Théorie des transformations de graphes : une approche algébrique

Xavier Seignard

14 juin 2010

1 Introduction

Dans cette article, nous allons présenter de manière formelle l'approche algébrique de la théorie des transformations de graphes adoptée par AGG [Tae00], outil de transformation de graphes. Nous allons d'abord présenter les notions de base, puis nous expliquerons quelle approche de transformation de graphe est mise en œuvre dans AGG.

2 Définitions de base

Définition 1 *Graphe*
Un graphe G dirigé et dont les arcs et les nœuds sont annotés, est un système défini comme suivant :

$$G = (V, E, s, t, l_E, l_V)$$

Avec :

- V : un ensemble fini de nœuds ;
- E : un ensemble fini d'arc ;
- s : une fonction $s : E \to V$, qui assigne une source $s(e) \in V$ pour chaque arc $e \in E$;
- t : une fonction $t : E \to V$, qui assigne une cible $t(e) \in V$ pour chaque arc $e \in E$;
- l_E : une fonction $l_E : E \to G$, qui assigne un label $l(e)$ pour chaque arc $e \in E$;

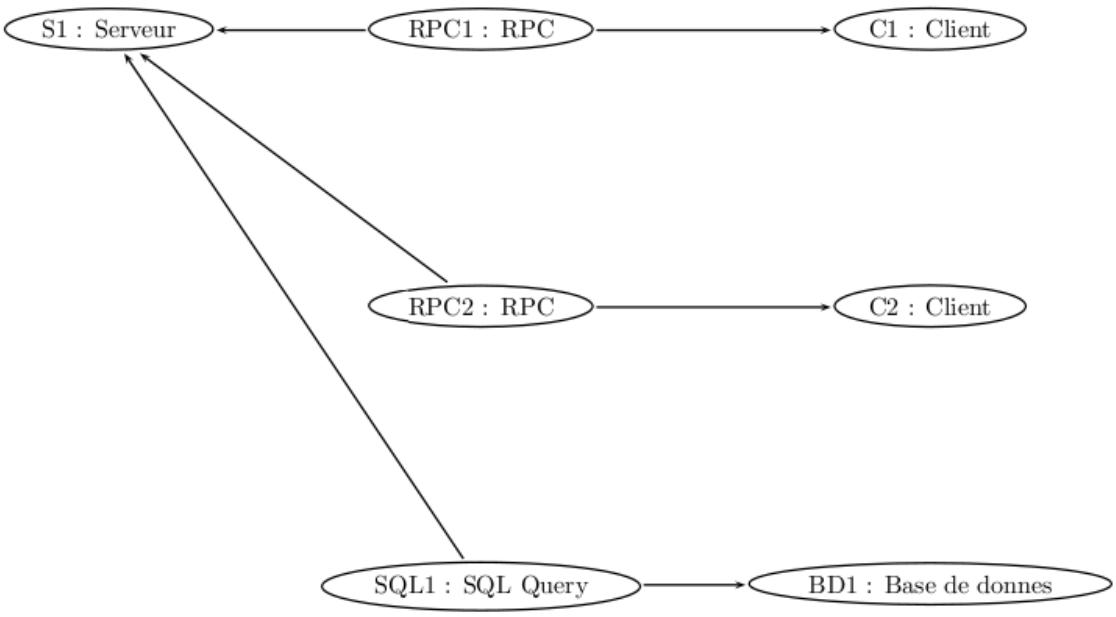


FIGURE 1 – Représentation d'une application Client/Serveur sous forme de graphe

– l_V : une fonction $l_V : V \to G$, qui assigne un label $l(v)$ pour chaque nœud $v \in E$.

Un graphe permet de représenter des structures de données, les différents états de systèmes concurrents et distribués, ou encore l'état d'un ensemble d'objets régis par des relations. Citons comme exemples, les réseaux de Petri [Rei85], les diagrammes UML [OMG07] ou encore les automates finis [HMU01]. Les graphes permettent ainsi de représenter de manière abstraite des structures complexes. La figure 1 présente un exemple Client/Serveur sous forme de graphe.

Afin de pouvoir expliquer comment modéliser les opérations d'évolution de SAEV sous forme de transformations de graphes, nous devons d'abord aborder les deux notions suivantes :

– sous-graphe ;
– morphisme de graphe.

