu'on peut les trouver dans les langages de programmation comme C++, ou comme on peut les trouver dans les logiciels de gestion de bases de données.

On utilise de manière préférentielle un langage spécialement développé à cet effet, permettant de faire aisément le pont entre les structures de données apparaissant dans les messages et les descriptions de structures de données telles qu'utilisée par les développeurs, comme celles permises par le langage de programmation C++.

Au cours de l'étape 3, un message est décrit de manière abstraite (i.e., indépendamment de l'encodage utilisé effectivement pour le transport) comme une suite d'actions s'appliquant aux données du contexte commun. Ceci diffère fondamentalement des descriptions usuelles, où les messages ou parties de messages sont décrits comme des structures de données.

Une description de message selon la méthode donne le sens du message, sous la forme de l'effet attendu de la réception du message par l'entité paire. Elle explicite aussi le sens des paramètres, en indiquant leur rôle dans la description de l'action attendue. Cela permet de formaliser de nombreux aspects de la

spécification d'un protocole qui sont traditionnellement couverts dans les descriptions de procédures par un texte en langage naturel.

L'action la plus courante est le changement d'une partie des données de contexte. Une telle action est nommée ci-après « SET ». Elle demande deux types de paramètres :

- ceux décrivant la partie du contexte concernée ; et
- ceux décrivant les nouvelles valeurs à affecter.

La description traditionnelle ne fait pas la distinction entre les deux types, et omet des paramètres « implicites ».

D'autres actions typiques sont « CREATE » ou « ADD » qui créent une partie des données (par exemple, création d'un nouveau contexte d'appel).

Une description des messages ou parties de message en CSN.1 n'adresse que la structure de ce qui est effectivement transmis. A l'opposé, une description abstraite d'un type message le décrit en termes sémantiques, plus liés à l'impact qu'a la réception du message qu'à la forme particulière utilisée pour la transmission.

Il est possible de décrire formellement la liaison entre les deux, de manière à permettre la génération automatique du logiciel depuis la réception de la chaîne de bit jusqu'à inclusivement la modification du contexte des données communes.

La description de cette liaison peut se faire de diverses manières. Une de ces manières est basée sur les principes suivants.

A chaque structure de chaîne de bit sont associées des fonctions nommées retournant des valeurs selon des types abstraits. Pour chaque fonction est décrit en détail l'algorithme permettant de calculer le résultat de la fonction à partir de la chaîne de bits reçue et, quand nécessaire, de la valeur courante du contexte commun. (Ce dernier point permet de formaliser des méthodes d'encodage dépendant du contexte, comme, par exemple, l'utilisation de tables d'index transmises au préalable.)

Corrélativement, des règles permettent d'associer à une description abstraite une liste de fonctions nommées correspondant exactement à tous les paramètres nécessaires pour faire les actions demandées par le message.

La méthode décrite a diverses applications, dont les principales sont les suivantes :

- le développement automatique des logiciels de mise en œuvre de protocoles de communication ;
- le développement de protocoles spécifiés selon des méthodes différentes ;
- la spécification directe de protocoles de communication ; et
- le développement de test de protocoles de communication.

L'un des buts principaux de l'invention est de permettre la génération automatique d'une partie substantielle des logiciels nécessaires à la mise en œuvre du protocole. Une formalisation complète du protocole selon l'invention permet la génération automatique d'un encodage et d'un décodage des messages, y compris la détection de chaînes erronées, et y compris les algorithmes d'encodage ou de décodage complexes prenant en compte des données de contexte.

Selon une variante de l'invention, une formalisation complète du protocole permet également la génération automatique des aspects suivants :

- données de contextes ; et/ou
- actions simples des messages (par exemple, le changement de valeur des données de contexte).

De plus, les aspects non générés automatiquement sont traités par ajout de blocs de programme attachés, selon l'approche de la programmation des données, aux classes de données (méthodes), éventuellement comme complément aux actions générées automatiquement.

Une application de la méthode consiste à traduire des protocoles spécifiés autrement (c'est-à-dire selon des méthodes différentes) dans une description conforme à l'invention. Parmi les avantages par rapport aux méthodes traditionnelles de développement, on peut citer :

- uniformisation des développements ;
- réduction du code généré par mise en commun des moteurs génériques (e.g., moteurs de décodage) ;
- meilleure modularisation, rendant plus aisées les modifications ultérieures.

L'invention permettant la spécification directe de protocoles peut aussi être mise en œuvre par des groupes de spécification.

Les descriptions formelles peuvent être utilisées pour générer directement des tests permettant de vérifier la conformité d'une réalisation avec la description.

La méthode complète de spécification selon le mode de réalisation décrit met en œuvre des syntaxes de description pouvant se décomposer en cinq groupes d'éléments syntaxiques (ou sous-langages formels):

a) une syntaxe de description de structure de données, pour les éléments du contexte commun ;

b) une syntaxe de description de l'encodage et du décodage de chaînes de symboles ;

c) une syntaxe de programmation, adaptée en liaison avec les syntaxes a) et b);

d) des éléments syntaxiques permettant de lier les descriptions selon les syntaxes a) et b) ; ces éléments peuvent être partie intégrante soit de la syntaxe a) soit de la syntaxe b), ou même se limiter à des règles de correspondance entre les descriptions selon les syntaxes a) et b) ; et

e) une syntaxe de description de messages comme actions agissant sur le contexte .

Selon une variante préférentielle de l'invention, la méthode de spécification met en œuvre les quatres premières syntaxes a) à d) sans utiliser la dernière syntaxe e) de description de messages.

Cette technique est décrite plus en détail dans les annexes 1 à 6, en relation avec la figure 1, de façon à ne pas surcharger la présente discussion. Il est clair cependant que ces annexes font partie intégrante de la description.

Les annexes 1, 4 et 5 décrivent des éléments de description formelle spécifiant un protocole de communication.

L'annexe 2 illustre une spécification de simple mise à jour de données.

Les annexes 3 et 6 illustrent un cas plus complexe de spécification d'un protocole GSM (« Global System for Mobile communication ») normalisé par l'ETSI (European Telecommunication Standard Institute) de gestion de mobilité entre une station mobile (MS) et une infrastructure de réseau.

Les spécifications illustrées en regard des annexes 2 et 3 (respectivement 6) utilisent les éléments de description formelle de l'annexe 1 (respectivement 4) et permettent une génération automatique de code selon l'invention.

Bien entendu, l'invention n'est pas limitée aux exemples de réalisation mentionnés ci-dessus.

En particulier, l'homme du métier pourra apporter toute variante dans la mise en œuvre de l'invention qui peut s'appliquer non seulement au domaine des télécommunications mobiles mais à toute conception de réseau (fixe et/ou mobile) ou de lien de communications ainsi qu'à la définition des protocoles correspondant.

L'invention s'applique également à différents types d'éléments communiquant, notamment terminaux de télécommunications, éléments d'infrastructure de réseau, cartes ou composants communicants à l'intérieur d'un dispositif...

L'invention peut, en outre, être mise en œuvre sur des machines ou outils de développement très variés, permettant, par exemple, la spécification de protocoles de communication, la génération de codes ou plus généralement la conception et la réalisation de protocoles de communication

On notera que l'invention est mise en œuvre sous la forme d'une séquence d'instructions d'un programme informatique. Aussi, la séquence d'instructions correspondante pourra être stockée dans un moyen de stockage amovible (tel que par exemple une disquette, un CD-ROM ou un DVD-ROM) ou non, ce moyen de

stockage étant lisible partiellement ou totalement par un ordinateur ou un microprocesseur.

ANNEXE 1.
Module de description formelle de protocole

Un module décrit dans l'annexe 1 comprend des éléments de description formelle spécifiant un protocole de communication associés à des entités paires.

Le module comprend trois types de description :

- des descriptions abstraites de types d'objet (section 3); cela correspond à une syntaxe de type a) illustrée précédemment;
- des descriptions de transfert s'appliquant soit à des messages, soit à des objets en tant que partie de message (section 6) ; cela correspond à une syntaxe de type b);
- et des descriptions abstraites d'actions (messages par exemple), section 5 (syntaxe de type e)) ou des programmations d'actions, selon une syntaxe de type c).

Selon une variante de l'invention, les descriptions abstraites d'actions gérées automatiquement par un système de spécification sont remplacées avantageusement par des programmations manuelles ou semi-automatiques d'actions.

A chaque type correspond un langage formel distinct.

Le module est essentiellement organisé comme une liste de types d'objets. A chaque type d'objet, sont associées une description abstraite, éventuellement une ou plusieurs descriptions de transfert, et éventuellement une ou plusieurs actions. A chaque action, sont associées une description abstraite et éventuellement une ou plusieurs description de transfert.

Pour des raisons pratiques, l'organisation générale d'un module permet soit de grouper les descriptions selon la logique ci-dessus, soit de séparer les éléments (par exemple pour grouper tous les messages ensemble), auquel cas les liaisons sont explicitées (par exemple la description d'un message indique alors le type d'objet auquel il est associé).

1. Syntaxe de module

Le module est composé d'un en-tête, suivi d'une liste de descriptions de type d'objet, de descriptions autonomes d'actions et de descriptions autonomes de structure de transfert.

1.1 En-tête de module

L'en-tête de module comporte son nom et, éventuellement, d'autres informations.

2. Spécifications de types d'objet

Une description (ou spécification) de type d'objet est constituée d'un nom suivi des caractères ' ::=', et terminée par un caractère ' ;'.

Une description comporte :

- une description abstraite, introduite par le mot clé 'abstract' ;
- zéro, une ou plusieurs descriptions de transfert, introduites chacune par le mot clé 'transfer' ;
- zéro, une ou plusieurs descriptions d'action, introduites chacune par le mot clé 'action'.

Une description de transfert décrit une structure de chaîne de symboles (par exemple binaire ou décrite par une chaîne de caractères) qui peut représenter correctement la structure abstraite (voir le chapitre sur le sujet).

Une action (message) associé au type d'objet décrit des modifications pouvant être demandées à une instance dudit type d'objet.

3. Descriptions abstraites de types d'objet

Une description abstraite de type d'objet se présente usuellement comme un ensemble de composants. Chaque composant est notamment caractérisé par :

- un nom de composant,
- des informations de présence, et
- une description de type.

3.1 Nom de composant

Le nom de composant est une chaîne libre utilisant les caractères disponibles à l'exception des caractères '[', '.',

3.2 Informations de présence

Par défaut le composant n'est pas indexé. Un composant indexé (tableau) peut avoir un nombre variable de copie, distinguées par un index. Un composant indexé est indiqué par une spécification d'intervalle comprise entre '[' et ']'. Différentes spécifications sont possibles :

- Nombre fixe : cela est indiqué par un entier, qui est le nombre de copies.
- Nombre variable libre : cela est indiqué par un intervalle (deux entiers séparés par '..'). Un cas particulier est 0..infinity qui indique un nombre quelconque de copies.
- Nombre variable conditionnel : cela est indiqué par une expression de type 'iif', indiquant différents intervalles ou nombre fixes à sélectionner selon des expressions booléennes portant sur les valeurs d'autres éléments de la structure.

3.3 Description de type

Cette description indique la structure du composant lui-même. Différentes descriptions sont possibles.

3.3.1 Composants élémentaires

Une telle description est introduite par un ' :'.

Cela inclus :

- entiers (mot clé 'integer', suivi éventuellement d'un intervalle entre parenthèses) ;
- type énuméré (mot clé 'enumerated' suivi d'une liste parenthésée d'identificateurs séparés par des virgules) ;
- chaîne de bit (mot clé 'bit string', suivi éventuellement d'une taille ou d'un intervalle de taille entre parenthèses)

3.3.2 Structure

Une structure est introduite par '{', et est décrite par une succession de lignes commençant par un '+' de plus que la ligne introduisant le composant, et terminée par un '}'.

3.3.3 Union

Une union est une écriture abrégée pour un ensemble de composants de présence exclusive.

3.3.4 Référence

Une référence est introduite par un caractère ' :', et est le nom d'un type de donnée décrit par ailleurs. Quand le module de description du type n'est pas le module englobant, il est indiqué entre parenthèse suivant le nom de type.

3.3.5 Index

Un type index est introduit par '->', suivi d'un nom de nœud. Un nom de nœud est composé d'un nom de type d'objet (éventuellement vide, auquel cas il s'agit du type courant), suivi d'une suite de nom de composants séparés par des '.', telle que chaque nom de composant existe dans la structure du composant ou type qui précède. Le dernier composant doit être indexé.

Un nom de nœud donne :

- L'information statique du type indexé (le type du dernier composant) ;

- Un lien dynamique vers une instance particulière de la liste. Ce lien est résolu en remontant l'arborescence de l'instance jusqu'à trouver un objet du type indiqué.

4. Description d'action

Une description d'action comporte le nom de l'action entre '<' et '>', suivi de ' ::=' puis d'une description abstraite d'action, suivie de zéro, une ou plusieurs descriptions de transfert, chacune introduite par le mot clé 'transfer'.

Une description de transfert décrit une structure de chaîne de symboles qui peut représenter correctement la structure abstraite (voir le chapitre sur le sujet).

5. Description abstraite d'action

Une description abstraite d'action comporte éventuellement une liste de paramètres, et éventuellement une liste d'instructions.

(Les instructions demandent en général des paramètres qui apparaissent par exemple dans la représentation de l'action. Ces paramètres ne sont pas listés dans la liste de paramètres : celle-ci ne contient les paramètres qui ne peuvent pas être décrits plus précisément comme liés à une instruction élémentaire.)

La liste de paramètres est introduite par le mot clé 'parameters' et est terminée par le mot clé 'end'. La liste suit la syntaxe de la description abstraite de type d'objet.

La liste d'actions est constituée d'une ou plusieurs lignes. Chaque ligne est constituée d'une éventuelle indication de condition, suivie d'une instruction parmi celles listées ci-dessous.

5.1 Indication de condition

Une indication de condition est soit le mot clé 'OPTIONAL', soit une instruction IF THEN.

Le mot clé OPTIONAL indique que l'instruction, ou la liste d'instructions entre '{' et '}', qui suit peut ou non être demandée.

Une instruction IF THEN indique (entre les mots clé 'IF' et 'THEN') une condition, exprimée à partir de valeurs de paramètres, si l'instruction est à faire ou non.

5.2 Instruction de sélection

Une instruction de sélection est introduite par le mot clé 'on' suivi du nom d'un composant (typiquement une structure) sur lequel s'applique une liste d'instructions comprises entre '{' et '}'. Si le composant est indexé, une indication d'index peut suivre le nom de composant (voir section sur l'index).

5.3 Instruction SET

Une instruction SET est introduite par le mot clé 'set' suivi d'un nom de composant (éventuellement avec un index). Cette instruction demande l'affectation d'une nouvelle valeur au composant (dans son intégrité, ce qui peut être un sous-arbre assez important). Il y a plusieurs manière dont la nouvelle valeur peut être indiquée :

5.3.1 Paramètre implicite de type implicite

Dans ce cas, l'instruction ne contient rien de plus. Le message contient alors un paramètre du type du composant, tel que décrit dans la description abstraite d'objet.

5.3.2 Paramètre implicite de type explicite

Dans ce cas l'instruction est suivi du caractère ' :' suivi d'un nom de type d'objet. Ce type doit être compatible avec le type du composant (sous-type).

5.3.3 Valeur explicite

Dans ce cas, l'instruction est suivie du caractère ' :=' suivi d'une expression dont le résultat doit être compatible avec le type.

5.3.4 Valeur calculée

Dans ce cas l'instruction est suivie du caractère' :=' suivi d'une liste d'instructions entre '{' et '}'. Ces instructions portent sur le nœud indiqué par le composant, et itémisent les actions aboutissant à l'affectation du composant (typiquement sous-composant par sous-composant).

5.3.5 Vidage

Dans ce cas, l'instruction est suivie de ' := null'. Cela ne s'applique qu'aux composants indexés (mais sans indication d'index dans l'instruction SET) et qui accepte un nombre nul de copies. L'instruction met le nombre de copies à zéro.

