Extraction automatique de modèles de processus pour l'étude de la résolution collective de problèmes dans les communautés du logiciel libre

Gabriel Ripoche [1,2]
gripoche@uiuc.edu

Les Gasser [1]
gasser@uiuc.edu

[1] Graduate School of Library and Information Science
University of Illinois at Urbana-Champaign
501 E. Daniel St., Champaign, IL 61820, USA
Téléphone: +1 217 265 5021 / Fax: +1 217 244 3302

[2] LIMSI-CNRS / Université Paris XI
BP 133, 91403 Orsay Cedex, France
Téléphone: +33 1 69 85 81 01 / Fax: +33 1 69 85 80 88

Résumé

Dans cet article nous présentons notre méthode d'analyse des aspects socio-techniques de la gestion de problèmes logiciels dans les communautés du logiciel libre (F/OSS : *Free/Open-Source Software*), à partir de bases de données de rapports de problèmes telles que celles du système de gestion de bugs Bugzilla ; et rapportons quelques résultats préliminaires obtenus en appliquant cette méthode. Etant donnée la quantité de données disponibles, il est nécessaire de recourir à des techniques automatisées pour extraire les informations requises par nos analyses et pour modéliser le comportement des collectifs étudiés. Nous utilisons différentes techniques qui associent des analyses qualitatives manuelles et des techniques automatiques d'extraction et de modélisation pour caractériser et analyser les processus impliqués dans la gestion de problèmes logiciels.

Mots clés

Inférence et modélisation de processus ; traitement du langage à partir de corpus ; extraction de motifs linguistiques ; gestion collective et distribuée de problèmes.

1 Introduction

La tâche d'identification et de résolution d'erreurs a toujours joué un rôle important dans la production et la maintenance de logiciels. Un logiciel « buggé » est au mieux irritant pour l'utilisateur, et dans le pire des cas peut avoir des conséquences désastreuses sur la sécurité, sur l'économie ou même mettre en danger des vies humaines. Alors que ces systèmes logiciels complexes se multiplient et deviennent incontournables, le coût social des erreurs logicielles et autres « bugs » risque d'augmenter. De plus, dans le paradigme émergeant de la conception continue (*continuous design*) incarné par le modèle du Free/Open-Source Software (F/OSS, le logiciel libre), la gestion des bugs joue un rôle majeur dans la conception des logiciels et dans leur évolution [6]. De ce fait, les bugs offrent un point de vue sur le plus vaste puzzle organisationnel du F/OSS.

Parce que les erreurs et bugs réduisent l'efficacité avec laquelle les systèmes logiciels sont construits, et parce qu'ils jouent un rôle de plus en plus important dans les processus de développement actuels, nous tentons de comprendre comment s'effectue la découverte de problèmes, comment les bugs peuvent être gérés de façon optimale et comment les personnes qui développent et utilisent des systèmes logiciels peuvent organiser leur travail pour éviter, accommoder, corriger, voire même bénéficier des problèmes rencontrés.

La plupart des approches concernant les problèmes logiciels se concentrent sur les failles techniques de conception et d'ergonomie. Il est évident que l'amélioration des phases de conception, de prototypage et d'analyse des besoins joue un rôle important dans ce contexte, mais le problème ne se limite pas à ces critères. Plus spécifiquement, la qualité d'un artefact logiciel dépend de la structure des processus techniques et organisationnels employés pour le construire et le maintenir [2]. Notre recherche étudie ces structures et leurs contraintes en s'interrogeant sur des questions telles que :

- Comment d'importantes communautés de développeurs (re)conçoivent et (re)développent-elles de façon continue leur(s) logiciel(s) sur de grandes durées ?
- Comment ces communautés gèrent-elles des flux continus de problèmes logiciels ?
- Comment ces communautés capturent-elles, gèrent-elles, et utilisent-elles leurs connaissances collectives pour faciliter la maintenance de leur(s) logiciel(s) ?

