CHASSIS — Une Plate-forme pour la Construction de Systèmes d'Information Ouverts *

Oscar Nierstrasz, Dimitri Konstantas†
Klaus Dittrich, Dirk Jonscher‡

Résumé

Les systèmes d'information d'aujourd'hui ont de plus en plus la nécessité d'être ouverts. Ceci implique qu'ils doivent répondre aux besoins de réseaux ouverts, de logiciel et de matériel hétérogènes et "inter-opérables," et, surtout, à des besoins évolutifs et changeants. Le projet CHASSIS vise le développement d'un cadre informatique et méthodologique pour (i) la conception et la construction de systèmes d'information hétérogènes, sûrs et fiables à partir de composants de logiciel et bases de données soit déjà existants soit développés pour l'occasion, et (ii) leur intégration sûre et fiable.
Dans CHASSIS, l'orientation-objet est la technologie clé pour la construction d'un tel système, car son interface uniforme est réalisée par un modèle de données orienté-objet, et la couche d'intégration est réalisée par du logiciel orienté-objet. CHASSIS consiste en des modèles objets pour l'intégration de base de données et langages de programmation, du logiciel orienté-objet pour l'intégration des systèmes, des méthodes de spécification pour soutenir le processus de conception, et des mécanismes de sécurité avancés qui permettent d'assurer un haut degré de sécurité pour le système d'information résultant. CHASSIS est un projet de collaboration Suisse entre l'Université de Zürich, l'Université de Genève, et le centre de recherche d'Asea Brown Boveri (Baden).
Mots-clefs: systèmes ouverts, sécurité, bases de données fédérées, systèmes objets.

*.Dans *AFCET '93 — Vers des Systèmes d'Information Flexibles*, Versailles, 8-10 juin, 1993, pp. 153-161.

†.Université de Genève, Centre Universitaire d'Informatique, 24 rue Général Dufour, CH-1211 Genève 4, SUISSE.
E-mail: {oscar,dimitri}@cui.unige.ch. *Tel:* +41 (22) 705.7664/7647. *Fax:* +41 (22) 320.2927.

‡.Universität Zürich, Institut für Informatik, Winterthurerstraße 190, CH-8057 Zürich, SUISSE.
E-mail:{dittrich,jonscher}@ifi.unizh.ch. *Tel*:+41 1 257.4312/4337. *Fax*:+41 1 363.0035.

1 Introduction

Ces dernières années, de spectaculaires changements dans le matériel et le logiciel utilisés dans les applications informatiques nous ont conduits vers des solutions réparties et ouvertes, nous éloignant ainsi des systèmes centralisés liés à un fournisseur unique. Les nouveaux systèmes sont ouverts en termes de plate-forme, topologie et évolution. Bien que les systèmes ouverts, sûrs et fiables soient bien compris au niveau des systèmes d'exploitation, ce n'est malheureusement pas le cas au niveau des applications. Un utilisateur typique d'un système informatique moderne peut facilement se trouver devant une riche variété d'outils, de bases de données et d'applications disponibles comme "services" sur le réseau, mais sans moyen cohérent et systématique pour les combiner afin de résoudre des problèmes quotidiens. En outre, il n'est actuellement pas possible dans un environnement réparti et hétérogène de partager de l'information, d'intégrer des systèmes nouveaux et des systèmes existants, ou de construire de nouveaux systèmes d'information basés sur des services, des données et des logiciels existants avec un effort raisonnable et d'une façon fiable.

CHASSIS** est un projet de recherche Suisse du Programme Prioritaire Informatique, qui vise le développement d'une plate-forme pour la conception et la construction sûre et fiable de systèmes d'information hétérogènes à partir de composants nouveaux ou existants d'applications et de bases de données. Les partenaires dans CHASSIS sont l'Université de Zürich, l'Université de Genève, et le centre de recherche d'Asea Brown Boveri (Baden).

