HAL
archives-ouvertes.fr

GCSS : Feuilles de Styles pour la Visualisation de Graphes

Guillaume Artignan, Mountaz Hascoët

▶ **To cite this version:**

Guillaume Artignan, Mountaz Hascoët. GCSS : Feuilles de Styles pour la Visualisation de Graphes. RR-09027, 2009. <lirmm-00416180>

HAL Id: lirmm-00416180
https://hal-lirmm.ccsd.cnrs.fr/lirmm-00416180

Submitted on 12 Sep 2009

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

GCSS: feuilles de styles pour la visualisation de graphes

Guillaume Artignan, Mountaz Hascoët

LIRMM, UMR 5506 - CC 477
161 rue Ada
34392 Montpellier Cedex 5 - France
artignan@lirmm.fr, mountaz@lirmm.fr

RESUME
Dans cet article, nous nous intéressons à la génération automatique de vues à partir de données abstraites dans le cas particulier où les informations abstraites sont modélisables par des graphes multi-valués. Notre objectif est de rendre ce processus entièrement paramétrable par un utilisateur final tout en rendant possible un large choix de représentations graphiques. Notre approche s'inspire du principe des CSS qui consacre la séparation entre fond (document HTML) et forme (CSS) pour les documents. Nous proposons ainsi GCSS, un langage dédié à la conception de feuilles de styles pour les graphes. Celles-ci adaptent, généralisent et étendent les concepts existants de codage graphique, d'héritage de propriétés ou encore de sélection. Elles introduisent en outre des notions spécifiques telles que les notions de structures graphiques permettant d'atteindre un niveau d'expression plus général que celui des CSS.

MOTS CLES : CSS, visualisation.

ABSTRACT
This paper describes GCSS a language devoted to the definition of graph-based visualizations. GCSS pays particular attention to the specification and control by the end-user of three important aspects of the automatic generation of such visualizations: coding, selection, and cascading.

CATEGORIES AND SUBJECT DESCRIPTORS: H5.m. Information interfaces and presentation (e.g., HCI): Miscellaneous.

GENERAL TERMS: Documentation.

KEYWORDS: CSS, visualisation.

INTRODUCTION

Séparer les données de leurs représentations graphiques est un enjeu important. L'utilisation de feuilles de styles est une façon d'effectuer cette séparation. Cela nous permet une réutilisation des données dans d'autres contextes. Les données n'étant pas modifiées. La feuille de style permet de construire un thème graphique utilisable pour d'autres données. Dans cet article, trois mécanismes sont importants : le codage graphique, la sélection et l'héritage.

- Le codage graphique est le processus permettant d'associer les attributs graphiques (couleur, taille, forme, etc) aux attributs des données (date, importance, etc). Dans le cadre de la visualisation de graphes, on peut par exemple vouloir faire en sorte que l'épaisseur des liens dans une représentation de type diagramme soit proportionnelle aux poids de ceux-ci.
- La sélection correspond au mécanisme permettant d'associer une représentation graphique, à un sous ensemble d'éléments. Par exemple, on peut souhaiter affecter une représentation graphique particulière à tous les nœuds ayant pour valeur d'un attribut date, une valeur supérieure à 2007.
- L'héritage est le mécanisme permettant à une représentation graphique d'hériter des propriétés d'une autre représentation graphique, puis, d'étendre ou de redéfinir ces propriétés. Ce principe d'héritage est connu sous le nom de « cascading » dans le langage CSS [9].

Dans cet article, nous proposons de définir un langage pour la création de feuilles de styles nommé GCSS (Graphe et CSS). Ce langage exploite les trois concepts exposés précédemment (codage, sélection, héritage) pour permettre à un utilisateur final de contrôler au mieux le mécanisme automatique de génération de représentations graphiques pour des informations modélisées par des graphes. Dans un premier temps nous exposons rapidement les travaux précédents concernant la séparation des données structurelles de leurs représentations graphiques, dans le cadre de la visualisation de données, mais aussi dans le cadre de la visualisation de graphes. Nous exposons dans un second temps, notre contribution, c'est-à-dire le langage proposé pour la représentation visuelle de graphes multivalués. Pour finir nous ferons un bilan sur la contribution apportée ainsi que les futurs travaux envisagés.

