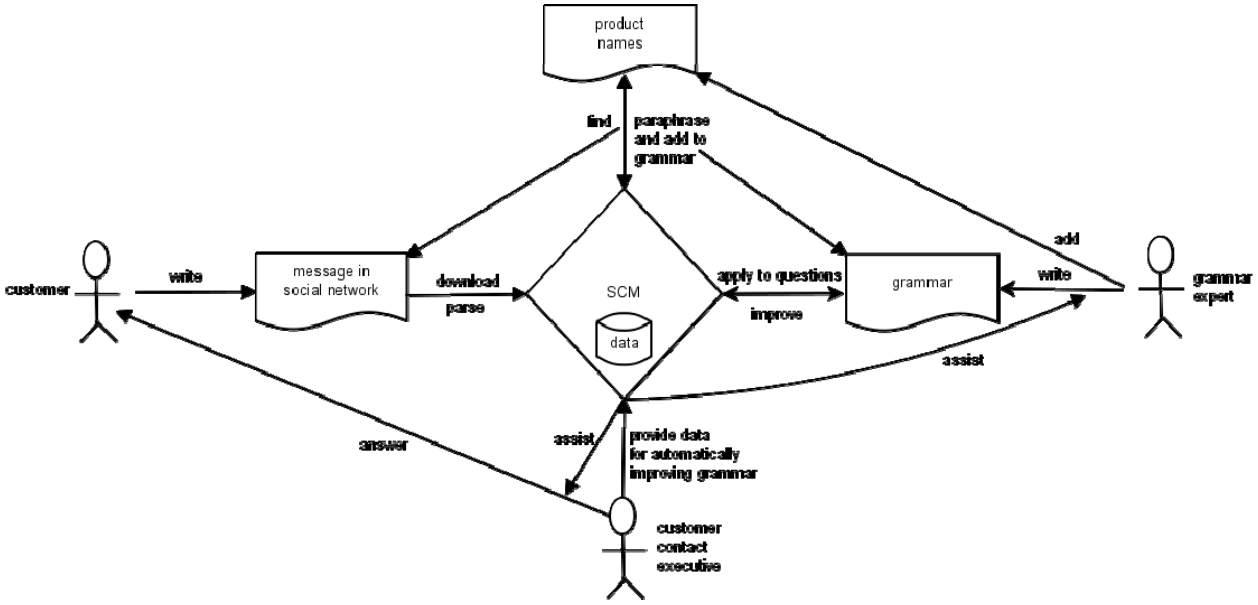

Figure 1. The basic design of SCM

3. System Overview

SCM is a web based monitoring and customer interaction management system that combines all technologies, data, software agents and human agents involved in the monitoring and customer interaction process, such as the technologies used for downloading, processing and storing data, the software agents used for classification and similar tasks and the human agents such as grammar experts and contact center agents. Figure 1 gives an overview of SCM's design: Customers post messages in social networks and other web sites. Messages typically concern products and/or services sold or provided by a company. Most messages state that a problem of a certain type occurred with a certain product or service. Some messages also contain expressions of praise or thank. SCM downloads all messages and stores them in a normalized format. It applies its grammar to the messages in order to recognize all product names in it and in order to assign a set of content tags (such as *hotline* or *delivery* or *malfunction*) and a sentiment tag (at the moment either *positive* or *negative*). (We will say immediately what we understand by a grammar.) The message, their tags and the list of product names are then forwarded to those contact center agents that are experts for the recognized type of problem. If a message has been assigned a wrong tag or no tag at all, contact center agents can assign a tag manually. They have to mark those parts of the message that justify its being assigned the tag. Thus, contact center agents unknowingly add new constraints to the grammar.

Actually, SCM does not only contain one grammar. It contains as many grammars as there are classifier objects. Each classifier has a (content or sentiment) tag, a set of positive and a set of negative constraints. (Figure 3 shows a simple content classifier.) If one of the positive constraints holds, the classifier's tag is assigned. If one of the negative constraints holds, the classifier's tag is not assigned, irrespective of the positive constraints. I.e. negative constraints can be used to override positive constraints.

Put simply, the constraints are regular expressions. Yet, as they can contain special terms that refer to other grammars (such as `\mobile_phone`), it would be more adequate to call them *Local Grammars* in the sense of [7]. Each instance of SCM comes with a predefined set of such special terms. (The idea is to tailor new instances of SCM to the needs of the companies or organizations that use it.) Grammar experts cannot add special terms or change their meaning. They just use them: If, e.g., a grammar line contains `foo \mobile_phone bar`, it will match *foo*, followed by a space character, followed by any variant of any mobile phone name known to the used instance of SCM, followed by a space, followed by *bar*.