Bien que CHASSIS vise le développement d'un cadre général, nos besoins initiaux seront dictés par le domaine des systèmes d'information pour l'ingénierie électrique (la spécialité de ABB). Ce domaine nécessite différents genres d'activités d'ingénierie pour construire un système complet, tels que CAO mécanique, CAO électrique (c.à.d., schémas de connexions électriques), calculs spécifiques à l'application (par ex., pressions mécaniques, transfert de chaleur, rendement énergétique, etc.), traitement de texte, composition (ou configuration) de systèmes spécifiques selon les besoins du client à partir de pièces standards, et programmation et paramétrisation selon les besoins du clients (par ex. d'un système de contrôle de processus). De plus, les divers aspects d'organisation jouent un rôle important, notamment pour la collaboration de plusieurs équipes souvent situées dans des régions différentes, pour l'optimisation et l'intégration du processus d'ingénierie de la préparation à l'acceptation finale, et pour la gestion et le contrôle d'un projet.

Le problème clé que CHASSIS essaie de traiter est l'intégration fiables de systèmes hétérogènes et ouverts. Les systèmes informatiques traditionnels sont fermés alors que l'on exige des systèmes modernes qu'ils soient ouverts sur de nombreux plans; en particulier, il est aujourd'hui impensable de définir les besoins d'un système en termes fermés et immuables. Les applications et systèmes d'information flexibles doivent être ouverts à des besoins changeants. CHASSIS prend comme cibles spécifiques (1) l'intégration fiable et sûre de

**.Configurable, Heterogeneous, And Safe, Secure Information Systems.

bases de données hétérogènes et autonomes, et (2) l'inter-opération de systèmes hétérogènes de logiciels et de bases de données. CHASSIS adopte une approche orientée-objet pour les deux problèmes, en introduisant (1) un modèle de données orienté-objet et un modèle de sécurité correspondant pour une base de données fédérée, et (2) un cadre orienté-objet pour l'inter-opération dans lequel chaque système de logiciel indépendant et chaque système d'information est considéré comme une *cellule* qui contient le système original comme *noyau*, et qui ajoute une *membrane* qui est responsable pour *la traduction de types, la traduction d'objets*, et *la négociation de connexion*. Dans une deuxième phase du projet, on envisage le développement d'un environnement de composants réutilisables pour l'intégration des systèmes hétérogènes, basé sur les concepts de CHASSIS.

2 L'environnement CHASSIS

Le but de CHASSIS est de permettre l'inter-opération sûre et fiable d'applications informatiques et de systèmes d'informations hétérogènes. En principe, l'inter-opération ne doit pas impliquer la modification des systèmes et des services existants. En particulier, les applications existantes doivent pouvoir fonctionner sans modification quand leur bases de données sous-jacentes deviennent accessibles à d'autres clients à travers le réseau. De plus, l'autonomie locale doit être conservée, et donc la sécurité doit être préservée selon les besoins *locaux*. Un plan de sécurité global doit prendre en considération les différences de politique entre les bases de données constitutives.

Si on regarde le problème de plus près, on peut distinguer deux caractéristiques qu'un composant logiciel peut fournir à un système d'information: la fonctionnalité (services) et les données. Par conséquent, rendre inter-opérable un système logiciel signifie que l'on considère les deux scénarios suivants:

- Si seuls les services offerts par le composant présentent un intérêt dans le système d'information intégré, il est suffisant de le traiter comme une boîte noire. On peut l'encapsuler comme un objet qui fournit sa fonctionnalité par une interface uniforme.
- Cependant, dès que le composant traite des données persistantes, il est probable que l'inter-opération signifie rendre accessibles les données dans le système d'information, et les partager avec les autres composants. Par conséquence, une vue uniforme et intégrée des données pour toutes les applications est indispensable, c.à.d., selon le modèle d'un système fédéré.

Ce système fédéré joue un rôle clé dans l'intégration du système, mais il prend une importance particulière au niveau de la sécurité. D'un côté, il fournit la base pour faire respecter la sécurité dans le système intégré, et de l'autre côté, il doit préserver autant que possible la sécurité des bases de données participantes. Dès lors, il doit comprendre des concepts complets pour la protection des données stockées dans les systèmes de bases de données autonomes et intégrés (par ex., contrôle d'accès et de flux d'information), car les systèmes locaux pourraient refuser de faire partie d'une fédération avec des propriétés de sécurité faibles. Dans CHASSIS, l'orientation-objet est la technologie clé pour la construction d'un tel système, car son interface uniforme est réalisée par un modèle de données orienté-objet, et la couche d'inter-opération est réalisée sur du logiciel orienté-objet.

