Case Study: Developing a Rule-Based Language for Mobile Robots

Joseph J. Pfeiffer, Jr.

Department of Computer Science

New Mexico State University

Las Cruces, NM 88003

`http://www.cs.nmsu.edu/~pfeiffer`

Abstract

The 1997 Visual Programming Challenge inspired the development of Altaira, a rule-based language for small mobile robots. This paper describes the original design of the language, the modifications made to render the language implementable in the time available, and further changes which became necessary as a result of experience with the language. The design of a more general second generation language is also introduced.

Keywords: End-User Languages, Rule-based Languages, Mobile Robots

1. Introduction

LEGOsheets, a visual language for controlling small mobile robots, was presented at VL'95[3]. In response to this work, Ambler proposed the first Visual Programming Challenge (VPC), which was held at VL'96. A second VPC was held at VL'97[1]. A team from NMSU developed Altaira, a rule-based language for small mobile robots, as an entry for the second VPC.

The time constraints of the Challenge forced an accelerated development cycle, while the team's experiences in developing solutions to the VPC tasks caused a similarly accelerated language evolution. Several of the original design decisions, relating both to the language itself and the user interface, needed modification due to experience using the language.

The language design and implementation experiences encountered in the development of Altaira should prove generally helpful to the designers and implementors of other visual languages, in particular rule-based languages.

2. Problem Statement and Initial Design

Competitors in the VPC were issued LEGO-based mobile robot kits (Zambonibugs), including sensors, motors, and a Motorola HC11-based single-board computer. Using this equipment, and a visual language of their choice, they were to explore a road course constructed from standard LEGO road tiles, modified by the addition of tape stripes in order to better accommodate the robot's sensors.

In order to solve this problem, it was necessary for a robot to follow the course while its controlling program maintained a map of the course, assured that the entire course was explored, and terminated when the entire course had been explored.

Due to the small size of the robot and the limited power available in its processor, the robot operated in a tethered mode, sending sensor readings to and receiving motor commands from a host computer.

2.1 Goals

The primary goal of the project was the development of a visual programming language that would be powerful enough to solve general problems in the tile-based environment, but simple enough to be used by children. The language was to be completely visual in the sense that it would be possible to use it to perform tasks with no recourse to text whatever

2.2 Language Design

In order to effectively decouple state information from program flow, a reactive strategy[4] implemented as a rule-based system was adopted. In each execution cycle, the system combined the inputs from the robot sensors with state information to provide motor outputs and state changes.

The original intent was to use a pattern-matching approach to tile recognition. Tile types would be represented as patterns portraying the tape marks present on the tiles; sensor inputs would be matched against possible tape marks to infer the pattern and hence the tile type. As the robot negotiated tiles, its orientation could be maintained either by using knowledge of the current tile type and the robot's position on the tile to infer the direction it must be heading, or by keeping track of steering commands sent to the robot and correcting the orientation on that basis.

It was immediately apparent that this approach was both too ambitious for implementation in the time available, and unnecessary in the limited problem domain.

Consequently, the pattern matching scheme was replaced with a state-based approach. In this approach the tile map is represented by a two-dimensional array of tile states, where each array element represents a LEGO tile. A tile state is capable of encoding the extent to which a tile has been explored, its type, its orientation, and (if it has been traversed previously) the side by which it was last exited. This approach is also simple to use in rule lookup; the state can simply be used as an array index.

This results in the following recognize-act cycle:

1. Read sensors
2. Use sensor values, the current state, and the current direction as indexes into an array containing motor commands, a new state, and a navigation command (this definition of rule lookup sidestepped the problem of selecting from a set of enabled rules; only a single rule could be enabled).
3. Feed the motor outputs to the robot, put the new state into the tile map, and use the navigation command to update the current robot direction and tile.

States were marked by the user as either terminal or non-terminal. Execution would terminate when every tile in the map had either never been explored, or was in a terminal state.

2.3 Program Design

The design of the program implementing Altaira closely followed that of the language itself.

Data Structures The central data structures were a three-dimensional array containing outputs for the rules as described above, and a two-dimensional array of tile states. The rule array was referred to as a ruleset, and the tile array as a map.

Execution Engine The execution engine implemented the recognize-act cycle described, and was also responsible for displaying information regarding the current state of the system (including the tile map, the robot, and the communications link). Menus available from this window allowed the user to switch between the physical and virtual robot (see below), to edit the ruleset, and to read and write ruleset files. Figure 1 shows a screen capture of the execution engine's user interface window.

