Modélisation Conceptuelle et Ingénierie des Systèmes d'Information*

Manuel KOLP Alain PIROTTE

Université catholique de Louvain, IAG-QANT
1 Place des Doyens, 1348 Louvain-la-Neuve, Belgique,
e-mail: kolp@qant.ucl.ac.be, pirotte@info.ucl.ac.be

Esteban ZIMÁNYI

Université libre de Bruxelles, INFODOC CP 175/2,
50 av. F. Roosevelt, 1050 Bruxelles, Belgique,
e-mail: Esteban.Zimanyi@ulb.ac.be

Abstract

L'utilisation de la modélisation conceptuelle dans le développement des systèmes d'information a pour objectif une prise en compte plus adéquate des besoins des applications dans leur environnement d'utilisation. La modélisation conceptuelle consiste à représenter de manière abstraite, c'est-à-dire en termes de concepts familiers aux domaines d'application et indépendamment des technologies d'implémentation, certains aspects des systèmes physiques ou humains et de leur environnement. Cet article suggère des analogies avec la science de la cognition et il motive l'importance de la modélisation conceptuelle dans la méthodologie moderne de développement de systèmes d'information. Enfin, il donne des exemples de travaux relatifs à l'utilisation de la modélisation conceptuelle dans divers domaines d'applications.

Mots-clé: modélisation conceptuelle, conception, systèmes d'information

1 Science de la cognition et informatique

La science de la cognition a été définie comme "l'étude des principes par lesquels des entités intelligentes interagissent avec leur environnement" [16]. Selon l'auteur, il s'agit d'un domaine multidisciplinaire, aux limites mal définies, qui touche, entre autres, à la psychologie, l'intelligence artificielle, la modélisation en informatique, la linguistique, l'anthropologie, la philosophie, la pédagogie, les mathématiques, l'ingénierie, la physiologie, la neurologie.

"L'objectif de la recherche est de découvrir les capacités de représentation et de traitement de l'information de l'esprit humain et leur représentation structurelle et fonctionnelle dans le cerveau humain" [16].

*Cet article traite de recherches effectuées dans le projet YEROOS (Yet another project on Evaluation and Research on Object-Oriented Strategies) basé principalement à l'Université libre de Buxelles et à l'Université de Louvain (http://yeroos.qant.ucl.ac.be).

La science cognitive et l'analyse de systèmes d'information ont en commun l'objectif de comprendre des actions complexes dirigées vers certains objectifs, indépendamment de leur mise en oeuvre, même si la première insiste surtout sur les formes d'intelligence de l'esprit humain, alors que la seconde met l'accent sur des tâches réalisables par les mécanismes de la technologie informatique.

La recherche et développement en informatique consiste bien sûr à améliorer les solutions existantes, mais aussi à imaginer de nouvelles applications plus ambitieuses dans de nouveaux domaines d'application.

Une évolution importante de l'informatique de ces dernières décennies est la séparation toujours plus claire entre la spécification de la fonction disponible pour les utilisateurs à l'interface des systèmes d'information et la réalisation ou implémentation de cette fonction à l'aide de la technologie du moment. Il s'agit d'une stratégie de rationalisation et de simplification[1] rendue possible par l'augmentation considérable de la puissance des ordinateurs et, en même temps, rendue nécessaire, sur le plan méthodologique, par l'inadéquation des générations précédentes de méthodes de développement de systèmes d'information. Ces méthodes, parfois qualifiées de "structurées" et encore largement utilisées en pratique, étaient trop rigides et optimistes quant à la complexité de la tâche.

La cognition et la modélisation en informatique ont quelques caractéristiques importantes en commun:

(1) l'importance du traitement symbolique: les systèmes étudiés (par exemple, l'esprit humain ou l'entreprise) ne le sont pas en termes de propriétés intrinsèques des systèmes eux-mêmes mais à l'aide de formulation et de manipulation de modèles et théories exprimés dans le style de formalismes typiques de la logique;
(2) la séparation entre fonctions et réalisation de ces fonctions: dans les phases du cycle de vie (voir section 2.2) des méthodes de développement de logiciel d'application et dans la séparation entre les fonctions de l'esprit humain, centre d'intérêt de la cognition, et leur réalisation biologique, gouvernée par les lois de la biochimie ou de la biophysique;
(3) un certain style d'analyse descendante (top-down) ou la compréhension de fonctions globales est privilégiée par rapport a l'étude expérimentale empirique.