Dans le premier des deux cas décrits ci-dessus, une application ne change pas sa façon de traiter les données persistantes, car la base de données utilisée est directement accédée par l'application sans passer par le composant du système fédéré responsable pour le stockage des données persistantes. Dans le deuxième cas, il *est* nécessaire de modifier l'application en remplaçant tous les éléments concernés par le stockage de données par des appels appropriés au système de base de données fédérées. Bien que le deuxième cas implique un effort considérable de programmation, c'est le seul moyen effectif de partager des données entre les applications en profitant de l'intégration et des mécanismes de sécurité offerts par les services de stockage.

La base de données fédérée est elle-même encapsulée dans le système d'information intégré en tant que *cellule* (voir Figure 1). Le modèle de cellules prévoit du support orienté-objet

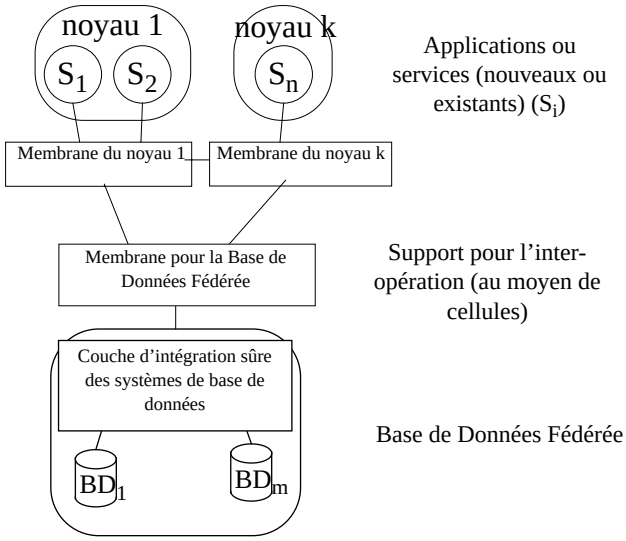

Figure 1 Architecture d'un système d'information intégré basé sur CHASSIS

pour l'inter-opération de systèmes hétérogènes au niveau des langages de programmation. Une cellule définit un espace de noms indépendant. Elle a deux parties: un *noyau* et une *membrane* (voir Figure 2). Le noyau est constitué d'un ensemble d'applications et de systèmes (bien que pas forcément orientés-objet), qu'on appelle en général des "objets," qui sont responsables de fournir des services et de gérer des ressources locales. La membrane, qui entoure le noyau, s'occupe de la communication avec le monde externe, c'est-à-dire avec les autres cellules. Les objets du noyau ne sont pas directement visibles par les autres cellules, et ne peuvent pas "voir" au-delà du noyau. C'est la membrane qui s'occupe de la correspondance entre les objets externes et les "mandataires" locaux, et qui prend en charge la traduction des types externes en types locaux.

Les services les plus importants offerts par la membrane d'une cellule sont *la traduction de types, la traduction d'objets*, et *la négociation de connexion.* La traduction de types est un

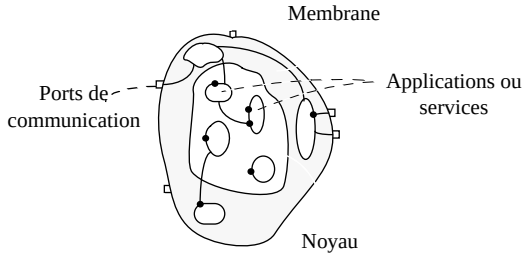

Figure 2 Une Cellule

service qui permet à "l'utilisateur" de définir une correspondance entre un type local et un type (ou plusieurs types) de la hiérarchie de types d'un autre site. La traduction d'objets permet l'accès d'objets externes par des noms locaux. La négociation de connexion gère les conditions sous lesquelles un objet externe devient accessible par un nom local. Quand un objet d'une cellule est accessible depuis le noyau d'une autre cellule, on dit que les deux cellules sont *connectées*.