4. Named Entity Paraphrasing

In our prototype named entities include mobile phones, chargers, headsets, batteries, operating systems, applications, tariffs, manufacturers and providers. All these types of named entities have their

Figure 2. Samsung Galaxy S in Greek, Korean and German

own paraphrasing mechanisms. As most of our named entities are product names and as product names usually have much more variants than e.g. manufacturer names and, therefore, more interesting paraphrasing mechanisms in SCM, we will concentrate on product names in this section. New instances of SCM tailored to any other set of types of named entities can easily be created. I.e. what we are going to say about product names in the remainder of this paper also holds for organization names, EU Guidelines, and many other types of named entities, *mutatis mutandis*.

Our first approach to product name paraphrasing was to use *paraphrasing classes*. Much as verbs are inflected according to their inflection class, product names were transformed according to their paraphrasing class. Yet, paraphrasing classes had to be assigned manually and quite many classes were needed. Therefore, we decided to use a simplified system: Each product or manufacturer name is stored in a canonical form: Thus, a name of the type *glofiish g500* is stored in the form *glofiish-g-500*, even if *glofiish g-500* or *glofiish g500* should be more frequent. The minus characters tell SCM where a new part of the product name begins. A product or manufacturer name has *permutations*: In German *o2 online-tarif* has the permutation *tarif-o2-online*. Standard permutations are added automatically: A product or manufacturer name with three parts has the standard permutation *123*. German tariff names of the type *o2 online-tarif* have the standard permutations *312* and *23 von 1* as in *online-tarif von O2* (*online-tariff by o2*), in addition. Grammar experts can add more permutations manually. Apart from their canonical name and its variants, product names can also have spelling variants. Thus, *android* has the spelling variants *androit*, *antroid*, *antroit*, *andorid*, *adroid*, *andoid* and *andoit*. (These are some of the most frequent ways *android* is actually spelt in the customer messages.) For each spelling variant, SCM automatically generates all paraphrases that exist according to the standard and the manually added permutations of the canonical name. I.e. the paraphrases of the mobile phone name *e-ten glofiish-g-500* include *e-ten klofisch-g-500*, *e-ten klofisch-g 500*, *e-ten klofisch g-500*, . . .

Apart from spelling variants, product names can also have lexical variants. The mobile phone *tct mobile ot-v-770-a* has the lexical variant *playboy-phone*. The regular permutation transformations are not applied to lexical variants. But lexical variants and their manufacturer-based variants (e.g. *tct playboy-phone* and *playboy-phone*) are, of course, paraphrases, too.

Depending on which characters are defined to be possible replacements of the minus character (the empty character, space, comma, . . .), SCM will generate a greater or smaller number of strings that actually never occur in the customer messages. But, for several reasons, overgeneration is not a problem. First, grammar experts can manually exclude variants. Secondly, most of the variants that never occur in the messages (such as *tct.mobile-ot,v770.a*) will hardly be used with the intent to refer to something else than the product of whose name they are claimed to be a paraphrase.

SCM's product name paraphrasing engine is aware of the fact that many products are sold in different countries under the same or under different names. SCM stores a unique international ID for each product. Product names and their paraphrases are language specific. SCM normalizes found product names to the international ID. This way, SCM provides the data for answering questions like: *Which statements are made about this mobile phone in which languages/in which social networks/in which countries/. . . ?* Figure 2 shows some of the Greek, Korean and German names of the product with the international ID *samsung galaxy-s.*

5. Development and Application of Grammars

Grammar experts can create any number of content and sentiment classifiers. As already pointed out, a classifier's grammar consists of a set of positive constraints and a set of negative constraints. To classify a message, SCM simply applies the grammars of all its (content or sentiment) classifier objects to the message. Put simply, tags are assigned in the following way: If a content classifier's grammar matches, then its tag is added to the message's tag list. Sentiment classification works analogously with the exception that each message gets exactly one sentiment tag. For a classifier to match means that its positive regular expressions match and its negative regular expressions do not match or that there are no negative regular expressions. For a set of regular expressions to match means that at least one of the regular expressions matches.

Actually, the application of content classifiers is a little bit more complicated than described above. The exact procedure will be described in the next section.

Content and sentiment classifiers are language- and url-specific: A classifier has exactly one language and a set of urls. It will only be applied to messages that have the same language and that stem from one of the urls in the classifier's set of urls.