Définition 2 *Sous-graphe*
Un graphe $G = (V_G, E_G, s_G, t_G, l_{E_G}, l_{V_G})$ *est sous-graphe de* $H = (V_H, E_H, s_H, t_H, l_{E_H}, l_{V_H})$ *(équivaut à* $G \subseteq H$*) si et seulement si :*

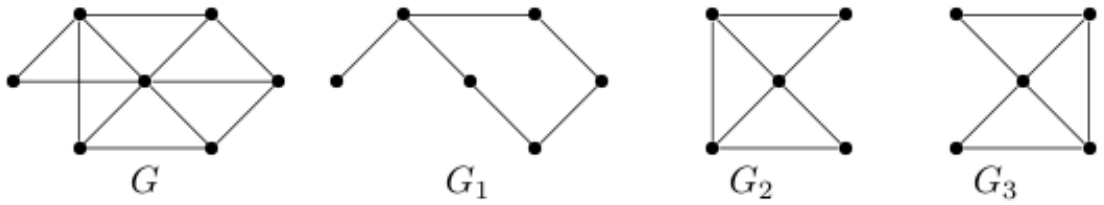


FIGURE 2 – Quatre graphes G ,G1 ,G2 , G3

- $V_G \subseteq V_H$ *: tous les nœuds de G existent dans H ;*
- $E_G \subseteq E_H$ *: tous les arcs de G existent dans H ;*
- $s_G(e) = s_H(e)$ *: chaque arc $e \in G$ a le même nœud source dans G et H ;*
- $t_G(e) = t_H(e)$ *: chaque arc $e \in G$ a le même nœud cible dans G et H ;*
- $l_{E_G}(e) = l_{E_H}(e)$ *: chaque arc $e \in G$ a le même label dans G et H ;*
- $l_{V_G}(v) = l_{V_H}(v)$ *: chaque nœud $v \in G$ a le même label dans G et H.*

L'obtention d'un sous-graphe n'est effective que si elle s'accompagne de la suppression de tous les arcs adjacents aux nœuds supprimés. L'obtention d'un sous-graphe H par la soustraction d'un ensemble nœuds et d'arcs $X = (V_x, E_x)$ à $G = (V, E, s, t, l_E, l_V)$, avec $V_x \subseteq V$ et $E_x \subseteq E$, est définie comme suivant :

$$H = G - X = (V - V_x, E - E_x, s', t', l'_E, l'_V)$$

avec :

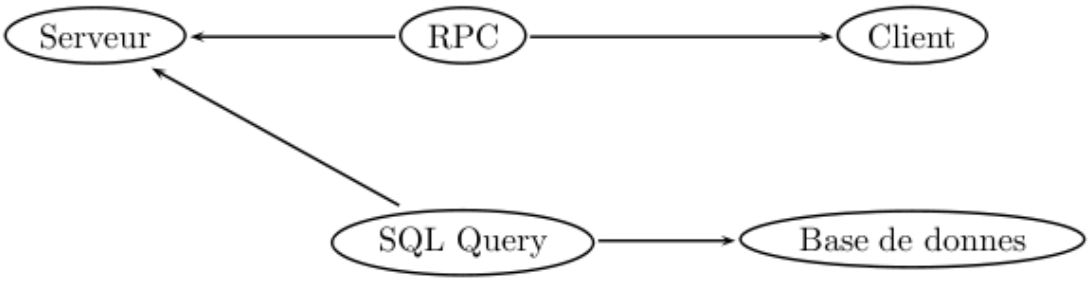
- $s'(e) = s(e)$;
- $t'(e) = t(e)$;
- $l'_E(e) = l_E(e), \forall e \in E - E_x$;
- $l'_V(v) = l_V(v), \forall e \in V - V_x$.

Dans la figure 2, nous voyons que G1 et G2 sont deux sous-graphes de G car ils sont conformes à la définition 2, mais G3 n'est pas un sous-graphe de G car EG3 EG .

Appliqué à notre architecture logicielle cela donne le figure 3.