La définition d'une traduction de types permet à "l'utilisateur" de spécifier la correspondance entre les opérations du type local et du type externe et le rapport entre leurs paramètres. La coercion entre opérations peut être établie, les paramètres peuvent être adaptés, et de nouveaux composants logiciels peuvent être introduits pour traiter les cas où une simple traduction n'est pas possible.

Selon la traduction définie, les objets externes deviennent visibles depuis le noyau de la cellule locale. De cette façon les objets provenant de différents environnements, sur des cellules lointaines qui peuvent être basées sur des langages de programmation très différents, peuvent devenir accessibles localement depuis un langage connu. De plus, la définition de traductions réciproques d'objets permet l'utilisation d'objets quelconques comme paramètre d'une opération.

Avant d'établir une connexion entre deux cellules, les termes de la connexion doivent être négociés. Ces termes peuvent comprendre la durée de la connexion, les algorithmes de sécurité utilisés pendant les échanges de message, les droits d'accès aux objets locaux et externes, les tarifs, etc.

Le modèle des cellules permet à des composants implantés avec des langages de programmation divers (orientés-objet ou pas) et fonctionnant sur des environnements d'exécution différents de coopérer et de communiquer à travers des types d'objets de niveau élevé, et pas seulement à travers des types de base (tel que les entiers, les chaînes de caractères etc.). De cette façon, les composants logiciels existant dans les cellules lointaines peuvent être réutilisés sans modification par des mandataires locaux qui suivent les conventions, les abstractions et les mécanismes de l'environnement local. Comme résultat, l'inter-opération des applications devient plus sûre et plus fiable, car elle est basée sur des composants existants et testés. De plus, le modèle des cellules permet la configuration dynamique de connexion d'objets afin que des applications puissent être introduites, modifiées ou supprimées à tout instant sans effets secondaires pour les applications existantes.

3 La Base de Données Fédérée

Comme l'on a déjà remarqué, une fédération peut être considérée comme une cellule particulière d'un système d'information. Pourtant, l'intégration de plusieurs bases de données autonomes existantes pose des problèmes de sécurité particuliers. La question clé dans ce domaine de traitement de données réparties est: comment offrir une vue uniforme et transparente des données stockées dans différentes bases de données, tout en préservant leur autonomie locale, y compris leur autonomie d'autorisation? Nous avons choisi un système étroitement associé pour permettre un contrôle global d'accès. Certains des problèmes auxquels nous sommes confrontés proviennent de l'hétérogénéité des différents systèmes (différents modèles de données et mécanismes de contrôle d'accès pour les bases de données individuelles). Le défi est de développer un modèle global de sécurité (basé sur le modèle global de données orienté-objet) et de traduire les autorisations globales en autorisations locales correspondantes, en préservant l'autonomie locale. De plus, la fédération permet d'établir des liens entre bases de données à protéger aussi.

Au niveau intégré, le modèle orienté-objet se conforme essentiellement au "manifeste des systèmes de base de données orienté-objet" [2] et peut être caractérisé comme suit: chaque entité dans le système est un objet qui consiste en un ensemble d'attributs (complexes) et un ensemble de méthodes qui encapsulent son comportement. Pourtant, nous ne nous limitons pas à l'encapsulation stricte. Un sous-ensemble d'attributs peut devenir visible (et accessible) au public. Un objet a une identité qui est indépendante de sa valeur, c.à.d., l'identité est une propriété qui permet de distinguer un objet d'un autre. Chaque objet est une instance d'un type unique. L'héritage dans notre modèle est basé sur l'inclusion et la spécialisation. Nous permettons l'héritage multiple uniquement pour le cas d'héritage préservant la compatibilité. Le mécanisme d'envoi de messages est très simple: le destinataire d'un message est toujours un objet, et le sélecteur est le nom de la méthode à invoquer. En résumé, le modèle est très proche de ZOO/$_{\text{IfI}}$* [5], un projet soeur à l'Université de Zürich qui vise le développement d'un environnement d'intégration pour les bases de données fédérées. Notre but est d'étendre ZOO/$_{\text{IfI}}$ avec la fonctionnalité de contrôle d'accès.