SCM supports grammar creation in various ways: All objects in SCM, such as content classifiers, sentiment classifiers, countries, languages and messages are searchable and can be filtered by various criteria. Grammar experts usually deal with messages: They apply the content classifiers they (or other grammar experts) have created (or, for short, they apply *the grammar*) to all or certain messages. They can then filter the result: They might want to see only the messages that have received at least one content tag. Or they might want to see the messages that have not received a sentiment tag. Or they might want to see only Korean messages, that have no content tag and are from a certain url, and so on. Grammars and constraints that contact center agents unknowingly generate can be supervised and manually corrected by grammar experts.

We think that the fact that contact center agents can invent new tags and assign new or old tags to (badly) classified messages, if they mark the strings that are supposed to justify the assignment of the tag, is a good reason for not using a statistical approach. If we used a statistical approach, human work would be necessary at some point of the development process: Some algorithm would have to be trained. In our approach, the human work is done in the customer management process. This way, two things are achieved in one step: The customer's request is answered and the classification algorithm is enhanced. An SCM instance is being enhanced while it is used. There is no need to interrupt the customer interaction in order to train SCM on new data that data specialists have created. Besides, manual intervention is much more straightforward and transparent, if a grammar of the type described above is used than it would be with a statistical algorithm. Our system is flexible in the sense that it can easily be modified in such a way that very specific requirements are met. If, e.g., a future user of SCM (a company that wants to interact with its customers) should want to assign every message that has the word *hotline* in it a certain tag – such as *hotline problem* –, then this requirement can be met by simply adding the line `hotline` to the positive constraints of the classifier called *hotline problem*.

SCM follows the DRY principle (**D**on't **r**epeat **y**ourself, see e.g. [10], page 35): Changes are only made in one place. An example: the Korean variants of the mobile phone name with the international ID *google nexus-s* include *google nexus s*, *google nexus-s*, *nexus s*, *nexus-s*, *구글 넥서스 에스*, *구글 넥서스에스*, *구글 넥서스 s*, *구글 넥서스-s*, *넥서스 에스*, *넥서스에스*, *넥서스 s*, *넥서스-s*, *구글 nexus s*, *구글 nexus-s*, *google 넥서스에스*, *구글의 넥서스에스*.

This phenomenon is represented in SCM as follows: The Korean manufacturer name corresponding to the international ID *google* has the variants *google* and *구글*. The Korean name for *nexus-s* has the variants *nexus-s*, *넥서스에스* and *넥서스-s*. This is the only information SCM users have to store in order to make SCM generate these and many other variants.