Définition 3 *Morphisme de graphes*
Un morphisme de graphe $g : G \to L$ est une paire de fonctions définies ainsi :

- $g_V : V_G \to V_L$
- $g_E : E_G \to E_L$



(a) Un graphe X

(b) Un sous-graphe de X

FIGURE 3 – Un sous-graphe (b) de l'architecture logicielle Client/Serveur

Où g_V est la fonction de correspondance entre les nœuds, et g_E la fonction de correspondance entre les arcs. Un morphisme est donc une famille de relations entre les ensembles porteurs du graphe source vers les ensembles porteurs du graphe cible préservant la structure des graphes ainsi que les labels [Gra00, CMR+97]. Les notions de morphisme et de sous-graphe vont nous permettre de comprendre comment la transformation de graphe s'effectue. Les notions suivantes sont liées et nous permettent de mieux comprendre l'approche qu'adopte AGG pour la transformation de graphes.

Définition 4 *Production*
Un production $p : L \rightsquigarrow R$ est un morphisme partiel[1]*, où L (respectivement R) est appelée partie gauche (respectivement droite) de la production, qui représente un ensemble potentiellement infini de dérivations directes. Dans la catégorie des graphes, une production est appelée pushout [Gra00].*

Si un match m (voir définition 5) permet d'identifier une occurence de L dans un graphe G donné, alors $G \Rightarrow p, mH$ représente un dérivation directe où p est appliquée, selon le match m, à G et produisant le graphe H [CMR+97, Gra00, Hec05].

1. Il existe des éléments sans image, ou/et sans antécédents.

(a) Partie Gauche (b) Partie Droite

FIGURE 4 – L'opération d'évolution ajout-client sous forme de production

Une production p est un morphisme de graphes partiel $L \to R$, ainsi les éléments de ce morphisme sont conservés par l'application de la production. Tous les autres éléments de L (respectivement R) sont supprimés (respectivement créés) par l'application de la production [Tae00].

La partie gauche de la règle est en correspondance avec un sous-graphe de l'application Client/Serveur présentée sous forme de graphe en figure 4.

Définition 5 *Match (correspondance)*
Un match $m : L \to G$ est un morphisme entre un graphe G et la partie gauche d'une production.

Le match de L dans G est donc un sous-graphe $m(G)$ régit par la paire d'images $m_V(V)$ et $m_E(E)$. Une dérivation $G \Rightarrow p, mH$ doit satisfaire deux conditions pour qu'elle soit applicable :

– la condition d'identification ;
– et la dangling condition

qui à eux deux forment la gluing condition [CMR+97, Hec05].

Définition 6 *Condition d'identification*
Soient deux éléments x et y de L, la partie droite d'une production p définie en 4. La condition d'identification est vérifiée si pour la fonction $o : L \to R$ nous avons [Hec05, EHK+97, CMR+97] :

$$o(x) = o(y) \Rightarrow (x = y) \vee (x, y \in L \cap R)$$

Cela signifie que chaque élément e du graphe R, et qui est supprimé par l'application de la production p, doit posséder un antécédent unique dans L.

Dans l'outil AGG, la fonction o est définie par l'utilisateur ou bien elle est générée si elle n'est pas précisée [Tea06].

Définition 7 *Dangling condition*
La dangling condition est satisfaite si et seulement si [Hec05, EHK+97, CMR+97] :

$$\forall e \in E_{g(R)} - E_{g(L)} \Rightarrow s(e), t(e) \in V_{g(R)} - V_{g(L)}$$

Cela signifie que la suppression d'un nœud implique forcément la suppression de ses arcs adjacents.

La somme des conditions définies en 6 et 7 forme ce que l'on appelle la gluing condition. Cette condition doit être validée pour qu'une production soit applicable.

Après avoir défini un ensemble de concepts de la théorie des transformations de graphes, voyons comment ceux-ci sont appliqués dans l'outil AGG.

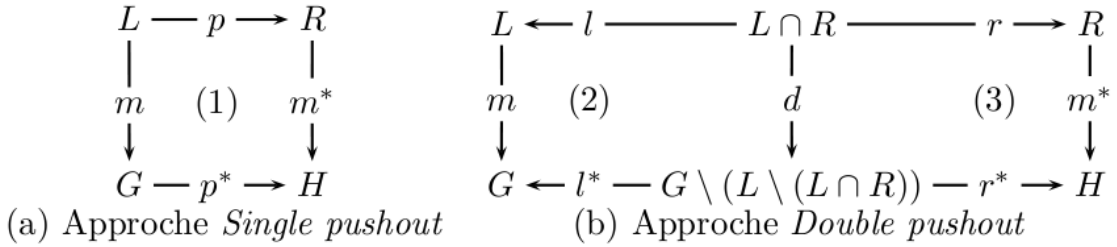
3 Transformations de graphes selon AGG

Les définitions précédentes nous permettent de mieux comprendre l'approche d'AGG pour la transformation de graphes. Voyons l'approche adoptée dans AGG.