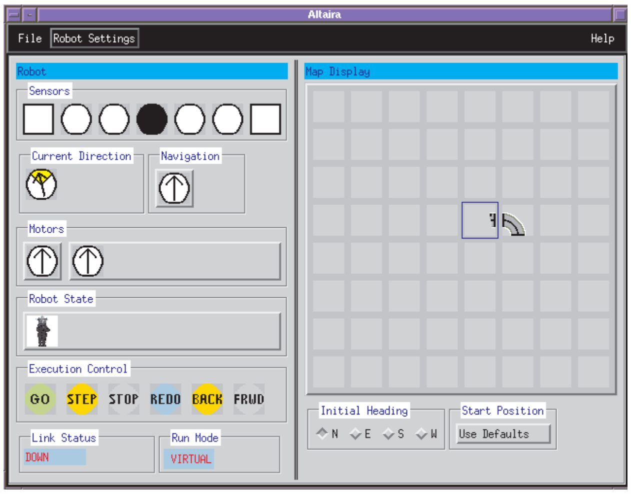

Figure 1: Execution Engine

Communication Substrate A simple communication protocol between the host and the robot was designed and implemented.

Virtual Substrate In order to have repeatable robot behavior for debugging, a virtual robot was developed. The programming interface for this virtual robot was identical to that for the actual communications substrate, easing development of the execution engine.

Rule Editor A rule editor allowed the user to define rules completely graphically. Icons were defined for sensor and direction inputs, and for motor and navigation outputs. The user was able to navigate the ruleset by selecting rule inputs (whenever a new rule input is selected, the appropriate rule is located in the ruleset and its outputs are displayed), and could modify the current rule by selecting new values for its outputs. Figure 2 shows a screen capture of the rule editor

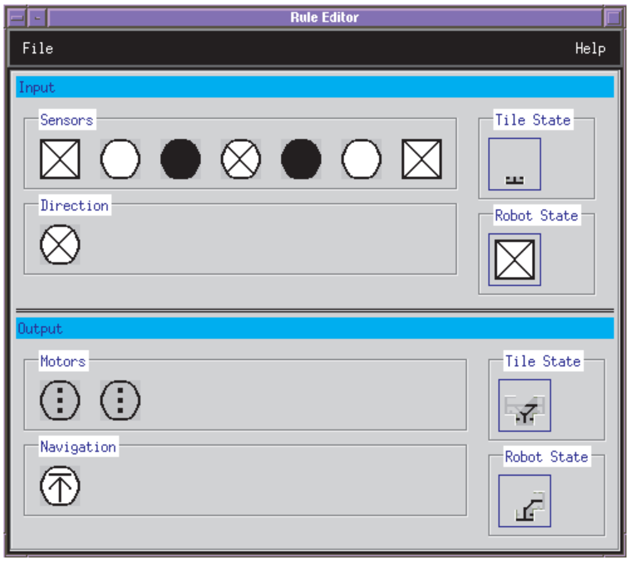

Figure 2: Rule Editor

Menus attached to each rule input and output allowed the user to make selections. The state input and output menus had special items which would invoke the state editor when selected, allowing the user to define new states.

State Editor The user was able to define icons for tile states using a simple icon editor. A state included the icon defined with this editor, an index assigned by the system, and a flag set by the user marking the state as terminal or nonterminal.

3. Implementation Decisions

Implementation decisions were made with a view to the programming expertise of the members of the Altaira team. As NMSU Computer Science uses C, Unix, and the X window environment for the majority of its programming classes, these were initially chosen as the implementation language and environment. As the member of the team with the greatest experience with X was more fluent with Motif than other widget toolkits, Motif was selected, along with UIL (a declarative, textual user interface definition language bundled with Motif) for implementation of the graphical user interface.

Early in the development of the system, a decision was made to switch to C++, both to take advantage of object-oriented programming and in order for the C programmers on the project to learn the language. This also allowed the project to make use of the Standard Template Library for developing data structures.

Version control was not used. Instead, one- and two-person subteams were given exclusive control over portions of the code on a file-by-file basis, and were responsible for all changes within those files.

4. Program Development

The initial language design was complete by the end of December, 1996. Program design was finished, and UIL code defining the windows complete, in late February, 1997.

While the program was being implemented, consideration was being given to the structure of the rulesets needed to perform a variety of tasks: simple line following, the road course exploration problem stated above, and simulation of a Turing machine. It became clear in early May that an error had been made in the language specification: all state information was tied to a particular tile, with no state information relating to the robot itself. This made it impossible to carry a notion of subtasks to be performed from tile to tile, or to have rules intended to accomplish tasks that were necessary in several different tile states. In response to this lack, a ‘‘robot state’’ was added to both the rule inputs and outputs. This change was completed in less than a day.

