An eXtreme method for developing lightweight ontologies

Maia Hristozova
The University of Melbourne
221 Bouverie Str.
Carlton 3010, Vic., Australia
+61 3 8344 9311
majah@cs.mu.oz.au

Leon Sterling
The University of Melbourne
221 Bouverie Str.
Carlton 3010, Vic., Australia
leon@cs.mu.oz.au

'There is no one correct way to model a domain - there are always viable alternatives. The best solution almost always depends on the application that you have in mind and the extensions that you anticipate.' [Noy & McGuiness, 2000]

Abstract

In this paper we propose a method for effective building of lightweight ontologies, applying the principles of eXtreme Programming. The method is based on a multi-layered approach, which combines the advantages of maximum monotonic extensibility with clarity of the desired terms. The method aims to make it easy to develop a simple ontology, being careful to avoid deviation between the user's expectations and the produced output. We emphasize on capturing system requirements and consider the intended use of the ontology by analyzing the user-provided competency questions.

The method is illustrated with the domain of Value-Added Publishing, which can be represented as a multi-agent system, where different types of information extraction agents deal with various aspects of processing information.

Keywords

Requirements for ontology modelling, applications of ontologies, ontology maintenance/evolution, practical experience.

1. Introduction

Agent-based systems are advancing in many different directions and the number of multi-agent solutions keeps increasing. Heterogeneity and diversity of the agent communication languages, protocols and building techniques give freedom and flexibility in developing, but at the same time create a major difficulty in communication between agents within a system and between different multi-agent systems [9]. The adoption of a shared ontology allows agents to interoperate without misunderstanding, while retaining a high degree of autonomy, flexibility and agility. By reducing semantic ambiguity for the purpose of sharing and reusing knowledge, ontologies help to achieve interoperability and robustness in a system. Agents can be highly adaptable yet are able to meaningfully communicate domain-specific knowledge. Also since they provide syntactic and semantic terms for describing knowledge about the domain in the multi-agent systems, ontologies are trying to find a place as specification mechanisms.

There are a number of issues covered in [4, 8] that leave a global ontology infrastructure unsuitable as a sole approach, and it is our belief that until all agents subscribe to a common ontology or a centralised language of understanding and representation, the development of multi-agent systems will necessitate building a number of small, domain-specific ontologies.

Various methodologies have been developed for building ontologies. However, most of these emerged from projects whose final goal was building the ontology itself. All that we are aware of are based on alteration and improvement on the produced output and even if the initial details and scope are well-defined an evolving prototype is the central core. But while visualizing and navigating ontologies are two relatively simple processes (due to the increasing number of ontology browsers), practical modification of an ontology, including improvement, expansion and updating, remains a very complex task. Removing classes or changing the structure (relationships) is a complex process that even the most intelligent agents lack capabilities to perform satisfactorily, i.e. without compromising the reliability of the conceptual model. Integration and maintenance of an existing ontology could also bring to the surface unconsidered difficulties. For these reasons maintaining a global ontology has so far been unsuccessful. On the other hand small lightweight methodologies for ontology-building that are easy to follow, with only a few rules and practices, have not become wide spread yet. Agent-systems developers keep building small domain-specific ontologies ignoring the guidelines and instinctively moving away from heavyweight methodologies. Most are moving back toward an earlier, simpler time of lightweight methodologies when a few rules were enough. The existing ontology-development methodologies, though, offer processes and rules that by simplifying can significantly advance the creation of reliable and quick small ontologies. In this paper we propose a method for effective building of extreme lightweight ontologies that maintain the balance between developing an ontology from scratch without any rules and practices, constantly updating the structure and on the other hand long-term specification, prototyping and maintenance. We will do this by introducing some few rules and practices that are light, concise, and effective, mainly based on the eXtreme Programming practices. The method is based on a multi-layered approach, utilising the advantages of monotonic extensibility and clarity of the desired terms. The method aims to avoid deviation between the user's expectations and produced output, by focusing on capturing system requirements as well as considering the intended use of the ontology by analyzing the competency questions, provided by the users.