5.4 Instruction ADD

Une instruction ADD est introduite par le mot clé 'set' suivi d'un nom de composant indexé (sans indication d'index dans l'instruction ADD). Cette instruction demande l'ajout d'une nouvelle copie au composant (dans son intégrité, ce qui peut être un sous-arbre assez important). Il y a plusieurs manières dont la nouvelle valeur peut être indiquée, comme dans le cas de l'instruction SET (sauf le vidage).

5.5 Instruction SEND

Une instruction SEND est introduite par le mot clé 'send' suivi d'un nom de composant (éventuellement avec un index). Cette instruction indique l'inclusion d'un paramètre dont la valeur est la valeur courante du composant du côté de la source. L'action du côté du récepteur est de vérifier la cohérence, mais la conséquence d'une détection d'incohérence n'est pas précisée.

5.6 Instruction CHOICE

Une instruction CHOICE est introduite par le mot clé 'choice' suivi d'une liste d'actions entre '{' et '}' et séparées par des ','. Cela indique qu'une et une seule des actions est demandée pour chaque occurrence de l'action.

5.7 Instruction DO

Une instruction DO consiste en le mot clé 'do' suivi du nom d'une action. Cette action doit être compatible avec le type de composant courant.

5.8 Instructions liées au retour en arrière

Une modification de données peut être demandée « à l'essai » et refusée éventuellement par le récepteur. Ceci est géré par les instructions PREPARE, COMMIT et UNDO.

5.8.1 L'instruction PREPARE

Cette instruction indique que les modifications qui la suivent et applicables au nœud courant peuvent être annulées.

5.8.2 L'instruction COMMIT

Cette instruction indique l'acceptation des modifications sujettes à un PREPARE sur le nœud courant.

5.8.3 L'instruction UNDO

Cette instruction indique le refus des modifications sujettes à un PREPARE sur le nœud courant.

6. Description de transfert

La description de transfert est à la base de type langage CSN.1 version 2.2, avec les adaptations suivantes.

6.1 En-tête

Quand la description de transfert est unique pour un type d'objet, le nom de structure CSN.1 peut être omis, et est alors le même que le nom de type.

6.2 Fin de structure

Le « ; » final est omis (la fin est détectée soit par le « ; » de fin d'objet, soit par un des mots clés « transfer » (indiquant une alternative de codage), ou « action ».

6.3 Fonctions calculées de liaison avec la description abstraite

L'en-tête d'une fonction calculée peut être de la forme « abstract. » suivi d'un identificateur d'élément abstrait. Le type n'est pas indiqué et est celui de l 'élément abstrait ainsi que défini dans la description abstraite.

Plus généralement, les règles de liaison entre une description de transfert et une description abstraite (de type d'objet ou de message) font l'objet d'un chapitre spécial.

6.4 Références au contexte

Une formule peut invoquer des valeurs d'élément de contexte.

7. Description séparée d'action

Une action peut être décrite en dehors de la description du type d'objet sur laquelle elle porte. La description d'action est alors précédé d'un en-tête composé du mot clé 'on' suivi du nom de type d'objet sue lequel porte l'action.

L'indication du type d'objet peut être omise dans le cas d'une description séparée d'action en suivant une autre s'appliquant au même type.

8. Description séparée de transfert

Une description de transfert peut être séparée de la description abstraite à laquelle elle correspond.

9. Règles de correspondance

La correspondance entre une description abstraite et une description de transfert est basée sur la correspondance entre des noms de composants abstraits et des noms de label, plus des formules de représentation.

10. Noms

Un nom (d'objet ou d'action) est composé de caractères libres, avec les exceptions nécessaires pour des raisons syntaxiques. Lors de la comparaison entre deux noms, les chaînes consécutives d'espace, tabulation et fin de ligne sont considérées comme équivalentes à un seul espace, sauf au début et à la fin , où elles sont considérées comme équivalentes à la chaîne vide ; la chasse n'est pas prise en compte.

ANNEXE 2

Premier exemple : spécification d'une simple mise à jour d'une donnée.

Un protocole est présenté ici dans le cadre d'un protocole unidirectionnel illustratif dont le seul propos est de mettre à jour une donnée. L'application de la méthode procède comme suit :

- description abstraite du contexte ;
- description abstraite des messages ;
- description de l'encodage et du décodage des données ;
- description de l'encodage et du décodage des messages ;

Les spécifications illustrées en regard de l'annexe 2 utilisent les éléments de description formelle de l'annexe 1 et permettent une génération automatique de code selon l'invention.

1. Description abstraite du contexte commun.

Cette description se fait selon un langage formel de description abstraite de données. Le langage utilisé ici n'est pas publié.

```
<type du contexte du protocole> ::=
    données gérées : type de données gérées ;
```

L'expression 'type de données gérées' peut représenter un type de données quelconque, par exemple :

```
<type de données gérées> ::=
    entier 1 : integer ,
    entier 2 : integer ;
```

2. Description abstraite des messages.

Le protocole est réduit à un seul message, qui modifie le contexte dans sa totalité. La description des messages se fait selon un langage formel adapté à la description d'actions portant sur le contexte de données :

```
on <type du contexte du protocole>
<protocol data unit> ::=
CHOICE
    <modifier : contenu du message modifier> ;

<contenu du message modifier> ::=
    on <données gérées> SET;
```

3. Description de l'encodage et décodage des données.

Cette description se fait en CSN.1, encapsulé dans un langage plus général combinant la description abstraite et la description concrète. Seuls sont décrites les données apparaissant comme paramètre de message.

```
<type de données gérées> ::=
abstract
    entier 1 : integer ,
    entier 2 : integer
transfer
    <entier 1 : bit(4)>      -- la liaison avec l'abstrait se fait par
l'identificateur
                    -- l'encodage par défaut est binaire, poids  fort en
premier
    <entier 2 : bit(6)> ;
```

4. Description de l'encodage et décodage des messages.

Cette description se fait en CSN.1, encapsulé dans un langage plus général combinant la description abstraite et la description concrète.

```
<protocol data unit> ::=
 abstract
      CHOICE
            <modifier : contenu du message modifier>
transfer
      00 <modifier : <contenu du message modifier>>
      {<extension : bit(*) = null}!   -- permet d'accepter une
extension future
      <message type errror : bit(*);        -- détection d'une err eur de
type

<contenu du message modifier> ::=
abstract
      on <données gérées> SET
transfer
      <données gérées.set.nouvelle données : <type de données
gérées>> ;
```

ANNEXE 3

Exemple de spécification d'un protocole réel (le protocole GSM de gestion de mobilité entre une station mobile (MS) et une infrastructure de réseau)

La plupart des fonctions ci-après ont des noms tels que spécifiés dans la norme GSM (« Global System for Mobile communication ») normalisée par l'ETSI (European Telecommunication Standard Institute).

Les noms ou accronymes utilisés dans les spécifications correspondent soit à des éléments de la syntaxe d'un mode particulier de réalisation de l'invention (tel que précisé en annexe 1) soit à des noms ou acronymes d'éléments spécifiés dans la norme GSM.

```
module MM 24.008 4.0.0

default scope : MM

RIL3.Location Area Code from RIL3 24.008 4.0.0
RIL3.Location Area Identification from RIL3 24.008 4.0.0
RIL3.Mobile Station Classmark 1 from RIL3 24.008 4.0.0
RIL3.Mobile Station Classmark 2 delta from RIL3 24.008 4.0.0
RIL3.Mobile Station Classmark 3 from RIL3 24.008 4.0.0
RIL3.Priority Level from RIL3 24.008 4.0.0
RIL3.Ciphering Key Sequence Number from RIL3 24.008 4.0.0
```

1 Contexte de protocole de gestion de mobilité (ou MM de l'anglais « mobility management »).

Il s'agit de la spécification du contexte de protocole, c'est-à-dire des données communes à l'infrastructure et à la station mobile, et gérées par le protocole.

1.1 Description abstraite

La description des éléments du contexte est indépendante des représentations, que ce soit pour la mémorisation dans les entités, ou pour la transmission dans des messages de protocole.

```
<MM context> : :=
 abstract
   IMSI[0..1] : IMSI
   Connect state : Enumerated (not connected, connected)
   Connection parameters[iif(Connect state = connected, 1, 0)] :
    + Serving cell : -> UE context.Cell list
    + IMEI[0..1] : IMEI
    + Software Version Number[iif(PRESENT (IMEI), 1, 0)] : Integer (0..99)
    + Mobile Station Classmark 1 : RIL3.Mobile Station Classmark 1
    + Mobile Station Classmark 2 delta[0..1] : RIL3.Mobile Station Classmark 2 delta
    + Mobile Station Classmark 3[0..1] : RIL3.Mobile Station Classmark 3
    + Has been authenticated : Boolean
    + Is ciphered : Boolean
    + Service data[0..infinite] :
    +  + CM service type : CM Service Type
    +  + Priority : RIL3.Priority Level
   Location update state : Enumerated (roaming not allowed, updated)
   Location parameters[iif(Location update state = updated, 1, 0)] :
    + LAI+TMSI[0..1] : LAI+TMSI
    + Registered location area : -> UE context.LA list
    + Detach flag : Boolean
    + CTS permission : Boolean
    + CKSN availability : Boolean
    + CKSN[iif(CKSN availability, 1, 0)] : RIL3.Ciphering Key Sequence Number
    + GSM ciphering key[0..1] : Bit string (64)
    + UMTS security data[0..1] :
    +  + AUTN : Authentication Parameter AUTN
    +  + UMTS ciphering key[0..1] : Bit string (128)
    +  + UMTS integrity key[0..1] : Bit string (128)
```

1.2 Actions

1.2.1 Gestion de mobilité dans le sens descendant (ou DL MM de l'anglais « downlink mobility management »)

1.2.1.1 Description abstraite

Ceci décrit abstraitement le message de protocole qui peut être transmis de l'infrastructure vers la station mobile.

```
action DL MM ::=
abstract
 CHOICE {
  do LOCATION UPDATING ACCEPT
  ,do LOCATION UPDATING REJECT
  ,do AUTHENTICATION REJECT
  ,do AUTHENTICATION REQUEST
  ,do TMSI REALLOCATION COMMAND
  ,do CM SERVICE ACCEPT
  ,do CM SERVICE REJECT
  ,do CM SERVICE PROMPT
  ,do IDENTITY REQUEST
  ,do ABORT
  ,do MM STATUS DOWNLINK
  ,do MM INFORMATION
  }
```

1.2.1.2 Syntaxe de transfert

Ceci indique la structure binaire du message. Cela fait apparaître en particulier un champ permettant de sélectionner une action particulière dans la liste des actions possibles.

```
transfer ::=
  -- flow of downlink MM messages
  {<skip indicator : 0000>
    <protocol discriminator : 0101>
    0 0
    {00 0010 <LOCATION UPDATING ACCEPT : LOCATION UPDATING ACCEPT>
```

```
        | 00 0100 <LOCATION UPDATING REJECT : LOCATION UPDATING REJECT>
        | 01 0001 <AUTHENTICATION REJECT : AUTHENTICATION REJECT>
        | 01 0010 <AUTHENTICATION REQUEST : AUTHENTICATION REQUEST>
        | 01 1010 <TMSI REALLOCATION COMMAND : TMSI REALLOCATION COMMAND>
        | 10 0001 <CM SERVICE ACCEPT : CM SERVICE ACCEPT>
        | 10 0010 <CM SERVICE REJECT : CM SERVICE REJECT>
        | 10 0101 <CM SERVICE PROMPT : CM SERVICE PROMPT>
        | 10 1000 <IDENTITY REQUEST : IDENTITY REQUEST>
        | 10 1001 <ABORT : ABORT>
        | 11 0001 <MM STATUS : MM STATUS DOWNLINK>
        | 11 0010 <MM INFORMATION : MM INFORMATION>}
      ! <erroneous type : bit** = <no string>>}
    <spurious extension : bit** = null>
;
```

1.2.2 Gestion de mobilité dans le sens montant (ou UL MM de l'anglais « uplink mobility management »)

1.2.2.1 Description abstraite

```
action UL MM ::=
abstract
  CHOICE {
   do AUTHENTICATION FAILURE
   ,do AUTHENTICATION RESPONSE
   ,do IDENTITY RESPONSE
   ,do TMSI REALLOCATION COMPLETE
   ,do CM SERVICE ABORT
   ,do CM SERVICE REQUEST SUBSEQUENT
   ,do MM STATUS UPLINK
  }
```

1.2.2.2 Syntaxe de transfert

```
transfer ::=
  -- flow of uplink MM messages once an RR connection is established
  {<skip indicator : 0000>
    <protocol discriminator : 0101>
    bit*2
    {01 1100 <AUTHENTICATION FAILURE : AUTHENTICATION FAILURE>
    | 01 0100 <AUTHENTICATION RESPONSE : AUTHENTICATION RESPONSE>
    | 01 1001 <IDENTITY RESPONSE : IDENTITY RESPONSE>
```

```
      | 01 1011 <TMSI REALLOCATION COMPLETE : TMSI REALLOCATION COMPLETE>
      | 10 0011 <CM SERVICE ABORT : CM SERVICE ABORT>
      | 10 0100 <CM SERVICE REQUEST : CM SERVICE REQUEST>
      | 11 0001 <MM STATUS : MM STATUS UPLINK>}
    ! <erroneous type : bit** = <no string>>}
    <spurious extension : bit** = null>
  ;
;
```

1.2.3 Abandon (ou « abort »)

1.2.3.1 Description abstraite

```
action ABORT ::=
abstract
  parameters
    Reject cause : MM.Reject cause
  end
  if iif (Reject cause.cause grouping = MS identification error, Reject cause.MS identification error
cause = illegal ME, false) then set Location update state := roaming not allowed
```

1.2.3.2 Syntaxe de transfert

```
transfer ::=
  <Reject cause : <M-V-IE (Reject cause downlink, 1)>>
;
```

1.2.4 Echec d'authentication (ou "AUTHENTICATION FAILURE")

1.2.4.1 Description abstraite

```
action AUTHENTICATION FAILURE ::=
abstract
  parameters
    Reject cause : MM.Reject cause
    Authentication Failure parameter[iif(iif (Reject cause.cause grouping = network related failure,
Reject cause.network-related failure cause = synch. failure, false), 1, 0)] : MM.Authentication
Failure parameter
  end
  on Location parameters {
    set UMTS security data := null
```

```
}
```

1.2.4.2 Syntaxe de transfert.

```
transfer ::=
  <Reject Cause : <M-V-IE (Reject Cause uplink, 1)>>
  < Authentication Failure parameter : <O-TLV-IE (0010 0010, Authentication Failure parameter,
16)>>
;
```

1.2.5 Rejet d'authentification (ou "AUTHENTICATION REJECT")

1.2.5.1 Description abstraite

```
action AUTHENTICATION REJECT ::=
abstract
  set Location update state := Roaming not allowed
```

1.2.5.2 Syntaxe de transfert

```
transfer ::=
  null
;
```

1.2.6 Requête d'authentification (ou "AUTHENTICATION REQUEST")

1.2.6.1 Description abstraite

```
action AUTHENTICATION REQUEST ::=
abstract
  parameters
    Authentication Parameter RAND : MM.Authentication Parameter RAND
    Authentication Algorithm : Enumerated (UMTS, GSM)
  end
  on Location parameters {
    PREPARE
    set CKSN availability := True
    set CKSN
    set GSM ciphering key := A8(Authentication Parameter RAND.RAND)
```

```
    if Authentication algorithm = UMTS then set UMTS security data := {
      set AUTN
      set UMTS ciphering key := f3 (Authentication Parameter RAND.RAND)
      set UMTS integrity key := f4 (Authentication Parameter RAND.RAND)
    }}
```

1.2.6.2 Syntaxe de transfert

```
transfer ::=
  <spare half octet : {bit*4 = 0000}>
  <Ciphering key sequence number : <M-VD-IE (Ciphering key sequence number Downlink)>>
  <Authentication parameter RAND : <M-V-IE (Authentication parameter RAND, 16)>>
  <Authentication parameter AUTN : <O-TLV-IE (0010 0000, Authentication parameter AUTN, 18,
18)>>

  abstract.Authentication algorithm returns
    -- if there is any error on AUTN, the authentication Algo will be considered GSM
    iif (exist(Authentication parameter AUTN), UMTS, GSM)
;
```