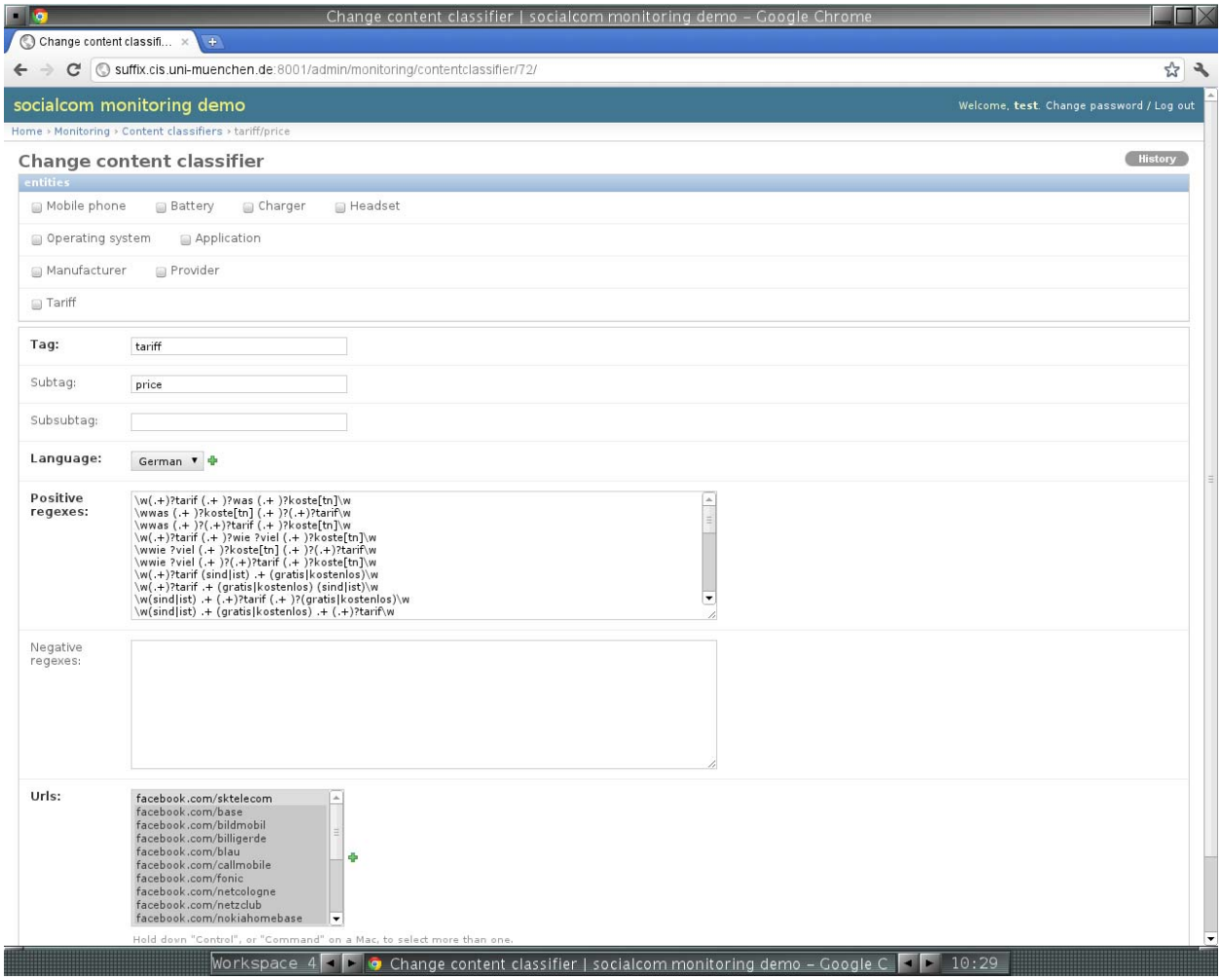

Figure 3. An unrestricted content classifier

SCM generates *google nexus s*, *구글 넥서스 s* and similar variants using the general rule that in any permutation of a product name any minus character may be replaced by a space character. SCM generates *넥서스에스*, *넥서스 s* and similar variants using the general rule that the manufacturer name may be omitted. And SCM generates *구글의 넥서스에스* using the two Korean variants of the manufacturer name and the general rule that phone names can have the form `<manufacturer name>`의 `<product name>`. (의 is a genitive affix, i.e. *구글의 넥서스에스* literally means *Google's Nexus S* or *Nexus S by Google*.)

We might, of course, add the general rule to SCM that any part of a Korean product name may be spelt either with Latin or with Hangul characters – according to several sets of transliteration conventions that are used in parallel.

Any change in a manufacturer, tariff or product name object, such as the Korean mobile phone name with the international ID *google nexus-s*, has implications for the grammars of the message classifiers: Newly generated variants of the product name must be matched by all instances of `\mobile_phone` in all grammars. For efficiency reasons, we compile all product names, tariff names, manufacturer names, content classification grammars, sentiment classification grammars, and so on, into one single function. The compiling and reloading of this function is done in the background, so the users of SCM do not need to know anything about it. They don't even have to understand the word *compile*. They just need to know that SCM sometimes needs a few seconds to be able to use changed objects. We tried to design SCM such that it is easy for grammar experts to create and maintain grammars and similar entities (such as product names and manufacturer names). We prefer longterm maintainability and usability over an algorithm that might perform better in a first demo version, but that is too rigid or complicated to be adjusted to new phenomena or to be otherwise maintained.

