A Comparison of a Rule Definition Language (RDL) and the JAVA Object Oriented Language for Implementing a Distributed System

Amarilis Cuaresma Zevallos
Advisor: Dr. Néstor Rodríguez
Software Sciences and Engineering Group
Electrical and Computer Engineering Department
Puerto Rico University, Mayagüez Campus
Mayagüez, Puerto Rico 00681-5829
amarilis@ece.uprm.edu

Abstract

The Event Rule Framework ERF was created for supporting the development of distributed systems with services and tools that specify the system behavior in a top level of abstraction based on events and rules. Through the Rule Definition Language (RDL) end users have the capability to define events, rules and event services. In this paper we present a study that compares RDL with JAVA for implementing distributed systems. The results of the study indicate that RDL requires significantly less code and less time to implement distributed systems applications based on events and rules.

1. Introduction

A Distributed System (DS) is a collection of autonomous computers with dissimilar platforms linked by a computer network and supported by software standards that enable the collection to operate as an integrated facility. These standards still lack of levels of abstraction, services and tools that enable to design, implement, monitor, debug and maintain a DS. They also lack of behavior specification. The existing environments do not allow incorporating changes in behavior dynamically. Any change in the behavior will involve changes in the implementation of functions or methods, which requires recompilation.

The Event Rule Framework (ERF) project has been developed in order to support the specification of behavior of DSs, in terms of how their components react to events. Behavior in a DS can be defined in terms of events and how the system reacts to their occurrences. These events can represent a change in the state of objects of an entire system. The rules are a high-level abstraction for specifying behavior in terms of events, conditions and actions. The heart of ERF is RUBIES (Rule Based Intelligent Event Service). RUBIES handles events using rules, which are used to specify, at a high level of abstraction, the behavior of a DS. In addition of handling events, this event service has the capabilities of creating and specifying events and rule properties, posting events, scheduling rules (e.g. active or inactive), and registering and notifying interested clients. Through the Rule Definition Language (RDL) end users have the capability to define events, rules and event services.

It is reasonable to expect that RDL be a better programming language than object oriented Java language for specifying behavior for DS. However, it is important to demonstrate how

much improvement RDL offers for specifying behavior through events and rules, how easy is it to use, how much time it takes to implement rules, how many lines of code are required and how much acceptance does it have with real users.

2. ERF: Event Rule Framework

ERF is a framework that provides a set of abstractions for specifying the behavior of DS in terms of events and rules, similar to CORBA, DCOM and Java RMI. ERF has an object-oriented model in which system components are treated as objects; however, in ERF, events and rules are also treated as objects. ERF is designed to support multiple standards for distributed system environments (e.g. CORBA, Java RMI) and has an open architecture that allows extensions for supporting new standards as well.

The ERF architecture [Arroyo02a] consists of two main structures: A static structure, which holds the objects that represent definitions (event types and rules); and a dynamic structure, which holds the objects that encapsulate the behavior of a distributed system at run time. One of these dynamic structures is a Rule Based Intelligent Event Service (RUBIES) [Arroyo01], the heart of ERF, which provides services for rules specification, service registration and event handling using rules.

A Rule Definition Language (RDL) has been created to specify the behavior of DS in terms of how distributed components react to the occurrence of events in a system. In RDL, a rule can be viewed as an algorithm that is executed due to the occurrence of events. The rule specifies the actions or alternative actions to be taken if the necessary events occur. How and when a rule is evaluated are issues of the computational model of the RDL [Arroyo02b]. A rule compiler is integrated to ERF in order to validate rules syntax and generating standard representation to be used by RUBIES.

3. RTFAS

A "Real-Time Flood Alert System" (RFTAS) was selected for this study because it has ideal characteristics of a DS, it comprises many applications which reside in dissimilar platforms, requires the processing of many simultaneous events, requires real-time processing of events, requires the processing of different sets of rules which may involve multiple distributed event services and needs continuous monitoring for debugging and validation purposes.

The flood detection and alert system relies on two major networks: ALERT and USGS. The first network is a cooperative flood-warning system called Automated Local Evaluation in Real Time (ALERT) maintained by the Civil Defense. This system maintains 40 real time gage stations that report rainfall measurements. The other network is maintained by the U.S. Geological Survey. It maintains about 123 real time gaging stations that report rainfall, discharge and stage measures.

The Major alerts generated by the system are: risk watch, moderate risk, high risk, low event in progress, high event in progress, and extraordinary event in progress. These alerts are generated from the triggering process of rainfall events.