1.2.7 Réponse d'authentification (ou "AUTHENTICATION RESPONSE")

1.2.7.1 Description abstraite

```
action AUTHENTICATION RESPONSE ::=
abstract
 parameters
  RES : Octet string (4..16)
 end
 COMMIT
 on Connection parameters {
   set Has been authenticated := True
transfer ::=
  <Authentication response parameter : <M-V-IE (Authentication Response Parameter, 4)>>
  <Authentication response extension : <O-TLV-IE (0010 0001, Authentication Response
Parameter extension, 3, 14>>

  function RES returns
    <instance(Authentication response parameter.V.RES part 1)>
    {<instance(Authentication response extension.V.RES part 2)
action CM SERVICE ABORT ::=
abstract
```

```
on Connection parameters {
  set Service data := null
 }
```

1.2.7.2 Syntaxe de transfert

```
transfer ::=
  null
;
```

1.2.8 Abandon de connexion pour service (ou "CM SERVICE ABORT")

1.2.8.1 Description abstraite

```
action CM SERVICE ABORT ::=
abstract
 on Connection parameters {
   set Service data := null
  }
```

1.2.8.2 Syntaxe de transfert

```
transfer ::=
  null
;
```

1.2.9 Acceptation de la demande de connexion pour service (ou "CM SERVICE ACCEPT")

1.2.9.1 Description abstraite

```
action CM SERVICE ACCEPT ::=
abstract
 COMMIT
```

1.2.9.2 Syntaxe de transfert

```
transfer ::=
  null
;
```

1.2.10 Affichage du service de gestion de connexion (ou "CM SERVICE PROMPT")

1.2.10.1 Description abstraite

```
action CM SERVICE PROMPT ::=
abstract
 parameters
  PD and SAPI : RIL3.PD and SAPI
 end
```

1.2.10.2 Syntaxe de transfert

```
transfer ::=
  <PD and SAPI of CM : <M-V-IE (PD and SAPI, 1)>>
;
```

1.2.11 Rejet de la demande de connexion pour service (ou "CM SERVICE REJECT")

1.2.11.1 Description abstraite

```
action CM SERVICE REJECT ::=
abstract
 parameters
  Reject cause : MM.Reject cause
 end
 UNDO
 if iif (Reject cause.cause grouping = MS identification error, Reject cause.MS identification error
cause = "illegal ME" or Reject cause.MS identification error cause = "IMSI unknown in VLR", false)
then set Location update state := roaming not allowed
```

1.2.11.2 Syntaxe de transfert

```
transfer ::=
  <Reject cause : <M-V-IE (Reject cause downlink, 1)>>
;
```

1.2.12 Demande de connexion additionnelle pour service (ou "CM SERVICE REQUEST SUBSEQUENT")

1.2.12.1 Description abstraite

```
action CM SERVICE REQUEST SUBSEQUENT ::=
abstract
 parameters
  Mobile Identity Type : MM.Mobile Identity Type
 end
 if Mobile Identity Type = IMSI then send IMSI
 on Location parameters {
  if Mobile Identity Type = TMSI/P-TMSI then on LAI+TMSI {
   send TMSI
  }
  send CKSN availability
  if CKSN availability then send CKSN
 }
 on Connection parameters {
  send Mobile Station Classmark 1
  send Mobile Station Classmark 2 delta
  PREPARE
  add Service data := {
   set CM Service Type
   set Priority
 }}
```

1.2.12.2 Syntaxe de transfert : CM SERVICE REQUEST

```
transfer <CM SERVICE REQUEST> ::=
  <Ciphering key sequence number uplink : <M-VD-IE (Ciphering key sequence number Uplink)>>
  <CM service type : <M-VD-IE (CM service type)>>
  <Mobile station classmark 2 : <M-LV-IE (Mobile station classmark 2, 4, 4)>>
  <Mobile identity : <M-LV-IE (Mobile identity, 2, 9)>>
  <Priority : <O-TVD-IE (1000, Priority Level)>>
```

```
abstract.Mobile Identity type returns
  Mobile identity.V.Type of identity.abstract

abstract.IMSI returns
  Mobile identity.V.IMSI.abstract

abstract.TMSI returns
  Mobile identity.V.TMSI/P-TMSI.abstract

abstract.IMEI returns
  Mobile identity.V.IMEI
;
```

1.2.13 Demande d'identité GSM (ou « IDENTITY REQUEST »)

1.2.13.1 Description abstraite

```
action IDENTITY REQUEST ::=
abstract
 parameters
  Mobile Identity Type : MM.Mobile Identity Type
 end
```

1.2.13.2 Syntaxe de transfert

```
transfer ::=
  <spare half octet : {bit*4 = 0000}>
  <Identity type : <M-VD-IE (Identity type)>>
;
```

1.2.14 Réponse d'identité (ou « IDENTITY RESPONSE »)

1.2.14.1.1 Description abstraite

```
action IDENTITY RESPONSE ::=
abstract
 parameters
  Mobile Identity Type : MM.Mobile Identity Type
 end
```

```
if Mobile Identity Type = IMSI then set IMSI
if Mobile Identity Type = IMEI then on Connection parameters {
  set IMEI
}
if Mobile Identity Type = IMEISV then on Connection parameters {
  set IMEI
  set Software Version Number := Integer (0..99)
 }
```

1.2.14.2 Syntaxe de transfert

```
transfer ::=
  <Mobile identity : <M-LV-IE (Mobile identity, 2, 10)>>
;
```

1.2.15 Acceptation de mise à jour de localisation (ou "LOCATION UPDATING ACCEPT")

1.2.15.1 Description abstraite

```
action LOCATION UPDATING ACCEPT ::=
abstract
 parameters
  TMSI allocation status : Enumerated (new TMSI, keep TMSI, erase TMSI)
  Follow on proceed : Boolean
 end
 set Location update state := updated
 on Location parameters {
  set Registered location area
  if TMSI allocation status = new TMSI then on LAI+TMSI {
   set Location Area Identification
   set TMSI
  }
  if TMSI allocation status = erase TMSI then set LAI+TMSI := null
  set CTS permission := Boolean
 }
```

1.2.15.2 Syntaxe de transfert

```
transfer ::=
```

```
<Location area identification : <M-V-IE (Location area identification, 5)>>
<Mobile identity : <O-TLV-IE (0001 0111, Mobile identity, 3, 10)>>
<Follow on proceed : <O-T-IE (1010 0001)>>
<CTS permission : <O-T-IE (1010 0010)>>

abstract.TMSI allocation status returns
  switch (Mobile identity = <null>, keep TMSI,
    Mobile identity.V.TMSI/P-TMSI = 1**, erase TMSI,
    Mobile identity.V.TMSI/P-TMSI = bit**, new TMSI)

abstract.TMSI returns
  Mobile identity.V.TMSI/P-TMSI.abstract
;
```

1.2.16 Rejet d'une mise à jour de localisation (ou « LOCATION UPDATING REJECT »)

1.2.16.1 Description abstraite

```
action LOCATION UPDATING REJECT ::=
abstract
 parameters
  Reject cause : MM.Reject cause
 end
 set Location update state := roaming not allowed
```

1.2.16.2 Syntaxe de transfert

```
transfer ::=
  <Reject cause : <M-V-IE (Reject cause downlink, 1)>>
;
```

1.2.17 Information de gestion de mobilité (ou "MM INFORMATION")

1.2.17.1 Description abstraite

```
action MM INFORMATION ::=
abstract
 parameters
  Universal time[0..1] : MM.Universal Time
```

```
end
on Location parameters {
  on Registered location area {
    OPTIONAL set Local time zone
    OPTIONAL set Daylight Saving Time
    on PLMN {
      OPTIONAL set Full name for network
      OPTIONAL set Short name for network
}}}
on Connection parameters {
  on Serving cell {
    OPTIONAL set LSA identity
 }}
```

1.2.17.2 Syntaxe de transfert

```
transfer ::=
  <Full name for network : <O-TLV-IE (0100 0011, Network Name, 3, 255)>>
  <Short name for network : <O-TLV-IE (0100 0101, Network Name, 3, 255)>>
  <Local time zone : <O-TV-IE (0100 0110, Time Zone, 2)>>
  <Universal time and local time zone : <O-TV-IE (0100 0111, Time Zone and Time, 8)>>
  <LSA Identity : <O-TLV-IE (0100 1000, LSA Identifier, 2, 5)>>
  <Network Daylight Saving Time : <O-TLV-IE (0100 1001, Daylight Saving Time, 3, 3)>>

  function Local time zone returns
    iif (exist (Local time zone), Local time zone,
      iif (exist (Universal time and local time zone),
        Universal time and local time zone.Local time zone,
        <no string>))

  function Universal time returns
    Universal time and local time zone.Universal time
;
```

1.2.18 Statut dans le sens descendant de la Gestion de mobilité (ou "MM STATUS DOWNLINK")

1.2.18.1 Description abstraite

```
action MM STATUS DOWNLINK ::=
abstract
 parameters
```

```
    Reject cause : MM.Reject cause
  end
```

1.2.18.2 Syntaxe de transfert

```
transfer ::=
  <Reject cause : <M-V-IE (Reject cause downlink, 1)>>
;
```

1.2.19 Statut dans le sens montant de la Gestion de mobilité (ou "MM STATUS UPLINK")

1.2.19.1 Description abstraite

```
action MM STATUS UPLINK ::=
abstract
 parameters
   Reject cause : MM.Reject cause
 end
```

1.2.19.2 Syntaxe de transfert

```
transfer ::=
  <Reject cause : <M-V-IE (Reject cause uplink, 1)>>
;
```

1.2.20 Commande de réallocation TMSI (ou "TMSI REALLOCATION COMMAND")

1.2.20.1 Description abstraite

```
action TMSI REALLOCATION COMMAND ::=
abstract
 on Location parameters {
   PREPARE
   set LAI+TMSI
  }
```

1.2.20.2 Syntaxe de transfert

```
transfer ::=
  <Location area identification : <M-V-IE (Location area identification, 5)>>
  <Mobile identity : <M-LV-IE (Mobile identity, 2, 9)>>

  abstract.TMSI returns
    Mobile identity.V.TMSI/P-TMSI.abstract
;
```

1.2.21 Réallocation TMSI effectuée (ou "TMSI REALLOCATION COMPLETE")

1.2.21.1 Description abstraite

```
action TMSI REALLOCATION COMPLETE ::=
abstract
 COMMIT
```

1.2.21.2 Syntaxe de transfert

```
transfer ::=
  null
;
```

2 Liste de zones de localisation (ou "LA list")

2.1 Description abstraite

```
<LA list> ::=
 abstract
   Location area description[0..infinite] :
    + PLMN : -> UE context.PLMN list
    + LAC : RIL3.Location Area Code
    + Local time zone[0..1] : Time Zone
    + Network daylight saving time[0..1] : Daylight Saving Time
;
```

3 Liste de réseaux (ou « PLMN list »)

3.1 Description abstraite

```
<PLMN list> ::=
 abstract
   PLMN description[0..infinite] :
    + PLMN Id : PLMN Identity
    + Full name for network[0..1] : Network Name
    + Short name for network[0..1] : Network Name
;
```

4 Paramètre d'authentification AUTN (ou "Authentication Parameter AUTN")

4.1 Description abstraite

```
<Authentication Parameter AUTN> ::=
 abstract
   Sequence number xor Anonymity Key : Bit string (48)
   Authentication management field : Bit string (16)
   Message authentication code : Bit string (64)
```

4.2 Syntaxe de transfert

```
transfer ::=
   <Sequence number xor Anonymity Key : bit*48>
   <Authentication management field : bit*16>
   <Message authentication code : bit*64>
;
```

5 Type service CM (ou "CM Service Type")

5.1 Description abstraite

```
<CM Service Type> ::=
 abstract
   service type : Enumerated (MO call or packet est, emergency call, SMS, SS, VGC, VBS, LCS)
```

5.2 Syntaxe de transfert

```
transfer ::=
```

```
    <service type : {0001|0010|0100|10 bit(2)}>
;
```

6 Correction d'heure d'été (ou "Daylight Saving Time")

6.1 Description abstraite

```
<Daylight Saving Time> ::=
 abstract
   Daylight Saving Time adjustment : Enumerated (0, +1 hour, +2 hour)
```

6.2 Syntaxe de transfert

```
 transfer ::=
   <spare bit> * 6
   <Daylight Saving Time adjustment : {00 | 01 | 10 }>
;
```

7 IMEI

7.1 Description abstraite

```
<IMEI> ::=
 abstract
   Type Approval Code[6] : Integer (0..9)
   Final Assembly Code[2] : Integer (0..9)
   Serial Number[6] : Integer (0..9)
```

7.2 Syntaxe de transfert

```
 transfer ::=
   < digit one : digit> 1 bit*3
   <pairs : <digit odd : digit> <digit even : digit>>*7

   function Digit(n) : integer returns
     iif(n=1, integer(digit one),
       iif(n%2 = 0, integer(pairs[n/2].digit even), integer(pairs[(n-1)/2].digit odd)))

   abstract.Type Approval Code[n] returns  -- n from 0 to 5, MSB to LSD
     Digit(n+1)
```

```
    abstract.Final Assembly Code[n] returns   -- n from 0 to 1, MSD to LSD
      Digit(n+7)

    abstract.Serial Number[n] returns         -- n from 0 to 5, MSD to LSD
      Digit(n+9)
;
```

8 IMSI

8.1 Description abstraite

```
<IMSI> ::=
 abstract
   MCC : MCC
   NMSI[3..12] : Integer (0..9)
```

8.2 Syntaxe de transfert

```
transfer ::=
  {< digit one : digit> <oddeven : bit> bit*3
    <pairs : <digit odd : digit> <digit even : digit>>**
    <pairs : <spare bit>*4 <digit even : digit>>*(1-integer(oddeven))}
  & octet*(4..8)

  function Digit(n) : integer returns
    iif(n=1, integer(digit one),
      iif(n%2 = 0, integer(pairs[n/2].digit even), integer(pairs[(n-1)/2].digit odd)))

  abstract.MCC returns
    Digit(1)*100 + Digit(2)*10 + Digit(3)

  abstract.NMSI.size returns
    (exist (pairs) *2) - 4 + integer (oddeven)

  abstract.NMSI[n] returns          -- n from 0 to (size-1)
    Digit(n+4)
;
```

9 LAI et TMSI

9.1 Description abstraite

```
<LAI+TMSI> ::=
 abstract
   Location Area Identification : RIL3.Location Area Identification
   TMSI : Temporary Mobile Station Identity
;
```

10 Identifiant LSA (ou « LSA Identifier »)

10.1 Description abstraite

```
<LSA Identifier> ::=
 abstract
   LSA available : Boolean
   LSA ID[iif(LSA available, 1, 0)] :
    + LSA identifier scope : Enumerated (PLMN significant, universal)
    + Localised service area identity : Bit string (23)
```

10.2 Syntaxe de transfert

```
 transfer ::=
   {<LSA ID : <Localised service area identity : bit(23)>
       <LSA identifier scope : bit>
     | null }

   abstract.LSA available returns
     exist (LSA ID)

   abstract.LSA ID.LSA identifier scope returns
     iif (LSA identifier scope = 0, PLMN significant, universal)
;
```

11 MCC

11.1 Description abstraite

```
<MCC> ::=
 abstract
   MCC value : Integer (0..999)
;
```

12 MNC

12.1 Description abstraite

```
<MNC> ::=
 abstract
   MNC value : Integer (0..999)
;
```

13 Nom du réseau (ou "Network Name")

13.1 Description abstraite

```
<Network Name> ::=
 abstract
   Coding scheme : Enumerated (cell broadcast default alphabet, UCS2)
   Add country's initials : Boolean
   Text string : String
```

13.2 Syntaxe de transfert

```
 transfer ::=
   <ext : 1>
   <Coding scheme : {000|001}>
   <Add country's initials : bit>
   <number of spare bits in last octet : bit(3)>
   <Text string :      -- order of bits within text string ?
     <octet>**
     <{0}*(number of spare bits in last octet)>
     <bit*(8 - number of spare bits in last octet)>>
;
```

14 Identité de réseau (ou « PLMN Identity »)

14.1 Description abstraite

```
<PLMN Identity> ::=
 abstract
   MCC : MCC
   MNC : MNC
;
```

15 Identité temporaire de station mobile (ou "Temporary Mobile Station Identity", TMSI).

15.1 Description abstraite

```
<Temporary Mobile Station Identity> ::=
 abstract
   temporary identity : Bit string (32)
```

15.2 Syntaxe de transfert

```
 transfer ::=
   <temporary identity : bit*32>
;
```

16 Zone horaire (ou « Time Zone »)

16.1 Description abstraite

```
<Time Zone> ::=
 abstract
   Delta with GMT : Integer (-99..99)
```

16.2 Syntaxe de transfert

```
 transfer ::=
   <sign : bit>
   <digit 1 : bit(3)>
   <digit 2 : bit(4)>

   abstract.Delta with GMT returns          -- units are minutes
     15 * iif (sign = 0, integer(digit 1)*10 + integer(digit 2), - integer(digit 1)*10 - integer(digit 2))
;
```

ANNEXE 4.
Module de description formelle de protocole selon une variante

Un module de description formelle de protocole selon une variante de l'invention est décrit dans l'annexe 4 et comprend des éléments de description formelle spécifiant un protocole de communication.