The system was first stable enough for user testing on Friday, May 23, 1997. Four children, aged seven to twelve, were given an opportunity to develop rulesets to follow lines and identify tiles; all four found the user interface to be intuitive and easy to use, except that the interface was based on palette selections and the children uniformly expected (and tried to use) drag-and-drop. All of the children were able to make the virtual robot follow a line, and the oldest child was able to develop a ruleset which recognized left and right curved tiles (but not intersections).

The rulesets for the road mapping problem and the Turing Machine simulation were developed by the author between Friday evening and the Monday, May 26 submission due date. The language description in [5] is based on the language as it existed at that point.

5. Language Changes Due to Experience

Following the VPC submission, the language was re-evaluated. The principal design objectives had been met (the children had been able to make effective use of the language, and it had been possible to solve the VPC tasks), however, improvement was possible. The following observations were made:

First, if an error occurred in a ruleset, the only way to locate it was to restart the simulation from the beginning, and single-step the rule executions when approaching the area of the map where the error took place. Locating an error was very time-consuming with the virtual robot, and nearly impossible with the physical robot (single-stepping changed the robot dynamics enough that it was likely the rule causing the error would never be executed). A rule recorder, capable of storing and playing back rule executions, was needed.

Second, when developing rulesets, the forced combination of sensors, direction, tile state, and robot state for inputs and outputs was not appropriate. It was quite frequently desired, for instance, to change a tile state based on an input from a particular sensor. The execution model required rules to be entered for all possible combinations of inputs that included the desired sensor. This led to excessively large rulesets (the Compulsory Problem ruleset contained approximately 2100 rules) and very little confidence that all situations had in fact been properly considered.

The language and interface were revised to address these weaknesses during Summer, 1997. As several team members had left, the previous division into subteams was no longer feasible. For the modifications, all remaining members could modify any code, requiring version control to be instituted. As RCS was readily available, it was used.

5.1 Implementation of Subsumption Architecture

The first modification was a change to the language, and the recognize-act cycle, to support an implementation of Brooks's Subsumption Architecture (it should be observed that this implementation of subsumption is very different from the implementation described in [2]). This was implemented as follows:

First, each rule input was augmented by a `DontCare`, selectable from the rule editor. A `DontCare` input is matched by any possible value for the particular rule input. This replaces the previous array indexing model of rule lookup with a more conventional model, in which a set of rules is enabled by matches between rule specifications and actual inputs.

This change to the rule enabling mechanism required a corresponding change to the rule selection mechanism. A simple scheme in which only a single enabled rule would be selected was seen as inappropriate, as it was desired that rules implementing line-following (for instance) should fire at the same time as rules causing tile state changes. Consequently a scheme that would amalgamate several rules to produce a result was necessary .

In order to permit this amalgamation, a `NoChange` was added to each output, so a rule would not need to provide all possible outputs.

The set of enabled rules is sorted by priority (rules with a robot state have highest priority, then rules with a tile state but no robot state, then rules with neither robot state nor tile state), and the highest priority rule providing a given output is used for each output. If no enabled rule provides a particular output, the output is inherited from the previous execution cycle.

5.2 Enhanced User Interface

Two changes were also made to the user interface.

First, a rule execution recorder was implemented. A history of 100 rule firings is stored, and can be reviewed. If the system is running as a simulation, the virtual robot is also moved to the position at which each rule firing occurred

Second, the language changes described in Section 5.1 made it very difficult to determine which rule was responsible for a given output. A search function, tying each output shown in the execution engine to the rule which caused it, was added. Whenever the user selects a rule output in the execution engine window, the rule editor is invoked with the rule responsible for the selected output as the current rule.

6. Further Observations

The modifications described in the previous section were all helpful in improving the language and interface.

The use of a subsumption architecture permitted the logical division of the ruleset, separating reflexive rules (rules that are independent of state), tactical rules (rules depending on the tile state), and strategic rules (rules depending on the robot state). In addition to a quantifiable reduction in the number of rules required to perform tasks (the solution to the road mapping problem described in Section 2 was reduced from over 2100 rules to 110 rules), the degree of confidence in the completeness of a solution is much higher.

The rule recorder and the search functions significantly eased ruleset debugging, as they simplified isolation of rules containing errors.

A description of the solutions to the VPC problems may be found in [6].