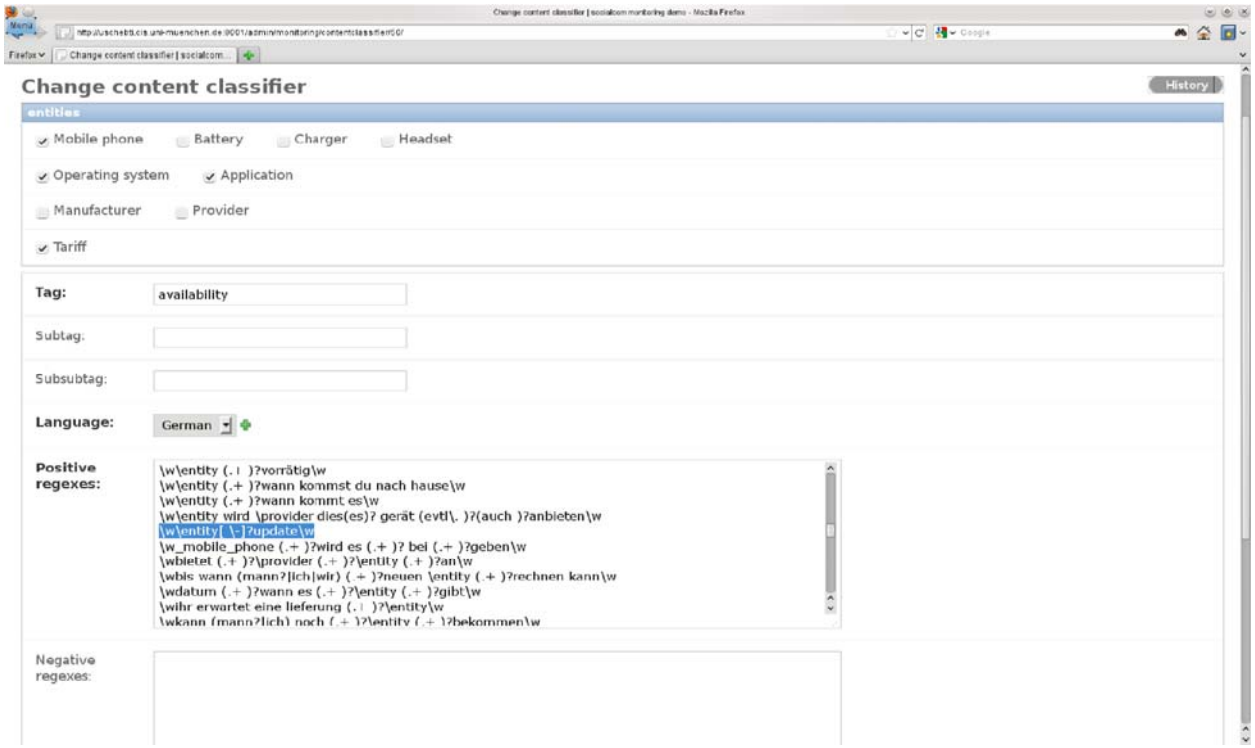

Figure 4. A restricted content classifier

6. Content Classification

Unlike sentiment classifiers, content classifiers come in two types: They may or may not be *restricted* to a set of entity types. If a content classifier is restricted to certain entity types, then a message must at least contain one instance of some of the chosen entity types in order for the content classifier to match. In the grammars of restricted content classifiers the special expression `\entity` refers to any instance of any of the chosen entity types. An example: The content classifier in figure 4 is restricted to the entity types *mobile phone*, *operating system*, *application* and *tariff*. This means that for this content classifier to match a message must at least mention one concrete mobile phone, operating system, application or tariff. In this grammar, the expression `\entity` matches any instance of any of these entity types. I.e. it would e.g. match *Iphone 4*.

Unrestricted content classifiers as the one shown in figure 3 do not, in general, require a message to contain any named entities. I.e. the expression `\entity` is meaningless in an unrestricted classifier's grammar. If a grammar author tries to use it, an error is raised. Grammar authors may use, though, expressions like `\mobile_phone` or `\tariff` in the grammars of unrestricted classifiers to require an instance of a specific type of named entities to be present in the message.

The idea of restricted content classifiers is to reuse code: We need to describe only once how customers express that some entity does not work or how they ask when some entity will be available. The more restricted content classifiers grammar authors use, the shorter will the grammar be. Yet, as not all phenomena can be described with restricted content classifiers, we need unrestricted classifiers, too.

Often grammar authors will have to create pairs of one restricted and one unrestricted content classifier that do almost the same. An example: To express that a mobile phone does not work one may either say *my mobile phone does not work* or, e.g., *my Nexus S does not work*. To classify the first message, we need an unrestricted content classifier with a grammar rule of the type `mobile phone .+(not|n't) work`. To classify the second message, we need a restricted content classifier with a grammar rule of the type `\entity .+(not|n't) work`. In the second case, we would assign a tag of the type *malfunction/mobile phone/google nexus-s*. In the first case, we would assign a tag of the type *malfunction/mobile phone*. We will immediately discuss how these tags come about.