ETAT DE L'ART

Nous nous intéressons aux trois concepts introduits dans la section précédente. Nous divisons donc cette section en trois parties : systèmes et outils réalisant le mécanisme de sélection, ceux réalisant le mécanisme de codage graphique, et pour finir ceux réalisant de l'héritage.

Codage graphique

Dans [5,10,12,9,3] les auteurs proposent différentes méthodes pour transformer des tuples en éléments

graphiques en se basant sur les attributs des tuples. Dans [14] une application est proposée pour le positionnement des nœuds en fonction de leurs attributs. Les auteurs de [4,13] proposent d'utiliser des algorithmes comme dans [8], pour l'appariement de valeur d'un ensemble continu ou discret avec des attributs graphiques comme la couleur. Dans [1] il est proposé une application proposant un lien entre attributs structurels et attributs graphiques. Dans toutes ces approches, le problème reste celui du contrôle par l'utilisateur final des mécanismes d'appariement entre attributs graphiques et attributs visuels.

Sélection

Le langage DOT [6] est un outil très utilisé pour la visualisation de graphes. Le langage permet de définir la structure d'un graphe, puis d'associer une représentation graphique sur chaque sommet ou sur chaque arête. La description structurelle du graphe n'est pas dissociée de sa description graphique. Ainsi, il est nécessaire de dupliquer le code pour affecter à deux sommets (resp. arêtes) une même représentation graphique. Dans [2] il est question d'une application permettant d'associer à un sous-ensemble de sommets (resp . arêtes) une représentation graphique. L'application fonctionne par envoi de requêtes, la sélection de sommets ou d'arêtes peut se faire sur des propriétés topologiques du graphe ou sur des propriétés communes. Dans [9], les auteurs proposent d'associer à chaque balise HTML une représentation graphique, la sélection s'effectue soit sur les types de balises, soit sur les classes des balises, préalablement définis par l'utilisateur. Un mécanisme de sélection très complet (par propriétés, moteur d'expression, évènements) est également présent dans Prefuse [7] au travers du mécanisme des actions.

Héritage.

Les auteurs de [9] proposent une implémentation de l'héritage nommé « cascading » dans le cadre de feuille de styles pour les documents HTML. Une balise HTML associée à plusieurs représentations se voit affecter d'une représentation héritant des propriétés graphiques des représentations graphiques associées.

Notre contribution se situe clairement dans les trois sections ci-dessus. Notre mécanisme de sélection se fait au travers d'expression booléennes ou fonctions systèmes. Ce mécanisme est plus expressif que dans les CSS dans lesquelles le mécanisme de sélection repose essentiellement sur la structure du document, la définition de classes et d'identifiant.

La représentation graphique des différents éléments est basée sur un concept nommé structure graphique. Chaque structure graphique possède des attributs graphiques pouvant eux-mêmes être affectés d'une liste de valeurs. Nous verrons que c'est à ce niveau là que notre mécanisme de codage graphique intervient. Les feuilles de style que nous proposons vont au délà du simple outil de « décoration » (changement de formes, ou de couleurs) les structures pouvant se composer entre-elles. Quant au mécanisme d'héritage, il est directement inspiré des CSS [9] mais, augmenté pour permettre l'intégration de structures graphiques plus générales.

LE LANGAGE GCSS

Notre contribution est un langage pour la création de feuilles de styles appliquée aux graphes. Cette section se divise en trois parties. La première partie décrit comment définir une représentation graphique, ainsi que le mécanise de codage graphique. La seconde partie traite du mécanisme de sélection. Nous terminons dans la dernière partie par le mécanisme d'héritage. Tout au long de cette section nous nous appuierons sur un exemple issu de l'implémentation de nos feuilles de styles. Cet exemple est spécialement conçu pour une bonne compréhension des concepts abordés dans cet article. La Fig. 1 donne un exemple de graphe multivalué. Les sommets A, C, E et G représentent des visiteurs. Les sommets B, D et F des sites internet. Un visiteur visite une page internet durant un certain temps, cette visite est représentée par un lien. Un site web peut référencer un autre site internet, cette référence ce traduit elle aussi par un lien. La Fig. 2 est le résultat de l'application de feuille de style Fig. 3 sur le graphe de la figure 1.