As a use case we take the domain of Value-Added Publishing. VAP can be represented as a multi-agent system, where different types of agents deal with different aspects of processing documents.

The rest of the paper is structured as follows. In section 2 we give a basic overview of eXtreme Programming principles. More details and descriptions are given in [1]. We then present some of the existing methodologies for ontology design and development. We have limited our review to those that we have tested and evaluated. In section 4 we briefly define the main aspects of the proposed eXtreme Ontologies method and present the general architecture of the prototype.

2. Extreme Programming Basics

EXtreme Programming (XP) has been developed on the basis of the results of many observations of what makes computer programming faster and what slows down it. As almost any other lightweight discipline of software development the main focuses of XP practices are simplicity, communication and feedback. The main XP rules that will be mentioned very briefly here and used for the purpose of this paper are: *The Planning Process.* It allows the customer to define the business value of desired features and uses cost estimates provided by the programmers to choose what needs to be done and what can be deferred. *Small Releases.* A simple system is released early, and updated frequently on a very short cycle. *Metaphor.* A common "system of names" and a common system description are used that guide development and communication. *Simple Design.* The program meets the current requirements. Instead of building "for the future", the focus is on providing business value. *Testing.* Focuses on validation at all times. Firstly the tests are developed and then software that fulfils the requirements. Customers provide acceptance tests that enable them to be certain that the features they need are provided. *Refractoring.* Improvement of the design throughout the entire development. *Collective Ownership*, *Continuous Integration.* Integration of the software system multiple times per day.

3. Extreme Ontologies Method

Ontologies may be distinguished by the requirements which they are able to solve; that is, one ontology may be able to represent and solve a different set of requirements to another ontology. Even if two ontologies are in the same domain, in different environments there is no guarantee that both will be interchangeable or reusable.

We propose to build the ontology at the end of the process of developing the system, after all the requirements have been specified, needs determined and specifications written. One might argue that the whole system will be dependent on what the user has specified initially, but the change of requirements won't affect much on the whole system, since rebuilding the ontology won't be time-consuming and there won't be a need to change the structure of the other modules. Its internal model can be well specified in the beginning and added to the other details of the specification. Making the ontology the last step of a software development cycle will reduce the development time, in addition to the benefits that come from building small, light-weight ontologies, instead of deep, broad and complex ones which are slow to parse and eventually retrieve information from.

The difference in the approach is the emphasis on the user. Rather than compiling all the existing knowledge in an ontology and teaching the system to browse it, fetch the valuable information and present it in a convenient way, the user's system plays a significant role in the initial process of developing the ontology. We are considering a multi-agent system, in which different types of agents exist each performing different task or service.

The process consists of:

1. Fetching the requirements of the system. In the traditional process this could be considered as scoping. The collected data will be used to create a baseline ontology. Baseline or input ontology [2] is an ontology that consists of a small number of concepts that are unavoidable for a particular domain.

Since we are taking value-added publishing (VAP) as a case study, potential concepts are: author, publication, awards of a paper, rating of the author, etc. In the context of XP this step of the process is defined as Collective Ownership, i.e. a full set of the system and domain requirements is available.

2. After the baseline ontology is created, the competency questions have to be defined. They are domain-specific and provided by the user. Competency questions provide the base for extracting the ontology concepts, or more formally they serve as test cases for completeness and validity of the ontological commitments. For example, analysis of a competency question "What is the ranking of this paper's author?" leads to including concepts such as 'paper', 'author', 'ranking'. This phase is very similar to the 'defining the content of the ontology' phase in the traditional methodologies.

3. Validation test. The main difference of eXtreme Ontologies approach is that once the list of the competency questions is determined, they are tested using the baseline (or extended) ontology. The validation test checks if the competency question could be answered using the existing ontology. If the result is successful the next competency question is asked. In terms of XP, since the ontology is being tested and updated frequently after each competency question, this step covers Small Releases.