2 Développement de systèmes d'information

2.1 Crise du Logiciel

Malgré les progrès constants et considérables des technologies de l'informatique, les systèmes d'information complexes répondent souvent bien moins que parfaitement aux besoins qu'ils sont supposés satisfaire. Le diagnostic de "crise du logiciel" de la fin des années 60 est encore d'actualité. L'informatique porte souvent le chapeau dans le grand public ("c'est la faute à l'ordinateur"), mais, bien sûr, dans la plupart des cas, les vraies causes de dysfonctionnement sont à rechercher dans l'inadéquation de décisions humaines ou dans

[1]Cette distinction plus nette entre interface et implémentation est, par exemple, une des caractéristiques principales de la technologie des objets, qui est en train d'introduire des progrès substantiels dans les techniques du logiciel et qui devrait améliorer la qualité des solutions informatiques [13].

des erreurs ou insuffisances méthodologiques, comme par exemple pour l'échec récent de la fusée Ariane 5[2].

Le terme de "génie logiciel" (software engineering), discipline qui peut se définir comme l'ensemble des processus méthodologiques et des produits pertinents au développement de logiciels à grande échelle, a d'ailleurs été inventé pour susciter une recherche méthodologique plus appropriée.

La crise du logiciel a plusieurs causes:

- la complexité toujours croissante des systèmes d'information modernes: les progrès constants de la technologie informatique suscitent et rendent possibles des applications toujours plus ambitieuses. De plus, l'accroissement de la demande excède constamment les gains bien réels de productivité, dûs à l'amélioration des méthodes, techniques et outils, et aux progrès méthodologiques et scientifiques de la discipline);
- la sous-estimation traditionnelle de la difficulté et donc du coût du développement de logiciels (les méthodes largement intuitives de la programmation individuelle ne s'étendent pas à la construction de grands systèmes complexes), avec comme conséquence un processus de développement qui se révèle être souvent complexe, long, coûteux, et des produits finaux souvent inadéquats;
- une fiabilité et une maturité plus faibles pour les technologies du logiciel que pour celles du matériel, alors que l'importance relative du logiciel dans la réalisation des fonctions de systèmes complexes ne cesse de croître;
- un retard substantiel des pratiques de l'industrie du logiciel par rapport à ce que permettrait une bonne application des principes (voir par exemple [7, 13, 17]), dû à la difficulté et au coût de l'enseignement, de la formation et du maintien au courant des développeurs dans une discipline en évolution constante;
- une complexité intrinsèque:
  - les problèmes auxquels s'adresse le logiciel peuvent être arbitrairement complexes; par exemple, les limites de la fonctionnalité d'un système logiciel peuvent être beaucoup moins claires que celles de produits plus tangibles;
  - le découpage séquentiel des phases de développement est moins naturel pour le logiciel que dans d'autres branches de l'ingénierie (fabrication mécanique, par exemple).

2.2 Méthodologie de développement

Les questions méthodologiques sont plus complexes que dans des disciplines plus anciennes et donc plus mûres. En informatique, il est possible de commencer la programmation avant que la phase de conception soit terminée, parce que des modifications incrémentales aux programmes sont moins onéreuses que les modifications correspondantes dans des productions plus tangibles (génie civil ou fabrication mécanique, par exemple).

Depuis le diagnostic de crise du logiciel, la recherche méthodologique s'est grandement développée et ce domaine est plus actif que jamais (voir par exemple [4, 8, 13]).

Les méthodes modernes de conception de logiciel s'accordent sur une définition générale des phases de ce qu'on appelle le **cycle de vie** du logiciel:

[2]Voir notre analyse du rapport officiel à l'adresse http://yeroos.qant.ucl.ac.be/dummies/ariane.html.