Le module contient trois types de description :

- des descriptions abstraites de types d'objet (correspondant à une syntaxe de type a) illustrée précédemment) ;
- des descriptions abstraites d'actions (messages par exemple) (correspondant à une syntaxe de type e) ;
- et des descriptions de transfert s'appliquant soit à des messages, soit à des objets en tant que partie de message (correspondant à une syntaxe de type b).

A chaque type correspond un langage formel distinct.

Le module est essentiellement organisé comme une liste de types d'objet. A chaque type d'objet sont associées une description abstraite, éventuellement une ou plusieurs descriptions de transfert, et éventuellement une ou plusieurs actions. A chaque action, sont associées une description abstraite et éventuellement une ou plusieurs descriptions de transfert.

Pour des raisons pratiques, l'organisation générale d'un module permet soit de grouper les descriptions selon la logique ci-dessus, soit de séparer les éléments (par exemple pour grouper tous les messages ensemble), auquel cas les liaisons sont explicitées (par exemple la description d'un message indique alors le type d'objet auquel il est associé).

1 Syntaxe de module

Un module est composé d'un en-tête, suivi d'une liste de description de type d'objet, de descriptions autonomes d'actions et de descriptions autonomes de structure de transfert.

1.1 En-tête de module

L'en-tête de module comporte son nom, le scope par défaut.

2 Descriptions de types d'objet

Une description comporte au minimum:

- une description abstraite;
- zéro, une ou plusieurs descriptions de transfert; et
- zéro, une ou plusieurs descriptions de commandes.

Une description de transfert décrit une structure de chaîne binaire qui peut représenter correctement la structure abstraite (voir le chapitre sur le sujet).

Une action (message) associée au type d'objet décrit des modifications pouvant être demandées à une instance dudit type d'objet.

3 Descriptions abstraites de types d'objet

Une description abstraite de type d'objet se présente usuellement comme un ensemble de composants. Chaque composant est caractérisé par :

- un nom de composant,
- des informations de présence, et
- une description de type.

3.1 Informations de présence

Un composant peut être muni d'un index (tableau). Par défaut un composant n'a qu'une copie, un index permet d'avoir un nombre variable de copies, distinguées par une valeur d'index. Différents cas sont distinguables pour le nombre de copies.

3.1.1 Nombre fixe de copies

Le nombre de copies est spécifié par un entier. Le cas par défaut, une seule copie, est un cas particulier de nombre fixe de copies.

3.1.2 Nombre variable libre

Le nombre de copies, possible est alors typiquement indiqué par un intervalle d'entiers. Un cas particulier est un nombre quelconque de copies (intervalle de 0 à l'infini).

3.1.3 Nombre variable conditionnel

Le nombre de copies, acceptable dépend des valeurs d'autres éléments de la structure.

3.2 Description de type

Cette description indique la structure du composant lui-même. Différentes descriptions sont possibles.

3.2.1 Composants élémentaires

Cela inclut au minimum:

- un composant de type entier, au sens mathématique du terme;
- un composant de type énuméré : la liste des valeurs possibles est donnée explicitement, commune liste de symboles abstraits ; et
- un composant de type chaîne de bits.

3.2.2 Structure

Une structure se présente comme une liste de composants.

3.2.3 Union

Une union est une écriture abrégée pour un ensemble de composants de présence exclusive, i.e., au plus un des composants est présent dans une instance de l'union.

3.2.4 Référence

Une référence est une structure dont le contenu est défini par référence à un autre type d'objet.

3.2.5 Index

Un composant de type index indique une instance particulière parmi un ensemble d'objets de même type. La définition d'un composant de type index doit inclure la référence à un composant indexé d'un objet particulier. Cela peut être soit un composant inclus dans la structure dans laquelle est défini le composant de type index, soit un composant inclus dans un contexte, un objet unique commun à l'émetteur et le récepteur.

4 Description d'action.

Une description d'action comporte un nom de commande, une description abstraite d'action, et une ou plusieurs descriptions de transfert.

Une description de transfert décrit une structure de chaîne binaire qui peut représenter correctement l'action (voir le chapitre sur le sujet).

5 Description abstraite d'action.

Une description abstraite d'action comporte éventuellement une liste de paramètres, et éventuellement une liste d'actions élémentaires. (Les instructions demandent en général des paramètres qui apparaissent par exemple dans la représentation de l'action. Ces paramètres ne sont pas inclus dans la liste de paramètres : celle-ci ne contient que les paramètres qui ne peuvent pas être décrits plus précisément comme liés à une instruction élémentaire.)

La liste d'actions est constituée d'une ou plusieurs actions élémentaires. Chaque action élémentaire est constituée d'une éventuelle indication de condition, suivie d'une instruction parmi celles listées ci-après.

5.1 Indication de condition

Par défaut, une instruction doit être exécutée. Une indication de condition permet d'indiquer les conditions sous lesquelles l'instruction doit être exécutée.

Une première syntaxe permet de décrire une condition comme une expression booléenne construite à partir des paramètres.

Une syntaxe simplifiée permet de décrire à la fois l'existence d'un paramètre booléen spécifique et la condition d'exécution correspondant à ce paramètre (e.g., mot clé OPTIONAL).

5.2 Instruction de sélection

Une instruction de sélection permet d'indiquer une sous-partie de l'objet sur lequel porte une séquence d'actions. On utilise par la suite le terme de 'nœud courant' pour désigner la sous-partie sur laquelle porte une instruction. Par défaut d'instruction de sélection, le nœud courant est un objet du type auquel est associée l'action. L'objet spécifique est déterminé implicitement.

5.3 Instruction d'affectation (SET)

Une instruction d'affectation demande l'affectation d'une nouvelle valeur à un composant du nœud courant (dans son intégrité, ce qui peut être un sous-arbre assez important). Il y a plusieurs manières dont la nouvelle valeur peut être indiquée : paramètre implicite de type implicite ou explicite, valeur explicite ou calculée.

5.3.1 Paramètre implicite de type implicite

Dans ce cas, l'instruction ne contient rien de plus. Le message (représentation de l'action) contient alors un paramètre du type du composant, tel que décrit dans la description abstraite d'objet.

5.3.2 Paramètre implicite de type explicite

Dans ce cas l'instruction inclut un nom de type d'objet. Ce type doit être compatible avec le type du composant (sous-type).

5.3.3 Valeur explicite

Dans ce cas, l'instruction inclut une expression dont le résultat doit être compatible avec le type de composant.

5.3.4 Valeur calculée

Dans ce cas, l'instruction inclut une liste d'instructions. Ces instructions itémisent les actions aboutissant à l'affectation du composant (typiquement sous-composant par sous-composant).

5.3.5 Vidage

Cela ne s'applique qu'aux composants indexés (mais sans indication d'index dans l'instruction SET) et qui accepte un nombre nul de copies. L'instruction met le nombre de copies à zéro.

5.4 Instruction d'ajout (ADD)

Une instruction d'ajout inclut un nom de composant indexé et une valeur. Cette instruction demande l'ajout d'une nouvelle copie au composant, ayant la valeur indiquée (dans son intégrité, ce qui peut être un sous-arbre assez important). Il y a plusieurs manières dont la nouvelle valeur peut être indiquée, comme dans le cas de l'instruction d'affectation.

5.5 Instruction d'envoi (SEND)

Une instruction d'envoi indique l'inclusion d'un paramètre dont la valeur est la valeur courante du composant du côté de la source. L'action du côté du récepteur est de vérifier la cohérence, mais la conséquence d'une détection d'incohérence n'est pas précisée.

5.6 Instruction de choix (CHOICE)

Une instruction de choix indique qu'une et une seule des actions est demandée pour chaque occurrence de l'action.

5.7 Instruction d'exécution d'une action (DO)

Une instruction d'exécution permet d'exécuter une action compatible avec le type de composant courant.

5.8 Instructions liées au retour en arrière

Une modification de données peut être demandée « à l'essai » et refusée éventuellement par le récepteur. Ceci est géré par des instructions spécifiques de préparation (PREPARE), d'acceptation de modifications associées à une préparation (COMMIT) et de refus de modifications (UNDO).

5.8.1 L'instruction de préparation (PREPARE)

Cette instruction indique que les modifications qui la suivent et applicables au nœud courant peuvent être annulées.

5.8.2 L'instruction d'acceptation (COMMIT)

Cette instruction indique l'acceptation des modifications sujettes à une préparation (PREPARE) sur le nœud courant.

5.8.3 L'instruction de refus de modification (UNDO)

Cette instruction indique le refus des modifications sujettes à une préparation (PREPARE) sur le nœud courant.

6 Description de transfert

La description de transfert est à la base CSN.1 version 2.2, avec les adaptations suivantes.

6.1 En-tête

Quand la description de transfert est unique pour un type d'objet, le nom de structure CSN.1 peut être omis, et est alors le même que le nom de type.

6.2 Fin de structure

Le « ; » final est omis (la fin est détectée soit par le « ; » de fin d'objet, soit par un des mots clés « transfer » (indiquant une alternative de codage), ou « action ».

6.3 Fonctions calculées de liaison avec la description abstraite

L'en-tête d'une fonction calculée peut être de la forme « abstract. » suivi d'un identificateur d'élément abstrait. Le type n'est pas indiqué et est celui de l'élément abstrait ainsi que défini dans la description abstraite.

Plus généralement, les règles de correspondance entre une description de transfert et une description abstraite (de type d'objet ou de message) font l'objet du chapitre 9.

6.4 Références au contexte

Une formule peut invoquer des valeurs d'élément de contexte.

7 Description séparée d'action

Une action peut être décrite en dehors de la description du type d'objet sur laquelle elle porte. La description d'action est alors précédée d'un en-tête composé du mot clé 'on' suivi du nom de type d'objet sue lequel porte l'action.

L'indication du type d'objet peut être omise dans le cas d'une description séparée d'action en suivant une autre s'appliquant au même type.

8 Description séparée de transfert

Une description de transfert peut être séparée de la description abstraite à laquelle elle correspond. Dans ce cas le nom de la structure de transfert est obligatoire.

9 Règles de correspondance

La correspondance entre une description abstraite et une description de transfert est basée sur la correspondance entre des noms de composants abstraits et des noms de label, plus des formules de représentation.

10 Noms et « scopes »

Le nom complet d'une action ou d'un type d'objet est composé d'un descriptif ou « scope » et d'un nom dans ce scope. L'écriture complète est alors le scope suivi d'un point suivi du nom dans le scope.

L'en-tête du module permet de donner un scope par défaut. On peut alors utiliser le nom local seul (non précédé du scope), le scope étant alors le scope par défaut.

Les noms du scope et de l'objet sont sensés être liés à la nature de l'objet.

Un nom (d'objet, d'action ou de scope) est composé de caractères libres, avec les exceptions nécessaires pour des raisons syntaxiques. Lors de la comparaison entre deux noms, les chaînes consécutives d'espace, tabulation et fin de ligne sont considérées comme équivalentes à un seul espace, sauf au début et à la fin, où elles sont considérées comme équivalentes à la chaîne vide ; la casse n'est pas prise en compte (ni la couleur ou autres attributs de même farine). En d'autres termes, dans la comparaison de chaînes de caractères, les attributs de formatage (majuscule/minuscule/autre, c'est-à-dire la casse en terme de typographie; italique/gras/autre, ...), ou de présentation (couleur, fonte, ...) n'interviennent pas dans la comparaison.

ANNEXE 5

Exemple de langage de description formelle.

1 Spécification d'un langage de description formelle

Cette annexe décrit un exemple de langage de description formelle selon l'annexe 4.

2 Classes d'objet

La notion de classe d'objet est commune à de nombreux langages de programmation ou de spécification, sous des noms différents comme "classe" ou "type". En toute généralité, une classe est un ensemble d'objets, ceux-ci étant décrits à partir de caractéristiques communes à tous ces objets. Dans le cadre des protocoles, les caractéristiques partagées importantes sont les représentations (comment transmettre l'information permettant de désigner un objet particulier au sein de la classe), la sémantique externe (comment les objets se rattachent à une réalité extérieure), et les opérations (traitements informatiques pouvant impliquer les objets).

La description d'une classe inclut en premier lieu la spécification d'un ensemble minimal de caractéristiques permettant de distinguer tout objet de la classe. Selon le langage décrit ici, cela correspond à une description abstraite de la classe. Cette description se présente comme une liste d'attributs, chaque attribut étant lui-même un objet. La description comprend, en général, des indications informelles (i.e., non utilisables par les traitements, mais destinées aux humains), comme les noms des attributs ou des commentaires.

Des descriptions CSN.1 permettent de spécifier des représentations par des chaînes binaires ou de caractères, ainsi que les règles implicites ou explicites permettant de mettre en relation une représentation et un élément de la classe selon la description abstraite.

Finalement, des traitements spécifiques aux objets de la classe peuvent être

inclus directement dans la description de la classe (méthodes). Le corps d'une méthode (la description détaillée du traitement) est décrit en C++ modifié, les modifications permettant le couplage avec la description abstraite.

La description d'une classe comprend à la base:

- un identificateur Ceci est une référence formelle non ambiguë à la classe, sous forme d'une chaîne de caractères;
- aucune ou une description abstraite ;
- aucune, une ou plusieurs représentations ; et
- aucune, une ou plusieurs méthodes.

Des informations formelles supplémentaires peuvent être fournies. Ceci est visible dans la description détaillée de la syntaxe.

Par souci de modularité, la syntaxe du langage permet de dériver une classe d'une autre, ainsi que de décrire des classes paramétrées, une manière de décrire en une seule fois plusieurs classes dont les différences peuvent être capturées par un petit nombre de paramètres formels.

2.1 Syntaxe

La syntaxe est la suivante:

```
classdecl ::=
  "class" id "{" classdesc "}" ";" |
  "class" id ":" "public" idlist "{" classdesc "}" ";"
```

Le deuxième cas permet de définir une classe à partir d'une ou plusieurs autres (dérivation de classe).

```
idlist ::=
  id |
  id "," idlist
```

```
classdesc ::=
  aspectlist
```

```
aspectlist ::=
  aspect |
  aspectlist aspect
```

```
aspect ::=
  "abstract" ":" abstractdesc |
  "transfer" ":" transferdesc |
  "parameters" ":" abstractdesc |
  "settings" ":" settingsdesc |
  "method" ":" methoddesc |
  "action" ":" methoddesc
```

Les chaînes d'identification acceptables sont décrites dans la section générale sur les identificateurs.

L'ordre d'apparition des entrées dans la liste n'a pas d'importance sauf mention contraire, et il peut y avoir plusieurs entrées de même nature.

S'il y a plusieurs entrées donnant une description abstraite, la description abstraite de la classe est la concaténation, dans l'ordre d'apparition, des différentes listes d'attributs.

Le même principe s'applique aux paramètres.

2.2 Dérivation de classe

Une forme simple de modularité consiste à décrire une classe comme l'extension d'une ou plusieurs classes préalablement décrites. Ce qui est étendu est limité à la description abstraite et aux traitements.

Les représentations ne peuvent pas être étendues.

La syntaxe permettant de décrire une classe par dérivation est la syntaxe normale à l'exception de l'en-tête qui mentionne alors la ou les classes dont la nouvelle est dérivée.