4. Redundancy test. A negative result to the previous step would lead to extending the existing ontology with new classes or relationships extracted from the competency question. For this purpose a redundancy test is run that checks for existing classes, attributes or relationships with similar or matching names. This phase ensures metaphor or unification of the name standards, which was introduced by Aspirez J. in 1993 and exists as a requirement for a good ontology design.

5. Planning. The planning process in ontology development is extended beyond the identification of requirements of the system. It also includes estimation of the cost and time for building and subsequently managing the developed ontology. The depth and width of the ontology to be created are analyzed by calculating the time for developing and use time (which is a function of time to parse, time to summarize and time to convert the ontology into the needed form). Since the rest of the system modules are already developed the workings of the parser, for example the expected output and the converted type of ontology concepts, are known. During the process of planning there is a number of inevitable issues to be considered. The tightness between classes or the type of the relationship between them is such. For example a relationship 'car_has_wheels' will always be slower to parse than 'tree_has_nests', because whenever the structure is created for a 'car', there will be need to create a structure for 'wheels' as well (cars always have wheels), while in parsing both 'tree' and 'nest' the existence of one does not lead to the creation of the second (trees can exist without nests). Planning also includes estimating

the optimal depth and width and answering questions such as "How much information should be encoded in the relations and how much in the hierarchy itself?", since ontologies with different structure perform differently even if they produce the same result.

6. After all the planning and estimations have been done the next step in the process is Continuous Integration. Ultimately any methodology needs to be customized to the circumstances and environment. No methodology is just a collection of rules to be performed in rote fashion. During the process of integration it is very important to explore how the ontology will react with the other modules of the environment. If for example there is a converter in the system that parses the ontology, it has to produce correct class structure output. If the purpose of the ontology is to fill up a database it must be checked for consistency and structure. The result and the benefit of the continuous integration is a clean and simple ontology without duplications.

7. Acceptance tests. One of the key tasks to be done (some might argue that they should be created even before the planning) is to define the acceptance tests. It is very important for the user to know in advance what the system should do and to understand it. Tests should be created for each iteration step. For each question and validation test there is a level of acceptance of the result. As in TOVE [5,6], the acceptance tests should be on more than one levels according to the priority. Ideally they should be defined in a hierarchy with higher-level questions requiring the solution of lower level questions. It is not a well-defined ontology if all the questions have the form of a simple query; the solutions of some questions should use the solutions of the others. It is important to mention that these tests do not generate ontology structure. Rather they are used to evaluate those concepts that are ontology commitments. They evaluate the expressiveness of the ontology that is required to represent the questions and to characterize their solutions.