La majeure partie des interactions dans les projets F/OSS se déroulant sur Internet (forums, IRC, systèmes collaboratifs, etc.), de très grandes quantités de données sur le fonctionnement des projets F/OSS et des communautés impliquées sont disponibles et aisément accessibles en ligne. Nous utilisons ces données comme base empirique pour aborder les questions suivantes, qui apparaissent dans les grandes communautés F/OSS, et dans une certaine mesure dans tout projet logiciel de grande envergure :

- Comment un groupe d'individus passe-t-il collectivement d'un ensemble de rapports de bugs, à un ensemble de bugs, puis à un ensemble de corrections ?
- Etant donné un nombre important de bugs simultanément actifs, comment une communauté décide-t-elle d'un plan d'action (et comment le réalise-t-elle) ?
- Pourquoi certains bugs persistent-ils pendant très longtemps alors que d'autres sont résolus rapidement ?

2 Données étudiées

Avec plusieurs centaines de milliers de rapports de problèmes provenant d'une variété de projets F/OSS, les systèmes de gestion de bugs offrent une source de données extrêmement vaste et variée pour analyser des questions telles que celles que nous avons énoncé ci-dessus. Bon nombre de ces systèmes sont utilisés depuis plusieurs années et contiennent des données permettant une analyse longitudinale du processus de résolution de problèmes et de son évolution. Notre étude actuelle se concentre sur des données provenant de Bugzilla [11], le système de gestion de bugs du projet Mozilla [10].

Le projet Mozilla a démarré en 1998—après la décision de Netscape de rendre publique le code source de la suite logicielle Communicator—avec pour objectif principal de développer un navigateur Internet libre, standard et performant. Depuis ses débuts, plusieurs milliers de programmeurs ont contribué du code, des dizaines de milliers d'utilisateurs ont soumis des rapports de bugs, et des millions de copies du logiciel ont été téléchargées.

Notre capture de la base de donnée de Bugzilla—effectuée en mars 2002—contient plus de 128 000 rapports de bugs et plus de 1,2 million de commentaires résultant de la participation de plus de 45 000 utilisateurs enregistrés.

3 Méthode

D'autres études basées sur des analyses quantitatives et concernant le temps de résolution des problèmes dans les projets logiciels de grande dimension ont déjà été menées, comme par exemple les travaux d'Herbsleb et Mockus [8]. Cependant, des analyses plus approfondies sont nécessaires et leur réalisation repose en particulier

sur une conceptualisation plus détaillée des processus de résolution de bugs ; la caractérisation de ces processus dépendant quant à elle principalement sur des analyses qualitatives. Etant donnée la quantité de données disponibles dans Bugzilla, ces études qualitatives doivent dépasser les méthodes manuelles traditionnelles et employer des techniques automatiques pour extraire et modéliser, à très grande échelle, les comportements du collectif. Pour extraire les instances d'évènements et de processus nécessaires à nos modèles, nous utilisons des techniques associant des annotations manuelles qualitatives à des techniques d'apprentissage et d'analyse automatiques [4,5].

3.1 Modélisation des processus

Notre modèle repose sur la méthode de modélisation de processus présentée par Cook et Wolf [1] (voir figure 1), que nous avons étendue pour intégrer les notions de modèle causal et de paramètres explicatifs. Nous avons développé des outils d'analyse qui utilisent les informations contenues dans Bugzilla pour créer des réseaux probabilistes sous la forme de modèles de Markov et d'automates probabilistes à états finis [1].

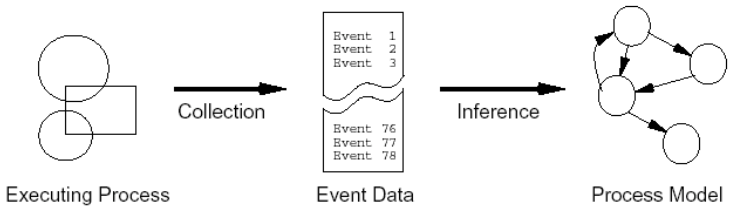

Figure 1 - Méthode d'inférence de modèles à partir de processus en exécution (Cook and Wolf).