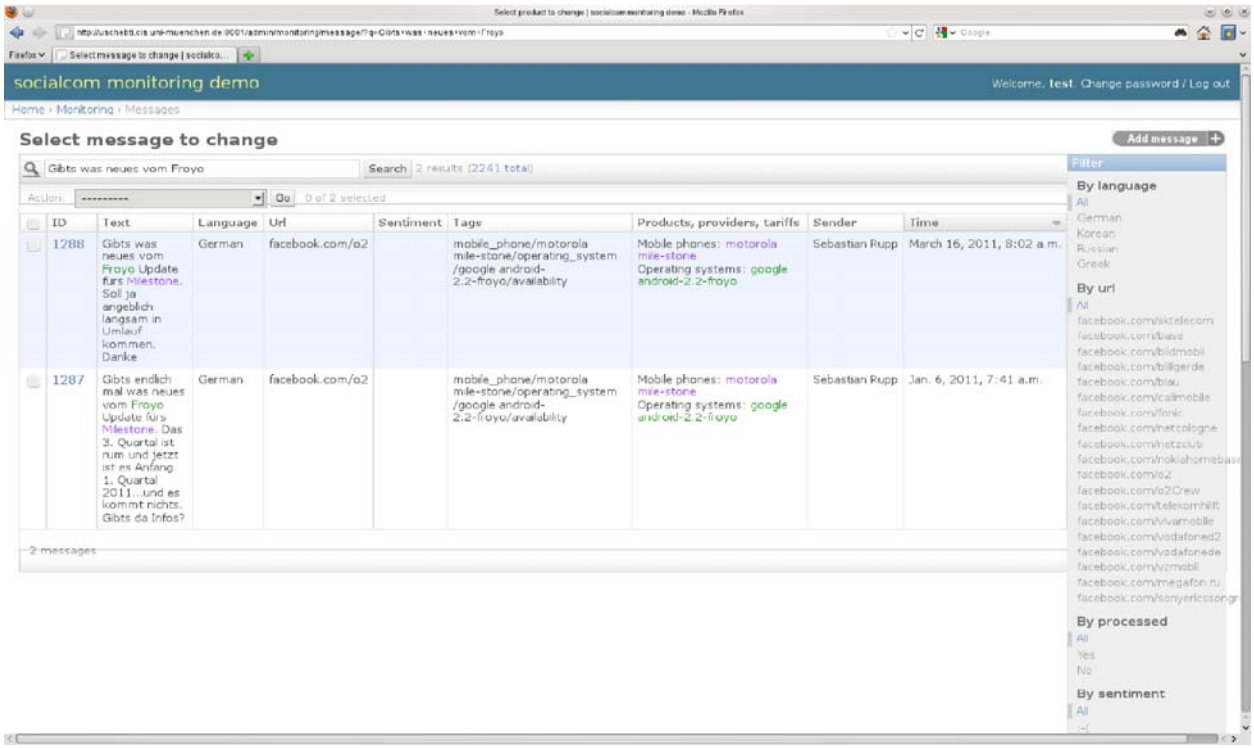

Figure 5. Messages in an SCM instance

Tags of content classifiers (restricted or unrestricted) may consist of up to three parts which we call *tag*, *subtag* and *subsubtag*. Only the field *tag* is obligatory. The three parts refer to three levels of generality. Sometimes we can only assign a rather general tag, such as *network*. Sometimes we might not only be able to guess that a message is about the network, but also, and more specifically, about *umts*. Sometimes we might even be able to guess that a message is about *network*, and, more specifically, about *umts* and, even more specifically, about the *malfunction* of the umts network. In this case, the content classifier with tag *network*, subtag *umts* and subsubtag *malfunction* would match and the tag *network/umts/malfunction* would be assigned. As experience taught us that more than three levels of generality (or specificity) are hardly needed, we limited the number of tag parts to three. A more specific tag part may only be present if the more general parts are present, too. Thus, an error message will be raised, if a grammar author tries to create a content classifier that has a *subsubtag*, but no *subtag* or no *tag*.

By using several levels of generality, we are able to distribute classified messages to contact center agents in a flexible way: A contact center agent may be an expert for a whole "branch" of the classification tree (such as *network*), or there may be one expert for umts-related issues and another expert for lte-related issues. Also, messages may be displayed and searched according to different levels of generality. And certain levels of generality may be ignored for certain tasks. In the case of unrestricted content classifiers, tag assignment is straightforward: The three tag parts are joined with slashes to form the tag to be assigned. But with resctricted content classifiers tag assignment is more complicated, as their application may involve *reasoning rules*.

Reasoning rules are a new and experimental feature of SCM. At the moment, grammar authors cannot create or change reasoning rules. They are not a feature of grammars or content classifiers, but of SCM. Here is an example of a very straightforward reasoning rule: Suppose a restricted content classifier with a tag such as *availability* or *malfunction* or *how does it work?* and with a set of entity types, such as *mobile phone*, *headset*, *battery*, *operating system* and *application* matches a certain message. Suppose the message contains exactly one named entity that matches the content classifier's entity types. The message might e.g. only mention the mobile phone *Google Nexus S* and no other mobile phones and no headsets and no batteries and no . . . . Then it is very probable that the topic of the message was if *Google Nexus S* is available or that a *Google Nexus S* does not work or how the *Google Nexus S* works, respectively. Therefore, we can assign a tag of the following type: *availability/mobile phone/google nexus s* (or *malfunction/mobile phone/google nexus s* or *how does it work?/mobile phone/google nexus s*). I.e. the tags that restricted content classifiers assign may contain

entity types and concrete named entities which they get from reasoning rules.