- l'**analyse** précise des besoins à satisfaire par le futur système, on parle aussi de phase d'**ingénierie des besoins**; il s'agit d'appréhender le domaine du problème posé et de spécifier le comportement externe (boîte noire) d'un système qui peut résoudre ce problème; il ne s'agit pas à ce stade de résoudre le problème et encore moins de faire intervenir les détails de la technologie; cette phase implique une modélisation adéquate des concepts pertinents du monde réel, la **modélisation conceptuelle**;
- la **conception** d'une solution au problème posé et de la structure générale d'un système qui va le résoudre;
- l'**implémentation** de la conception à l'aide de la technologie choisie (typiquement, un langage de programmation);
- le **test** et la **validation** de l'implémentation, et la mise en **opération** du système;
- l'adaptation, ou **maintenance**, du système opérationnel aux changements des besoins et de l'environnement d'utilisation.

Les idées sur les qualités souhaitables des méthodes de développement d'applications ont beaucoup évolué. Par exemple, un apport important de la technologie des objets est que la structure des logiciels développés avec cette technologie peut refléter bien mieux celle du problème posé, ce qui simplifie le développement et sa compréhension.

Le changement peut-être le plus important découle de la reconnaissance plus explicite que, si un système reste en activité pendant un certain temps (ce qui devrait être la situation habituelle), alors il est normal que tant les besoins auxquels il répond que l'environnement dans lequel il évolue (le monde réel avec lequel le système interagit) évoluera et que le système devra être modifié. Auparavant, la maintenance et l'évolution étaient plutôt considérées comme un mal nécessaire, bien qu'on estime à plus de 50% des ressources globales la part typiquement consacrée à l'amélioration et à la maintenance après la première livraison du système.

Il est donc nécessaire, pour améliorer la qualité des produits logiciels et mieux contrôler leur coût, que l'évolution et la maintenance soient anticipées et organisées dès le début du processus de développement.

L'évolution, toujours en cours, des idées sur la méthodologie de développement des systèmes d'information tend donc à consacrer une part plus grande de l'effort global aux phases amont du processus (c'est-à-dire l'analyse des besoins et la modélisation conceptuelle). C'est un changement important par rapport aux pratiques, ou souvent, ces phases amont ne sont considérées que comme des aides servant à maîtriser une phase de programmation complexe, considérée comme centrale au processus de développement.

L'intérêt croissant porté à la modélisation conceptuelle et à la recherche dans ce domaine tient aussi à l'évolution des coûts de l'informatique. Les phases de maintenance et d'évolution des systèmes d'information impliquent des activités créatives qui ne peuvent être automatisées. Or, en gros, il y a 20 ans, une heure de temps machine coûtait autant qu'une semaine de travail d'un programmeur, alors qu'aujourd'hui, une semaine de programmation coûte la même chose qu'un ordinateur personnel. Par conséquent, faire le meilleur usage des ressources humaines est devenue une préoccupation majeure du développement d'applications.

3 La modélisation conceptuelle

L'idée qui sous-tend la modélisation conceptuelle est très simple: il s'agit de développer un modèle d'un domaine d'application particulier en termes des concepts familiers aux acteurs de ce domaine et non en termes de la technologie informatique tels que "fichiers", "stratégies d'accès", "programmation", ou "architecture client-serveur", par exemple. Ces considérations technologiques ne deviennent pertinentes que plus tard, dans la phase d'implémentation, postérieure à celle de modélisation conceptuelle.

Un bon modèle conceptuel implique un processus d'**abstraction**, de simplification d'une situation perçue comme complexe dans le monde réel et de suppression des détails dont l'effet sur la solution au problème posé est minime ou inexistant. La solution au problème est donc aussi simplifiée, et son implémentation rendue plus performante, par une focalisation sur un modèle traitant de l'essence du problème.

Comme toute activité de modélisation, la modélisation conceptuelle est une activité créative non automatisable et non déterministe. Un modèle conceptuel n'est pas correct ou incorrect, il est plus ou moins adéquat à certains besoins. Un modèle plus détaillé n'est donc pas nécessairement un modèle meilleur.

Les modèles conceptuels pourraient remplir plusieurs fonctions dans le cycle de vie d'un produit logiciel:

- améliorer la compréhension des structures et fonctions d'un domaine du monde réel, grâce à un modèle basé sur des concepts clairs et intuitifs proches de ceux utilisés par les acteurs du domaine. Un modèle conceptuel peut être utile pour mieux comprendre la complexité, même si on n'envisage pas de construire un système informatique;
- fournir un vecteur de communication précis entre modélisateurs et développeurs du système d'une part, et spécialistes du domaine d'application et utilisateurs finaux d'autre part;
- spécifier et guider les phases aval du cycle de vie du système telles l'implémentation, la programmation et les optimisations diverses, et constituer une trace de haut niveau, compréhensible par les acteurs du domaine d'application et utilisable pour l'évolution et la maintenance du système en fonctionnement;
- fournir une documentation de haut niveau, à destination des utilisateurs, basée sur les concepts du domaine d'application.