Par exemple, nous avons utilisé cette technique pour modéliser une partie des informations concernant les changements de statut des bugs dans Bugzilla. Nous nous sommes basés sur la « trajectoire » (succession de statut d'un bug) de 88 000 rapports de bugs résolus, que nous avons codé utilisant un alphabet attribuant une lettre à chaque statut (par exemple : 'R' pour '*Resolved*') et un format permettant l'encodage des transitions d'un statut à l'autre (la figure 2 montre des exemples de trajectoires). La génération d'un modèle probabiliste (automate à états finis) basé sur le calcul des fréquences d'occurrence de chacune de ces trajectoires dans le corpus Bugzilla nous a permis d'obtenir un modèle du cycle de vie d'un rapport de bug (voir figures 3 et 4). Nous sommes actuellement en train d'évoluer vers des modèles d'ordre *n* (chaînes de Markov) afin d'éviter les limitations des modèles d'ordre 1 (pas de prise en compte de l'historique d'un processus) et d'améliorer les capacités explicatives (et éventuellement prédictives) du modèle.

```
# freq / process
 11037   ZZNZ-NZRD-RDVD
 10847   ZZUZ-UZRD-RDVD
 10510   ZZNZ-NZAZ-AZRF-RFVF
         [...]
    16   ZZNZ-NZAZ-AZNZ-NZAZ-AZRF-RFOF-OFOZ-OZRF-RFVF
         [...]
     2   ZZNZ-NZRF-RFOF-OFRF-RFOF-OFAF-AFRF-RFOF-OFOZ-OZRF-RFVF
         [...]
```

Figure 2 - Exemples de « trajectoires » de statut de rapports de bug.

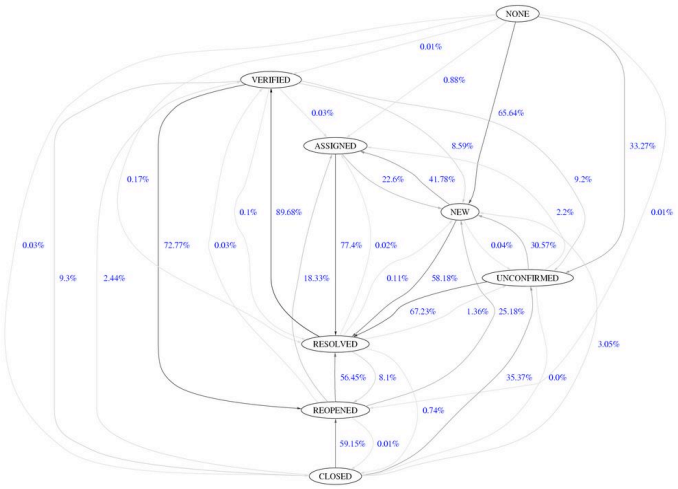

Figure 3 - Modèle probabiliste intégrant uniquement la variable 'Status'.

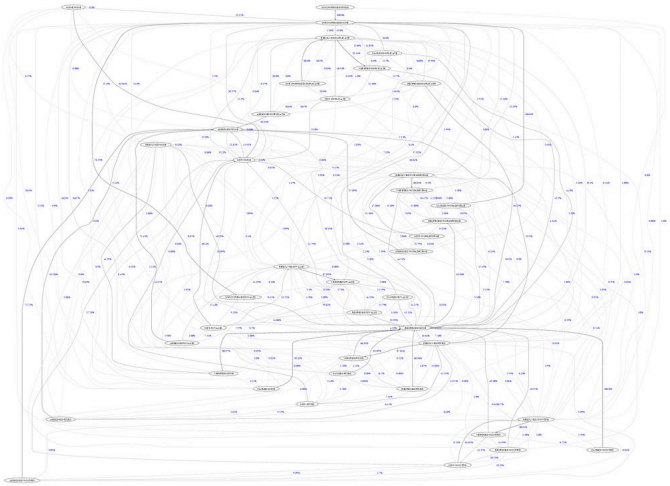

Figure 4 - Modèle probabiliste intégrant deux variables ('Status' + 'Resolution type').

La structure de ces modèles probabilistes nous fournit de solides fondations statistiques pour la compréhension du processus de résolution de bugs, et pour relier les décisions effectuées au cours de ce processus aux résultats de ce dernier. Cependant, dans le modèle de la figure 3, nous n'avons utilisé qu'un seul type—ou « dimension »—de données (les changements de statut). Dans la plupart des cas, un modèle aussi simple ne suffira pas à expliquer ou à prévoir les trajectoires d'un processus de résolution. Il est donc nécessaire d'intégrer d'autres aspects de ces processus.

3.2 Expliquer les variations de trajectoires

Au risque de simplifier, considérons l'activité de gestion de bugs comme un type (probablement très complexe) d'action heuristique. Chaque processus en jeu est composé d'une séquence d'états (tels qu'illustrés dans les figures 2, 3, 4) et chaque état possède un historique des états précédents dans lequel un processus s'est trouvé (états d'entrée) ainsi qu'un ensemble d'états futurs possibles (états de sortie). La figure 5 résume cette configuration. Dans ce contexte, la question devient : quels sont les paramètres explicatifs d'un processus qui vont, à partir d'un état donné, causer une transition vers un état de sortie spécifique ?

