Explicit Domain Knowledge in Software Applications [*] .

Maja D'Hondt
System and Software Engineering Lab
Vrije Universiteit Brussel, Belgium
mjdhondt@vub.ac.be

ABSTRACT

This research is concerned with representing knowledge about the domain of software applications independently and separately from other concerns of the software – henceforth referred to as the implementation strategy – at the implementation level. As a result both the domain knowledge and the implementation strategy become more reusable and maintainable because changes made in the one part do not propagate to the other. Additionally, the average software developer – who is typically not a domain expert – does not have to deal with the domain knowledge and vice versa. For expressing domain knowledge in a suitable medium we investigate existing hybrid knowledge representation technologies that use frames and rules for representing knowledge. For composing the domain knowledge and the implementation strategy into a operational software application that exhibits the required behavior, we are inspired by the principles Aspect-Oriented Software Development.

1. TECHNICAL PROBLEM

The complexity of software domains is steadily increasing and knowledge management of businesses is becoming more important. The real-world domains of many software applications, such as e-commerce, the financial industry, television and radio broadcasting, hospital management and rental business, are inherently knowledge-intensive. Current software engineering practices result in software applications that contain implicit *domain knowledge* tangled with the *implementation strategy*. An implementation strategy might result in a distributed or real-time application, or in an application with a visual user interface or a database, or a combination of above. Domain knowledge consists of a conceptual model containing *concepts* and *relations* between the concepts. It also contains *constraints* on the concepts and the relations, and *rules* that state how to infer or "calculate" new concepts and relations [25]. There is a strong analogy between the rules and constraints on the one hand, and *Business Rules* on the other. Business Rules are defined on a *Business Model*, analogous to the conceptual model of the domain knowledge.

A first problem is that real-world domains are subject to change and businesses have to cope with these changes in order to stay competitive. Therefore, it should be possible to identify and locate the software's domain knowledge easily and adapt it accordingly while at the same time avoiding propagation of the adaptations to the implementation strategy. Similarly, due to rapidly evolving technologies, we should be able to update or replace the implementation strategy in a controlled and well-localized way. A second problem is that the development of software where domain knowledge and implementation strategy are tangled is a very complex task: the software developer, who is typically a technology expert but not a domain expert, has to concentrate on two aspects of the software at the same time and manually compose them. This violates the principle of *separation of concerns* [8] [22] [10], which states that the implementation strategy should be separated from other concerns or aspects such as domain knowledge. In short, the tangling of domain knowledge and implementation strategy makes understanding, maintaining, adapting, reusing and evolving the software difficult, time-consuming, error-prone, and therefore expensive.

The cause of this problem can be found in current software development methodologies. Older software development methodologies such as OOSE [13] and Booch [4], and even the newer standard Rational Unified Process [24] employ a use case-driven approach as a result of which domain knowledge and implementation strategy are modelled together from the start. Other approaches to software development such as domain engineering [20] [14] try to take a whole family of applications into account by allowing anticipated variations of (among others) domain knowledge, but do not offer an explicit and separate model of the real-world domain knowledge.

Note that knowledge elicitation is not the major issue in this problem and not an ingredient in the answer to tangled and implicit domain knowledge in software applications. Domain knowledge is currently already elicited from the stakeholders, but unfortunately in this process it is not separated from the other requirements of the application (which may lead

[*]This research is funded by *het Instituut voor de bevordering van Wetenschappelijk en Technologisch onderzoek in Vlaanderen* (IWT)

to a specific implementation strategy) and modelled explicitly. Moreover, domain knowledge is quite tangible since it contributes actively to the behavior of the application.