Il existe plusieurs façons d'appliquer une production p afin de dériver de G (le graphe avant transformation) vers H (le graphe après transformation). L'approche retenue par l'outil AGG est l'approche dite de single pushout [Tae00].

Définition 8 *Single pushout*
L'approche dite de single pushout est une application d'une production p. Cependant, celle-ci n'est pas régie par la validation de la gluing condition contrairement à l'approche double pushout.

Malgré tout, de par la définition même d'un graphe, un arc ne peut exister sans nœud à ses extrémités. Une stratégie de suppression de tels arcs est alors adoptée, même si cette suppression n'est pas définie dans la production p. De plus, comme cité en définition 6, l'identification est soit définie par l'utilisateur d'AGG ou bien définie par défaut par AGG, de manière à ce qu'elle favorise la suppression des nœuds et arcs qui ne sont pas identifiés en cas de conflits.



(a) Approche *Single pushout* (b) Approche *Double pushout*

FIGURE 5 – Approches «Single pushout» et «Double pushout»

L'outil AGG, bien qu'il adopte une approche single pushout, dispose donc de mécanismes de vérification de l'applicabilité des productions proches de ceux présentés précédemment. Cependant, les approches single pushout et double pushout sont équivalentes, si les conditions de vérifications se limitent à la gluing condition [EHK+97, BHKR07]. La figure 5 présente les deux concepts de single pushout et double pushout. Le concept de double pushout consiste en une production p décomposée en deux morphismes totaux [2] de graphes $l : L \cap R \to L$ et $r : L \cap R \to R$. Ainsi, l permet la suppression des éléments et r l'ajout et la modification. Le graphe de contexte $D = G\ (L\ (L \cap R))$ est obtenu depuis G , par le co-morphisme (ou morphisme associé) l^{invXX} , en y supprimant les éléments de l'ensemble $L\ L \cap R$: les éléments de L n'ayant pas d'image par la production $p : L \rightsquigarrow R$. Une fois les éléments supprimés de G, il faut ajouter ou modifier ceux qui doivent l'être. Cette opération s'effectue par l'insertion des éléments de R, n'ayant pas d'antécédent dans $L \cap R$, dans le graphe D : il s'agit du co-morphisme r^{invXX} . La décomposition de la production p en deux morphismes permet de s'assurer que la gluing condition est vérifiée.

L'approche single pushout, quant à elle, est l'application directe d'une production p selon un match m. La production p est un morphisme partiel de graphes, car il existe des éléments sans image, et d'autres sans antécédent. Dans cette approche, la suppression d'un nœud de G implique la suppression de tous ses arcs adjacents même si cela n'est pas spécifié dans p. Du fait que p soit un morphisme partiel, son co-morphisme p^{invXX} est aussi un morphisme partiel : la suppression de certains arcs est effective alors qu'elle n'est pas spécifiée dans p, par exemple. Il se peut alors que que le co-match m^{invXX} soit aussi partiel.

2. Tout élément possède une image.

Malgré tout, la dérivation $G \Rightarrow p, mH$ est possible. Cela apporte une souplesse qu'il n'est pas possible d'avoir avec l'approche double pushout. Imaginons, par exemple, un réseau pair à pair et un client qui tombe en panne, l'approche single pushout permet de supprimer automatiquement les connexions adjacentes au pair en panne.

4 Enrichissement de l'approche algébrique

L'outil AGG permet d'exploiter des concepts supplémentaires de l'approche algébrique des transformations de graphes. Nous allons présenter ceux-ci.

La définition d'un graphe ne permet pas d'exprimer des contraintes sur les connexions entre arcs et nœuds, comme par exemple : tel nœud, portant tel label, ne peut être source de tel arc, portant tel label. De telles contraintes peuvent être exprimées à l'aide d'un graphe type [Hec05].

