Error checking in the Process Interchange Format(PIF) Ontology *

Ioannis Kalfoglou† David Robertson†
† Department of Artificial Intelligence, University of Edinburgh,
80 South Bridge, Edinburgh, EH1 1HN, Scotland
Email: {yannisk,dr}@dai.ed.ac.uk

Abstract

It is widely accepted that conceptual errors in the early stages of a software process are often the most pernicious. This is because they reflect a misunderstanding of the domain of application, hence we can have a conceptual error in a description which "looks good" from an abstract mathematical point of view. A section of the Knowledge Based Systems community is now beginning to forge agreements on the use of terminology for particular domains. Amongst the more formal of these is the Process Interchange Format. The existence of these formal agreements allows limited forms of checking for conceptual errors as a supplement to the normal testing process. This paper describes a method for performing such checks.

1 Introduction

Like all other technologies, software development needs blueprints from which developers can accurately work. These blueprints are precise and independent descriptions of the desired program behaviour. One form of blueprint is a formal description expressed in logic, the purpose of which is to[3] "define all required characteristics of the software to be implemented, and thus form the starting point of any software development process". The precise role of formality in software development is still a matter of debate(see for example [7], [4]).

Formal methods, support tests for some forms of completeness and consistency and there exist methods for methodological refinement of some types of formal specification into executable form, via appropriate interpreters. This provides an additional advantage, an executable specification represents not only a conceptual, but also a behavioural model of the software system to be implemented[3], allowing early validation. This might increase the correctness and the reliability of the software, and reduce development costs and time. The value of this approach in clarifying modelling concepts has been noted by the SP community(see for example, [10]).

2 Errors in specifications

It is difficult, even with executable formal specifications, to make blueprints error free. In fact, it is easy(maybe easier) to make errors in this phase with pernicious side-effects for the remainder of the life-cycle. That is because they may not be detected by those who use the formal blueprint in subsequent design and may affect the functionality of entire systems. As stated in [1], "It is often stated that Prolog programs can be seen as executable specifications. If the programmer can get the specification right then there will be no program code errors. But it is not so simple to get the specification right..."

The problem of errors in the early phases of software development is treated with awe by the community[7] largely because the most damaging errors are conceptual errors. A conceptual error occurs when we misuse the terminology of a domain, thus describing a problem in a way which is inappropriate to that domain. Such errors are not detectable without knowledge of the domain itself and, since such knowledge isn't normally in the specifications, we can have deep conceptual errors in specifications which are mathematically elegant.

The Knowledge Based Systems(KBS) community has had to face this problem directly because it relies on describing domains. However, the methods used in KBS to help detect errors have been applied only sparingly to formal specifications-perhaps because these methods tend to be approximate and heuristic rather than deductive. In other words, they help suggest potential errors but don't prove their existence.

In the next few sections we demonstrate how we can improve this situation with the use of ontologies as a basis of our error-checking method.

*This paper submitted to the 6th European Workshop on Software Process Technology(EWSPT-6)

3 Emerge of Ontologies

Although, there are numerous ways of constructing an ontology(see [12]), the purpose of an ontology is to record a set agreements by a group of engineers about how they describe a given domain. An ontology prescribes a formal syntax and may also define axioms constraining the way in which it will be used. Potentially, such axioms could be used to help protect specifications against conceptual errors. However, we are aware of no system which deploys ontological axioms to check for such errors. This paper describes our early efforts in building this kind of system in the process modelling domain, using the Process Interchange Format(PIF) ontology[5].

The aim of PIF is to develop an interchange format to help automatically exchange process descriptions among a variety of business modelling and support systems. The benefit of using PIF is that it acts as an interlingua among a variety of systems, reducing the number of translators needed from $O(n^2)$ to $O(n)$. The core of PIF consists of the minimal sets of constructs necessary to translate simple but non-trivial process descriptions. In addition, PIF can be extended to represent local needs of individual groups with the use of Partially Shared Views(PSV) described in [6]. The PIF's ontology focal point is a process, which is a set of activities that stand in certain relations to one another and to objects over timepoints.[1]

4 Checking conceptual errors

Our method, which illustrated in Figure 1, starts from the available syntax and axioms[2] of PIF. We can then follow either of two paths. The left path is followed by those who develop the implementation and the right path by those who will check it with respect to error occurrences.