La notion de dérivation de classe n'est qu'un raccourci textuel. La classe ainsi décrite est celle correspondant à la description obtenue en incluant tous les aspects des classes parentes entre l'en-tête et la première entrée.

2.3 Classes paramétrées

Il arrive que plusieurs classes partagent la plupart de leurs caractéristiques, et que les différences peuvent être décrites par un nombre limité de paramètres de nature élémentaire, comme des booléens ou des entiers. Par exemple les listes d'entiers, finies ,ordonnées et de taille fixe peuvent être décrites comme une seule méta-classe ne différant que par un seul paramètre entier, la taille de la liste.

2.3.1 Description d'une méta-classe

La liste des paramètres est introduite par le mot-clé « parameters », et suit la même syntaxe qu'une liste d'attributs.

Les paramètres peuvent (et doivent) apparaître dans le reste de la description de la classe, en toute place où une constante nommée peut apparaître.

2.3.2 Invocation d'une classe paramétrée

Le nom d'une classe paramétrée ne peut être utilisé qu'accompagné d'un jeu de valeurs des paramètres. La syntaxe est décrite plus loin.

3 Description abstraite

La description d'une structure de donnée fait partie de nombreux langages de programmations, de programmes de gestion de base de données ou de spécifications de protocoles. Selon l'état de l'art, la description mélange des aspects abstraits (ce qui est spécifique à la structure indépendamment de toute représentation) et des aspects concrets (représentations, que ce soit pour la transmission ou la mémorisation).

Dans le langage de description formelle selon l'invention, ici décrit, les

deux aspects sont gardés aussi séparés que possible, et des syntaxes distinctes sont utilisées pour les descriptions abstraites et les représentations. La syntaxe de description contient des adaptations aux problèmes spécifiques de la spécification des protocoles.

Une structure abstraite est décrite de manière arborescente. A chaque noeud (qui correspond à un composant) non terminal, est associée une liste de descriptions de noeud. A un noeud terminal, ou feuille, est associé une classe élémentaire.

Chaque description de noeud comprend un identificateur, qui permet d'y référer. Cet identificateur est usuellement choisi de manière à donner aux lecteurs humains une idée de la sémantique réelle de l'attribut.

Il arrive que la liste contient des noeuds de même structure, et dont les identificateurs ne se distinguent que par un nombre (notion de tableau, de liste, de collection). Cela amène à l'idée de multiplicité, i.e., les valeurs possibles du nombre de noeuds dans un tel groupe.

Ainsi, une liste de noeuds est décrite comme une liste de triplets, un triplet comprenant:

- un identificateur;
- la multiplicité; et
- une description abstraite de structure.

La description abstraite peut être explicite, soit une référence à une structure nommée par ailleurs, soit une description de classe élémentaire.

La syntaxe permet d'ajouter à un noeud des informations supplémentaires, par exemple une valeur dite par défaut, dont l'application est décrite plus loin.

3.1 Syntaxe

Une description abstraite de structure est une séquence de description de noeuds:

```
abstractdesc ::=
```

```
complist
```

```
complist ::=
  compdesc ";" | complist compdesc ";"
```

Chaque noeud est décrit par un triplet identificateur/multiplicité/description :

```
compdesc ::=
  id "[" multipliplicitydesc "]" ":" typeanddef |
  id ":" typedesc |
  choiceconstruct
```

La première syntaxe est la fondamentale, les autres pouvant s'y ramener par réécriture formelle.

Dans le second cas, la multiplicité est implicitement 1..1. La construction "choice" est une notation abrégée pour le cas d'une présence exclusive d'un noeud parmi une liste.

3.2 Multiplicité

La multiplicité d'un noeud est le nombre de noeuds dans une structure abstraite partageant les mêmes identificateur et structure, la distinction étant faite par un numéro d'ordre. Ce nombre est souvent dynamique, i.e., varie d'une instance à une autre: ce qui doit être décrit est donc l'ensemble des valeurs acceptables pour ce nombre. La multiplicité est donc un sous-ensemble des entiers positifs ou nuls.

L'ensemble acceptable peut être lui-même dynamique, dépendant de la valeur d'autres attributs (présence conditionnelle). Le langage permet une description en détail de ces cas.

Un cas classique de présence conditionnelle est celui où l'ensemble total des nombres est 0..1.

Une syntaxe simplifiée permet de décrire de tels cas comme une expression booléenne.

La syntaxe est la suivante:

```
multiplicitydesc ::=
  integerset |
  condset
```

Le premier cas correspond aux multiplicités libres, i.e., non conditionnelles; les deux autres à des présences conditionnelles.

Pour la multiplicité libre, la syntaxe est la suivante:

```
integerset ::=
  integersetelem "," integerset
```

```
integersetelem ::=
  expression |
  interval
```

```
interval ::=
  expression ".." expression |
  expression ".." "infinite"
```

Les expressions « integersetelem » et « interval » doivent retourner une valeur entière positive ou nulle.

La deuxième expression « interval » doit retourner une valeur supérieure ou égale à celle de la première.

Pour une multiplicité conditionnelle, la syntaxe est la suivante:

```
condset ::=
  integerset "|" expression |
  expression
```

L'ensemble d'entiers doit inclure les multiplicités possibles. Cette information est redondante puisqu'elle peut se dériver de l'expression qui suit; sa présence obligatoire améliore la compréhension et simplifie la compilation.

L'expression conddesc doit retourner soit un ensemble d'entiers, soit un entier, soit un booléen.

Par convention un entier seul est assimilé à l'ensemble réduit à sa seule valeur. Un booléen est traduit comme suit: la valeur "vrai" correspond à l'ensemble 1..1, la valeur fausse à l'ensemble 0..0.

3.2.1 La construction CHOICE

Cette syntaxe permet de simplifier l'écriture des cases où la présence de plusieurs noeuds est exclusive (au plus un d'entre eux peut être présent), et dépend dynamiquement de la valeur d'un noeud terminal, appelé le discriminant.

Par exemple, la liste suivante:

```
discri: 0..2;
el1 [0..1 | discri=0]: typ1;
el4 [0..1 | discri=2]: typ4;
el2 [0..1 | discri=1]: typ2;
el3 [0..1 | discri=2]: typ3;
```

peut s'écrire alternativement

```
CHOICE discri:0..2 {
0 : { el1 : typ1;}
1 : { el2 : typ2;}
2 : { el3 : typ3;
```

el4 : typ4;}
};

La syntaxe est la suivante:

```
choiceconstruct ::=
  "CHOICE" id ":" typedesc "{" branchlist "}"
```

```
branchlist ::=
  branchdesc ";" |
  branchlist ";" branchdesc ";"
```

```
branchdesc ::=
expr ":" "{" complist "}"
```

Si la description d'un noeud dans une branche inclut une multiplicité explicite, la multiplicité effective est 0 si le discriminant n'a pas la valeur adéquate, et la multiplicité indiquée sinon.

Incompatibilité des identificateurs.

L'équivalence est exacte. En particulier, ni la construction elle-même, ni une branche n'est une structure nommée; les listes de noeuds sont au même niveau hiérarchique, celui du point où apparaît le mot clé CHOICE. Par conséquence, il n'est pas autorisé d'avoir le même identificateur pour un noeud dans une branche et pour un noeud apparaissant au même niveau que le CHOICE.

Par contre, il est autorisé que le même nom de noeud apparaisse dans deux ou plus branches d'un même CHOICE: cela correspond alors à un même et unique noeud, la structure abstraite devant être la même. La multiplicité peut être différente.

4 Description de structure de noeud

4.1 Descriptions de type.

Les différentes possibilités sont résumées par la syntaxe suivante:

```
typedesc ::=
  "{" complist "}" |
  id | leafdesc |
  id "(" exprlist ")" |
  "*" typedesc
```

La première possibilité est une description abstraite de syntaxe, parenthésée.

La seconde possibilité est une référence à une classe pré-définie, décrite par son identificateur.

La troisième possibilité est une description de classe élémentaire, prédéfinie par le langage. Les syntaxes possibles et leurs significations sont décrites plus loin.

La quatrième possibilité est une référence à une classe paramétrée, constituée de l'identificateur et d'une liste de paramètres effectifs.

La cinquième possibilité est une référence à une classe pré-définie, comme dans le premier cas.

L'étoile indique une différence de réalisation, précisément qu'il s'agit d'une instance partagée avec d'autres structures. (En terme pratique, cela indique une implémentation par référence dynamique (pointeur) plutôt que par valeur explicite).

Les classes élémentaires acceptées comprennent les classes suivantes:

- énuméré strict;
- booléen;
- caractère;
- cardinal fini (entier naturel positif ou nul);

- dénombrable (entier étendu de la valeur infinie) ;
- différences (entier signé);
- indéfini ou "void".

Les syntaxes sont:

```
leafdesc ::=
  enumdesc |
  integerdesc |
  chardesc |
  "boolean" |
  "void"
```

Chaque cas est décrit séparément ci-après.

4.1.1 Référence avec paramètres

La liste des expressions qui suit l'identificateur doit correspondre en nombre et en type à la liste des paramètres définis pour la classe.

4.1.2 Enuméré strict

Il s'agit d'un type fini (i.e., les valeurs possibles pour une instance de la classe sont en nombre fini). La liste des valeurs est donnée explicitement et complètement par une liste d'identificateurs.

Cette liste est abstraite, i.e., n'inclut aucune autre signification que le fait que deux identificateurs distincts correspondent à deux valeurs distinctes, ce qui guide l'implémentation du test à l'égalité de valeur de deux instances.

La sémantique formelle est limitée à ce test. La sémantique externe est en général indiquée pour les humains par le choix des identificateurs.

La syntaxe stipule par nécessité un ordre, mais celui n'a pas de signification formelle.

La syntaxe est la suivante :

```
enumdesc ::=
  "enumerated" "(" valuelist ")" |
  "enumerated" id "(" valuelist ")"
```

```
valuelist ::=
  value |
  valuelist "," value
```

```
value ::=
  id "=" number |
  id
```

L'identificateur qui peut apparaître entre enumerated et le caractère « (» est une notation abrégée pour déclarer et invoquer la classe simultanément.

4.1.3 Cardinaux finis

Cela correspond au concept mathématique d'entiers positifs ou nuls, et a la sémantique formelle correspondante. La sémantique externe normal de cette classe est le décompte de choses, et l'usage de cette classe devrait en toute rigueur être limitée à de tels cas.

La sémantique formelle inclut, entre autres, les tests de comparaison et d'égalité, ainsi que les opérations arithmétiques usuelles.

Un cas évident d'usage légitime est le décompte du nombre de copies d'un noeud indexé. Ainsi, une multiplicité effective est un décompte, et une description de multiplicité est un sous-ensemble des entiers positifs ou nul.

La syntaxe est incluse dans la description des cardinaux dénombrables.

4.1.4 Différences

Il s'agit des entiers négatifs, positifs ou nul.

La syntaxe est incluse dans la description du cas suivant.

4.1.5 Cardinaux dénombrables

Il s'agit de l'ensemble des entiers positifs ou nul, étendu d'une valeur spéciale, l'infini. Cette classe est très utile en pratique, par exemple pour une borne supérieure.

Les entiers, étendus ou non, sont rarement utilisés avec l'intervalle complet des valeurs. La syntaxe permet de décrire des classes dérivées, limitées à un sous-ensemble fini des valeurs.

La syntaxe est la suivante :

```
integerdesc ::=
  "integer" |
  "integer" "(" integerset ")"
```

Le premier cas correspond aux entiers positifs ou nuls (cardinaux finis).

```
integerset ::=
  integersetelem "," integerset
```

```
integersetelem ::=
  expression |
  interval
```

```
interval ::=
  expression ".." expression |
  expression ".." "infinite"
```

Aucun élément de syntaxe permet d'étendre les entiers négatifs vers -infini ($-\infty$).

Voici, quelques exemples dans le contexte de la norme UMTS :

- "TGPRC" : Integer (1..511, infinite); (Le nombre d'intervalles non alloués à la transmission (ou « transmission gap pattern » en anglais) parmi la séquence de d'intervalle de temps non alloués à la transmission (ou « Transmission Gap Pattern Sequence » en anglais).
- "TGSN" : Integer (0..14); (le numéro d'intervalle de temps élémentaire de la première période non allouée à la transmission (en anglais « slot number of the first transmission gap slot ») dans le TGCFN.
- "TGL1" : Integer (1..14); (la longueur de la période non allouée à la transmission dans le motif des creux de transmission exprimé en nombre d'intervalles de temps élémentaires.

Dans ces exemples, les commentaires ont été gardés pour montrer qu'il s'agit bien de décompter une valeur.

4.1.6 Booléen

C'est un énuméré strict particulier, qui intervient comme résultat des opérations booléennes. Le langage de description formelle suit la convention la plus usuelle soit le couple (faux, vrai) ou (false, true), avec comme sémantique (formelle et réelle) que "true" est le résultat du test « 1 = 1 ».

La classe boolean est donc prédéfinie comme : enumerated (false, true).

4.1.7 Type indéfini ou void

« Void » indique un type général, non défini.

5 Représentations

Une représentation capture tous les attributs d'un objet sous forme d'une chaîne de symboles binaires ou caractères.

Les représentations sont décrites en CSN.1.

6 Règles de correspondance

Les règles de correspondance permettent de mettre en relation une description abstraite et une représentation.

Dans le langage de description formelle, la correspondance est définie à partir de fonctions. Une fonction prend comme paramètre un objet et renvoie une valeur. Vu ainsi, une fonction a un nom et un type de résultat.

La description abstraite de la classe de l'objet permet de construire une liste de fonctions comme suit. A chaque noeud terminal correspond une liste de fonctions, la première renvoie un décompte, la multiplicité effective du noeud, c'est-à-dire le nombre de copies du noeud. Ensuite, pour chaque entier i (par convention positif ou nul, i.e. 0 est inclus), une fonction renvoie la valeur du $i^{ème}$ éléments.

Le type renvoyé par ces dernières fonctions est celui associé au noeud.

Pour les besoins de cette spécification, les notations suivantes sont introduites (l'usage formel des noms de fonction est développé au sein des adaptations du langage C++).

- "id .size()" est l'attribut renvoyant le nombre effectif de noeuds de id .
- "id" retourne la valeur du noeud dans le cas où la multiplicité est implicitement 1..1.
- "id [i]", où i est un entier naturel retourne la valeur de la $(i+1)^{ème}$ copie.

Quand un noeud n'est pas terminal, la notation avec point permet de construire récursivement toutes les fonctions.

6.1 Fonctions définies pour une représentation

Les labels d'une description CSN.1 déterminent une liste de fonctions, chacune renvoyant soit 'non instancié' soir une chaîne de symboles.

Pour chaque fonction définie à partir de la description abstraite, une représentation doit fournir une fonction correspondante. La correspondance est basée sur l'identité des noms, ainsi que, pour les fonctions correspondant aux noeuds terminaux, sur l'existence d'une règle de transcodage permettant de passer

d'une chaîne de symboles à la valeur abstraite.

Par exception, l'existence d'une valeur par défaut permet d'accepter un renvoi 'non instancié' pour un noeud. (La valeur est alors la valeur par défaut.)

Le transcodage pour un noeud terminal est obtenu soit par application de règles par défaut, soit par la spécification dans la description de transfert d'une fonction de transcodage.

6.2 Règles par défaut

6.2.1 Enumérés

Si des valeurs numériques d'encodage sont spécifiées dans la description abstraite, le décodage est celui d'un entier comme spécifié dans la section suivante, suivi du transcodage dérivé de la description abstraite.

Si aucune valeur d'encodage n'est donnée, le décodage par défaut est le décodage d'un entier suivi par le transcodage appliquant à i la $(i+1)^{ème}$ valeur de la liste défini dans la description du type abstrait.

6.2.2 Entiers

Ces règles ne s'appliquent qu'aux cas où la description abstraite est celle d'un entier non étendu.

6.2.3 Symboles binaires

Si la borne inférieure de l'ensemble d'entiers est positive ou nulle, le décodage est celui de la numérotation binaire, le premier bit de la chaîne étant le bit de poids fort. Le décodage est complet (toute chaîne est acceptable), mais non bijectif (l'ajout de 0 à l'avant ne change pas la valeur). Typiquement, l'encodage cherchera à limiter au mieux le nombre de 0 à l'avant.