Le modèle global de sécurité dans son état actuel a les propriétés suivantes [3]: il est basé sur un contrôle d'accès discrétionnaire, c.à.d., sur l'identité du sujet et celle de l'objet protégé. Nous avons choisi un système mixte, permettant des autorisations positives et négatives, avec l'hypothèse de monde fermé: une requête est rejetée si ni une autorisation positive ni une autorisation négative ne peut être déduite à partir de l'ensemble des autorisations explicites. De plus, nous appliquons un procédé simple pour la résolution de conflits: les autorisations négatives prennent toujours le dessus sur les positives. L'autorisation décentralisée est basée sur un paradigme de possession et sur les "grant options" [4]. Des considérations sur les paradigmes d'administration suivront. Nous faisons une dis-

*.Zürich object-oriented integration framework for building Heterogeneous Database Systems

tinction entre deux genres de sujets: les utilisateurs et les rôles. Les rôles décrivent la situation organisationnelle, fonctionnelle ou sociale des utilisateurs dans le domaine de discours, et servent, par exemple, à modéliser la structure des sociétés. Les utilisateurs peuvent jouer plusieurs rôles (aussi en même temps), héritant leurs droits d'accès. Le déclenchement et la terminaison des rôles sont faits explicitement et sont contraints par la résolution des conflits, c.à.d., il est possible d'empêcher le déclenchement simultané de rôles qui peuvent mener à une accumulation non sûre de droits. Le système utilise largement l'autorisation implicite. Les rôles peuvent avoirs des rapports subordonnés (ce genre de rapport binaire établit un ordre partiel). Les rôles supérieurs héritent des autorisations positives de leurs sous-rôles, alors que les rôles subordonnés héritent des autorisations négatives de leurs rôles supérieurs. Ceci aboutit à un système dans lequel les rôles supérieurs ont un pouvoir (ou une autorité) strictement plus fort que leurs subordonnés. De plus, les objets complexes constituent une unité d'autorisation, c.à.d., les droits d'un objet complexe sont hérités par tous ses éléments. D'autres implications nécessitées par les modèles de données orientés-objet ne sont pas présentées en détail ici.

Le concept d'un *domaine* est également présent. Nous faisons la distinction entre deux différents genres de domaines: les domaines de sujets, et les domaines d'objets protégés. Un domaine de sujets consiste en un ensemble d'utilisateurs, de rôles ou d'autres domaines de sujets. Un domaine d'objets protégés peut consister en un sous-ensemble quelconque d'objets à protéger, c.à.d., des types, des extensions de types, objets (complexes) et d'autres domaines d'objets protégés. Ils servent aussi pour les autorisations implicites. Les autorisations pour les domaines (positives ou négatives) sont héritées par leurs éléments. Les domaines constituent un outil puissant pour modéliser plusieurs concepts de sécurité tel que les groupes de travail imbriqués et les bases de données privées.

Le modèle global de sécurité sera implanté d'une façon orientée-objet, utilisant ObjectStore (une base de données orientée-objet commerciale) pour stocker les méta-données.

L'architecture du système de contrôle d'accès global est basée sur un moniteur de référence ("reference monitor") distribué qui intercepte les messages, évalue leur autorisation, et les fait suivre à leurs destinataires. Une instance de ce moniteur tourne sur chaque noeud où la fédération tourne elle-même. Il est à noter que ce moniteur ne constitue qu'une première ligne de défense, car chaque base de données locale se munit (théoriquement) de son propre système de contrôle d'accès.

Nous ne prenons en considération aucun problème de sécurité de réseau, car ceci constitue une direction indépendante de recherche, et, de plus, il existe déjà des solutions applicable (c.à.d., techniques d'encryption et d'identification mutuelle).