2. EXAMPLES OF DOMAIN KNOWLEDGE IN SOFTWARE APPLICATIONS

To clarify our above description of domain knowledge, two small representative case studies that are used in this research are briefly presented here. They are both based on existing industrial applications, but scaled down without reducing the inherent complexity related to this research topic. The first case study is an e-commerce application for an online book and cd shop. The second case study is an application for the management and support of planning television programs for broadcast companies. These cases provide a balanced combination of domain knowledge and implementation strategy: in e-commerce applications there are obviously technological challenges such as transactions, replication, remote procedure calling and concurrency, whereas the second case study has to deal with a visual user interface and persistency.

2.1 e-Commerce

The domain of a the first application contains for example the obvious concepts customer, shopping cart, product (of which book and cd are specializations), customer profile, and some obvious relationships between them. Constraints on this static domain model are for example "a customer can buy at most 10 products at the same time" or "if the purchased products are shipped, the order cannot be cancelled". Related to calculating the price of an order there are a number of rules such as "if a customer has previously bought 10 products, he or she is entitled to a 10% discount on the next order", "if it is Christmas, everybody gets a 5% discount" and "if a customer's last purchase was a cd in the category of classical music, then he or she gets a discount of 15% on the next classical music cd". It becomes interesting when one thinks of the possible interferences of these rules and constraints and how to deal with them. What happens when a customer who has already purchased more than 10 products orders something during Christmas?

2.2 Broadcast Planning

In the domain of broadcasting there are concepts such as transmission (a time slot in the schedule), program (concrete program to be broadcasted), contract (for programs that were purchased), tape, snap (a rebroadcast), trailer (announcement for a number of programs), group of programs, chain of programs, and so on. Again, there are relationships between these concepts, some more obvious than others. Constraints limit the scheduling of these concepts, for example stating that "a snap should always be scheduled after its original" and that "the contract should be valid for the period in which the program will be broadcasted'. When a scheduled entity is moved in the program schedule a number of rules become active, such as "if the anchor of a chain of programs is moved, the entire chain has to be moved". Again, the rules and constraints interact: if the rules dictate that other programs have to be moved as a result of the move of a program, all the constraints have to be checked on these programs as well.

3. SCOPE, GOALS, AND HYPOTHESIS

According to the technical problem described earlier, the domain knowledge and the implementation strategy of software applications should be represented as separated as possible. Although this principle can and should be applied throughout the entire software development life cycle, we will concentrate on representing domain knowledge and implementation strategy separately at the implementation level.

Object-oriented programming languages are the state of the art today for expressing the implementation strategy. Given the structure of concepts and relations and the declarative nature of the constraints and the rules, a suitable representation will be selected from existing hybrid (i.e. frames and rules) knowledge representation languages for expressing domain knowledge. Purely static representation of domain knowledge will not suffice since it has a very active role to play in achieving the overall behavior of the system. Therefore, suitable reasoning mechanisms have to be selected for checking the constraints and chaining the rules. A combination of forward and backward reasoning seems a good candidate for the latter.

Our research hypothesis is *Using knowledge representation technologies for expressing domain knowledge of a software application explicitly and separately from the implementation strategy of the software application which is expressed in a standard (object-oriented) programming language will improve software understandability, software maintenance and software reuse.* Although it is difficult to test subjective properties such as improved understandability, it was already shown in [30] that an explicit model of the domain knowledge achieves exactly this. Furthermore, we will show that the separation of domain knowledge and implementation strategy reduces the propagation of changes from the one part to the other, thus facilitating maintenance. Finally, the AOSD community among others, promotes decomposing parts of the software into loosely coupled and independently evolvable components because it improves reusability.