L'état des pratiques est cependant bien différent. Le plus souvent, les modèles conceptuels sont uniquement utilisés (s'ils le sont) lors du développement initial du système, puis traduits dans le langage d'implémentation, et la maintenance et l'évolution se font uniquement sur les programmes sans être répercutés dans le modèle conceptuel, qui cesse ainsi d'être à jour et devient inutile.

4 Applications de la modélisation conceptuelle

La modélisation conceptuelle consiste donc à construire des représentations abstraites de certains aspects des systèmes physiques et sociaux et de leur environnement dans le monde

qui nous entoure. La modélisation est une entreprise interdisciplinaire, ou les informaticiens se doivent d'étudier les domaines d'application suffisamment pour en comprendre les concepts importants et les termes dans lesquels se posent les problèmes, et les spécialistes des domaines d'application se doivent de pouvoir appréhender les possibilités de la technologie du moment. Sans cette collaboration interdisciplinaire, l'informatique risque de se contenter d'analyser les pratiques, souvent en retard par rapport à ce que permet une bonne utilisation de la technologie.

La modélisation conceptuelle nécessite des notations, outils et techniques de représentation de l'information et des traitements concernant un domaine d'intérêt. Des progrès en modélisation conceptuelle consistent à définir des langages de modélisation simples, puissants et de haut niveau, qui permettent de réduire la distance entre les concepts familiers aux acteurs dans les domaines d'application et leur représentation dans les modèles.

C'est ainsi que nous étudions des associations sémantiques comme la *matérialisation* [15], une association qui caractérise le lien entre des objets abstraits (par exemple, des modèles de voiture) et des objets concrets (par exemple, des voitures individuelles) ou encore l'agrégation (ou association *partie-de*) [9] qui caractérise le lien entre un objet composite et des objets composants.

Les associations sémantiques permettent de mieux structurer les informations dans les modèles conceptuels et permettent de capturer des phénomènes complexes typiques des domaines d'application non traditionnels. Cependant les langages d'implémentation d'aujourd'hui ne supportent que quelques associations sémantiques de base comme la généralisation et la classification. Dès lors, nous étudions [5, 6, 10, 11, 18] différentes approches pour implémenter les associations sémantiques dans les langages de programmation, notamment en utilisant les mécanismes de métaclasses et de métaobjets.

Les domaines d'application de la modélisation conceptuelle sont très variés. Ceci est dû à l'évolution des systèmes d'information qui abordent aujourd'hui des domaines nouveaux qui n'étaient pas envisagées par l'informatique traditionnelle. C'est par exemple le cas des systèmes d'information géographiques (SIG) [14] ou le traitement des informations temporelles [19]. Les modèles conceptuels spatio-temporels existants ne sont pas satisfaisants puisque ils sont très liés au mécanismes d'implémentation propres aux logiciels SIG. C'est ainsi que nous avons travaillé dans le développement d'un modèle conceptuel spatio-temporel permettant d'offrir aux utilisateurs une interface conceptuelle de définition et d'accès aux bases de données spatio-temporelles.

D'autres domaines d'application que nous avons abordé incluent les bases de données administratives [1, 2, 3], la modélisation de macromolécules biologiques [12], l'automatisation de la gestion des bibliothèques, la gestion d'un secrétariat medical inter-entreprise, la gestion de dossiers de production dans l'industrie chimique, la gestion du crédit bancaire international et l'organisation de conférences internationales.

References

[1] I. Boydens. La banque de données LATG de l'ONSS. Les flux de l'information traitée à partir d'une banque de données : étude critique. Mémoire de Licence Spéciale en Sciences de l'Information et de la Documentation, INFODOC, Université Libre de Bruxelles. Lauréat du Concours des Bourses de Voyage de la Communauté Française de Belgique, 1992.