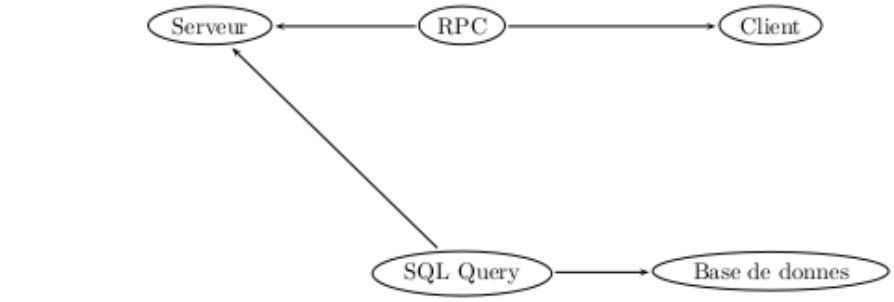
Définition 9 *Graphe type*
Soient deux graphes $TG = (V_{TG}, E_{TG}, s_{TG}, t_{TG}, l_{E_{TG}}, l_{V_{TG}})$ *et* $\langle G, tg^G \rangle$ *. On dit que* TG *est le graphe type de* G *si le morphsime* $tg^G : G \rightarrow TG$ *préserve les labels (types) et si les nœuds source et cible d'un arc de label* x *de* G *portent les mêmes labels que dans* TG*.*

Rajoutons que le graphe G est appelé instance de TG [Hec05].

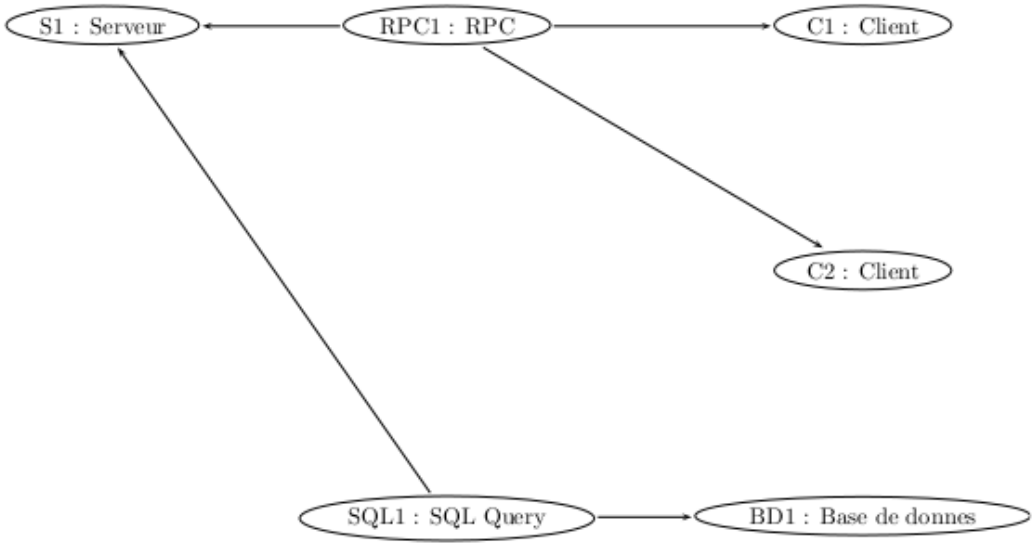
Cette notion s'applique parfaitement à la description des méta et instance. La théorie de transformations de graphe propose de facto des notions pour représenter l'abstraction architecturale, comme le montre la figure 6.

Définition 10 *Condition de non application*
Une condition de non application est un graphe N *, associé à une production* p*. Pour que* $G \Rightarrow p, mH$ *soit applicable, il faut que* $N \nsubseteq G$ *[Tea06].*

Cela signifie que la dérivation est inapplicable si N est un sous-graphe de G. Ces conditions de non application permettent de définir plus précisément les productions. En effet, il est possible que l'on veuille exprimer une production selon une condition particulière.