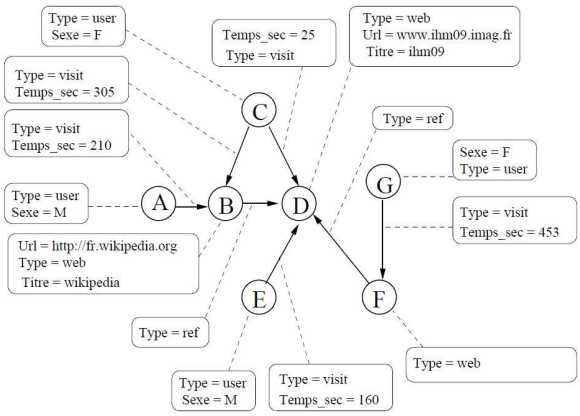

Figure 1 : un exemple de graphe multi-valué

Définition de la représentation graphique.

Dans GCSS, les feuilles de styles se découpent en deux sections notées R et A sur la Fig. 3. Les représentations graphiques sont définies dans la section R. Elles sont marquées de R1 à R8. Une représentation est définie par un nom, un lien vers le type de l'élément auquel elle s'applique ainsi qu'une liste de structures graphiques. Chaque structure graphique possède une liste non vide d'attributs graphiques pouvant eux-mêmes être affectés d'une liste non vide de valeurs. Par exemple, dans la Fig. 3 la représentation graphique notée R3 se nomme « Utilisateur » elle est associée aux éléments de type nœuds et possède deux structures graphiques : un polygone et un cercle. Le polygone possède un attribut

noté « points » affecté à une liste de valeurs, qui deux à deux définissent une liste de coordonnées du polygone.

Le codage graphique s'effectue par le biais des attributs. Ils peuvent prendre des valeurs fixes (entiers, réels, chaine de caractère, nœud et arête), des valeurs dynamiques (attributs structurels, fonctions paramétrées), des valeurs calculées (expressions, opérations). La représentation graphique R8 montre un cas d'appariement d'un attribut visuel avec un attribut structurel. Ici l'épaisseur du lien est définie comme étant un dixième de l'attribut « temps_sec » qui correspond au temps qu'un utilisateur a passé sur la page internet.

Dans les contributions étudiées dans la section précédente, les liens entre les sommets sont dessinés de la représentation graphique de la source du lien vers la représentation de la cible du lien. Se pose généralement la question du point d'ancrage du lien que ce soit sur la cible ou sur la source. Ce point d'ancrage dépend non seulement de la forme des structures graphiques représentants les objets source et cible, il dépend également des préférences de l'utilisateur final. Afin de donner un contrôle fin sur la définition du point d'ancrage nous avons donc introduit la notion d'ancre. Toute structure graphique peut être une ancre. Lorsque c'est une ancre, comme par exemple la structure 'oval' figure 3(R3), la structure graphique possède un attribut nommé « anchor_name » , définissant le nom de cette ancre. Le nom de l'ancre est ensuite utilisé dans les structures graphiques représentant les arêtes pour définir précisément les points d'ancrage sur la source et la cible comme par exemple dans la représentation 'Lien'.

Mécanisme de sélection

Le mécanisme de sélection est effectué dans la section marquée A de la figure 3. Elle se compose d'une liste d'associations. Une association contient une expression booléenne ainsi que la représentation graphique qui correspond au cas où l'expresssion booléenne est valide. L'expression booléenne peut tester la valeur d'un attribut des données (par le biais des opérateurs =, >, <), l'existence d'un attribut ou faire appel à n'importe quelle fonction système retournant une valeur booléenne. Ainsi, dans l'exemple de la figure 3, on définit une arête de type visite à partir d'une arête pour laquelle la source est un utilisateur et la cible est une page web.