The developers path starts with a manual implementation of the desired specification representing software processes. This implementation conforms to the syntax of PIF but may contain conceptual errors(some of which could be detectable via the PIF axioms). The specification is then translated to an executable formal specification, in our case using Prolog. We could test this specification in the normal way(by giving Prolog goals) but this would not include checks against the ontological axioms. To include these we need to look at the other elements of the diagram.

The right path starts with the transformation of the ontology's axioms to First Order Predicate Calculus(FOPC). We then translate automatically this axiomatisation to Normal Form using standard *Lloyd-Topor* transformations [8]. Finally the translated axioms are stored in a database of constraints expressed as Prolog clauses.

The two paths converge at a meta-interpreter error-checking program which is used in order to detect errors with respect to ontological constraints.

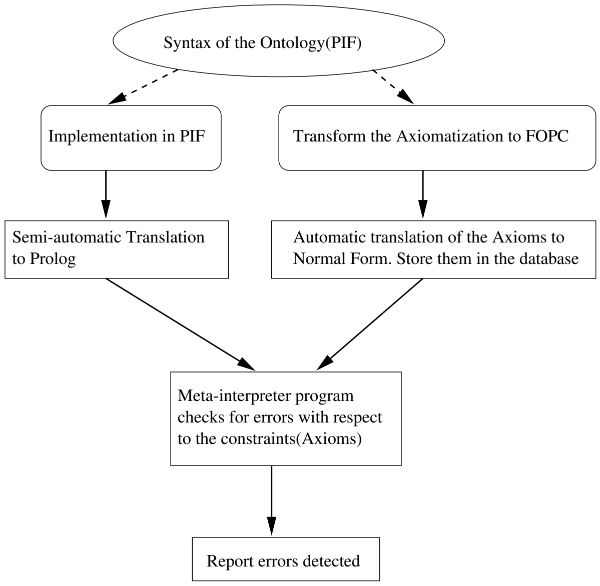

Figure 1: Method for error detection

The role of meta-interpreters in logic programming is normally to describe explicitly some pattern of inference. In our case, this description is the normal Prolog mechanism for satisfying Prolog goals - so our interpreter replicates the normal behaviour experienced by someone checking the specification. However, the mechanism is extended by checking, whenever the solution to a goal is contained in the execution, whether relevant ontological constraints apply and, if so, whether they are satisfied. Thus our interpreter can tell us, for each individual test whether any part of it strayed outside the formalised ontological constraints.

Notice that this does not mean that our specification is error free. They could be conceptual errors which we fail to reveal because we choose the wrong tests. Our mechanism, as is normal, can show errors but cannot prove their absence.

In the following section we present an example case, simplified in order to demonstrate the operation of the method.

[1] PIF examples can be found in [5] and in [11]
[2] The axiomatisation was proposed in [9]

5 Example case

We have chosen one of the proposed scenarios for the PIF evaluation, the *"supply chain scenario"*.[3] The main context for this proposed scenario involves the development and coordination of supply chain processes between a manufacturer, retailer, distributor, warehouse company and transportation company[2]. For the sake of brevity we will present the detection of an error at the process 'process document request' in the transportation company. The description of this process is given in [11]:

> "...Shipping orders are received at the documentation department by a manager. The manager delegates the task to an employee. The delegated employee completes the shipping forms and the customs documents and returns them to the manager for approval. The manager then approves the forms and sends a notice of completion".

One of the PIF axioms states that "the only agents which are allowed to perform an activity are those who have the appropriate capability that permits them to have a performs relation to that activity". The axiom is given as:

$performs(Actor, Activity) \rightarrow object(Actor)$
$\wedge activity(Activity) \wedge (capable\_of(Actor, Capability)$
$\wedge refers\_to(Capability, Activity))$.

Another PIF axiom stipulates that a 'performs' relation must be defined over objects and activities. An actor is a specialisation of an object. Lets assume that the designer of this process chooses to represent the 'performs' relation in the following way:

$performs(Actor, Activity) \rightarrow$
$object(Actor) \wedge activity(Activity)$.

