EXPLICIT DOMAIN KNOWLEDGE REPRESENTATION-AOSD APPROACH

Rajeev Ranjan*

ABSTRACT

This research is concerned with representing knowledge about the domain of AOSD 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 behaviour, we are inspired by the principles **Aspect-Oriented Software Development**.

Keywords: - *Explicit, Domain Knowledge representation, Aspect-oriented software development.*

***Computer Science, S.R.K.G. College, Sitamarhi/ B.R.Bihar University, Bihar, India**

1.Introduction:-

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.

a. 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.

b. 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. 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 and Booch** , and even the newer standard Rational Unified Process 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 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.

2. Examples of domain Knowledge representation AOSD:-

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.

2.1 E-Commerce

The domain of a 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 15 products, he or she is entitled to a 10% discount on the next order", "if it is Holi, 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 Holi?

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 behaviour 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 understand ability, software maintenance and software reuse.

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 because they achieve exactly this. In these technologies, aspects such as error handling, error reporting, and persistence and so on, are expressed in an aspect language separate from the implementation strategy.

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
- Prototype-based frames, as in KRS

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 , AspectJ, Composition Filters and Demeter.** 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.

5. RELATEDWORK

Apart from the aforementioned technologies and approaches that will be actively (re)used in this research project, some other work is also relevant. Bank Objects is a project we were involved in together with an industrial partner specialized in producing and maintaining digital Banking data to be used in Bank 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 Banking 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 Banking data via API calls.

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. However, the business rules are still tangled in the implementation strategy and not ex-

pressed declaratively. We still need to look into approaches such as Common Rules and Business Rule Beans.

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.

7. REFERENCES

1. Aspect-Oriented Software Development. http://www.aosd.net/.
2. G. Booch. Object-Oriented Analysis and Design with Applications. Benjamin/Cummings
3. IBM. Business Rule Beans. http://www.research.ibm.com/AEM/brb.html.
4. IBM. CommonRules. http://www.research.ibm.com/rules/commonrulesoverview.html.