Explicit Domain Knowledge in Software Engineering

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

January 6, 2002

1 Research Areas

This research is about making knowledge about the domain of software applications explicit and separating it from other concerns of the software, thus enhancing software understandability, software maintenance, and software reuse. To achieve this we use existing knowledge representation technologies from AI in state-of-the-art software engineering. Our approach is not unlike the principles Aspect-Oriented Software Development, where aspects are separated from the other implementation components and expressed explicitly.

2 Technical Problem

The complexity of software domains – such as 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. The domains of many software applications, such as the ones mentioned above, are inherently knowledge-intensive and this knowledge is often not explicitly dealt with in software development. 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 [22].

Current software engineering practices result in software applications where domain knowledge is implicit and tangled in the implementation strategy. This encumbers understandability, maintenance, adaptability, reuse - in short evolution of the software. This is because
the domain knowledge is difficult to identify and locate, and because changes to the domain knowledge propagate to the implementation strategy. Because this evolution cannot be performed locally and in a controlled manner, it is difficult, time-consuming, error-prone and thus expensive. 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 [10] [15], that states that the basic algorithm should be separated from other concerns or aspects.

The cause of this problem can be found in current software development methodologies. Older software development methodologies such as OOSE [16] and Booch [2], and even the standard Rational Unified Process [21] employ a use case-driven approach as a result of which domain knowledge and implementation strategy are modeled together from the start. Other approaches to software development such as domain engineering [17] [19], allow anticipated variations in (among others) domain knowledge, but unanticipated changes require an entirely new domain design and implementation.

There exist a number of techniques to improve evolution of software applications. Object-oriented programming [5] introduces encapsulation and abstraction. Framework technology [14] only takes anticipated variations into account. Finally design patterns [13] are descriptions of recurring designs as they are found in existing software applications. Design patterns can be used up to some point to make explicit and factor out domain knowledge, but the disadvantage is they are descriptive and non-enforcing. Furthermore, a number of techniques can and have been used in software engineering to capture domain knowledge such as conceptual modeling [9] and business rules [6]. However, it is unclear how these artefacts can survive at the implementation level where they are translated to the paradigm of the programming language thereby discarding the declarative nature of domain knowledge.

3 Research Hypothesis

The goal of this research is to express domain knowledge in software applications explicitly and as separated as possible from the implementation strategy. We are inspired by Aspect-Oriented Programming [18] [1], where 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 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 [11] [12]. Our research hypothesis is thus

Expressing domain knowledge of a software application explicitly 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.

In order to prove this hypothesis, we will investigate how to express domain knowledge explicitly and how to compose it with the implementation strategy in order to get the required behaviour of the software application. The domain knowledge has an operational role to play together with the implementation strategy which will be expressed in a standard (object-oriented) programming language. Composition of the domain knowledge and the implementation strategy will be crucial, which leads us to our second hypothesis:

Composition of an aspect with a component - more often referred to as weaving - is a knowledge-intensive task and will benefit from being expressed explicitly in a suitable medium.

4 Contributions

The first contribution of this research should be a raised awareness of the implicit knowledge that is inherently present in software applications, such as domain knowledge and composition knowledge which are addressed in this research. Moreover, this work should clearly show that making this knowledge explicit has substantial benefits for software engineering. In devising a solution to the problem, we will also make a contribution in combining already existing knowledge engineering techniques and current software engineering, more specifically knowledge representation languages and object-oriented programming languages respectively. This combination will be investigated on a conceptual level, but will also result in a specific research artefact for proof of concept which will result in technological contributions. Furthermore, we expect a limited but important contribution with respect to software engineering methodologies, which undoubtedly will be affected by this research. Finally, if as a by-product the secondary hypothesis can also be proven, there will be an additional contribution to the field of Aspect-Oriented Software Development.

5 Methods and Plan

For expressing domain knowledge, we will explore existing frame-based knowledge representation languages that were developed in the field of AI, such as KL-ONE [3] and Classic [4]. Since we believe that composing domain knowledge with the implementation strategy is also knowledge-intensive we will investigate the advantages of using the same knowledge representation language as meta-language for expressing 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 [23]. A similar configuration was already successfully developed at our computer science department, more specifically logic meta-programming [8] [27] 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 [25] [26]. Moreover, a simple prototype of a rule-based meta-language for an object-oriented base language is also developed in [7] 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 artefact we will take the industry as laboratory approach. Through contacts with industrial partners, several case studies will be set up. One of them is a Belgian company that has been making software for managing everything related to television or radio broadcasting for a decade and count among their customers most major European broadcast companies as well as others outside Europe. They have to cope with exactly the same problems as we are trying to tackle in this research.

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 [24] and the ECOOP '01 Workshop on Feature Interaction in Composed Systems [20], we will now organise the workshop on Knowledge-Based Object-Oriented Software Engineering at ECOOP '02. Participation through technical papers and posters will be continued.

References