7. Conclusions and Future Work

First and foremost, a rule-based visual language is an appropriate medium for implementing reactive robot control strategies (further evidence for this conclusion is that both our solution, and the other finalist's solution, used this approach[1]). However, seemingly minor differences in language semantics can have major ramifications for ruleset size, and editor requirements.

The importance of the simulation facilities, including both the simulation environment itself and the rule recording facilities, cannot be overstated. Very little time in ruleset development and debugging actually took place with the physical robot, as repeatable observations were impossible to obtain. Instead, multiple simulation runs were made to accomplish nearly all of the ruleset debugging, with comparatively very few runs made with the actual robot. The rule recorder is important for use in the simulation environment, and indispensable for the actual robot, due to the changes in robot dynamics described above.

The form of the ruleset search function (essentially, ''how'd *that* get there?'') was very appropriate for debugging.

The programming tools used were, in general, satisfactory to the job at hand. Using UIL as the user interface definition tool was particularly helpful, as it permitted rapid prototyping of the interface before development of other code. The definition of the user interface using UIL also made modifications to the interface much simpler, assisting the addition of the robot states, rule recorder, and search functions. The Motif toolkit itself, however, introduced severe overhead; the user interface is unacceptably slow when used with Intel 486 platforms. It is unclear whether the change from C to C++ was an advantage or not. It seems clear that, due to the inexperience of most of the team with the language, development was slowed by the change; however, the support for classes encouraged a highly modular programming style that contributed markedly to the project's success (when the project was begun, Java interpreters were viewed as being unacceptably slow for the project, so it was not seriously considered as an implementation language. It is not clear how the use of Java would have affected the project). Use of the Standard Template Library was probably harmful to the project; many of the restrictions it places on the use of iterators caused a great deal of time to be lost while work-arounds were implemented.

The greatest deficiency seen in the language itself at this time is the lack of support for any analog to separate program modules. This effectively prevents collaboration on ruleset development, which proved to be a handicap to ruleset development.

There are several avenues for further development of Altaira.

First, the language is currently tied to the Zambonibug's configuration of motors and sensors. A robot definition editor, permitting users to define the geometry and characteristics of the sensors and actuators, would provide much greater generality.

Second, tethering the robot to the host should not be necessary. A ruleset interpreter should be developed for eight bit microcontrollers such as the HC11, 8051, or PIC, allowing rulesets to be downloaded to robots and run autonomously. The rule execution model is simple enough that this should be possible, even given the limitations of these processors.

Third, as mentioned previously in this section, a mechanism for merging rulesets should be developed to permit collaboration in ruleset development.

Finally, the direction currently being explored is to replace the tile-based navigation scheme with the more general pattern-based scheme originally envisioned. A rule-based system for geometric reasoning, with rules based on compass-and-straightedge constructions, is currently being developed by the team. This is being explored as a general-purpose geometric system, usable for robotics by tying particular rule inputs to sensors, and particular rule outputs to actuators.

8. Acknowledgments

Altaira was implemented by a Kit Crumpton, Tim James, Endy Mendoza, Karen Payne, and Heather Pfeiffer. Hardware and travel support were provided by NSF Grant IRI-9616242. Additional travel support was provided by the NMSU College of Arts and Sciences and Burlington Resources.

9. References

1. Ambler, A. L. ‘‘1997 Visual Programming Challenge Summary,’’ in *Proceedings of the 1997 IEEE Symposium on Visual Languages*, Capri, Italy, pp 11-18, 1997.
2. Brooks, R. A. ‘‘A robust layered control system for a mobile robot,’’ in *IEEE Journal of Robotics and Automation* RA-2, pp 14-23, 1986.
3. Gindling, J., A. Ioannidou, J. Loh, O. Lokkebo, and A. Repenning. ‘‘LEGOsheets: a rule-based programming, simulation and manipulation environment for the LEGO programmable brick,’’ in *Proceedings of the 11th IEEE Symposium on Visual Languages*, Darmstadt, Germany, pp 172-179, 1995.
4. Martin, F. J. *The Art of Robotics: An Introduction to Engineering*, Addison Wesley Longman, forthcoming.
5. Pfeiffer, J. J. Jr. ‘‘A Rule-Based Visual Language for Small Mobile Robots,’’ in *Proceedings of the 1997 IEEE Symposium on Visual Languages*, Capri, Italy, pp 164-165, 1997.
6. Pfeiffer, J. J. Jr. ‘‘Altaira: A Rule-Based Visual Language for Small Mobile Robots.’’ *Journal of Visual Languages and Computation*, 9(2), 1998.