Pour mettre en évidence ces paramètres, il est nécessaire de prendre en compte beaucoup plus de types d'information concernant un processus que le seul statut formel d'un rapport de bug. Par exemple, la figure 4 présente un modèle résultant de l'ajout d'une autre dimension ('*Resolution type*') au modèle contenant uniquement la composante '*Status*'. Ce nouveau modèle possède beaucoup plus d'états et une plus grande précision est obtenue concernant les transitions entre états. D'autres dimensions potentiellement intéressantes que nous envisageons d'étudier sont par exemple : temps écoulé entre changements d'états, type de problème, présence/absence de certains processus sociaux (voir plus loin), et présence/absence de types d'information spécifiques.

Le concept de paramètre explicatif peut être illustré de la façon suivante : si, par exemple, un processus atteint l'état '*Resolved*', nous voulons savoir si la présence de certains types de processus sociaux ou d'informations est associée de manière consistante à la transition de ce processus vers l'état '*Reopened*'. Ces paramètres explicatifs devraient nous permettre d'obtenir un modèle causal du processus de résolution de problèmes.

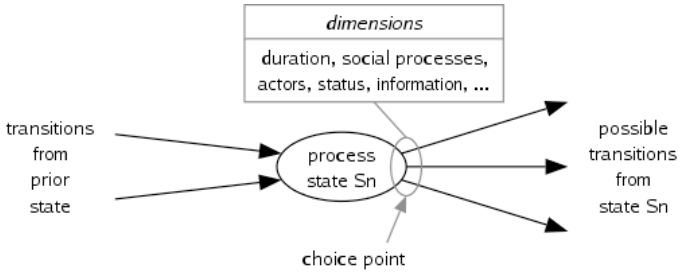

Figure 5 - Point de choix dans une trajectoire.

3.3 Computational Amplification

Cependant, une large partie des dimensions que nous souhaitons intégrer dans nos modèles n'est pas représentée et codée directement dans le corpus, comme le sont les dimensions '*Status*' et '*Resolution type*'. En effet, une grande part des interactions se déroulant dans Bugzilla—et plus généralement dans les communautés F/OSS—s'effectue sous la forme d'échanges textuels en langue naturelle (ce que nous appelons données informelles). Ces échanges capturent des données clés à la représentation de certaines dimensions, mais seulement de manière implicite à travers le langage. Pour pouvoir utiliser ces sources d'information, nous concevons des outils de traitement du langage pour automatiser l'annotation et l'extraction des ces données.

Nous développons actuellement des techniques basées sur la génération semi-supervisée de motifs d'extraction [7,12,13,14,15] et sur l'analyse de discours [9] qui ont pour but d'identifier des instances spécifiques caractéristiques de certains processus, tels que les processus liés à l'acquisition et l'utilisation d'information ou certains processus sociaux fondamentaux. Par exemple, figure 6 montre une ébauche du premier niveau de notre taxonomie de *Basic Social Processes* (BSPs—processus sociaux fondamentaux), élaborée à partir d'analyses qualitatives manuelles.

Ambiguity	Description	Rationale
Articulation	Evaluation	Statement
Conflict	Negotiation	Uncertainty

Figure 6 - Niveau supérieur de notre taxonomie de « Basic Social Processes ».

A partir d'une telle taxonomie et d'une série de motifs linguistiques élaborés à partir d'un échantillon de rapports annotés, nous sommes parvenus à extraire et coder des instances potentielles de BSPs sur l'intégralité de notre corpus. Ces annotations automatiquement générées ont été analysées et vérifiées par les annotateurs de l'échantillon et les motifs utilisés pour l'extraction automatique ont été modifiés selon leur efficacité. La répétition de ce processus permet de sélectionner des motifs d'extraction spécialisés pour un type de BSP.

Nos premières évaluations[1] indiquent un taux de précision de l'ordre de 80% pour les BSPs '*conflict*' et '*evaluation*'. Ces outils nous ont permis de filtrer de manière automatique d'importantes quantités de données et d'identifier des instances potentielles pour un certain nombre de concepts. A partir de ces données, nous avons pu obtenir des informations statistiques concernant les processus sociaux présents, et nous sommes en train d'intégrer ces données dans nos modèles.