Whereas the aforementioned suite of technologies achieves the desired separation or decomposition of explicitly described domain knowledge from the implementation strategy, it does not consider the composition of the two in order to achieve a working software application. Since the structural part of the domain knowledge should be mapped onto the implementation strategy and the operational part should be dynamically inserted in very specific places in the implementation strategy, we will look at *Aspect-Oriented Software Development* technologies [2] because they achieve exactly this. In these technologies, *aspects* such as error handling, error reporting, persistence and so on, are expressed in an *aspect language* separate from the implementation strategy. A *weaver* composes the aspect with the implementation strategy which results in an executable program. A weaver uses *join points* which indicate dynamic places in the implementation strategy where the aspect should be inserted. We believe that in some cases the "weaving" of domain knowledge and implementation strategy will be quite straightforward, but that in others it will benefit from applying ideas if not actual techniques from aspect-oriented programming. An original contribution of this research is to consider *domain knowledge as an aspect* [6] [7]. We are cur-

rently investigating the suitability of existing AOSD technologies for composing domain knowledge with implementation strategy. The goal is to come up with a required set of features to achieve this. A slightly more adventurous and ambitious side track of this story is the idea that weaving is a knowledge-intensive process (as is also shown in [28]). Therefore we will investigate the advantages of using the same knowledge representation language as meta-language for guiding the composition.

4. PLAN, METHOD AND CONTRIBUTION

The following sections correspond to the most important steps in this research project.

4.1 Representing Domain Knowledge

After a literature study of hybrid knowledge representation systems containing representation mechanisms for both frames and rules, the following minimal set of features for representing domain knowledge was decided upon:

- basic frame-based representation, with frames having slots that can contain values or rules (specifying how to infer values), and daemons that watch the slots and trigger rules when slots are accessed or changed
- prototype-based frames, as in KRS [18]
- a mixture of forward and backward chaining rules

We will further investigate if a constraint checker or truth maintenance systems is required. If possible, an existing system will be reused, but given the context in which it has to operate it is more likely that we will implement a light-weight knowledge representation system with the above features.

4.2 Composing Domain Knowledge and Implementation Strategy

This step in the research project consists of two parts: first, investigating the suitability of existing AOSD technologies for composing domain knowledge with the implementation strategy, and second, establishing a symbiosis between the selected knowledge representation system (KRS) and the object-oriented programming language (OOPL) for facilitating the composition of domain knowledge and implementation strategy.

4.2.1 AOSD technologies

Currently we are investigating the state of the art in AOSD technologies such as HyperJ [21], AspectJ[16], Composition Filters [3] and Demeter [17]. The goal is to find out how well they support composing domain knowledge and implementation strategy using the small case studies explained above. The result will be a set of necessary features that are required for composing domain knowledge and implementation strategy. Since it is very likely that this set will contain features from the different approaches – some AOSD approaches have different capabilities that complement each other well – we predict that no single approach will be most suitable.

4.2.2 Symbiosis between a KRS and a OOPL

For enabling the composition of domain knowledge and implementation strategy, the symbiosis between the chosen knowledge representation system and the object-oriented programming language will have to be investigated. Research on symbiosis between two object-oriented languages is already conducted in [26]. A similar configuration was already successfully developed at our computer science department, more specifically logic meta-programming [28] [31] where a logic language serves as a meta-language to reason about object-oriented base code, which can be used for example to enforce architectural or design choices in the code [32]. Moreover, a simple prototype of a forward-chaining rule-based meta-language for an object-oriented base language was also developed for reasoning with design knowledge for interactively supporting framework reuse [19].

4.3 Validation

For the proof of concept using the developed artifact we will take the industry as laboratory approach. Through contacts with industrial partners, several case studies will be set up. One of our partners is a Belgian company that has been making software for managing everything related to television or radio broadcasting for a decade. It counts among its customers many major European broadcast companies as well as others outside Europe.

Moreover, our ideas and findings are and will be subject to inspection by the international research community. After gaining experience in organizing workshops through the co-organisation of the ECOOP '00 Workshop on Aspects and Dimensions of Concerns [27] and the ECOOP '01 Workshop on Feature Interaction in Composed Systems [23], we will now organize the workshop on Knowledge-Based Object-Oriented Software Engineering at ECOOP '02. Participation through technical papers and posters will be continued.