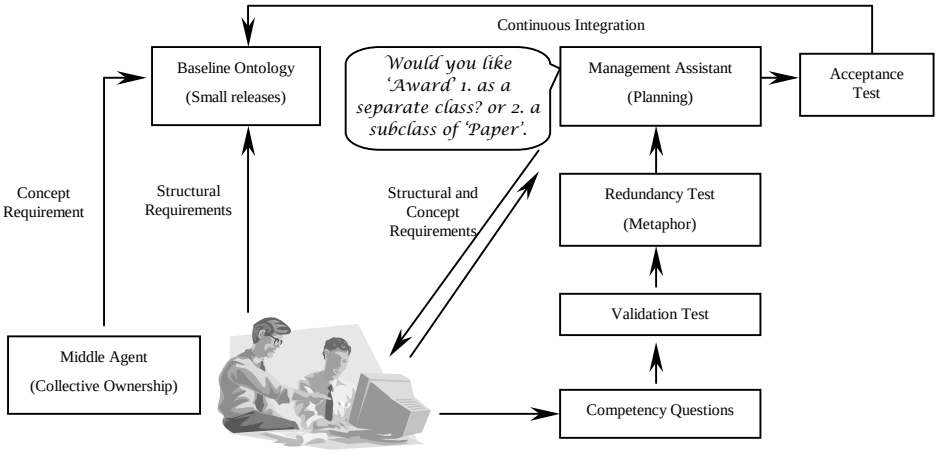

Figure 1

This method does not include maintenance as a final stage of the ontology development. The main reason is that since the requirements are carefully specified in the initial stage, once an ontology is created it is guaranteed that it meets the specific needs of the current application. All the inconsistencies or potential problems with the developed ontology are avoided during the testing and planning stages. A situation commonly encountered during software development is that of changing requirements. Drawing on the principles of XP, the eXtreme Ontology method handles changes by focusing on keeping ontologies lightweight and highly adaptable. Frequent iterations and continuous integration ensure that any necessary changes are identified immediately, and no unnecessary changes are made. The fact that the ontology is developed after the system is specified indicates that the requirements will be much more stable during the ontology development phase. In this way, the problems are avoided that traditional methods face when the ontology is frozen long before any prototypes can be presented to the client for initial feedback (the point at which the client usually realises that their initial requirements are not what they really want). Also in case of adding new modules of components to the system, due to the semi-automated tools, the process of building the ontology is easy and quick.

Following the described method at the moment we are developing a prototype to test the usability and effectiveness of the process. The final purpose is to measure and compare the extreme ontologies approach to other traditional ontology-development methodologies in terms of success of using the provided tools, techniques and integration of the produced ontology in a complex organization. The prototype is still in development but the draft version of its architecture is given in Figure 1.

A module that has the 'full' knowledge of the system achieves the Collective Ownership principle. The way we propose that this be done is by using one or more middle agents [4,11,12] that collect agents' capabilities. Capabilities briefly define the purpose of an agent, for example finding the rating of an author. In a multi-agent system whenever agents 'come' into a system they register their capabilities to the middle agent by sending to it a basic description of their input and output. There are a number of ways this can done, generally by using an interface description language, or an agent description language. Another comparison to XP is the fact that the requirements are extracted from some form of system by one module and further processed by another module. We consider the second stage to be performed by humans and the requirements are expressed in a primitive and simple enough way to act as tokens. So far the existing methodologies suggest that ontology engineers create the baseline ontology manually. In our case it is done semi-automatically - the middle agent provides concepts and attributes, but engineers define the structure. For example, based on the description of agent's capabilities, a middle agent will suggest concepts such as 'author' and 'paper', the type of the output, i.e. 'list of papers' [and maybe other information, depending on what is provided by the information extraction agent] but the relationship between them - 'writes' - will be defined manually.

The validation test module ensures small releases of the ontology. For example in the case of VAP a competency question might be:

"Provide a list of books written by this author?". If the question can be answered using the current ontology the ontological commitments are valid and the next question is tested. Otherwise the redundancy test checks if the competency question contains words similar to the already existing concepts in the ontology. At this stage the comparison is mainly syntactic, based on pattern matching, but further it could be extended to semantic mapping [8,12].

In [2] is proposed a semi-automated approach for modifying ontologies, using management assistants. For the different modifications of the ontology corresponding assistants analyze the model to identify the consequences of the planned actions. The assistant then works in cooperation with the user to select and perform operations without violating the consistency of the knowledge base in order to achieve the user's goal. An intelligent agent can predict the consequences of an ontology structure, thus the evaluation will be performed before the actual change.

Referring back to XP the management assistant does not play its initial role of interface between requirements and development, but acts rather as facilitator to the customer. In this sense little freedom is given to the user - they must choose between a number of options provided by the management assistant. Thus rather than translating the informal requirements into formal, the assistant helps the user to express the requirements clearly but still in their own words, as close as possible to the ontology representation. According to XP practices, in cases of multiple users it is not the assistant's responsibility to prioritize the requirements - it only helps the customers to sort out and schedule in time the things between themselves. In an ideal system they can even negotiate and distribute the time to perform each one's requirements (or during a different iteration). But in the end, feedback is given prior to the release about what will be done, how and why.