3.4 Etude d'un cas particulier : la durée de vie des bugs

Nous avons mis en pratique la méthode élaborée dans cet article en nous penchant sur la question de la durée de vie des bugs [3]. Des travaux antérieurs concernant la gestion des erreurs logicielles ont montré que l'existence de « désalignements » dans des réseaux d'activité distribués augmentaient la difficulté de résolution des problèmes émergeant de cette activité [2]. Dans le contexte de Bugzilla, une façon de concevoir cette observation est de considérer que ces « désalignements » risquent de conduire à des temps de résolution plus longs.

Notre étude est basée sur un échantillon comparatif de 159 rapports de bugs contenant 59 rapports à « longue durée de vie » (avec des temps de résolution supérieurs à 1000 jours) et 100 rapports à « courte durée de vie » (avec des temps de résolution égaux à 30 jours). L'annotation qualitative manuelle de cet échantillon nous à indiqué que les rapports à longue durée de vie avaient tendance à contenir une plus grande proportion d'épisodes caractérisant les BSPs '*conflict*', '*ambiguity*', et '*uncertainty*'—et donc plus de « désalignements »—que les rapports de courte durée.

Ces observations nous ont amené à formuler l'hypothèse que le temps de résolution d'un rapport de bug dépend (au moins en partie) du temps mis pour atteindre un consensus (*time to consensus*) sur plusieurs aspects centraux à la définition même d'un bug. En d'autres termes, le temps de résolution dépend du temps pris pour aligner ces différents aspects.

Bien sûr, d'autres contraintes telles que l'allocation de ressources ou la définition de priorités ont un impact sur le temps de résolution. Cependant, ceci ne contredit pas notre hypothèse et notre modèle permet d'ajouter ultérieurement ces autres dimensions afin d'approfondir d'avantage notre connaissance des processus de résolution.

[1] Nous n'avons pas encore évalué le taux rappel de nos motifs d'extraction car cela nécessite l'annotation d'un autre échantillon—tâche relativement longue que nous n'avons pas encore achevé.

4 Projets en cours et futurs

Dans la section précédente nous avons donné un résumé succinct de notre méthode d'analyse à grande échelle des processus de gestion de problèmes. Notre recherche actuelle s'applique à raffiner les outils utilisés dans les différentes étapes de cette méthode et à les mettre à l'œuvre en prenant en compte un plus grand nombre de dimensions et de contextes.

Une des difficultés, que nous avons déjà brièvement mentionnée, concerne le besoin d'échantillons annotés pour amorcer nos mécanismes d'extraction. Bien qu'il ne soit pas possible de totalement éviter ce problème, nous étudions des techniques plus autonomes, largement inspirés des orientations actuelles dans le domaine de l'extraction d'information [14,15,12], afin de réduire l'effort manuel nécessaire à nos études.

Nous travaillons également sur des modèles de processus plus élaborés permettant de prendre en compte plus de paramètres et d'augmenter ainsi leur capacité explicative. Par exemple, nos modèles actuels sont basés sur des automates à états finis qui ne prennent pas en compte l'historique d'un processus. Plus récemment nous avons expérimenté avec des modèles utilisant des chaînes de Markov d'ordre *n* [1] et d'autres méthodes stochastiques sont également à l'étude.

Par ailleurs, l'importance de Bugzilla comme outil de coordination dans la communauté Mozilla fait que notre corpus contient une grande partie de l'activité liée à la gestion de problèmes dans cette communauté. Cependant, il serait intéressant de prendre en compte d'autres archives du projet (telles que les listes de diffusion, les « *minutes* », les archives CVS, etc.), à la fois pour compléter notre perception de la gestion de problèmes au sein de Mozilla et pour élargir nos investigations vers la question plus générale de la conception continue dans les communautés F/OSS.

Enfin, bien que cet article se soit concentré principalement sur les méthodes que nous développons pour automatiser nos études, nous n'ignorons pas la nécessité de mener des analyses qualitatives manuelles à partir d'échantillons tirés du corpus pour de caractériser de façon précise les dimensions que nous intégrons dans nos modèles.