5. RELATED WORK

Apart from the aforementioned technologies and approaches that will be actively (re)used in this research project, some other work is also relevant.

GeoObjects is a project we were involved in together with an industrial partner specialized in producing and maintaining digital geographic data to be used in Geographic Information Systems. In this project we delivered a means for describing quality constraints, used for checking the well-formedness of the geographic data, in an application independent, modular and declarative way on a conceptual model of the geographic data. This representation of the quality constraints is translated by means of a code generator into a classical programming language which has access to the actual geographic data via API calls [29] [5].

There is some work done on *Business Rules*, where rules and constraints are modelled separately from the core application at the specification level. At the design and implementation level *patterns* are provided for making the business rules as reusable and maintainable as possible [15]. However, the business rules are still tangled in the implementation strategy and not expressed declaratively. We still need to look into approaches such as *CommonRules*[12] and *Business Rule Beans*[11].

There are other efforts that advocate the explicit modelling of domain knowledge. The *framework for requirements models (RMF)* represents real-world knowledge explicitly in the requirements specification [9]. In [1] the authors argue that conventional software engineering and knowledge engineering are complementary and both essential for developing the increasingly larger systems of today. They propose a single common life-cycle methodology. These approaches, however, do not offer support at the implementation level.

6. CONCLUSION

We are currently halfway through this research project. The exploratory phase is coming to an end and all the ideas that we picked up on the way are beginning to crystallize into a coherent research topic, goal and plan. It is time to make some final decisions before developing an artifact that will be used as proof of concept. Once this artifact, a development environment in this case, is tuned at the hand of the academic case studies described earlier, it is ready to be used with an industrial case in order to validate our work.

7. ACKNOWLEDGMENTS

We would like to thank the reviewer of this paper for the valuable comments and suggestions.

8. REFERENCES

[1] F. Alonso, N. Juristo, J. L. Maté, and J. Pazos. Software engineering and knowledge engineering: Towards a common life cycle. *The Journal of Systems and Software*, 33:65–79, 1996.

[2] Aspect-Oriented Software Development. http://www.aosd.net/.

[3] Lodewijk Bergmans and Mehmet Aksit. Composing crosscutting concerns using composition filters. *Communications of the ACM*, 44(10):51–57, 2001.

[4] G. Booch. *Object-Oriented Analysis and Design with Applications*. Benjamin/Cummings, 1994.

[5] M. Casanova, M. D'Hondt, and T. Wallet. Explicit domain knowledge in geographic information systems. In *Proceedings of the 14th Conference on Software Engineering and Knowledge Engineering (SEKE '01)*. Knowledge Systems Institute, 2001.

[6] M. D'Hondt and T. D'Hondt. Is domain knowledge an aspect? In *ECOOP 99, Workshop on Aspect-Oriented Programming*, 1999.

[7] M. D'Hondt, W. De Meuter, and R. Wuyts. Using reflective logic programming to describe domain knowledge as an aspect. In *First Symposium on Generative and Component-Based Software Engineering (to appear)*, 1999.

[8] E. W. Dijkstra. *A Discipline of Programming*. Prentice-Hall, 1976.

[9] S. J. Greenspan, J. Mylopoulos, and A. Borgida. Capturing more world knowledge in the requirements specification. In *Proceedings of the 6th International Conference on Software Engineering (ICSE '82)*, 1982.

[10] W.L. Hürsch and C.V. Lopes. Separation of concerns. Technical report, North Eastern University, 1995.

[11] IBM. *Business Rule Beans*. http://www.research.ibm.com/AEM/brb.html.

[12] IBM. *CommonRules*. http://www.research.ibm.com/rules/commonrules-overview.html.

[13] I. Jacobson, M. Christerson, P. Jonsson, and G. Overgaard. *Object-Oriented Software Engineering*. Addison-Wesley, 1992.