[2] I. Boydens. Statistical exploitation method for administrative databases. In *Proc. of the Int. Conf. on New Techniques and Technologies and Statistics*, pages 63–70, Bonn, Germany, Nov. 1995.

[3] I. Boydens, A. Pirotte, and E. Zimányi. Managing constraint violations in administrative information systems. In S. Spaccapietra and F. Maryanski, editors, *Data Mining and Reverse Engineering: Searching for semantics, Proc. of the 7th IFIP Conf. on Data Semantics, DS-7*, pages 347–372, Leysin, Switzerland, Oct. 1997. Chapman & Hall.

[4] D. Coleman, P. Arnold, S. Bodoff, C. Dollin, H. Gilchrist, F. Hayes, and P. Jeremaes. *Object-Oriented Development: The Fusion Method.* Prentice Hall, 1994.

[5] M. Dahchour, A. Pirotte, and E. Zimányi. Metaclass implementation of materialization. Technical Report YEROOS TR-96/06, IAG-QANT, Université catholique de Louvain, Belgium, Jan. 1996. Older version of TR-99/01.

[6] M. Dahchour, A. Pirotte, and E. Zimányi. Metaclass implementation of generic relationships. Technical Report YEROOS TR-97/25, IAG-QANT, Université catholique de Louvain, Belgium, 1997. Submitted for publication.

[7] A. Davis. *201 principles of software development.* McGraw-Hill, 1995.

[8] D. D'Souza and A. Wills. *Objects, Components, and Frameworks with UML: The Catalysis Approach.* Addison-Wesley, 1998.

[9] M. Halper, J. Geller, and Y. Perl. An OODB part relationship model. In Y. Yesha, editor, *Proc. of the 1st Int. Conf. on Information and Knowledge Management, CIKM'92*, LNCS 752, pages 602–611, Baltimore, USA, 1992. Springer-Verlag.

[10] M. Kolp. A metaobject protocol for reifying semantic relationships into open systems. In *Proc. of the 4th Doctoral Consortium of the 9th Int. Conf. on Advanced Information Systems Engineering, CAiSE'97*, pages 89–100, Barcelona, Spain, June 1997.

[11] M. Kolp and A. Pirotte. An aggregation model and its C++ implementation. In M. Orlowska and R. Zicari, editors, *Proc. of the 4th Int. Conf. on Object-Oriented Information Systems, OOIS'97*, pages 211–224, Brisbane, Australia, Nov. 1997.

[12] D. Massart and J. Richelle. Object-oriented conceptual modeling of databases for macromolecular structures. In P. Charrel, H. Jaakkola, H. Kangassalo, and E. Kawaguchi, editors, *Information Modelling and Knowledge Bases IX*, pages 146–159. IOS Press, 1998.

[13] B. Meyer. *Object-oriented Software Construction.* Prentice Hall, second edition, 1997.

[14] C. Parent, S. Spaccapietra, and E. Zimányi. Conceptual modeling for federated geographical information systems over the Web. In *Proc. Int. Symp. on Information Systems and Technology for Network Society*, pages 173–182, Fukuoka, Japan, Sept. 1997.

[15] A. Pirotte, E. Zimányi, D. Massart, and T. Yakusheva. Materialization: a powerful and ubiquitous abstraction pattern. In J. Bocca, M. Jarke, and C. Zaniolo, editors, *Proc. of the 20th Int. Conf. on Very Large Data Bases, VLDB'94*, pages 630–641, Santiago, Chile, 1994. Morgan Kaufmann.

[16] Z. Pylyshyn. Cognitive science. In S. Shapiro and D. Eckroth, editors, *Encyclopedia of artificial intelligence*, pages 120–124. John Wiley & Sons, 1987.

[17] I. Sommerville. *Software Engineering.* Addison-Wesley, fourth edition, 1992.

[18] E. Zimányi. Implementing materialization in Logtalk. Technical Report YEROOS TR-97/09, Laboratoire de Bases de Données, Département d'Informatique, Ecole Polytechnique Fédérale de Lausanne, Switzerland, Apr. 1997.

[19] E. Zimányi, C. Parent, S. Spaccapietra, and A. Pirotte. TERC+: a temporal conceptual model. In *Proc. Int. Symp. on Digital Media Information Base, DMIB'97*, Nara, Japan, Nov. 1997.