



## Copyright Statement

The digital copy of this thesis is protected by the Copyright Act 1994 (New Zealand). This thesis may be consulted by you, provided you comply with the provisions of the Act and the following conditions of use:

- Any use you make of these documents or images must be for research or private study purposes only, and you may not make them available to any other person.
- Authors control the copyright of their thesis. You will recognise the author's right to be identified as the author of this thesis, and due acknowledgement will be made to the author where appropriate.
- You will obtain the author's permission before publishing any material from their thesis.

To request permissions please use the Feedback form on our webpage.

<http://researchspace.auckland.ac.nz/feedback>

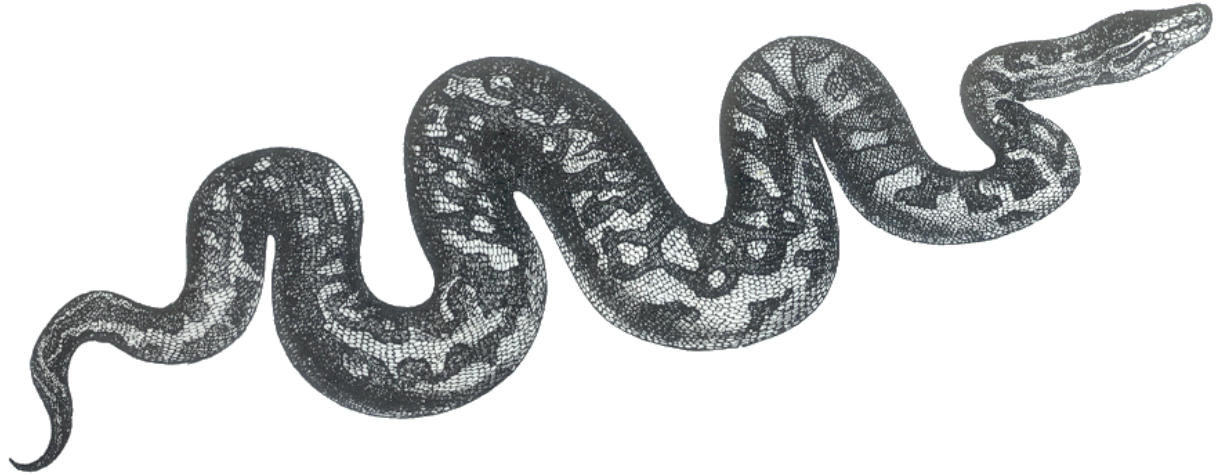
## General copyright and disclaimer

In addition to the above conditions, authors give their consent for the digital copy of their work to be used subject to the conditions specified on the Library

[Thesis Consent Form](#)

---

*Department of Electrical & Electronic Engineering  
The University of Auckland  
New Zealand*



---

# **Designing an application– specific programming language for mobile robots**

---

*Geoffrey Biggs*

*May 2007*

*Supervisor: Dr Bruce MacDonald*



A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF THE  
REQUIREMENTS OF DOCTOR OF PHILOSOPHY IN ENGI-  
NEERING



# Abstract

The process of programming mobile robots is improved by this work. The tools used for programming robot systems have not advanced significantly, while robots themselves are rapidly becoming more capable because of advances in computing power and sensor technology. Industrial robotics relies on simple programming tools usable by non-expert programmers, while robotics researchers tend to use general purpose languages designed for programming in other domains.

The task of developing a robot cannot be assumed to be identical to developing other software-based systems. The nature of robot programming is that there are different and additional challenges when programming a robot than when programming in other domains. A robot has many complex interfaces, must deal with regular and irregular events, real-time issues, large quantities of data, and the dangers of unknown conditions. Mobile robots move around and are capable of affecting everything in the environment. They are found in cluttered environments, rather than the carefully-controlled work spaces of industrial robots, increasing the risk to life and property and the complexity of the software.