In addition, our program database of the process contains facts concerning the rest of the process entities in the following format:

$object(manager)$.
$object(employee)$.
$activity(approve\_forms)$.
$activity(filling\_forms)$.
$capable\_of(manager, approves)$.
$capable\_of(employee, filling\_forms)$.
$refers\_to(approves, approve\_forms)$.
$refers\_to(filling\_forms, filling\_forms)$.
...

[3] This scenario was adopted from the Workflow Management Coalition's(WfMC) workflow interoperability demonstration presented at the 1996 Business Process and Workflow Conference in Amsterdam[2]

An error might occur in our specification, where we are allowed to assign an agent to perform an activity that it has not the capability to perform. This error could be demonstrated by the following erroneous Prolog goal:

$performs(employee, approve\_forms)$.

the result of which will be a correct answer since there is nothing to prevent the occurrence of this error, although it is syntactically correct. Assuming the ontological axioms given above, our meta-interpreter can detect this error.

The result of the error-checking mechanism pinpoints the discrepancy found:

```
Conceptual error detected at:
(capable_of(employee,filling_forms),
refers_to(filling_forms,approve_forms)).
```

where the first line shows that an *employee* is capable of *filling_forms* and, if so, then it must has a 'performs' relation to the *filling_forms* activity according to the ontological axiom. However, as we can see from the second line a discrepancy detected where the *filling_forms* capability of the employee, does not refer to the correct activity, which is the *filling_forms* but to *approve_forms* which is the activity that the *manager* performs.

6 Conclusions

Our motivation is to present a method which will be capable of detecting errors at the early phases of a system design. In particular, errors that are not easily detected such as conceptual errors, which plague the software design process. Our method is capable of detecting such errors as long as the designer of the processes decides to use a formal syntax and the supported axiomatisation provided by an ontology. In the area of SPs such an ontology is PIF which provides the additional flexibility of acting as an interlingua between a variety of PM languages and tools. The method of meta-interpretation which we have described is "lightweight" to use because those testing specifications in Prolog give goals in the normal way, with checking for conceptual errors being given as a supplement to the results in satisfying the test goals.

Acknowledgements

The research described in this paper is supported by a European Commission's Marie Curie Fellowship, under the TMR contract: ERBFMB-ICT-97-1955 for the first author and a EPSRC IT Advanced Fellowship for the second author.

References

[1] P. Brna, M. Brayshaw, A. Bundy, M. Elsom-Cook, P. Fung, and T. Dodd. An Overview of Prolog Debugging Tools. Research paper 398, Department of Artificial Intelligence, University of Edinburgh, 1988.

[2] Workflow Management Coalition. Workflow interoperability demonstration. Technical report presented at business process and workflow conference, Amsterdam, October 1996. http://www.wfmc.org.

[3] N. Fuchs. Specifications are (preferably) executable. *Software Engineering Journal*, pages 323–334, September 1992.

[4] N. Gehani and D. McGettrick. *Software Specification Techniques*. Addison-Welsey, 1986.

[5] J. Lee, M. Gruninger, Y. Jin, T. Malone, A. Tate, and G. Yost. The PIF Process Interchange Format and framework, version 1.1. Technical report working paper no 194, MIT center for Coordination Science, 1996.

[6] J. Lee and T. Malone. Partially Shared Views: A Scheme for Communicating between Groups Using Different Type Hierarchies. *ACM Transactions on Information Systems*, 8(1):1–26, 1990.

[7] B.H Liskov and J. Guttag. *Abstraction and Specification in Program Development*. MIT Press, 1986.

[8] L.W. Lloyd. *Foundations of Logic Programming*. Symbolic Computation. Springer-Verlag, 1984.

[9] C. Menzel. An Axiomatization of the PIF Core. 3rd PIF Workshop, Stanford, USA, August 1997.

[10] C. Montangero. Open Issues in the Design of PM Languages. In Wilhelm Schafer, editor, *4th European Workshop on Software Process Technology, Noordwijkerhout, The Netherlands*, page 93. Springer, April 1995.

[11] S.T. Polyak. A Supply Chain Process Interoperability Demonstration using the Process Interchange Format(PIF). Research paper 889, Department of Artificial Intelligence, University of Edinburgh, February 1998.

[12] M. Uschold and M. Gruninger. Ontologies: principles, methods and applications. *The Knowledge Engineering Review*, 11(2):93–136, November 1996.