5 Conclusions

Nous avons présenté notre méthode générique d'analyse des processus de résolution de problèmes basée sur l'extraction automatique de différents aspects permettant de caractériser ces processus et sur la construction de modèles explicatifs reposant sur les interactions entre ces dimensions. Nous avons appliqué cette technique à la question de la durée de vie d'un bug et avons obtenu des résultats préliminaires encourageants. La possibilité de manipuler de gros volumes de données (par rapport à des études qualitatives traditionnelles), les différentes techniques d'extraction de données ainsi que la modélisation de multiples dimensions rendent cette méthode bien adaptée à l'analyse complexe de processus de développement logiciel fortement distribués. Nos travaux actuels consistent à raffiner les différentes techniques d'extraction utilisées et à développer de meilleurs modèles afin de caractériser plus précisément les mécanismes socio-techniques du fonctionnement des communautés F/OSS.

6 Remerciements

Nous tenons à remercier Jean-Paul Sansonnet du LIMSI-CNRS et Bob Sandusky de GSLIS à l'Université d'Illinois à Urbana-Champaign (UIUC). Bryan Penne (GSLIS, UIUC) a également contribué à l'annotation des données. Ce travail est financé par le programme ITR de la National Science Foundation (Digital Society and Technologies) sous le financement n° 0205346. Les avis, résultats, conclusions et recommandations exprimés dans cet article sont ceux des auteurs et ne reflètent pas nécessairement les positions de la National Science Foundation.

7 Références

[1] Cook, J. E., Wolf, A., Automating Process Discovery through Event Data Analysis. In Proceedings of the 17th International Conference on Software Engineering, 1995.

[2] Gasser, L., The Social Organization of Errors in Computing Work. Working Paper LG-2000-13, Graduate School of Library and Information Science, University of Illinois, 2000.

[3] Gasser, L., Dubin, D., Penne, B., Ripoche, G., Sandusky, R., A Comparative Study of Resolution Times and Processes for Mozilla/Bugzilla Bug Reports. SQA Project Memo UIUC-2003-06. 22 February 2003.

[4] Gasser, L., Penne, B., Ripoche, G., and Sandusky, R., Computational Amplification of Qualitative Analysis: Mining Social Processes with Language Models. SQA Project Memo UIUC-2003-14. 25 April 2003.

[5] Gasser, L., Ripoche G., Building and Analyzing PFSAs as Process Representations, SQA Project Memo UIUC-2003-17. 20 May 2003.

[6] Gasser, L., Scacchi, W., Ripoche, G., Penne, B., Understanding Continuous Design in F/OSS. To appear in Proceedings of the 16th International Conference on Software & Systems Engineering and their Applications (ICSSEA-03). 2003.

[7] Grishman, R., Adaptive Information Extraction and Sublanguage Analysis. In Proceedings of the Workshop on Adaptive Text Extraction and Mining at the 17th International Joint Conference on Artificial Intelligence. 2001.

[8] Herbsleb, J. D., Mockus, A., An Empirical Study of Speed and Communication in Globally-Distributed Software Development. IEEE Transactions on Software Engineering, 29:6. June, 2003.

[9] Herring, S., Computer-Mediated Discourse Analysis: An Approach to Researching Online Behavior. In Designing for Virtual Communities in the Service of Learning, Barab S. A., Kling R., and Gray J. H. (Eds.). New York: Cambridge University Press. 2003.

[10] The Mozilla Project website. `http://www.mozilla.org/` (last visited: 10/13/2003).

[11] The Mozilla Project's Bugzilla repository. `http://bugzilla.mozilla.org/` (last visited: 10/13/2003).

[12] Poibeau, T., Dutoit, D., Generating Extraction Patterns from a Large Semantic Network and an Untagged Corpus. SemaNet Workshop at COLING'02. 2002

[13] Riloff, E., Automatically Generating Extraction Patterns from Untagged Text. In Proceedings of the 13th National Conference on Artificial Intelligence (AAAI-96), 1996.

[14] Riloff, E., Jones, R., Learning Dictionaries for Information Extraction by Multi-Level Bootstrapping. In proceedings of the 16th National Conference on Artificial Intelligence (AAAI-99), 1999.

[15] Yangarber, R., Grishman, R., Tapanainen, P., Huttunen, S., Unsupervised Discovery of Scenario-Level Patterns for Information Extraction, In Proceedings of the 6th Conference on Applied Natural Language Processing, 2000.