(a) Graphe type



(b) Graphe conforme au graphe type

FIGURE 6 – (a) Niveau architecture et (b) niveau application

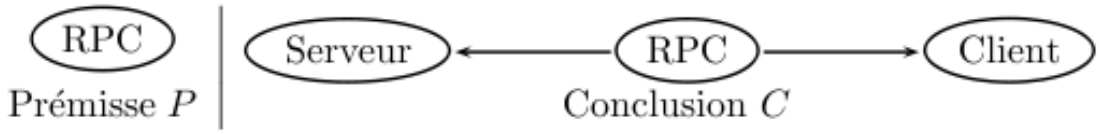


FIGURE 7 – Un connecteur RPC doit être relié à un composant Serveur et à un composant Client

Définition 11 *Contraintes de consistance*
Une contrainte de consistance sur un graphe G est un morphisme injectif[3]

$$c : P \to C$$

où P est appelé prémisse et C conclusion. P et C étant des sous-graphes de G. [HHS02, Tea06, HW95]

Définir une contrainte de consistance c entre P et C revient à dire que tout prémisse P dans G possède une image C par c dans G.

Définissons la contrainte suivante : dans une architecture Client/Serveur, tout connecteur RPC doit être relié à un composant Serveur et à un composant Client. Cette contrainte s'exprime par la contrainte de consistance définie en figure 7.

Effectivement, un connecteur RPC connecté à rien n'a aucun sens.

Définition 12 *Paires critiques*
Soient deux dérivations :

$$H_1 \Leftarrow p_1, m_1 G \Rightarrow p_2, m_2 H_2$$

telles que celles-ci soient en conflit. Ajoutons qu'il est nécessaire que G soit l'union des parties gauches L_1 et L_2 de p_1 et p_2 pour que p_1 et p_2 soient potentiellement applicables [HHT02] [Tea06] :

$$G = m_1(L_1) \cup m_2(L_2)$$

Une paire critique (p_1, p_2) est donc une paire de dérivations telle que l'application de l'une empêche l'application de l'autre.

3. Tout élément de C n'a qu'n antécédent unique dans P par c.

5 Conclusion

Voici une première approche algébrique de la théorie des transformation de graphes. Pour plus de détails se reporter à la bibliographie.

Références

[BHKR07] I.B. Boneva, F. Hermann, H. Kastenberg, and A. Rensink. Simulating multigraph transformations using simple graphs. In *Proceedings of the Sixth International Workshop on Graph Transformation and Visual Modeling Techniques*, pages 42–56, 2007.

[CMR+97] A. Corradini, U. Montanari, F. Rossi, H. Ehrig, R. Heckel, and M. Löwe. *Handbook of Graph Grammars*. World Scientific Publishing Co., Inc., 1997.

[EHK+97] H. Ehrig, R. Heckel, M. Korff, M. Löwe, L. Ribeiro, A. Wagner, and A. Corradini. *Handbook of Graph Grammars*. World Scientific Publishing Co., Inc., 1997.

[Gra00] P. Gradit. L'approche catégorique des grammaires de graphes. *Formalisation des Activités Concurrentes (FAC'2000)*, pages 70–81, 2000.

[Hec05] R. Heckel. Stochastic analysis of graph transformation systems : A case study in p2p networks. In *2nd International Colloquium on Theoretical Aspects of Computing (ICTAC'05)*, 2005.

[HHS02] J. Hausmann, R. Heckel, and S. Sauer. Extended model relations with graphical consistency conditions. In *Workshop on Consistency Problems in UML-based Software Development*, pages 61–74, 2002.

[HHT02] J. Hausmann, R. Heckel, and G. Taentzer. Detection of conflicting functional requirements in a use case-driven approach : a static analysis technique based on graph transformation. In *ICSE '02 : Proceedings of the 24th International Conference on Software Engineering*, pages 105–115, 2002.

[HMU01] J.E. Hopcroft, R. Motwani, and J.D. Ullman. Introduction to automata theory, languages, and computation, 2nd edition. *SIGACT News*, 32 :60–65, 2001.

[HW95] R. Heckel and A. Wagner. Ensuring consistency of conditional graph rewriting - a constructive approach. *Electronic Notes Theorical Computer Science*, 2, 1995.

[OMG07] OMG. Unified modeling language specification (version 2.1), 2007.

[Rei85] W. Reisig. *Petri nets : an introduction*. Springer-Verlag, 1985.

[Tae00] G. Taentzer. Agg : A tool environment for algebraic graph transformation. In *AGTIVE '99 : Proceedings of the International Workshop on Applications of Graph Transformations with Industrial Relevance*, pages 481–488, 2000.

[Tea06] AGG Team. The agg 1.5.0 development environment, the user manual, 2006.