As most messages in social networks are very short, reasoning rules work well, even if they are very simple. An example: Our prototype assigned the second message in figure 5 the tag *mobile phone/motorola mile-stone/operating system/google android-2.2-froyo/availability*. I.e. the system stated that the meaning of the message is whether the operating system Android 2.2 is available for the mobile phone Motorola Milestone. Instead of using slash-joined tags, we might, of course, also use formulas of predicate logic or plain English to represent "meanings" of messages. Interestingly, the sender of the message asked the same question again about two months later. But he used a slightly different wording. Our system assigned exactly the same tag to the second message. That means our system recognized the two messages as paraphrases of each other. As questions with this type of meaning ("when is X available for Y") are asked very often, they might be answered semiautomatically in the future. A contact center agent might create standard answers for certain values of X and Y.

The reasoning rule necessary for correctly classifying the messages of figure 5 works roughly as follows: If the message's tag is *availability* and the message contains exactly one named entity of type *mobile phone* and exactly one named entity of type *operating system*, then the sender probably wants to know if (when) the operating system is (will be) available for the mobile phone. There are many similar cases: Thus, if a message's tag is *availability* and if it contains exactly one named entity of type *application* and exactly one named entity of type *mobile phone*, then the question is probably if this application is available for this mobile phone. Or, if a message's tag is *installation* and the only two named entities are of type *application* and *operating system*, then the message is probably about how to install this application on this operating system.

As these and other reasoning rules are very specific and domain-dependent, they should not be part of SCM, but of concrete SCM instances. We might implement this in the future. There are basically two ways to implement it: An easy and ugly way and a difficult and beautiful way. We chose the easy and ugly way for our current SCM prototype. We made a copy of SCM – of the "pure", the domain independent SCM – and added domain-specific reasoning rules to it. This can also be done for other domains. The difficult and beautiful way would be to enable grammar authors – or other agents – to create domain specific reasoning rules for instances of SCM via the web interface. This solution would be beautiful in so far as it does not involve copying and adapting of source code. There would be one and only one SCM and many different instances of it. Reasoning rules would be data just like product names or content classifiers. Yet, experience teaches us that it is often not efficient to create systems that are overly beautiful. In our case, a disadvantage would be that grammar authors would have to learn one more feature of the system – and a very complicated feature. Besides, it is not clear how the abstract form of reasoning rules should look like. Which elements does it take to create a reasoning rule? Which relations between these elements can hold? How are grammar authors supposed to describe the reasoning rules?

One possibility to give grammar authors responsibility for reasoning rules would be the following: Grammar authors might use tags as predicates and entity types as arguments. Thus, they might write a grammar for the topic *availability* and choose an arity for this topic, e.g. two. Then they might write rules of the type `Is \1 available for \2?` and decide that – in this specific content classifier – `\1` may refer to an application and `\2` to a mobile phone or an operating system, or, respectively, `\1` might refer to a battery or a headset and `\2` might refer to a mobile phone (but, obviously, not to an operating system). This way we could still share code, but hand over responsibility for reasoning to the grammar authors. And we would have more precise and powerful grammar rules. Yet, it would also make the system much more complicated – for its authors as well as for its users.

7. Conclusion

In this article, we presented SCM, a system for web monitoring and customer interaction. We tried to show how these tasks can be automatized to a large degree without making the system too complicated and intransparent. Our system is highly flexible and modular, because it has a software agent for each of the entities involved in the monitoring and customer interaction process. In SCM companies are mapped to company objects, products are mapped to product objects, text types are mapped to text type objects, and so on. Each of the objects can easily be modified by grammar experts. SCM makes sure that any change in an object is communicated to all the other objects that must know about it.

We hope to be able to enhance SCM in several ways in the future. The most important enhancements are perhaps the following two. On one hand, we should find a general and domain-independent solution for creating reasoning rules. On the other hand, SCM should be able to learn more things automatically. As many of SCM's grammars use special terms for matching certain types of expressions, such as `\mobile_phone`, `\charger`, `\headset`, `\tariff`, `\software`, `\operating_system` or `\battery`, minor code changes will suffice to enable SCM to find unknown mobile phone, charger or headset names in customer messages: As our grammars are designed for classifying messages of

specific types, they contain very good contexts of named entities. Thus, a grammar that describes messages written by customers who want to install a certain software on a certain mobile phone will probably contain good contexts for software and mobile phone names.