Si la borne inférieure est strictement négative, le décodage est celui du code par complément à 2, le bit de poids fort étant le premier. Le décodage est complet, mais non bijectif (l'ajout à l'avant d'un bit identique au premier bit ne change pas la valeur).

6.2.4 Caractères

Le décodage est celui de la numérotation positionnelle usuelle, le premier chiffre étant celui de poids fort. Un caractère de signe ("+" ou "-") peut apparaître optionnellement avant le premier chiffre. Le décodage n'est pas complet: tout caractère autre que les chiffres de 0 à 9 entraîne une erreur de décodage, ainsi que l'usage d'un caractère de signe ailleurs qu'en position initiale. L'encodage n'est pas bijectif.

7 Méthodes et actions

Cette aspect du langage n'est pas directement en rapport avec la spécification de protocole. La syntaxe est indiquée ici pour mémoire.

Dans le cadre de ce langage, une action est un cas particulier de méthode, et n'est citée ici que pour mémoire.

7.1 Syntaxe déclarative

Une méthode ou une action est déclarée comme suit:

```
methoddesc ::=
  "function" id "(" paramdesclist ")" ":" id "{" ccode "}"
```

ANNEXE 6

Exemple de spécification d'un protocole réel (le protocole GSM de gestion de mobilité entre une station mobile (MS) et une infrastructure de réseau) utilisant le langage selon l'annexe 5.

L'exemple présente dans cette annexe reprend l'exemple précédemment décrit en regard de l'annexe 3, selon le langage illustré dans l'annexe 5.

1 Contexte de protocole de gestion de mobilité (ou MM).

```
module MM
#include "RIL3"
```

Le module RIL3, non illustré plus en détail, comprend des descriptions de détail invoquées dans la suite par des identificateurs commençant par 'RIL3'. Ces descriptions sont communes avec d'autres protocoles inclus dans la la couche trois radio (RIL3 est une abbréviation de l'anglais « Radio Interface Layer 3 » ou « Couche 3 d'interface radio » en français).

1.1 Contexte MM

1.1.1 Description abstraite

```
class "MM context" {
 abstract :
  "IMSI" [0..1] : "IMSI" ;
  "Connect state" : Enumerated ("not connected", "connected") ;
  "Connection parameters" [0..1 | "Connect state" = "connected"] : {
```

La partie décrivant une présence conditionnelle peut être ignorée. Elle permet d'éviter une perte d'information.

```
    "Serving cell" : * "cell" //-> UE context.Cell list
    "IMEI" [0..1] : "IMEI" ;
    "Software Version Number" [0..1 | present ("IMEI")] : Integer (0..99) ;
    "Mobile Station Classmark 1" : RIL3."Mobile Station Classmark 1" ;
"Mobile Station Classmark 2 delta" [0..1] : RIL3."Mobile Station Classmark 2 delta" ;
    "Mobile Station Classmark 3" [0..1] : RIL3."Mobile Station Classmark 3" ;
    "Has been authenticated" : Boolean ;
    "Is ciphered" : Boolean ;
    "Service data" [0..infinite] : {
      "CM service type" : "CM Service Type" ;
      "Priority" : RIL3."Priority Level" ;
    } ;
    "Location update state" : Enumerated ("roaming not allowed", "updated") ;
    "Location parameters" [0..1 | iif("Location update state" = "updated", 1, 0)] : {
     "LAI+TMSI"[0..1] : "LAI+TMSI" ;
     "Registered location area" : *"Location area" ;//UE context.LA list
```

Selon une variante, la syntaxe n'indique que le type, alors que le mode de réalisation décrit indique l'ensemble indexé dont fait partie l'objet à pointer. L'ensemble est laissé ici en commentaire.

```
    "Detach flag" : Boolean ;
    "CTS permission" : Boolean ;
    CKSN availability : Boolean ;
    "CKSN" [0..1 | iif(CKSN availability, 1, 0)] : RIL3."Ciphering Key Sequence Number" ;
    "GSM ciphering key" [0..1] : Bitstring (64) ;
    "UMTS security data" [0..1] : {
      "AUTN" : "Authentication Parameter AUTN" ;
      "UMTS ciphering key" [0..1] : Bitstring (128) ;
      "UMTS integrity key" [0..1] : Bitstring (128) ;
    } ;
} ;
```

1.1.2 Commandes

1.1.2.1 DL MM

1.1.2.1.1 Description abstraite

Selon le mode de réalisation décrit, on ne distingue pas les commandes (messages) des objets. Les descriptions qui suivent se présentent comme des classes.

```
class "DL MM" : {
abstract :
  CHOICE "message type" :   ( "LOCATION UPDATING ACCEPT",  "LOCATION UPDATING
REJECT",  "AUTHENTICATION REJECT",  "AUTHENTICATION REQUEST",  "TMSI
REALLOCATION COMMAND",  "CM SERVICE ACCEPT",  "CM SERVICE REJECT",  "CM
SERVICE PROMPT",  "IDENTITY REQUEST", "ABORT ",  "MM STATUS DOWNLINK",  "MM
INFORMATION")
{
  "LOCATION UPDATING ACCEPT" : "LOCATION UPDATING ACCEPT" ;
  "LOCATION UPDATING REJECT" : "LOCATION UPDATING REJECT" ;
  "AUTHENTICATION REJECT" : "AUTHENTICATION REJECT" ;
  "AUTHENTICATION REQUEST" : "AUTHENTICATION REQUEST" ;
  "TMSI REALLOCATION COMMAND" : "TMSI REALLOCATION COMMAND" ;
  "CM SERVICE ACCEPT" : "CM SERVICE ACCEPT" ;
  "CM SERVICE REJECT" : "CM SERVICE REJECT" ;
  "CM SERVICE PROMPT" : "CM SERVICE PROMPT" ;
  "IDENTITY REQUEST" : "IDENTITY REQUEST" ;
  "ABORT " : "ABORT " ;
  "MM STATUS DOWNLINK" : "MM STATUS DOWNLINK" ;
  "MM INFORMATION" : "MM INFORMATION" ;
 }
```

1.1.2.1.2 Syntaxe de transfert

```
transfer :
  -- flow of downlink MM messages
  <DL MM> ::=
  {<skip indicator : 0000>
    <protocol discriminator : 0101>
    0 0
    {00 0010 <LOCATION UPDATING ACCEPT : LOCATION UPDATING ACCEPT>
      | 00 0100 <LOCATION UPDATING REJECT : LOCATION UPDATING REJECT>
      | 01 0001 <AUTHENTICATION REJECT : AUTHENTICATION REJECT>
      | 01 0010 <AUTHENTICATION REQUEST : AUTHENTICATION REQUEST>
      | 01 1010 <TMSI REALLOCATION COMMAND : TMSI REALLOCATION COMMAND>
      | 10 0001 <CM SERVICE ACCEPT : CM SERVICE ACCEPT>
      | 10 0010 <CM SERVICE REJECT : CM SERVICE REJECT>
      | 10 0101 <CM SERVICE PROMPT : CM SERVICE PROMPT>
      | 10 1000 <IDENTITY REQUEST : IDENTITY REQUEST>
      | 10 1001 <ABORT : ABORT>
      | 11 0001 <MM STATUS : MM STATUS DOWNLINK>
      | 11 0010 <MM INFORMATION : MM INFORMATION>}
```

```
    ! <erroneous type : bit** = <no string>>}
  <spurious extension : bit** = null>
;
};
```

1.1.2.2 UL MM

1.1.2.2.1 Description abstraite

```
class "UL MM" {
abstract :
  CHOICE "message type" : ( "AUTHENTICATION FAILURE", "AUTHENTICATION RESPONSE",
"IDENTITY RESPONSE", "TMSI REALLOCATION COMPLETE", "CM SERVICE ABORT", "CM
SERVICE REQUEST SUBSEQUENT", "MM STATUS UPLINK") {
  "AUTHENTICATION FAILURE" : "AUTHENTICATION FAILURE" ;
  "AUTHENTICATION RESPONSE" : "AUTHENTICATION RESPONSE" ;
  "IDENTITY RESPONSE" : "IDENTITY RESPONSE" ;
  "TMSI REALLOCATION COMPLETE" : "TMSI REALLOCATION COMPLETE" ;
  "CM SERVICE ABORT" : "CM SERVICE ABORT" ;
  "CM SERVICE REQUEST SUBSEQUENT" : "CM SERVICE REQUEST SUBSEQUENT" ;
  "MM STATUS UPLINK" : "MM STATUS UPLINK" ;
  }
```

1.1.2.2.2 Syntaxe de transfert

```
transfer :
<UM MM> ::=
  -- flow of uplink MM messages once an RR connection is established
  {<skip indicator : 0000>
    <protocol discriminator : 0101>
    bit*2
    {01 1100 <AUTHENTICATION FAILURE : AUTHENTICATION FAILURE>
    | 01 0100 <AUTHENTICATION RESPONSE : AUTHENTICATION RESPONSE>
    | 01 1001 <IDENTITY RESPONSE : IDENTITY RESPONSE>
    | 01 1011 <TMSI REALLOCATION COMPLETE : TMSI REALLOCATION COMPLETE>
    | 10 0011 <CM SERVICE ABORT : CM SERVICE ABORT>
    | 10 0100 <CM SERVICE REQUEST : CM SERVICE REQUEST>
    | 11 0001 <MM STATUS : MM STATUS UPLINK>}
  ! <erroneous type : bit** = <no string>>}
  <spurious extension : bit** = null>
;
};
```

1.1.2.3 Abandon (ou « abort »)

1.1.2.3.1 Description abstraite

```
class "ABORT" {
abstract
"Reject cause" : "MM.Reject cause" ;
bind <ABORT> {
}
```

Le corps de la commande introduite par 'bind' n'est pas inclus dans l'exemple. Il est à décrire préférentiellement en C++. Ceci s'applique a toutes les commandes (introduites par 'bind' ou 'load') dans les sections qui suivent.

1.1.2.3.2 Syntaxe de transfert

```
transfer : <ABORT> ::=
   <Reject cause : <M-V-IE (Reject cause downlink, 1)>>
} ;
```

1.1.2.4 Echec d'authentification (ou « AUTHENTICATION FAILURE »)

1.1.2.4.1 Description abstraite

```
class "AUTHENTICATION FAILURE" {
abstract :
   "Reject cause" : "MM.Reject cause" ;
   "Authentication Failure parameter"[0..1 | iif ("Reject cause.cause grouping" = "network related failure", "Reject cause.network-related failure cause" = "synch. failure", false)] : "MM.Authentication Failure parameter"
bind <AUTHENTICATION FAILURE> {
}
```

1.1.2.4.2 Syntaxe de transfert

```
transfer <AUTHENTICATION FAILURE> ::=
   <Reject Cause : <M-V-IE (Reject Cause uplink, 1)>>
```

```
    < Authentication Failure parameter : <O-TLV-IE (0010 0010, Authentication Failure parameter,
16)>>
  };
```

1.1.2.5 Rejet d'authentification (ou « AUTHENTICATION REJECT »)

1.1.2.5.1 Description abstraite

```
class "AUTHENTICATION REJECT" {
abstract :
bind <AUTHENTICATION REJECT> {
}
```

1.1.2.5.2 Syntaxe de transfert

```
transfer <AUTHENTICATION REJECT> ::=
  null
};
```

1.1.2.6 Requête d'authentification (ou « AUTHENTICATION REQUEST »)

1.1.2.6.1 Description abstraite

```
class "AUTHENTICATION REQUEST" {
abstract :
   "Authentication Parameter RAND" : "MM.Authentication Parameter RAND" ;
   "Authentication Algorithm" : Enumerated ("UMTS", "GSM") ;
bind <AUTHENTICATION REQUEST> {
}
```

1.1.2.6.2 Syntaxe de transfert

```
transfer <AUTHENTICATION REQUEST> ::=
  <spare half octet : {bit*4 = 0000}>
  <Ciphering key sequence number : <M-VD-IE (Ciphering key sequence number Downlink)>>
  <Authentication parameter RAND : <M-V-IE (Authentication parameter RAND, 16)>>
  <Authentication parameter AUTN : <O-TLV-IE (0010 0000, Authentication parameter AUTN, 18,
18)>>

  abstract.Authentication algorithm returns
    -- if there is any error on AUTN, the authentication Algo will be considered GSM
```

```
    iif (exist(Authentication parameter AUTN), UMTS, GSM)
};
```

1.1.2.7 Réponse d'authentification (ou « AUTHENTICATION RESPONSE »)

1.1.2.7.1 Description abstraite

```
class "AUTHENTICATION RESPONSE" {
abstract :
   RES : Octet string (4..16) ;
load  <AUTHENTICATION RESPONSE> {
}
```

1.1.2.7.2 Syntaxe de transfert

```
transfer <AUTHENTICATION RESPONSE> ::=
   <Authentication response parameter : <M-V-IE (Authentication Response Parameter, 4)>>
   <Authentication response extension : <O-TLV-IE (0010 0001, Authentication Response
Parameter extension, 3, 14>>

   function RES returns
      <instance(Authentication response parameter.V.RES part 1)>
      {<instance(Authentication response extension.V.RES part 2)> | null}
};
```

1.1.2.8 Abandon de connexion pour service (ou « CM SERVICE ABORT »)

1.1.2.8.1 Description abstraite

```
class "CM SERVICE ABORT" {
abstract :

bind <CM SERVICE ABORT> {
   }
```

1.1.2.8.2 Syntaxe de transfert

```
transfer <CM SERVICE ABORT> ::=
   null
;
```

1.1.2.9 Acceptation de la demande de connexion pour service (ou « CM SERVICE ACCEPT »)

1.1.2.9.1 Description abstraite

```
class "CM SERVICE ACCEPT" {
abstract :
load <CM SERVICE REQUEST> {
}
```

1.1.2.9.2 Syntaxe de transfert

```
transfer <CM SERVICE ACCEPT> ::=
  null
};
```

1.1.2.10 Affichage du service de gestion de connexion (ou « CM SERVICE PROMPT »)

1.1.2.10.1 Description abstraite

```
class "CM SERVICE PROMPT" {
abstract :
  "PD and SAPI" : RIL3."PD and SAPI" ;
```

1.1.2.10.2 Syntaxe de transfert

```
transfer <CM SERVICE PROMPT> ::=
  <PD and SAPI of CM : <M-V-IE (PD and SAPI, 1)>>
} ;
```

1.1.2.11 Rejet de la demande de connexion pour service (ou « CM SERVICE REJECT »)

1.1.2.11.1 Description abstraite

```
class "CM SERVICE REJECT" {
abstract :
  Reject cause : MM.Reject cause ;
load <CM SERVICE REJECT> {
}
```

1.1.2.11.2 Syntaxe de transfert

```
transfer <CM SERVICE REJECT> ::=
  <Reject cause : <M-V-IE (Reject cause downlink, 1)>>
};
```

1.1.2.12 Demande de conexion additionnelle pour service (ou « CM SERVICE REQUEST SUBSEQUENT »)

1.1.2.12.1 Description abstraite

```
class "CM SERVICE REQUEST SUBSEQUENT" {
abstract :
   "Mobile Identity Type" : "MM.Mobile Identity Type" ;
  "IMSI" [0..1 | "Mobile Identity Type" = "IMSI"] : "IMSI" ;
  "TMSI" : [0..1 | "Mobile Identity Type" = "TMSI/P-TMSI"] : "TMSI" ;
  "CKSN availability" : "CKSN availability" ;
    "CKSN" [0..1 | "CKSN availability"] : CKSN ;
    "Mobile Station Classmark 1" :  "Mobile Station Classmark 1" ;
    "Mobile Station Classmark 2 delta" : "Mobile Station Classmark 2 delta" ;
      "CM Service Type" : "CM Service Type" ;
      "Priority" : "Priority" ;
```