An analysis of the process of developing robots provides insight into how robot programming environments can be improved to make the task of robot development easier. Three analyses have been performed: a task analysis, to determine the important components of the robot development process, a use case analysis, to determine what robot developers must do, and a requirements analysis, to determine the requirements of a robot programming environment. From these analyses, the important features of a robot development environment were found. They include features such as data types for data commonly found in robotics, semantics for managing reactivity, and debugging facilities such as simulators.

The analyses also found that the language is an important component of the programming environment. An application-specific language designed for robot programming is proposed as a solution for providing this component. Application-specific languages are designed for a particular domain of programming, allowing them to overcome the difficulties in that domain without concern for their usefulness in other domains. To test

the hypothesis that such a language would improve robot development a set of language extensions has been created. These extensions, named RADAR, provide explicit support for robotics. The prototype implementation uses the Python programming language as the base language.

RADAR provides support for two of the necessary features found in the analyses. The first is support for dimensioned data via a new primitive data type, ensuring all dimensioned data is consistent throughout a program. Dimensional analysis support is provided, allowing the safe mixing of data with compatible units, the creation of more complex units from simple single-dimension units, and built-in checking for errors in dimensioned data such as performing operations involving incompatible dimensioned data. For example, the data type will prevent the addition of distance and speed values. Several dimensional analysis systems have been developed for general purpose languages in the past. However, this is the first application of the concept specifically to robotics.

The second feature is semantics for managing reactivity. In RADAR, the principle of ease-of-use through simplicity is followed. Special objects represent events and responses, and a special syntax is used for both specifying these objects and managing the connections between them in response to the changing state of the program. This reduces programming complexity and time. There are many other languages for managing reactivity, both the more general languages, such as Esterel, and languages for robotics, such as TDL and Colbert. RADAR is simpler than the general languages, as it is aimed solely at the needs of robot developers. However, it takes a different approach to its design than other languages for reactivity in robotics. These are designed to provide support for a specific architecture or architecture style; RADAR is designed based on the needs of robot developers and so is architecture-independent.

RADAR's design philosophy is to provide robot-specific features with simple semantics. RADAR is designed to support what robot developers need to do with the language, rather than providing a special syntax for supporting a particular robot, architecture or other system. RADAR has been shown to provide an improvement in dimensioned data management and reactivity management for mobile robot programming. It increases the readability, writability and reliability of robot software, and can reduce programming and maintenance costs. RADAR shows that an application-specific approach to developing a robot programming language can improve the process of robot development.

# Acknowledgements

When I first started my Phd, the three years I had ahead of me until my minimum submission date seemed like an eternity of work. Now that I have finished, the three and a half years it has taken seems like a roller coaster of activity squashed into a few memories. Needless to say, I would not have survived without the support of numerous people.

First on the list is my supervisor, Dr Bruce MacDonald, who has provided guidance and support since before I knew I wanted to do a PhD. Due to his support applying for a summer research scholarship at the end of my third undergraduate year, he is probably one of the most responsible for my being here today. For showing me during that summer that a PhD could be fun, Lee Middleton, Sylvia Wong and David Yuen must also be thanked.

I would like to thank my parents, who have provided me with the necessities, such as a home, food and Internet access, and so helped me to live while carrying out this research.

My friends have provided me with support in many ways. Toby Collett has been a friend since we began our time at Auckland University and has shared the revelations and suffering during his own PhD. Lee Middleton has played the role of L<sup>A</sup>T<sub>E</sub>X Guru and experienced senior throughout my PhD. These and other friends have been responsible for keeping me sane, particularly during the harder parts of the research.

Other members of the Robotics Group have been an invaluable sounding board for ideas, including Rick Chen, Kathy Fung, Luke Gumbley, Ben Moores, Sigrid Roehling, Grant Sargent, Dr Karl Stol, Slobodan Vukanovic, Sylvia Wong, Jack Yu, and David Yuen.

The technical and administrative staff of the department have helped ensure I had what I need when I needed it. Grant Sargent in particular has created a well-run Robotics Lab where research is fun and exciting. Lynda Jones, Bev Painter, Jamie Walker, Stephen Ward and Peter Wigan have all been very helpful.