[14] K. Kang, S. Cohen, J. Hess, W. Nowak, and S. Peterson. Feature-oriented domain analysis (foda) feasibility study. Technical report, Software Engineering Institute, Barnegie Mellon University, Pittsburgh, Pennsylvania, 1990.

[15] Gerti Kappel, S. Rausch-Schott, Werner Retschitzegger, and Markku Sakkinen. From rules to rule patterns. In *Conference on Advanced Information Systems Engineering*, pages 99–115, 1996.

[16] G. Kiczales, E. Hilsdale, JJ. Hugunin, M. Kersten, J. Palm, and W. Griswold. An overview of aspectj. In *Proceedings of the 15th European Conference on Object-Oriented Programming (ECOOP '01)*, 2001.

[17] Karl Lieberherr, Doug Orleans, and Johan Ovlinger. Aspect-oriented programming with adaptive methods. *Communications of the ACM*, 44(10):39–41, 2001.

[18] K. Van Marcke. *The Use and Implementation of the Representation Language KRS*. PhD thesis, Vrije Universiteit Brussel, 1988.

[19] W. De Meuter, M. D'Hondt, S. Goderis, and T. D'Hondt. Reasoning with design knowledge for interactively supporting framework reuse. In *Proceedings of the Second International Workshop on Soft Computing Applied to Software Engineering (SCASE '01)*, pages 31–36, 2001.

[20] J.M. Neighbors. Draco: A method for engineering reusable software systems. In *Domain Analysis and Software Systems Modeling*. IEEE Computer Society Press, 1991.

[21] Harold Ossher and Peri Tarr. Using multidimensional separation of concerns to (re)shape evolving software. *Communications of the ACM*, 44(10):43–50, 2001.

[22] D. L. Parnas. On the criteria to be used in decomposing systems into modules. *Communications of the ACM*, 15(12):1053–1058, 1972.

[23] E. Pulvermueller, A. Speck, M. D'Hondt, W. De Meuter, and J. O. Coplien. Report from the ecoop2001 workshop on feature interaction in composed systems. In *Workshop Reader of the 15th European Conference on Object-Oriented Programming (ECOOP '01)*. Springer-Verlag, 2001.

[24] Rational Software Corporation. *Rational Unified Process*. http://www.rational.com/products/rup/index.jsp.

[25] A. Th. Schreiber, J. M. Akkermans, A. A. Anjewierden, R. de Hoog, N. R. Shadbolt, W. Van de Velde, and B. J. Wielinga. *Knowledge Engineering and Management: The CommonKADS Methodology*. MIT Press, 2000.

[26] P. Steyaert. *Open Design of Object Oriented Languages*. PhD thesis, Vrije Universiteit Brussel, 1994.

[27] P. Tarr, M. D'Hondt, L. Bergmans, and C. Videira Lopes. Report from the ecoop2000 workshop on aspects and dimensions of concern: Requirements on, and challenge problems for, advanced separation of concerns. In *Workshop Reader of the 14th European Conference on Object-Oriented Programming (ECOOP '00)*, pages 203–240. Springer-Verlag, 2000.

[28] K. De Volder. *Type-Oriented Logic Meta Programming*. PhD thesis, Vrije Universiteit Brussel, 1998.

[29] T. Wallet, M. Casanova, and M. D'Hondt. Ensuring quality of geographic data with uml and ocl. In *Third International Conference on the Unified Modeling Language (<<UML >>2000)*, pages 225–239. Springer-Verlag, 2000.

[30] C. A. Welty. *An Integrated Representation for Software Development and Discovery*. PhD thesis, Rensselaer Polytechnic Institute, 1995.

[31] R. Wuyts. *A Logic Meta-Programming Approach to Support the Co-Evolution of Object-Oriented Design and Implementation*. PhD thesis, Vrije Universiteit Brussel, 2001.

[32] R. Wuyts and K. Mens. Declaratively codifying software architectures using virtual software classifications. In *Proceedings of TOOLS Europe'99*, 1999.