4. The experiment

For this experiment 10 participants were asked to implement a subset of the RTFAS domain problem using RDL and JAVA. The participants had experience programming in

JAVA but no experience programming in RDL and no knowledge of RTFAS. The task of the participants was concentrated on the development of algorithms to generate some specific flood alerts using as input a real time sequence of rainfall events. The modules that handled the input, and output of events and the evaluation of the rules were preprogrammed to allow the participants to focus their effort on the implementations of the algorithms to generate the alerts.

The experiment followed a within subjects design. The participants were asked to perform the same task with the two programming languages. To compensate for the learning effect the participants were divided in two groups. Group A implemented the RDL version first and then the JAVA version. Group B implemented the JAVA version first and then the RDL version.

The details of the experiment were explained to all the participants in an orientation session. A tutorial on the RDL language was given to all the participants before they started with the RDL implementation. After completing each implementation the participants were asked to answer a cognitive dimension questionnaire [Green96] and [Blackwell00].

The variables measured for the experiment were: time needed to complete each implementation, number of lines of code generated, and the cognitive dimensions variables.

5. Results

In table 1 we present a summary of the results in terms of the average number of lines of code generated for each implementation by each group.

Table 1. Average number of lines of code generated.

	RDL Implementation	JAVA Implementation
Group A	85	398
Group B	136	469
Overall	111	434

Overall the JAVA implementation required 3.9 times the lines of codes than it was required to complete the RDL implementation. An interesting finding was that the participants that implemented the RDL version first (Group A) generated fewer number of lines of codes than the participants that implemented the RDL version second (Group B). It seems that the knowledge of the problem acquired by Group B from the JAVA implementation had a negative influence on their implementation of the RDL implementation.

In table 2 we present a summary of the results in terms of the average number of time required to complete each implementation by each group.

Table 2. Average time to complete each implementation.

	RDL Implementation	JAVA Implementation
Group A	8.1	18.3
Group B	7.0	30.5
Overall	7.6	24.4

Overall the JAVA implementation required 3.2 times the overall time required to complete the RDL implementation.

A summary of the most relevant comments made by the participants for each of the cognitive dimension variables follows.

Abstraction: It is easy to abstract the ideas in terms of events and rules with RDL. RDL operators and operations are very intuitive.

Closeness of mapping: RDL lets the user represent system dynamic behavior in a natural way and the results obtained are the expected ones.

*Consistency***:** RDL operators and operations are easy to use and to infer for specifying system behavior.

*Viscosity***:** RDL requires little effort to make changes.

Diffuseness: RDL syntax is simple and short; rules only need very few lines of code.

Error-proneness: The RDL compiler does not provide appropriate error messages, and it creates error-proneness in users.

*Hard mental operations***:** No hard mental effort is required to specify rules using RDL.

Premature commitment: RDL forces to users to think in terms of events and rules to specify behavior.

Progressive evaluation: RDL language does not have a tool to verify and debug events at run-time

Role-expressiveness: It is easy to infer the meaning of any rule implemented using RDL.

Secondary notation: RDL lets the user to add comment lines; they are indispensable in any programming language.

6. Conclusion

In this work we demonstrated that RDL is a better paradigm for implementing distributed systems applications based on events and rules. Distributed applications are developed easily, in less time, with less code and with greater user satisfaction with RDL than with JAVA.

In general the participants expressed that RDL was easier to learn than Java, that rules specifications was easier and natural even for modifying existing rules or adding new ones. The participants also expressed that RDL does not provide good error messages and that the messages provided were not enough to verify the state of the events and attributes at run-time.

7. References

[Arroyo01] Arroyo, J., Moulier E. "A Rule-Based Intelligent Event Service (RUBIES)", Proceedings of the Computing Research Conference 2001, University of Puerto Rico, Mayagüez Campus, March 2001.

[Arroyo02a] Arroyo J., Borges. J., Rodríguez N., Moulier E., Rivas M., Cuaresma A., Yeckle J., "An Event/Rule Framework (ERF) for specifying the Behavior of Distributed Systems", 3rd International Workshop on Software Engineering and Middleware, 2002, pp 59 – 71.

[Arroyo02b] Arroyo J., Moulier E., Yeckle J., "RDL: A Rule Definition Language for Specifying the Behavior of Distributed Systems", Electrical and Computer Engineering Department University of Puerto Rico, Mayagüez Campus Mayagüez, Puerto Rico, 2002.

[Blackwell00] Blackwell, A.F. & Green, T.R.G. (2000)., "A Cognitive Dimensions questionnaire optimised for users". In A.F. Blackwell & E. Bilotta (Eds.) Proceedings of the Twelth Annual Meeting of the Psychology of Programming Interest Group, 137-152.

[Green96] Green, T. R. G. & Petre, M. 1996, "Usability analysis of visual programming environments: a 'cognitive dimensions' framework." J. Visual Languages and Computing, 1996, 7, 131-174.