Making Software Knowledgeable

[Extended Abstract]

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

The complexity of software domains – for example the financial industry, television and radio broadcasting, hospital management and rental business – is steadily increasing and knowledge management of businesses is becoming more important with the demand for capturing business processes. Software domains such as the ones mentioned above, are inherently knowledge-intensive but this domain knowledge is often hard to detect in the resulting application. This is because the domain knowledge is not separately and explicitly dealt with, but implemented together with the implementation strategy of the application. This also results in changes to the domain knowledge propagating to the implementation strategy. Hence, current software engineering practices result in software applications where domain knowledge is implicit and tangled in the implementation strategy, thus encumbering understandability, maintenance, adaptability, and reuse. 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 not a domain expert has to concentrate on two aspects of the software at the same time, and moreover manually compose them. This violates the principle of *separation of concerns* [5] [9], that states that the basic algorithm should be separated from other concerns or aspects.

A category of applications where separating explicit domain knowledge would be advantageous is concerned with support of a real-world task and management of real-world information, such as software for banks, hospitals or television broadcasters. For example, in the latter domain there exists software for the support of seasonal and daily planning of the broadcasts. Such an application typically consists of domain knowledge about the kinds of programs that exist (films, series, commercials) and planning rules and constraints (children's programs should be scheduled before 8pm, moving a program results in moving the commercials that are scheduled in its breaks, a film should be broadcast in the period for which its contract is valid). This domain knowledge is typically tangled with the implementation strategy, which is concerned with for example the user interface and persistency. These are appropriately implemented using object-oriented programming and techniques such as design patterns (model-view-controller, bridge) [8], whereas the domain knowledge ideally should be expressed in a declarative medium using rules and constraints.

Another example is distributed applications, where the domain knowledge of the application and distribution issues such as transactions, replication, remote procedure calling and concurrency control are also difficult to separate.

The goal of this research is to express domain knowledge in software applications explicitly and as separated as possible from the implementation strategy. Athough some (domain) knowledge is notoriously hard to elicit and capture, as was discovered in building expert systems, the domain knowledge we intend to make explicit is quite tangible as is illustrated by the aforementioned examples. In fact, the domain knowledge is currently "implemented" using a (object-oriented) programming language. When expressed in a suitable medium, domain knowledge consists of concepts and relations between the concepts, constraints on the concepts and the relations, and rules that state how to infer new concepts and relations [12].

We are inspired by Aspect-Oriented Programming [10] [1], where *aspects* such as error handling, error reporting, synchonisation and so on, are expressed in an *aspect language* separately 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 places in the implementation strategy where the aspect should be inserted. An original contribution of this research is to consider domain knowledge as an aspect [6] [7]. 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.

Our research hypothesis is *Expressing domain knowledge of a software application explicitly and separately in a suitable medium alongside 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 [15] 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, modularity in software applications where the parts are as independent and loosely coupled as possible improves reusability.

The most tangible contribution of this research will be a programming environment where it is possible to express domain knowledge and implementation strategy separately, both in a suitable medium. For expressing domain knowledge, we are exploring existing frame-based knowledge representation languages that employ production rules, developed in the field of AI. The implementation strategy will be implemented using a standard object-oriented programming language. Hence, we will not contribute to advances in either one of these areas, but more to their combination. As mentioned earlier, we believe that composing domain knowledge with the implementation strategy is in some cases similar to aspect weaving. Moreover we claim that this process is knowledge-intensive (as is also shown in [4]). Therefore we will investigate the advantages of using the same knowledge representation language as meta-language for guiding the composition. In this configuration, a symbiosis between the chosen knowledge representation language and the object-oriented programming language is indispensable for fluently addressing elements of one language in the other. Research on symbiosis between two object-oriented languages is already conducted in [13]. A similar configuration was already successfully developed at our computer science department, more specifically logic meta-programming [4] [17] 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 [16]. Moreover, a simple prototype of a rule-based meta-language for an object-oriented base language is also developed in [3] for reasoning with design knowledge for interactively supporting framework reuse. These past experiences offer an excellent starting point for the artefacts that will have to be developed in the context of this research.

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 organising workshops through the co-organisation of the ECOOP '00 Workshop on Aspects and Dimensions of Concerns [14] and the ECOOP '01 Workshop on Feature Interaction in Composed Systems [11], we will now organise the workshop on Knowledge-Based Object-Oriented Software Engineering at ECOOP '02 [2]. Participation through technical papers and posters will be continued.

1. REFERENCES

[1] Web site of Aspect-Oriented Software Development: `http://www.aosd.net`

[2] Web site of the ECOOP workshop on Knowledge-Base Object-Oriented Software Engineering: `http://infoweb.vub.ac.be/ mjdhondt/KBOOSE/index.htm`

[3] De Meuter, W., D'Hondt, M., Goderis, S.: Reasoning with design knowledge for interactively supporting framework reuse. In Proceedings of the Second International Workshop on Soft Computing Applied to Software Engineering (2001)

[4] De Volder, K.: Type-Oriented Logic Meta-Programming. PhD thesis, Vrije Universiteit Brussel, Belgium (1998)

[5] Dijkstra, E. W.: A discipline of programming. Prentice-Hall (1976)

[6] D'Hondt, M., D'Hondt, T.: Is domain knowledge an aspect? Aspect-Oriented Programming Workshop, ECOOP (1999)

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

[8] Gamma, E., Helm, R., Johnson, R., Vlissides, J.: Design Patterns, Elements of Reusable Object-Oriented Software. Addison-Wesley (1995)

[9] Hürsch, W.L., Lopes, C.V.: Separation of Concerns. Technical report, North Eastern University (1995)

[10] Kiczales, G., Lamping, J., Mendhekar, A., Maeda, C., Lopes, C., Loingtier, J., Irwin, J.: Aspect-oriented programming. In Proceedings of ECOOP (1997)

[11] Pülvermüller, E., Speck, A., D'Hondt, M., De Meuter, W., Coplien, J.: Report from the ECOOP2001 Workshop on Feature Interaction in Composed Systems. In Workshop Reader of ECOOP, Springer-Verlag (2001)

[12] Schreiber, G., Akkermans, H., Anjewierden, A., de Hoog, R., Shadbolt, N., Van de Velde, W., Wielinga, B.: Knowledge Engineering and Management: The CommonKADS Methodology. MIT Press (2000)

[13] Steyaert, S.: Open design of object-oriented languages, a foundation for specialisable reflective language frameworks. PhD thesis, Vrije Universiteit Brussel, Belgium (1994)

[14] Tarr, P., D'Hondt, M., Bergmans, L., Lopes, C.V.: 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 ECOOP, Springer-Verlag (2000)

[15] Welty, C. A.: An Integrated Representation for Software Development and Discovery PhD thesis, Rensselaer Polytechnic Institute, USA (1995)

[16] Wuyts, R.: Declaratively codifying software architectures using virtual software classifications. In Proceedings of TOOLS Europe (1999)

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