Le prototype sera implanté en C++, intégrant une base de données orientée-objet (ObjectStore) et une base de données relationnelle (Oracle). Puisque ObjectStore n'offre aucun mécanisme de contrôle d'accès, le vrai défi est de traduire les concepts globaux de sécurité en concepts locaux à Oracle. Le problème clé est que les décisions locales d'Oracle sont toujours prioritaires, et que le modèle global est beaucoup plus puissant que les mécanismes disponibles localement. Nous avons l'intention d'assurer un état d'autorisation cohérent entre le niveau global et le niveau local. Les décisions locales de contrôle d'accès peuvent être basées sur deux paradigmes différents. La fédération doit fournir l'identificateur de l'utilisateur global qui a lancé la requête au niveau fédéré, ou alors la fédération elle-même doit se montrer comme un utilisateur local (ou, plus précisément, comme une application locale). Dans le deuxième cas, les systèmes locaux doivent faire confiance aux mécanismes de sécurité de la fédération, c.à.d., ils doivent sacrifier leur autonomie d'autorisation locale dans une certaine mesure. Pourtant, ces solutions ne sont que deux extrêmes d'une gamme continue, car la fédération peut représenter plusieurs utilisateurs globaux par des utilisateurs locaux virtuels [10]. L'approche à adopter doit être négociée entre les administrateurs locaux et globaux.

4 Inter-opérabilité Orientée-objet

Puisque les applications peuvent échanger de l'information à travers leur système de base de données fédéré, on peut dire qu'un tel système fédéré fournit une base pour connecter les applications "par en dessous." Pourtant, toutes les applications n'utilisent pas forcément un système de base de données fédéré, et il est aussi possible que plusieurs systèmes fédérés cohabitent sur un même réseau. Une application peut aussi avoir besoin d'accéder directement aux services offerts par d'autres applications pouvant être toutes connectées à la même base de données fédérée. Dans un environnement ouvert, un service donné pourrait être offert par plusieurs serveurs avec des interfaces différentes. Par conséquent, une application désirant accéder à un service particulier offert par différents serveurs est obligée d'utiliser une interface différente pour chaque serveur. La situation devient encore plus pénible si les différents serveurs utilisent des protocoles et des mécanismes de sécurité différents.

Une solution au problème d'interfaces multiples — le Common Object Request Broker Architecture (CORBA) [1] — a été proposée par le Object Management Group (OMG). Pourtant, le CORBA ne permet ni des transformations flexibles d'interfaces, ni l'échange de types de données autres que les types de bases et leurs aggrégats (simples). En outre, CORBA ne prend aucun aspect de sécurité en considération. Une meilleure solution pour CHASSIS se présente dans le *support d'inter-opérabilité orientée-objet* [7] dans le cadre d'un environnement à *cellules* [6][8]. Ce support est basé sur trois services: *la traduction de types, la traduction d'objets,* et *la négociation de connexion.*

4.1 Traduction de Types

La traduction de types consiste à définir les liens et les transformations de l'interface des requêtes posées par le client avec l'interface offerte par le serveur. Ces liens et ces transformations sont exprimés dans un *langage de traduction de types*, qui est spécifique à chaque cellule, et qui peut donc avoir une syntaxe compatible avec celle du langage de programmation de la cellule.

Le langage de traduction de types permet à l'utilisateur de spécifier les liens entre les opérations d'un type local et celles

d'un type externe, et les règles de transformation de paramètres entre les deux. Le langage de spécification est assez puissant pour exprimer non seulement le rapport entre opérations, mais aussi le pré- et post- traitement des paramètres et des résultats, le groupement et la séparation d'opérations et de types, et la définition de fonctions d'adaptation.

Pendant la traduction de types, on peut distinguer trois genres de rapport entres les types locaux et externes: équivalence, traduction et translitération.

Les type équivalents sont des types qui existent dans les deux cellules avec la même sémantique et structure interne. C'est le cas le plus commun pour les types de données, tel que les entiers, les chaînes de caractères et leurs aggrégats. Les types équivalents peuvent migrer d'une cellule à une autre sans modification (à l'exception de modifications dues à la représentation interne, comme l'ordre des bytes).

