Identifying and Implementing Relationships

Stephen Nelson, David J Pearce (Advisor), and James Noble (Advisor)

{stephen,djp,kjx}@mcs.vuw.ac.nz
Victoria University of Wellington, New Zealand

Abstract. Relationships have been an essential component of OO design since the 90s and, although several groups have attempted to rectify this, mainstream OO languages still do not support *first-class* relationships. This requires programmers to implement relationships in an ad-hoc fashion which results in unnecessarily complex code. This abstract describes research towards first-class language support for relationships using collections. In particular, we discuss studies profiling the use of equality and collections in existing Java programs to determine how relationships are currently implemented. The results of these studies should allow language designers better criteria for evaluating relationship support, and we discuss example changes.

1 Problem Description

Object-oriented practitioners are frequently faced with a dilemma when they design and implement object-oriented systems: modelling languages describe object systems as a graphs of objects connected by relationships [2, 7], but most object-oriented languages have no explicit support for relationships. This results in a trade-off between high-level models which are de-coupled from their implementations and low-level models which are confusing to use as they contain irrelevant detail.

There have been many proposals for adding relationship support to object-oriented languages. Rumbaugh proposed a language with relationship support in 1987 and there has been a recent resurgence interest with proposals for language extensions [1, 3, 8] and library support [5, 6].

The potential benefits of such support are clear: improved traceability between design and implementation, reduced boilerplate code, better program understanding by programmers, and the opportunity to introduce new high-level language features such as queries, relationship (function) operations, and program structure constraints.

In spite of the clear advantages of relationship support none of the solutions proposed so far have achieved widespread use, and while this may be due to limited time and exposure we believe that existing solutions fail to capture the intent of object-oriented models, and so do not provide the necessary tools to replace the relationships already existing in programs.

2 Goal Statement

This work plans to identify how relationships are implemented in existing programs and provide criteria for evaluation proposals for supporting them in new languages.

We will produce these criteria using two extensive runtime studies. The first study will examine the use of equality and immutability in Java programs, in particular in the context of collections. The second study plans to identify particular patterns in running programs which indicate of relationships, and a taxonomy of those relationships. These studies have been inspired by a detailed examination of existing proposals for relationships. At the conclusion of these studies, the results will be used to provide particular feedback on existing proposals for supporting relationships and develop our own proposal if there are particular criteria which none of the existing proposals satisfy.

3 Method

We have conducted an extensive survey of existing work in this area, and produced several prototypes to examine the importance of particular language features on relationships. Major observations include that almost all interesting relationships are many-to-many, and almost always involve the use of collections in their implementations. This leads to a tight coupling between decisions made by collection implementers and a language's support for relationships which is perhaps surprising. In particular, in Java the decision to use a programmer defined equality.

3.1 Experiment 1

A survey of existing Java programs using a custom profiling tool implemented in AspectJ to examine correlations between the use of equality, immutability, and collections. This work has been completed and published [4]. The profiler is interesting for several reasons: it actively injects code to determine when particular properties (such as equality) change, and it's implemented in a popular language (AspectJ) without the use of additional VM tools, so it could be repeated for other languages.

3.2 Experiment 2

A study of existing Java programs using a custom profiler implemented in a mixture of c and Java to provide whole-program traces. These will then be analysed later without significantly impacting the performance of the program. We particularly interested in seeing whether there are some relationships which are entirely structural (accessed by external objects) and others which are entirely behavioural (accessed only by themselves). These types of relationships could not be identified by snapshot-based tools, and would be very difficult to identify using static analysis. This study is currently in progress.

3.3 Criteria for Relationship Support

We will use the results of our studies to produce a list of salient features of good relationship systems by highlighting certain features of the Java ecosystem which are used by relationships in implementation and discussing the merits of existing proposals in light of these criteria. If appropriate, we will outline how a new language could make use of these observations to remain 'relationship-friendly'.

4 Validation

Our goal is that both of the studies should be sufficiently self-contained that they are independently publishable. This will provide invaluable feedback on the individual experiments and hopefully stimulate discussion which we can feed into our selection of criteria. The criteria for relationships, and any language proposal we make will be supported by the results we have obtained in the experiments. Once completed, there will be a criteria for evaluating relationship support, and indeed, we will use them to retrospectively examine existing work.

These studies will be based around a corpus of Java programs and may be vulnerable to bias as a consequence. It would be valuable to repeat this work in other languages and systems, but we believe that implementing the requisite profiling tools and compiling suitable corpuses of programs to study would be well outside the scope of a PhD. However, Java is a popular and influential language with a diverse audience, which will hopefully help to mitigate any bias, and our work should be replicable in other languages.

References

1. Balzer, S., Gross, T. R., and Eugster, P. A relational model of object collaborations and its use in reasoning about relationships. In *ECOOP* (2007).
2. Beck, K., and Cunningham, W. A laboratory for teaching object oriented thinking. In *OOPSLA* (1989), ACM, pp. 1–6.
3. Bierman, G. M., and Wren, A. First-class relationships in an object-oriented language. In *ECOOP* (2005), pp. 262–286.
4. Nelson, S., Pearce, D. J., and Noble, J. Understanding the impact of collection contracts on design. In *TOOLS Europe* (2010).
5. Østerbye, K. Design of a class library for association relationships. In *LCSD* (2007).
6. Pearce, D. J., and Noble, J. Relationship aspects. In *AOSD* (2006), ACM Press, pp. 75–86.
7. Rumbaugh, J., Jacobson, I., and Booch, G. *The Unified Modelling Language Reference Manual.* Addison-Wesley, 1999.
8. Vaziri, M., Tip, F., Fink, S., and Dolby, J. Declaritive object identity using relation types. In *ECOOP* (2007), pp. 54–78.