1.1.2.12.2 Syntaxe de transfert

```
transfer : <CM SERVICE REQUEST> ::=
  <Ciphering key sequence number uplink : <M-VD-IE (Ciphering key sequence number Uplink)>>
  <CM service type : <M-VD-IE (CM service type)>>
  <Mobile station classmark 2 : <M-LV-IE (Mobile station classmark 2, 4, 4)>>
  <Mobile identity : <M-LV-IE (Mobile identity, 2, 9)>>
  <Priority : <O-TVD-IE (1000, Priority Level)>>

abstract.Mobile Identity type returns
    Mobile identity.V.Type of identity.abstract

  abstract.IMSI returns
    Mobile identity.V.IMSI.abstract

  abstract.TMSI returns
    Mobile identity.V.TMSI/P-TMSI.abstract
```

```
    abstract.IMEI returns
      Mobile identity.V.IMEI
};
```

1.1.2.13 Demande d'identité GSM (ou « IDENTITY REQUEST »)

1.1.2.13.1 Description abstraite

```
class "IDENTITY REQUEST" {
abstract :
   "Mobile Identity Type" : "MM.Mobile Identity Type" ;
```

1.1.2.13.2 Syntaxe de transfert

```
transfer : <IDENTITY REQUEST> ::=
   <spare half octet : {bit*4 = 0000}>
   <Identity type : <M-VD-IE (Identity type)>>
};
```

1.1.2.14 Réponse d'identité (ou « IDENTITY RESPONSE »)

1.1.2.14.1 Description abstraite

```
class "IDENTITY RESPONSE" {
abstract :
   CHOICE "Mobile Identity Type" : "MM.Mobile Identity Type"  {
   "IMSI" : {"IMSI": "IMSI"} ;
   "IMEI" : {"IMEI" : "IMEI"} ;
   "IMEISV" : {
     "IMEI" : "IMEI" ;
     "Software Version Number" : Integer (0..99) ;
   }} ;
```

1.1.2.14.2 Syntaxe de transfert

```
transfer : <IDENTITY RESPONSE> ::=
   <Mobile identity : <M-LV-IE (Mobile identity, 2, 10)>>
};
```

1.1.2.15 Acceptation de mise à jour de localisation (ou « LOCATION UPDATING ACCEPT »)

1.1.2.15.1 Description abstraite

```
class "LOCATION UPDATING ACCEPT" {
abstract :
   "TMSI allocation status" : Enumerated ("new TMSI", "keep TMSI", "erase TMSI") ;
   "Follow on proceed" : Boolean ;
    set Registered location area
    [0..1 | "TMSI allocation status" = "new TMSI"] {
      "Location Area Identification" :  "Location Area Identification" ;
      "TMSI" : "TMSI" ;
    } :
   "CTS permission" : Boolean ;

bind {
}
```

1.1.2.15.2 Syntaxe de transfert

```
transfer : <LOCATION UPDATING ACCEPT> ::=
  <Location area identification : <M-V-IE (Location area identification, 5)>>
  <Mobile identity : <O-TLV-IE (0001 0111, Mobile identity, 3, 10)>>
  <Follow on proceed : <O-T-IE (1010 0001)>>
  <CTS permission : <O-T-IE (1010 0010)>>

  abstract.TMSI allocation status returns
    switch (Mobile identity = <null>, keep TMSI,
      Mobile identity.V.TMSI/P-TMSI = 1**, erase TMSI,
      Mobile identity.V.TMSI/P-TMSI = bit**, new TMSI)

  abstract.TMSI returns
    Mobile identity.V.TMSI/P-TMSI.abstract
};
```

1.1.2.16 Rejet d'une mise à jour de localisation (ou « LOCATION UPDATING REJECT »)

1.1.2.16.1 Description abstraite

```
class "LOCATION UPDATING REJECT" {
abstract :
```

```
    "Reject cause" : "MM.Reject cause" ;

bind <LOCATION UPDATING REQUEST> {
//   set Location update state := roaming not allowed
}
```

1.1.2.16.2 Syntaxe de transfert

```
transfer : {LOACTION UPDATING REQUEST> ::=
  <Reject cause : <M-V-IE (Reject cause downlink, 1)>>
;
```

1.1.2.17 Information de gestion de mobilité (ou « MM INFORMATION »)

1.1.2.17.1 Description abstraite

```
class "MM INFORMATION" {
abstract :
   Universal time[0..1] : MM.Universal Time
      "Local time zone"[0..1] : "Time zone" ;
      "Network daylight Saving Time"[0..1] :  "Daylight Saving Time" ;
        "Full name for network"[0..1] :  "network name" ;
        "Short name for network"(0..1] :  "network name" ;
      "LSA identity"[0..1] : "LSA identity" ;
```

1.1.2.17.2 Syntaxe de transfert

```
transfer : <MM INFORMATION>  ::=
  <Full name for network : <O-TLV-IE (0100 0011, Network Name, 3, 255)>>
  <Short name for network : <O-TLV-IE (0100 0101, Network Name, 3, 255)>>
  <Local time zone : <O-TV-IE (0100 0110, Time Zone, 2)>>
  <Universal time and local time zone : <O-TV-IE (0100 0111, Time Zone and Time, 8)>>
  <LSA Identity : <O-TLV-IE (0100 1000, LSA Identifier, 2, 5)>>
  <Network Daylight Saving Time : <O-TLV-IE (0100 1001, Daylight Saving Time, 3, 3)>>

  function Local time zone returns
    iif (exist (Local time zone), Local time zone,
      iif (exist (Universal time and local time zone),
        Universal time and local time zone.Local time zone,
        <no string>))
```

```
    function Universal time returns
      Universal time and local time zone.Universal time
  };
```

1.1.2.18 Statut, dans le sens descendant, de la gestion de mobilité (ou « MM STATUS DOWNLINK »)

1.1.2.18.1 Description abstraite

```
class "MM STATUS DOWNLINK" {
 abstract :
   "Reject cause" : "MM.Reject cause" ;
```

1.1.2.18.2 Syntaxe de transfert

```
transfer <MM STATUS DOWNLINK> ::=
  <Reject cause : <M-V-IE (Reject cause downlink, 1)>>
};
```

1.1.2.19 Statut, dans le sens montant, de la gestion de mobilité (ou « MM STATUS UPLINK »)

1.1.2.19.1 Description abstraite

```
class "MM STATUS UPLINK" {
abstract :
  "Reject cause" : "MM.Reject cause" ;
```

1.1.2.19.2 Syntaxe de transfert

```
transfer : <MS STATUS UPLINK>  ::=
  <Reject cause : <M-V-IE (Reject cause uplink, 1)>>
};
```

1.1.2.20 Commande de réallocation TMSI (ou « TMSI REALLOCATION COMMAND »)

1.1.2.20.1 Description abstraite

```
class "TMSI REALLOCATION COMMAND" {
abstract :
```

```
"LAI+TMSI" : "LAI+TMSI" ;
```

1.1.2.20.2 Syntaxe de transfert

```
transfer : <TMSI REALLOCATION COMMAND> ::=
  <Location area identification : <M-V-IE (Location area identification, 5)>>
  <Mobile identity : <M-LV-IE (Mobile identity, 2, 9)>>

  abstract.TMSI returns
    Mobile identity.V.TMSI/P-TMSI.abstract
};
```

1.1.2.21 Réallocation TMSI effectuée (ou « TMSI REALLOCATION COMPLETE »)

1.1.2.21.1 Description abstraite

```
class "TMSI REALLOCATION COMPLETE" {
abstract :

bind <TMSI REALLOCATIO N COMPLETE> {
}
```

1.1.2.21.2 Syntaxe de transfert

```
transfer : <TMSI REALLOCATION COMPLETE> ::=
  null
};
```

1.2 Listes des zones de localisation (ou « LA list »)

1.2.1 Description abstraite

```
class "LA list" {
 abstract :
   Location area description[0..infinite] : {
     "PLMN » : *"PLMN" ;  //-> UE context.PLMN list
     "LAC" : RIL3."Location Area Code" ;
     "Local time zone"[0..1] : "Time Zone" ;
     "Network daylight saving time[0..1] : "Daylight Saving Time" ;
};
```

1.3 Liste de réseaux (ou « PLMN list »)

1.3.1 Description abstraite

```
class "PLMN list" {
 abstract :
   "PLMN description"[0..infinite] : {
     "PLMN Id" : "PLMN Identity" ;
     "Full name for network"[0..1] : "Network Name" ;
     "Short name for network"[0..1] : "Network Name" ;
 }
;
```

1.4 Paramètres d'authentification AUTH (ou « Authentication Parameter AUTN »)

1.4.1 Description abstraite

```
class "Authentication Parameter AUTN" {
 abstract :
   "Sequence number xor Anonymity Key" : Bitstring (48) ;
   "Authentication management field" : Bitstring (16) ;
   "Message authentication code" : Bitstring (64) ;
```

1.4.2 Syntaxe de transfert

```
 transfer : < Authentication Parameter AUTN> ::=
   <Sequence number xor Anonymity Key : bit*48>
   <Authentication management field : bit*16>
   <Message authentication code : bit*64>
};
```

1.5 Type de service CM (ou « CM Service Type »)

1.5.1 Description abstraite

```
class "CM Service Type" {
 abstract :
   "service type" : Enumerated ("MO call or packet est", "emergency call", "SMS", "SS", "VGC",
"VBS", "LCS")
```

1.5.2 Syntaxe de transfert

```
transfer <CM Service Type> ::=
   <service type : {0001|0010|0100|10 bit(2)}>
};
```

1.6 Correction d'heure d'été (ou « Daylight Saving Time »)

1.6.1 Description abstraite

```
class "Daylight Saving Time" {
 abstract :
   "Daylight Saving Time adjustment" : Enumerated (0, "+1 hour", "+2 hour")
```

1.6.2 Syntaxe de transfert

```
transfer : <Daylight Saving Time> ::=
   <spare bit> * 6
   <Daylight Saving Time adjustment : {00 | 01 | 10 }>
};
```

1.7 IMEI

1.7.1 Description abstraite

```
class "IMEI" {
 abstract :
   "Type Approval Code"[6] : Integer (0..9) ;
   "Final Assembly Code"[2] : Integer (0..9) ;
   "Serial Number"[6] : Integer (0..9) ;
```

1.7.2 Syntaxe de transfert

```
transfer : <IMEI> ::=
   < digit one : digit> 1 bit*3
   <pairs : <digit odd : digit> <digit even : digit>>*7

   function Digit(n) : integer returns
     iif(n=1, integer(digit one),
       iif(n%2 = 0, integer(pairs[n/2].digit even), integer(pairs[(n-1)/2].digit odd)))
```

```
    abstract.Type Approval Code[n] returns   -- n from 0 to 5, MSB to LSD
      Digit(n+1)

    abstract.Final Assembly Code[n] returns   -- n from 0 to 1, MSD to LSD
      Digit(n+7)

    abstract.Serial Number[n] returns          -- n from 0 to 5, MSD to LSD
      Digit(n+9)
};
```

1.8 IMSI

1.8.1 Description abstraite

```
class "IMSI" {
 abstract :
   "MCC" : "MCC" ;
   "NMSI"[3..12] : Integer (0..9) ;
```

1.8.2 Syntaxe de transfert

```
 transfer : <IMSI> ::=
   {< digit one : digit> <oddeven : bit> bit*3
     <pairs : <digit odd : digit> <digit even : digit>>**
     <pairs : <spare bit>*4 <digit even : digit>>*(1-integer(oddeven))}
   & octet*(4..8)

   function Digit(n) : integer returns
     iif(n=1, integer(digit one),
       iif(n%2 = 0, integer(pairs[n/2].digit even), integer(pairs[(n-1)/2].digit odd)))

   abstract.MCC returns
     Digit(1)*100 + Digit(2)*10 + Digit(3)

   abstract.NMSI.size returns
     (exist (pairs) *2) - 4 + integer (oddeven)

   abstract.NMSI[n] returns          -- n from 0 to (size-1)
     Digit(n+4)
;
```

1.9 LAI et TMSI

1.9.1 Description abstraite

```
class "LAI+TMSI" {
  abstract
    "Location Area Identification" : RIL3."Location Area Identification" ;
    "TMSI" : "Temporary Mobile Station Identity" ;
;
```

1.10 Identifiant LSA (ou « LSA Identifier »)

1.10.1 Description abstraite

```
class "LSA Identifier" {
  abstract :
    "LSA available" : Boolean ;
    "LSA ID"[0..1 | "LSA available"] : {
      "LSA identifier scope" : Enumerated ("PLMN significant", "universal") ;
      "Localised service area identity" : Bitstring (23) ;
```

1.10.2 Syntaxe de transfert

```
  transfer : <LSA Identifier> ::=
    {<LSA ID : <Localised service area identity : bit(23)>
        <LSA identifier scope : bit>
      | null }

    abstract.LSA available returns
      exist (LSA ID)

    abstract.LSA ID.LSA identifier scope returns
      iif (LSA identifier scope = 0, PLMN significant, universal)
};
```

1.11 MCC

1.11.1 Description abstraite

```
class "MCC" {
  abstract :
    "MCC value" : Integer (0..999) ;
```

```
};
```

1.12 MNC

1.12.1 Description abstraite

```
class "MNC" {
 abstract :
   "MNC value" : Integer (0..999) ;
};
```

1.13 Network Name

1.13.1 Description abstraite

```
class "Network Name" {
 abstract :
   "Coding scheme" : Enumerated ("cell broadcast default alphabet", "UCS2")
   "Add country's initials" : Boolean ;
"Text string" : String ;
```

1.13.2 Syntaxe de transfert

```
 transfer : <Network Name>  ::=
   <ext : 1>
   <Coding scheme : {000|001}>
   <Add country's initials : bit>
   <number of spare bits in last octet : bit(3)>
   <Text string :      -- order of bits within text string ?
     <octet>**
     <{0}*(number of spare bits in last octet)>
     <bit*(8 - number of spare bits in last octet)>>
};
```

1.14 Identité de réseau (ou « PLMN Identity »)

1.14.1 Description abstraite

```
class "PLMN Identity" {
 abstract :
   "MCC" : "MCC" ;
   "MNC" : "MNC" ;
```

```
};
```

1.15 Identité temporaire de station mobile (ou « Temporary Mobile Station Identity »)

1.15.1 Description abstraite

```
class "Temporary Mobile Station Identity" {
  abstract :
    "temporary identity" : Bitstring (32) ;
```

1.15.2 Syntaxe de transfert

```
  transfer : <Temporary Mobile Station Identity> ::=
     <temporary identity : bit*32>
;
```

1.16 Zone horaire (ou « Time Zone »)

1.16.1 Description abstraite

```
class "Time Zone" {
  abstract :
    "Delta with GMT" : Integer (-99..99) ;
```

1.16.2 Syntaxe de transfert

```
  transfer : <Time Zone> ::=
     <sign : bit>
     <digit 1 : bit(3)>
     <digit 2 : bit(4)>

     abstract.Delta with GMT returns          -- units are minutes
        15 * iif (sign = 0, integer(digit 1)*10 + integer(digit 2), - integer(digit 1)*10 - integer(digit 2))
};
```

REVENDICATIONS

1. Système de spécification et de mise en œuvre d'un protocole de gestion de communication et de transmission entre au moins un émetteur et au moins un récepteur, caractérisé en ce qu'il comprend:

- des moyens de détermination d'un contexte commun (2) comprenant des moyens de description d'un ensemble de données gérées par ledit ou lesdits émetteurs et/ou ledit ou lesdits récepteurs ;
- des moyens d'identification des messages de protocole (3) susceptibles d'être échangés entre ledit ou lesdits émetteurs et ledit ou lesdits récepteurs, ladite identification étant indépendante de la ou des représentations utilisées pour lesdits messages;
- des moyens de définition de représentations de données stockées et/ou transmises par ledit ou lesdits émetteurs et/ou ledit ou lesdits récepteurs à partir de ladite détermination d'un contexte commun et de ladite identification des messages, ladite définition de représentations permettant de déterminer des instructions d'implémentation dudit protocole dans au moins un desdits émetteurs et au moins un desdits récepteurs ; et
- des moyens de mise en œuvre dudit protocole dans au moins un desdits émetteurs et au moins un desdits récepteurs à partir desdites instructions d'implémentation, de façon à ce que ledit ou lesdits émetteurs soient aptes à communiquer avec ledit ou lesdits récepteurs.