Les types traduits sont des types qui ont la même sémantique dans les deux cellules, mais avec structure et représentation différentes. Par exemple, les chaînes de caractères peuvent être représentées dans une cellule comme des tableaux de caractères, alors qu'une autre cellule peut avoir le type *string* (avec une autre sémantique qu'un tableau de caractères). Dans ce cas, on peut migrer une chaîne de caractères (représentée comme un tableau de caractères) vers la deuxième cellule en effectuant une traduction en un objet de type *string*.

Les types translitérés sont des types dont les interfaces sont reliées par des transformations définies. Ce n'est pas seulement le cas pour les serveurs principaux d'une cellule, mais aussi pour tout autre objet, tel qu'une console, un processeur de tableaux, une base de données, etc., qui ne peut pas migrer.

4.2 Traduction d'Objets

Alors que la traduction de types maintient l'information statique des modèles d'inter-opérabilité, la traduction d'objets fournit le support dynamique et l'implantation des liens d'inter-opérabilité. Nous distinguons deux éléments dans la traduction d'objets: un statique et un dynamique. L'élément statique est responsable de la création des classes implantant les liens d'inter-opérabilité comme ils sont spécifiés dans la traduction de types correspondante. L'élément dynamique, en revanche, s'occupe de la création et la gestion des objets utilisés pendant la connexion.

L'essentiel de la traduction d'objets et d'introduire dynamiquement dans le noeud local les services des serveurs des autres noeuds. Cependant, la réalisation de l'accès aux services externes doit respecter toutes les conventions locales. Dans un noeud orienté-objet, ceci est accompli grâce à la création d'un objet local représentant le serveur externe, que nous appelons un *inter-objet.* Un inter-objet est une instance d'un type pour lequel une transformation a été définie. La classe (c'est à dire, l'implantation du type) de l'inter-objet est générée automatiquement à partir de la spécification de la transformation, et on l'appelle une *inter-classe*. Les inter-classes contiennent tout le logiciel nécessaire pour réaliser les liens avec le serveur externe.

Après la création d'un inter-objet et l'établissement des liens avec le serveur externe, l'application cliente peut commencer à appeler les opérations de l'inter-objet, donnant d'autres objets comme paramètres. Le support de l'inter-opérabilité permet l'utilisation d'objets de n'importe quel type comme paramètres aux opérations. Le service de traduction d'objets gère les objets fournis comme paramètres selon les rapports définis avec le noeud externe. De cette façon les objets avec un type équivalent ou traduit sur l'autre noeud migreront, alors que les objets avec un type translitéré seront accessible par un inter-objet sur l'autre noeud.

4.3 La Négociation de Connexion

La négociation de connexion est déclenchée quand une connexion est demandée. Il existe deux modes d'opération pour la négociation de connexion: *maître* et *esclave*. Le client se met en mode esclave et le serveur en mode maître. Cette situation est due au fait que c'est le serveur qui doit dicter ses termes au client. Si le client n'aime pas ces termes, il peut essayer auprès d'un autre serveur!

Avec l'encapsulation par cellule, le noyau est débarrassé de toutes considérations de sécurité concernant le monde externe (ce qui est géré par la membrane). Chaque accès au système d'information sera précédé par une séance de négociation de connexion dans laquelle les besoins de sécurité et les autres paramètres de connexion seront définis. Une fois que la négociation de connexion est achevé avec succès, les services véritables seront fournis. Néanmoins, les paramètres de connexion peuvent être dynamiquement renégociés, et de nouvelles procédures de sécurité peuvent être établies. Les services peuvent même être interrompus si une rupture de sécurité est soupçonnée. De plus, même pendant une connexion, des besoins de sécurité différents pour des opération spécifiques ou pour l'échange de données peuvent être imposés dynamiquement par des séances spéciales de négociation de connexion. Cependant, la négociation dynamique des procédures de sécurité sera transparente au noyau, de sorte à ce que les objets aient l'illusion d'offrir leurs services à d'autres objets locaux.

5 Conclusions

Le projet CHASSIS vise l'inter-opération sûre et fiable d'applications et de bases de données hétérogènes au moyen d'un cadre d'inter-opérabilité orienté-objet et d'un modèle de données orienté-objet pour la fédération de bases de données. Dans un premier temps, les spécifications de CHASSIS proviendront du domaine de systèmes d'information pour l'ingénierie électrique, et les résultats seront appliqués à un prototype dans ce domaine.