Mécanisme d'héritage

Un nœud (ou arête) peut être associé à plusieurs représentations graphiques. Prenons l'exemple du nœud D dans la figure 1. Ce nœud est associé en premier à la représentation graphique nommée 'Web' puis en second à la représentation nommée 'ImageWeb'. La représentation 'Web' contient une structure image nommée 'web_icon'. La représentation nommée 'ImageWeb' contient également une image portant le même nom. Lorsque la représentation graphique du nœud D est calculée, c'est l'image correspondant à l'image de la représentation graphique ImageWeb qui sera utilisée. Elle se substituera ainsi à l'image autrement utilisée dans la représentation graphique 'Web'. En revanche le reste des propriétés de 'Web' non redéfini dans 'ImageWeb' sera hérité directement. Par exemple, les attributs de position définis dans 'Web' et non redéfinis dans 'ImageWeb' seront conservés. Ce mécanisme est conforme au principe de cascading des CSS et se rapproche du mécanisme d'héritage typique des langages de programmation orientée objet. Cependant, ce mécanisme de cascading est étendu à nos représentations à base de structures graphiques. En conservant le même exemple, on peut remarquer que la structure 'TextBox' n'est pas définie dans la représentation 'Web'. Cette structure est donc ajoutée à la représentation finale de notre nœud D. De manière analogue, les nœuds A et E sont associés aux représentations 'utilisateur' et 'masculin' alors que les nœuds C et G sont associés aux représentations 'utilisateur' et 'féminin'. Si l'on s'intéresse de plus prêt à la définition des représentations masculin et féminin on s'aperçoit qu'elles ne définissent que la couleur du cercle et du polygone définis dans la représentation 'utilisateur' le reste des attributs de cercles et de polygone sont directement définis et hérités de la représentation 'utilisateur'. C'est ainsi dans utilisateur qu'est définie la forme du bonhomme qui représente un utilisateur indépendamment de sa couleur.

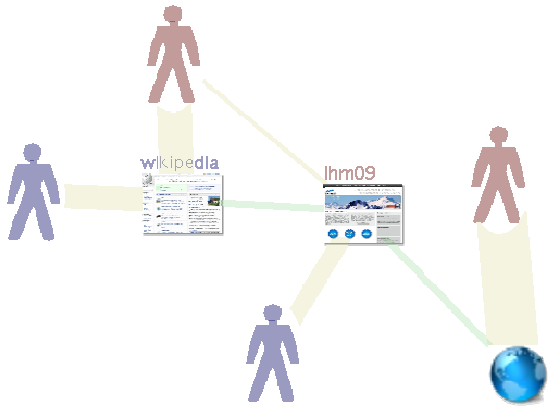

Figure 2 : Représentation du graphe après application de la feuille de style de la Figure 3.

CONCLUSION

Nous avons présenté un langage permettant à un utilisateur de contrôler finement les caractéristiques graphiques d'une visualisation de données modélisables par des graphes multi-valués. Cette représentation s'inspire des CSS et de la séparation entre fond et forme. Nous avons montré qu'il convenait néanmoins de redéfinir les principaux mécanismes (sélection, codage, et cascading) et d'introduire la notion de représentation et structure graphique pour donner la généricité nécéssaire dans le contexte des graphes. Nous avons

également introduit et intégré la notion d'ancre pour permettre un contrôle fin sur les points d'ancrage des arêtes. Avec GCSS, il est possible de construire des représentations graphiques différentes du même graphe et/ou de réutiliser les mêmes représentations graphiques pour des graphes différents. L'usage de GCSS dans un projet de visualisation collaborative a récemment permis de faciliter les ajustements de codage. Comme pour l'écriture de CSS, l'écriture de GCSS peut néanmoins se révéler fastidieuse et l'étape suivante consiste donc à réaliser une interface graphique facilitant la conception de GCSS.