8. Acknowledgment

The Socialcom project was funded by the German Federal Ministry of Economics and Technology.

9. References

[1] Gryc, W., and Moilanen, K. (2010) 'Leveraging Textual Sentiment Analysis with Social Network Modelling: Sentiment Analysis of Political Blogs in the 2008 U.S. Presidential Election', in Proceedings of the From Text to Political Positions Workshop (T2PP 2010), Vrije Universiteit, Amsterdam, April 9–10 2010; http://users.ox.ac.uk/~wolf2244/gryc_moilanen_t2pp_2010_final.pdf (26 July 2011).

[2] Krauss, J., Nann, S., Simon, D., Fischbach, K., and Gloor, P. A. (2008) 'Predicting Movie Success and Academy Awards Through Sentiment and Social Network Analysis', ECIS European Conference on Information Systems.

[3] Pal, J. K., and Saha, A. (2010) 'Identifying Themes in Social Media and Detecting Sentiments', HP Laboratories, Tech. Rep. HPL-2010-50.; http://www.hpl.hp.com/techreports/2010/HPL-2010-50.pdf (26 July 2011).

[4] Mishne, G. (2007) Applied Text Analytics for Blogs, University of Amsterdam, Enschede/Netherlands.

[5] Jijkoun, V., Khalid, M. A., Marx, M., and de Rijke, M. (2008) 'Named entity normalization in user generated content', in Proceedings of the second workshop on Analytics for noisy unstructured text data, ser. AND '08: New York, NY, USA, ACM, pp. 23–30.

[6] Raghavan, H., and Allan, J. (2005) 'Proper Names and their Spelling Variations in Automatic Speech recognition output', University of Massachusetts, IR 361.

[7] Gross, M. (1989) 'The Use of Finite Automata in the Lexical Representation of Natural Language', in Proceedings of the LITP Spring School on Theoretical Computer Science, Electronic Dictionaries and Automata in Computational Linguistics: Springer-Verlag, London, pp. 34–50.

[8] Browne, R., Clements, E., Harris, R., and Baxter, S. (2009) 'Business and consumer communication via online social networks: a preliminary investigation', in Australian and New Zealand Marketing Academy (ANZMAC) Conference; http://www.duplication.net.au/ANZMAC09/papers/ANZMAC2009-328.pdf.

[9] Friedrich, G., and Shchekotykhin, K. (2006) 'NameIt: Extraction of product names', Data Mining Workshops, International Conference on: vol. 0, pp. 29–33.

[10] Colbaugh, R., and Glass, K. (2010) 'Estimating sentiment orientation in social media for intelligence monitoring and analysis', in ISI: Vancouver, BC, Canada, pp. 135–137.

[11] Locke, B., and Martin, J. (2009) 'Named Entity Recognition: Adapting to Microblogging', University of Colorado.

[12] Bothos, E., Apostolou, D., and Mentzas, G. (2010) 'Using Social Media to Predict Future Events with Agent-based Markets', IEEE Intelligent Systems, vol. 99.

[13] Laboreiro, G., Sarmento, L., Teixeira, J., and Oliveira, E. (2010) 'Tokenizing Micro-Blogging Messages using a Text Classification Approach', in AND'10: Toronto, Ontario, Canada.

[14] Kaufmann, M. (2010) 'Syntactic Normalization of Twitter Messages', studies, pp. 1–7; http://www.cs.uccs.edu/_kalita/work/reu/REUFinalPapers2010/Kaufmann.pdf (26 July 2011).

[15] Key-Sun, J.-S. N., and sun Choi, K. (1997) 'A Local Grammar-based Approach to Recognizing of Proper Names in Korean Texts', in Proceedings of the Fifth Workshop on Very Large Corpora: ACL/Tsing-hua University/Hong-Kong University of Science and Technology, pp. 273–288.

[16] Kozareva, Z. (2006) 'Bootstrapping named entity recognition with automatically generated gazetteer lists', in Proceedings of the Eleventh Conference of the European Chapter of the Association for Computational Linguistics: Student Research Workshop, ser. EACL '06: Stroudsburg, PA, USA, Association for Computational Linguistics, pp. 15–21.

[17] Volk, M., and Clematide, S. (2001) 'Learn – Filter – Apply – Forget. Mixed Approaches to Named Entity Recognition' in Proceedings of the 6th International Workshop on Applications of Natural Language to Information Systems: GI, pp. 153–163.

[18] Kaplan, A. M., and Haenlein, M. (2010) 'Users of the world, unite! The challenges and opportunities of Social Media': Business Horizons, vol. 53, pp. 59–68.

