

A Multiparadigm Programming Case Study with RESOLVE Components

Matthew Thornton (thorntom@vt.edu)

Abstract—A formal specification language for programs written in a multiparadigm fashion would combine the best of two worlds: a programming platform that allowed developers the freedom to write programs in whatever methodology fits the problem with an unambiguous specification for the program behavior. A proof of concept for a behavioral specification language has already been devised by extending the syntax of the Java Modeling Language. A theoretical validation of this language is already underway. However, a more—practical validation of the language would be ideal, one that would show that the specification language would be “useful” to developers and specifiers alike.

To demonstrate this, a case study will be developed to show that the programs specified using behavioral specification language devised (JML-MP) have similar attributes in terms of specified collected metrics as an existing specification language—in this case, RESOLVE. This paper will discuss some of the issues that will need to be addressed and raise discussion points for discussions during the workshop.

I. INTRODUCTION

A multiparadigm programming language is a programming language that allows a developer to program in more than just one paradigm. Programming languages such as Oz, Leda, JavaMP, and others like it that include Object-Oriented, Functional, Logical, and Imperative paradigms give the developer a rich tool chest to develop software that uses the best tool for the job [1]. At the moment, there is no behavioral specification language that allows one to specify the behavior of programs written in a multiparadigm fashion.

JML-MP is a multiparadigm behavioral specification language that allows one to model behavior of Object-Oriented, Functional, Logical, and Imperative programming paradigms in an extension of Java called JavaMP[7]. JML-MP functions basically as an extension to the Java Modeling Language (JML) [6].

A syntactic definition of the language JML-MP has been completed [9, 10] and a theoretical validation of the language against the programming language it was written for (JavaMP) is currently underway. In many cases, however, a programming language can have all of the desirable theoretical attributes of soundness and completeness and still be a

worthless language if people won’t use it. Ideally, there would be some way of comparing JML-MP against existing specification tools and techniques and see that writing specifications in that language is similar to writing specifications in another specification language. The question, then, is whether or not this is actually possible. The following sections will discuss our position on this issue and bring to light some of the issues that will need to be addressed moving forward.

II. POSITION

The position that this paper takes is that it is possible to develop a comparative study of JML-MP using specifications and implementations of components that already exist. This will be accomplished through a case study of an existing program or software component written in RESOLVE, which has many characteristics of existing specification languages [4]. A comparable multiparadigm specification and implementation will then be created and data regarding both sets of specifications/implementations will be gathered and metrics will be generated against the data and the results will be compared. Ideally, the results should show that a JML-MP-specified program or software component is comparable to the comparable RESOLVE software component.

III. RELATED WORK

The idea of comparing and contrasting *programming* languages is nothing new but evaluations of formal specification languages are much harder to come by. Consequently, it will be necessary to make use of some of the work that has already been done with evaluating programming languages. Countless books have been written on the subject [8]. The concepts discussed in books like this are questions of semantic differences, differences in features between languages, the paradigms that the language provides, and others. However, in these cases, the comparison is more of a “black and white” feature comparison. Some work has been done in the area of testing the *usability* of a language [2], but not a lot of work has been done in evaluating more empirical data that can be collected about comparing one language against another.

IV. CASE STUDY SETUP

The case study will, again, involve comparing an existing

specification and implementation of a RESOLVE software component. What follows is a brief synopsis of the current plan for the case study.

A. RESOLVE Component to be Evaluated

The RESOLVE component that was chosen to be evaluated is the *Assertion_And_Query_Machine* component. The *Assertion_And_Query_Machine* component is a unit of software that takes in as input assertions. These assertions consist of a label and a sequence of values that that label can be. After that, a structured query can be fed into the machine. The query can have bound and unbound variables in it. The machine then generates the appropriate bindings based on the query and allows the client to process the bindings by arbitrarily removing the bindings from the machine.

There are several reasons why this component was chosen. For one, the component is sufficiently long and complex so as to allow for a useful case study to be done. Another (potentially more important reason) is that the *Assertion_And_Query_Machine* component has many attributes that would lend itself to a multiparadigm design. Processing a list is something that the Functional programming paradigm is used for. Executing search and query functionality is something that immediately lends itself to a logical programming paradigm [1]. Developing a JavaMP component based on the RESOLVE component would allow for many interesting multiparadigm design issues to be addressed.

B. Multiparadigm Design and Specification

In Section A, it was stated that there were several interesting design issues that would have to be addressed in implementing a design and specification for the *Assertion_And_Query_Machine*. First of all, class design differs considerably between RESOLVE C++ and Java (and its JavaMP variant). To implement a *proper* class design in Java, appropriate design patterns [3] should be instituted, where appropriate. This brings up the immediate concern of how to integrate additional paradigms into the Java design methodology. In [5], the authors address the issue by providing several examples of patterns in a multiparadigm fashion, including iteration patterns, command patterns, and generators. These concepts and others need to be investigated to see how to best integrate them into the design of the *Assertion_And_Query_Machine* for JavaMP.