Le cadre orienté-objet pour l'inter-opérabilité est basé sur la notion de *cellules*, qui encapsule un système existant avec une *membrane* qui gère la négociation de la communication avec les clients et les serveurs externes. Dans CHASSIS, nous avons l'intention d'étudier les besoins de la négociation de connexion, en particulier les aspects de sécurité. De plus, nous allons étudier les implications pour la sécurité de la traduction de types et d'objets lors de la connexion de différents systèmes d'information.

Le modèle de données orienté-objet pour la fédération de bases de données cache les modèles de données des bases de données individuelles au moyen d'une couche d'intégration

dans laquelle un schéma orienté-objet global est défini. Un modèle de sécurité global sera développé pour répondre aux différents besoins et politiques de sécurité des bases de données individuelles.

En dernier lieu, nous envisageons le développement d'un environnement de composants logiciels réutilisables pour connecter des systèmes hétérogènes d'une façon sûre et fiable. Ce travail sera abordé avec un *langage de motifs* ("pattern language") qui est actuellement en développement au sein d'un autre projet ("Objets Actifs et Multi-médias"). Le langage de motifs [9] est destiné à simplifier et à généraliser les mécanismes orientés-objet d'encapsulation et de réutilisation en introduisant des objets "actifs" comme les éléments de base à l'évaluation, et les *motifs* comme un mécanisme de base d'abstraction pour la conception et le développement de composants réutilisables (remplaçant ainsi les classes, l'héritage, etc.). Puisque le langage de motifs sera lui-même sous développement pendant la phase initiale de CHASSIS, le travail de développement du cadre des motifs commencera forcément plus tard, et sera d'une nature plus expérimentale.

Remerciements

Un grand merci à Serge Renfer et à Laurent Dami, pour leur aide à la traduction en français.

Références

[1] *The Common Object Request Broker: Architecture and Specification*, Object Management Group and X Open, Document Number 91.12.1 Revision 1.1

[2] M. Atkinson et al., "The Object-Oriented Database System Manifesto," Proc. 1st Int. Conf. on Deductive and Object-Oriented Databases, Kyoto, Japan, Dec. 1989

[3] D. Denning; "Cryptography and Data Security," Addison-Wesley, Reading Massachusetts, 1982

[4] P.P. Griffith and B.W. Wade, "An Authorization Mechanism for a Relational Database System," ACM TODS, Vol. 1, No. 3, Sep. 1976, 242-255

[5] M. Haertig and K. R. Dittrich, "An Object-Oriented Integration Framework for Building Heterogeneous Database Systems," in *Proc. of the IFIP DS-5 Conf. on Semantics of Interoperable Database Systems*, Lorne, Australia, Nov. 1992.

[6] D. Konstantas, "Design Issues of a Strongly Distributed Object Based System," in *Proceedings of 2nd IEEE International Workshop for Object-Orientation in Operating Systems (I-WOOOS '91)*, Palo-Alto, October 17-18 1991, pp. 156-163.

[7] D. Konstantas, "Object-Oriented Interoperability," in *Proceedings ECOOP '93*, ed. O. Nierstrasz, LNCS, Springer-Verlag, Kaiserslautern, Germany, July 1993, to appear.

[8] D. Konstantas, "Hybrid Cell: An Implementation of an Object Based Strongly Distributed System," in *Proceedings of the International Symposium on Autonomous Decentralized Systems — ISADS 93*, Kawasaki, Japan, March 30 1993, to appear.

[9] O. Nierstrasz, "Composing Active Objects — The Next 700 Concurrent Object-Oriented Languages," in *Research Directions in Concurrent Object Oriented Programming*, ed. G. Agha, P. Wegner and A. Yonezawa, MIT Press, 1993, to appear.

[10] M. Templeton, E. Lund and P. Ward, "Pragmatics of Access Control in Mermaid," Data Engineering, Vol. 10, No. 3, Sep. 1987 (Special Issue on Federated Database Systems).