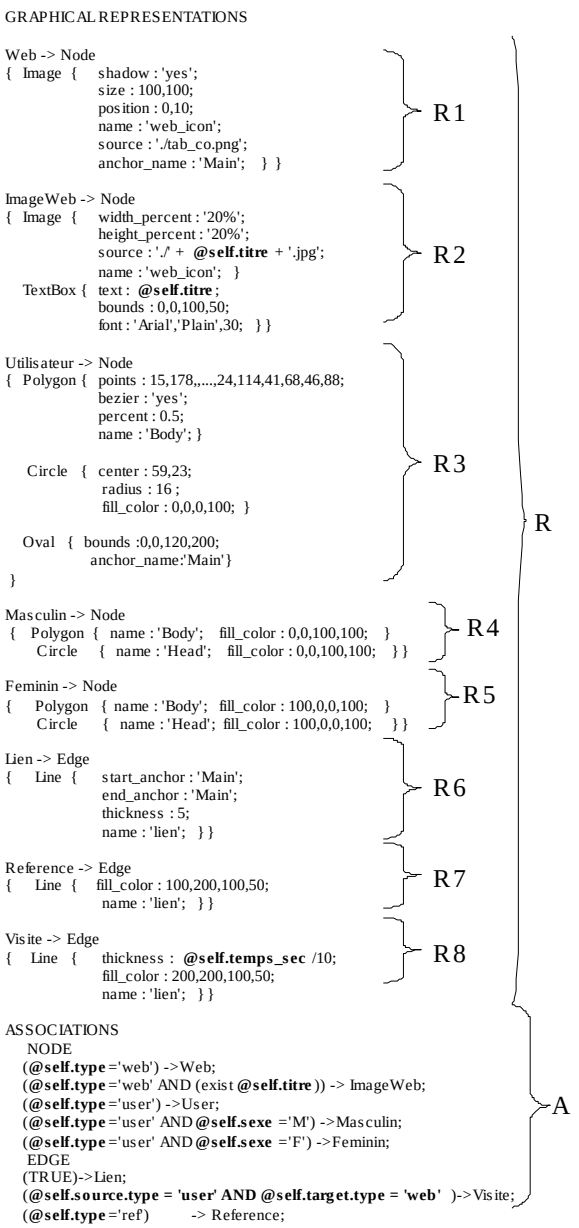

Figure 3 : Exemple de feuille de style en GCSS

BIBLIOGRAPHIE

1. http://www.netminer.com/.
2. Adar, E. Guess: a language and interface for graph exploration. In *CHI '06: Proceedings of the SIGCHI conference on Human Factors in computing systems*, pages 791–800, New York, NY, USA, 2006. ACM.
3. Aiken, A., Chen, J., Stonebraker,M., andWoodruff, A. Tioga-2: A direct manipulation database visualization environment. pages 208–217, 1996.
4. Auber, D. Tulip : A huge graph visualisation framework. In Mutzel, P., and J¨unger, M., editors, *Graph Drawing Softwares*, Mathematics and Visualization, pages 105–126. Springer-Verlag, 2003.
5. Bosch, R., Stolte, C., Tang, D., Gerth, J., Rosenblum, M., and Hanrahan, P. Rivet: a flexible environment for computer systems visualization. *SIGGRAPH Comput. Graph.*, 34(1):68–73, 2000.
6. Gansner, E. R., and North, S. C. An open graph visualization system and its applications to software engineering. *Softw. Pract. Exper.*, 30(11):1203–1233, 2000.
7. Heer, Jeffrey, Card, S., Landay, James A., *Prefuse: a toolkit for interactive information visualization*, in CHI 2005, Human Factors in Computing Systems (2005)
8. Herman, I.,Marshall,M., andMelanc¸on, G. Density functions for visual attributes and effective partitioning in graph visualization. In Roth, S., and Keim, D., editors, *Proceedings of the IEEE Information Visualization Symposium 2000*, pages 49–56. IEEE CS Press, 2000.
9. Lie, H. *Cascading Style Sheets: Designing for the Web*. Addison-Wesley Educational Publishers Inc, May 2005.
10. Livny, M., Ramakrishnan, R., Beyer, K., Chen, G., Donjerkovic, D., Lawande, S., Myllymaki, J., and Wenger, K. Devise: Integrated querying and visual exploration of large datasets (demo abstract). In *In Proceedings of ACM SIGMOD*, pages 301–312, 1997.
11. Mackinlay, J. Automating the design of graphical presentations of relational information. *ACM Trans. Graph.*, 5(2):110–141, 1986.
12. Roth, S. F., Lucas, P., Senn, J. A., Gomberg, C. C., Burks, M. B., Stroffolino, P. J., Kolojejchick, J. A., and Dunmire, C. Visage: A user interface environment for exploring information. In *Proc. Information Visualization, IEEE*, pages 3–12, 1996.
13. Shannon, P., Markiel, A., Ozier, O., Baliga, N. S., Wang, J. T., Ramage, D., Amin, N., Schwikowski, B., and Ideker, T. Cytoscape: a software environment for integrated models of biomolecular interaction networks. *Genome Res*, 13(11):2498–2504, November 2003.
14. Wattenberg, M. Visual exploration of multivariate graphs. In *CHI '06: Proceedings of the SIGCHI conference on Human Factors in computing systems*, pages 811–819, New York, NY, USA, 2006.