The specification of the JavaMP class(s) that are defined in creating a JavaMP counterpart to the RESOLVE component will be done using the draft version of JML-MP. This will allow us to iron out the bugs of the specification language and verify that we actually have everything we need to effectively write a specification for such a component. Many of the issues that will need to be addressed include exactly how specific a specification should be given to the lambda functions that are created as members of a class or that are parameters to member methods.

C. Data Collection

Once the design, specification, and implementation of the JavaMP counterpart to the *Assertion_And_Query_Machine* is completed, we will begin to gather data about both components. This data will include easily-accessible data such as the number of lines of code, numbers of lines of specification, number of methods, and other data. It would also include calculation of complexity measures of the implementation. For *that* matter, some form of complexity measure could be devised for the specification, as well (including some calculation involving the number of nested quantifiers, etc.). Measures taken from the specification such as the number of quantifiers, number of decision paths in the specification, and other will be included. Additionally, as much information related to the design and implementation *process* will be gathered as possible (time it took to design, specify, and implement, measures of numbers of iterations/versions, etc). This data could then be used in gathering useful information about comparing and contrasting the two specifications and their implementation.

D. Metrics Evaluation

There are a number of interesting metrics that have already been considered for the evaluation of the two specifications. The goal of the evaluation is to see that writing the specification in JML-MP is comparable to writing it in a language such as RESOLVE. What do we mean by “comparable”? It would be nice if the specifications were about the same length, complexity, difficulty to write, and other measures. Some of the metrics that would allow us to glean more information out of the data that was collected would be metrics like complexity per line of code/specification (this would give us a measure of how “dense” the specification was), how long the implementation is versus how long and/or complex the specification is, how long it took to write a line of the specification, and other calculations.

If it were possible to gather similar metrics of other RESOLVE specifications and then do a statistical analysis of the results gathered and then include the JML-MP specification in the mix and we see that the metrics and data we collected from the JML-MP specification is within tolerance or an acceptable alpha, then we could say that JML-MP is comparable to writing a specification for a RESOLVE component.

V. CONCLUSIONS AND FUTURE WORK

Coming up with a case study that shows how *similar* two languages are seems to be hard to come by. Furthermore, coming up with a way of showing how similar two *specification* languages are seems even more difficult. The case study approach that has been adopted should allow for us to see how similar the JML-MP specification is to other similar RESOLVE components.

The preceding has just been a proposal. Since the case study has not yet begun, it would be nice to get feedback on

this paper. Particularly, information in the following topic areas would be useful:

- Suggestions on design and how to incorporate other paradigms into design and implementation of a JavaMP *Assertion_And_Query_Machine* class.
- Suggestions on data points to collect in the design, specification, and implementation of the *Assertion_And_Query_Machine* class.
- Recommendations on how to do the analysis of the data and metrics collected.
- Suggested metrics to take based on the data that has been collected.

There is already a theoretical model for the viability of writing program specifications for a multiparadigm program. If we can demonstrate that it is also *practical* to write such a specification, it could go a long way to improving the multiparadigm community of adopting formal methods into their research.

REFERENCES

- [1] Budd, T., T. Justice and R. Pandey, *General-Purpose Multiparadigm Programming Languages: An Enabling Technology for Constructing Complex Systems*, Engineering of Complex Computer Systems 1995, IEEE, Ft. Lauderdale, FL, 1995, pp. 334-337.
- [2] Clarke, S., *Evaluating a New Programming Language*, Psychology of Programming Interest Group, Bournemouth University, UK, 2001.
- [3] Gamma, E., R. Helm, R. Johnson and J. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison-Wesley, Indianapolis, IN, 1995.
- [4] Heym, W. D., T. J. Long, W. F. Ogden and B. W. Weide, *Mathematical Foundations and Notation of RESOLVE--OSUCISRC-8/94-TR45*, in Department of Computer and Information Science--The Ohio State University, ed., 2000.
- [5] Knutson, C. D., T. A. Budd and C. R. Cook, *Multiparadigm Patterns of Thought and Design*, Pattern Languages of Programs Conference, Allerton Park, Illinois, 1996.
- [6] Leavens, G., E. Poll, C. Clifton, Y. Cheon, C. Ruby, D. Cok, P. Muller and J. Kiniry, *JML Reference Manual*, 2005.
- [7] Naik, R., *Multiparadigm Programming with JavaMP*, Electrical Engineering and Computer Science, Oregon State University, Corvallis, 2003.
- [8] Sebesta, R., *Concepts of Programming Languages*, Addison Wesley, 2003.
- [9] Thornton, M., *Behavioral Specification of Multiparadigm Programs*, in Stephen Edwards, ed., RESOLVE 2006 Workshop, Virginia Tech, Blacksburg, VA, 2006.
- [10] Thornton, M., *JML-MP: a Formal Specification Language for a Multiparadigm Programming Language (Abstract)*, Fall Conference of the Mid-Southeast Chapter of the ACM, Gatlinburg, TN, 2006.