4. Conclusions and Future Work

After reviewing some of the existing methodologies for ontology development, in this paper we have proposed a new approach for developing lightweight ontologies. We are combining some of the beneficial characteristics of the existing methods, but avoiding the attitude that "ontologies are the core element" of the systems being developed. EXtreme Ontologies approach involves humans to create the baseline ontology, but all the consequent changes are only guided by the user, while the internal processes are performed by automated tools. The method ensures that development effort is not spent on unnecessary implementation of classes and relations. The derived ontology only meets the requirements that match the validation rules and test cases.

Despite the fact that our project is still in an initial phase, and tools that reduce risk and gain benefit from the validation tests and rules before the actual implementation of an ontology have not yet been developed, we believe that it will significantly contribute to this area. Additionally, the evaluation of the method and subsequent experiments will reveal the effectiveness of applying the eXtreme Programming principles and rules to ontology design and development. The final purpose is to explore and compare the extreme ontologies approach to other traditional ontology-development methodologies in terms of success of using the provided tools, techniques and integration of the produced ontology in a complex organization.

5. REFERENCES

[1] Beck, K. Extreme Programming Explained: Embrace Change, Addison Wesley, 2000.

[2] Boicu, M., Tecuci, Gh. et al. Ontologies and the Knowledge Acquisition Bottleneck. In Proceedings of the IJCAI-01 Workshop on Ontologies and Information Sharing. 2001.

[3] Fernandez L. Overview Of Methodologies For Building Ontologies, 1996.

[4] Giampapa, J., Paolucci, M., Sycara, K. Agent Interoperation Across Multagent System Boundaries. In Proceedings of the Fourth International Conference on Autonomous Agents (Agents 2000). Association for Computing Machinery, ACM, 1515 Broadway, New York, NY, 10036, USA. June, 2000.

[5] Grüninger, M., Fox, M. Methodology for the Design and Evaluation of Ontologies. Canada M5S 1A4, April 1995.

[6] Grüninger, M., Fox M. The Design and Evaluation of Ontologies for Enterprise Engineering. IFIP WG5.7 Workshop on Benchmarking - Theory and Practice. Norway, 1994.

[7] KBSI. The IDEF5 Ontology Description Capture Method Overview. KBSI Report. USA, 1994.

[8] Lister, K., Sterling, L. Agents in a Multi-Cultural World: Towards Ontological Reconciliation, Markus Stumptner, Dan Corbett and Mike Brooks (eds), AI 2001. Advances in Artificial Intelligence. In Proceedings of the 14th Australian Joint Conference on Artificial Intelligence. Adelaide, 2001.

[9] Subrahmanian, V. S., Bonatti, P., Dix, J., Eiter, T., Ozcan, F. Heterogeneous Agent Systems. MIT Press, 2000.

[10] Swartout, W.R., Patil, R., Knight, K. and Russ, T. Towards Distributed Use of Large-Scale Ontologies. AAAI-97. Spring Symposium on Ontological Engineering, Stanford University. 1997.

[11] Sycara, K., Chi Wong, H. A Taxonomy of Middle-agents for the Internet. Proceedings of the Fourth International Conference on MultiAgent Systems, 2000.

[12] Sycara, K., Klusch, M., Widoff, S., Lu, J. Dynamic Service Matchmaking among Agents in Open Information Environments. In ACM SIGMOD Record, Special Issue on Semantic Interoperability in Global Information Systems, 1999.

[13] Uschold, M. and King, M. Towards a Methodology for Building Ontologies. IJCAI-95. Workshop on Basic Ontological Issues in Knowledge Sharing. Montreal, Canada. 1995.

[14] Uschold, M. Building Ontologies: Towards A Unified Methodology. Proceedings Expert Systems 96. Cambridge, 1996.