2. Système selon la revendication 1, caractérisé en ce que lesdits moyens de détermination d'un contexte commun comprennent eux-mêmes des moyens de description d'une structure du contexte, ladite description étant indépendante de la ou des représentations utilisées pour le stockage et/ou la transmission desdites données.

3. Système selon la revendication 2, caractérisé en ce que lesdits moyens de détermination d'un contexte commun comprennent en outre des moyens

d'identification et de description des types d'objets permettant d'analyser ladite structure du contexte, ladite identification et ladite description des types d'objets étant indépendantes de la ou des représentations utilisées pour le stockage et/ou la transmission.

4. Système selon l'une quelconque des revendications 1 à 3, caractérisé en ce que tout ou partie desdites données gérées sont les données que ledit protocole vise à maintenir cohérent dans au moins un émetteur parmi ledit ou lesdits émetteurs et au moins un récepteur parmi ledit ou lesdits récepteurs.

5. Système selon l'une quelconque des revendications 1 à 4, caractérisé en ce qu'au moins une partie de ladite identification des messages est effectuée en termes d'actions et/ou de ses effets sur ledit contexte commun.

6. Système selon l'une quelconque des revendications 1 à 5, caractérisé en ce qu'il comprend des moyens de liaison entre :

- une description de la forme de signal effectivement transmis ; et
- ladite identification des messages de protocole.

7. Système selon la revendication 6, caractérisé en ce que ladite forme du signal appartient au groupe comprenant :

- les suites de données binaires ;
- les suites d'éléments pris dans un alphabet prédéterminé; et
- les suites de formes d'ondes modulées.

8. Système selon l'une quelconque des revendications 6 et 7, caractérisé en ce que ladite liaison se fait dans un langage de spécification de données comprenant des fonctions permettant la description desdites données sous forme d'attributs d'objets abstraits et de fonctions s'appliquant sur des représentations concrètes.

9. Système selon l'une quelconque des revendications 6 à 8, caractérisé en ce que ladite spécification de la forme de signal se fait en langage CSN1 enrichi de fonctions permettant la description desdites données sous forme d'attributs d'objets abstraits et de fonctions s'appliquant sur des représentations concrètes.

10. Système selon l'une quelconque des revendications 1 à 9, caractérisé en ce qu'il comprend des moyens d'utilisation des données de contexte dans une description formelle d'un encodage et/ou d'un décodage.

11. Système selon l'une quelconque des revendications 1 à 10, caractérisé en ce que lesdits moyens de détermination d'un contexte commun, lesdits moyens d'identification des messages de protocole et lesdits moyens de définition de représentations de données mettent chacun en oeuvre un langage formel permettant une détermination automatique du logiciel de mise en œuvre dudit protocole.

12. Système selon la revendication 11, caractérisé en ce qu'il comprend des moyens de détermination automatique du logiciel de mise en œuvre dudit protocole.

13. Système selon l'une quelconque des revendications 11 et 12, caractérisé en ce qu'il comprend des moyens de détermination automatique de tests de dispositifs mettant en œuvre ledit protocole.

14. Système selon l'une quelconque des revendications 1 à 13, caractérisé en ce que lesdits moyens de détermination d'un contexte commun comprennent des moyens de traduction d'un mode de représentation d'un protocole en ladite description d'un ensemble de données.

15. Système selon l'une quelconque des revendications 1 à 14, caractérisé en ce que lesdits moyens d'identification des messages de protocoles comprennent des moyens de traduction d'un mode de représentation de protocoles.

16. Système selon l'une quelconque des revendications 1 à 15, caractérisé en ce qu'il comprend en outre des moyens de visualisation de la sémantique desdits messages de protocoles échangés entre ledit ou lesdits émetteurs et/ou ledit ou lesdits récepteurs.

17. Procédé de spécification et de mise en œuvre d'un protocole de gestion de communication et de transmission entre au moins un émetteur et au moins un récepteur, caractérisé en ce qu'il comprend les étapes suivantes :

- détermination d'un contexte commun (2) comprenant une description d'un ensemble de données gérées par ledit ou lesdits émetteurs et/ou ledit ou lesdits récepteurs ;
- identification des messages de protocole (3) susceptibles d'être échangés entre ledit ou lesdits émetteurs et ledit ou lesdits récepteurs, ladite identification étant indépendante de la ou des représentations utilisées pour lesdits messages;
- définition de représentations de données stockées et/ou transmises par ledit ou lesdits émetteurs et/ou ledit ou lesdits récepteurs à partir de ladite détermination d'un contexte commun et de ladite identification des messages, ladite définition de représentations permettant de déterminer des instructions d'implémentation dudit protocole dans au moins un desdits émetteurs et au moins un desdits récepteurs ; et
- mise en œuvre dudit protocole dans au moins un desdits émetteurs et au moins un desdits récepteurs à partir desdites instructions d'implémentation, de façon à ce que ledit ou lesdits émetteurs soient aptes à communiquer avec ledit ou lesdits récepteurs.

18. Dispositif de communication, caractérisé en ce qu'il comprend des moyens permettant l'émission et/ou la réception de données vers ou d'un dispositif tiers un autre dispositif selon un protocole obtenu par la mise en œuvre du procédé selon la revendication 17.

19. Produit programme d'ordinateur comprenant des éléments de programme, enregistrés sur un support lisible par au moins un microprocesseur, caractérisé en ce que lesdits éléments de programme contrôlent le ou lesdits microprocesseurs pour qu'ils effectuent les étapes suivantes adaptées à la spécification de protocole de communication et de transmission entre au moins un émetteur et au moins un récepteur:

- détermination d'un contexte commun (2) comprenant une description d'un ensemble de données gérées par ledit ou lesdits émetteurs et/ou ledit ou lesdits récepteurs ;

- identification des messages de protocole (3) susceptibles d'être échangés entre ledit ou lesdits émetteurs et ledit ou lesdits récepteurs, ladite identification étant indépendante de la ou des représentations utilisées pour lesdits messages;
- définition de représentations de données stockées et/ou transmises par ledit ou lesdits émetteurs et/ou ledit ou lesdits récepteurs à partir de ladite détermination d'un contexte commun et de ladite identification des messages, ladite définition de représentations permettant de déterminer des instructions d'implémentation dudit protocole dans au moins un desdits émetteurs et au moins un desdits récepteurs ; et
- mise en œuvre dudit protocole dans au moins un desdits émetteurs et au moins un desdits récepteurs à partir desdites instructions d'implémentation, de façon à ce que ledit ou lesdits émetteurs soient aptes à communiquer avec ledit ou lesdits récepteurs.

20. Produit programme d'ordinateur, caractérisé en ce que ledit programme comprend des séquences d'instructions adaptées à la mise en œuvre d'un procédé de spécification de protocole de communication entre au moins un émetteur et au moins un récepteur selon la revendication 17 lorsque ledit programme est exécuté sur un ordinateur.

1/1

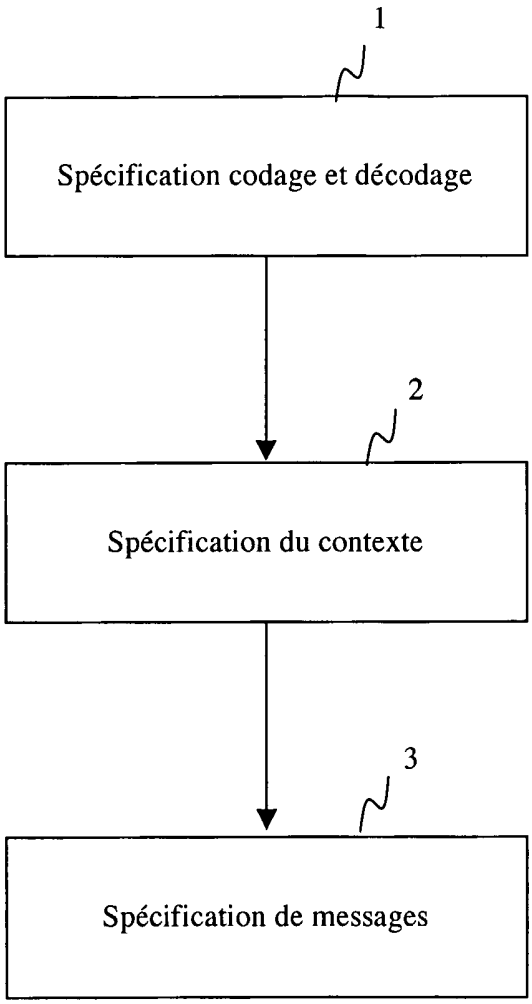

1
Spécification codage et décodage
2
Spécification du contexte
3
Spécification de messages

Fig. 1

INTERNATIONAL SEARCH REPORT

International Application No
PCT/FR2004/001310

A. CLASSIFICATION OF SUBJECT MATTER
IPC 7 H04L29/08

According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)
IPC 7 H04L

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practical, search terms used)

EPO-Internal, INSPEC, COMPENDEX, IBM-TDB, PAJ, WPI Data

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category °	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
A	US 5 930 264 A (NGUYEN VIET) 27 July 1999 (1999-07-27) column 2, line 25 - line 41 column 3, line 30 - column 4, line 2; claims 1-7; figure 2	1-19
A	US 2002/004830 A1 (HYATT MICHAEL ET AL) 10 January 2002 (2002-01-10) paragraph '0008! - paragraph '0009! paragraph '0024! - paragraph '0026!; claims 1-14; figure 1	1-19

-/--

[X] Further documents are listed in the continuation of box C.

[X] Patent family members are listed in annex.

° Special categories of cited documents :

"A" document defining the general state of the art which is not considered to be of particular relevance

"E" earlier document but published on or after the international filing date

"L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)

"O" document referring to an oral disclosure, use, exhibition or other means

"P" document published prior to the international filing date but later than the priority date claimed

"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention

"X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone

"Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art.

"&" document member of the same patent family

Date of the actual completion of the international search	Date of mailing of the international search report
29 September 2004	05/10/2004

Name and mailing address of the ISA	Authorized officer
European Patent Office, P.B. 5818 Patentlaan 2 NL - 2280 HV Rijswijk Tel. (+31-70) 340-2040, Tx. 31 651 epo nl, Fax: (+31-70) 340-3016	Schwibinger, H-P

INTERNATIONAL SEARCH REPORT

International Application No
PCT/FR2004/001310

C.(Continuation) DOCUMENTS CONSIDERED TO BE RELEVANT

Category °	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
A	KONING J-L: "Algorithms for translating interaction protocols into a formal description" SYSTEMS, MAN, AND CYBERNETICS, 1999. IEEE SMC '99 CONFERENCE PROCEEDINGS. 1999 IEEE INTERNATIONAL CONFERENCE ON TOKYO, JAPAN 12-15 OCT. 1999, PISCATAWAY, NJ, USA,IEEE, US, 12 October 1999 (1999-10-12), pages 810-815, XP010363604 ISBN: 0-7803-5731-0 page 810, left-hand column, line 27 - right-hand column, line 16 page 811, left-hand column, line 6 - page 812, left-hand column, line 13 -----	1,17,19

INTERNATIONAL SEARCH REPORT

International Application No
PCT/FR2004/001310

Patent document cited in search report	Publication date	Patent family member(s)	Publication date
US 5930264 A	27-07-1999	AU 6008598 A	26-08-1998
		BR 9807182 A	25-01-2000
		CA 2279368 A1	13-08-1998
		WO 9835526 A2	13-08-1998
US 2002004830 A1	10-01-2002	CA 2307403 A1	02-11-2001

RAPPORT DE RECHERCHE INTERNATIONALE

Demande Internationale No
PCT/FR2004/001310

A. CLASSEMENT DE L'OBJET DE LA DEMANDE
CIB 7 H04L29/08

Selon la classification internationale des brevets (CIB) ou à la fois selon la classification nationale et la CIB

B. DOMAINES SUR LESQUELS LA RECHERCHE A PORTE

Documentation minimale consultée (système de classification suivi des symboles de classement)
CIB 7 H04L

Documentation consultée autre que la documentation minimale dans la mesure où ces documents relèvent des domaines sur lesquels a porté la recherche

Base de données électronique consultée au cours de la recherche internationale (nom de la base de données, et si réalisable, termes de recherche utilisés)
EPO-Internal, INSPEC, COMPENDEX, IBM-TDB, PAJ, WPI Data

C. DOCUMENTS CONSIDERES COMME PERTINENTS

Catégorie °	Identification des documents cités, avec, le cas échéant, l'indication des passages pertinents	no. des revendications visées
A	US 5 930 264 A (NGUYEN VIET) 27 juillet 1999 (1999-07-27) colonne 2, ligne 25 - ligne 41 colonne 3, ligne 30 - colonne 4, ligne 2; revendications 1-7; figure 2	1-19
A	US 2002/004830 A1 (HYATT MICHAEL ET AL) 10 janvier 2002 (2002-01-10) alinéa '0008! - alinéa '0009! alinéa '0024! - alinéa '0026!; revendications 1-14; figure 1	1-19

-/--

[X] Voir la suite du cadre C pour la fin de la liste des documents

[X] Les documents de familles de brevets sont indiqués en annexe

° Catégories spéciales de documents cités:

"A" document définissant l'état général de la technique, non considéré comme particulièrement pertinent

"E" document antérieur, mais publié à la date de dépôt international ou après cette date

"L" document pouvant jeter un doute sur une revendication de priorité ou cité pour déterminer la date de publication d'une autre citation ou pour une raison spéciale (telle qu'indiquée)

"O" document se référant à une divulgation orale, à un usage, à une exposition ou tous autres moyens

"P" document publié avant la date de dépôt international, mais postérieurement à la date de priorité revendiquée

"T" document ultérieur publié après la date de dépôt international ou la date de priorité et n'appartenenant pas à l'état de la technique pertinent, mais cité pour comprendre le principe ou la théorie constituant la base de l'invention

"X" document particulièrement pertinent; l'inven tion revendiquée ne peut être considérée comme nouvelle ou comme impliquant une activité inventive par rapport au document considéré isolément

"Y" document particulièrement pertinent; l'inven tion revendiquée ne peut être considérée comme impliquant une activité inventive lorsque le document est associé à un ou plusieurs autres documents de même nature, cette combinaison étant évidente pour une personne du métier

"&" document qui fait partie de la même famille de brevets

Date à laquelle la recherche internationale a été effectivement achevée	Date d'expédition du présent rapport de recherche internationale
29 septembre 2004	05/10/2004

Nom et adresse postale de l'administration chargée de la recherche internationale	Fonctionnaire autorisé
Office Européen des Brevets, P.B. 5818 Patentlaan 2 NL - 2280 HV Rijswijk Tel. (+31-70) 340-2040, Tx. 31 651 epo nl, Fax: (+31-70) 340-3016	Schwibinger, H-P

RAPPORT DE RECHERCHE INTERNATIONALE

Demande Internationale No
PCT/FR2004/001310

C.(suite) DOCUMENTS CONSIDERES COMME PERTINENTS

Catégorie °	Identification des documents cités, avec, le cas échéant, l'indication des passages pertinents	no. des revendications visées
A	KONING J-L: "Algorithms for translating interaction protocols into a formal description" SYSTEMS, MAN, AND CYBERNETICS, 1999. IEEE SMC '99 CONFERENCE PROCEEDINGS. 1999 IEEE INTERNATIONAL CONFERENCE ON TOKYO, JAPAN 12-15 OCT. 1999, PISCATAWAY, NJ, USA,IEEE, US, 12 octobre 1999 (1999-10-12), pages 810-815, XP010363604 ISBN: 0-7803-5731-0 page 810, colonne de gauche, ligne 27 - colonne de droite, ligne 16 page 811, colonne de gauche, ligne 6 - page 812, colonne de gauche, ligne 13	1,17,19

RAPPORT DE RECHERCHE INTERNATIONALE

Demande Internationale No
PCT/FR2004/001310

Document brevet cité au rapport de recherche	Date de publication	Membre(s) de la famille de brevet(s)	Date de publication
US 5930264 A	27-07-1999	AU 6008598 A	26-08-1998
		BR 9807182 A	25-01-2000
		CA 2279368 A1	13-08-1998
		WO 9835526 A2	13-08-1998
US 2002004830 A1	10-01-2002	CA 2307403 A1	02-11-2001