Finally, I would like to thank The University of Auckland for providing the significant financial support I have received, and the members of the FirstOnes.com website, whose constant insanity has helped me avoid my own.



# Contents

|          |   |           |
|----------|---|-----------|
| <b>1</b> | <b>Introduction</b>                                   | <b>1</b>  |
| 1.1      | Robotics now and in the future . . . . .              | 3         |
| 1.2      | Overview of robot development . . . . .               | 4         |
| 1.2.1    | Application-specific languages . . . . .              | 6         |
| 1.3      | Description of the thesis . . . . .                   | 7         |
| 1.3.1    | Scope of the research . . . . .                       | 7         |
| 1.3.2    | Overview of the thesis . . . . .                      | 8         |
| 1.4      | Contributions of the thesis . . . . .                 | 9         |
| <b>2</b> | <b>Programming robots</b>                             | <b>11</b> |
| 2.1      | Reviewing robot programming . . . . .                 | 11        |
| 2.2      | Manual programming methods . . . . .                  | 12        |
| 2.2.1    | Text-based methods . . . . .                          | 13        |
| 2.2.2    | Graphical methods . . . . .                           | 19        |
| 2.3      | Automatic programming methods . . . . .               | 21        |
| 2.3.1    | Learning systems . . . . .                            | 21        |
| 2.3.2    | Programming by Demonstration . . . . .                | 22        |
| 2.3.3    | Instructive systems . . . . .                         | 24        |
| 2.4      | Programming language design trends . . . . .          | 25        |
| 2.5      | Discussion . . . . .                                  | 26        |
| <b>3</b> | <b>Robot Programming Environment Design</b>           | <b>29</b> |
| 3.1      | Analysing robot development . . . . .                 | 30        |
| 3.1.1    | Task analysis . . . . .                               | 30        |
| 3.1.2    | Use case analysis . . . . .                           | 38        |
| 3.1.3    | Requirements analysis . . . . .                       | 42        |
| 3.2      | Features of a robot programming environment . . . . . | 43        |
| 3.2.1    | Essential features . . . . .                          | 44        |
| 3.2.2    | Non-essential features . . . . .                      | 46        |



---

|          |  |           |
|----------|--|-----------|
| 3.2.3    | Other features . . . . .                             | 47        |
| 3.3      | Providing feature support . . . . .                  | 47        |
| 3.3.1    | Using an existing environment . . . . .              | 48        |
| 3.3.2    | Using a custom environment . . . . .                 | 50        |
| 3.4      | RADAR . . . . .                                      | 51        |
| 3.4.1    | Why Python? . . . . .                                | 52        |
| 3.5      | Discussion . . . . .                                 | 52        |
| <b>4</b> | <b>Dimensional Analysis</b>                          | <b>55</b> |
| 4.1      | The importance of dimensional analysis . . . . .     | 55        |
| 4.2      | Requirements . . . . .                               | 58        |
| 4.3      | Literature survey . . . . .                          | 59        |
| 4.4      | Solutions . . . . .                                  | 62        |
| 4.4.1    | New variable information . . . . .                   | 62        |
| 4.4.2    | Object Oriented . . . . .                            | 63        |
| 4.4.3    | Preprocessor . . . . .                               | 63        |
| 4.4.4    | Primitive data type . . . . .                        | 64        |
| 4.5      | RADAR's dimensioned data type . . . . .              | 65        |
| 4.5.1    | Design . . . . .                                     | 65        |
| 4.5.2    | Implementation . . . . .                             | 70        |
| 4.5.3    | Implementing RADAR in C++ . . . . .                  | 73        |
| 4.6      | Higher level support . . . . .                       | 74        |
| 4.6.1    | Class framework . . . . .                            | 74        |
| 4.6.2    | Ensuring correctness in coordinate systems . . . . . | 77        |
| 4.7      | Integration with existing systems . . . . .          | 77        |
| 4.8      | Discussion . . . . .                                 | 78        |
| <b>5</b> | <b>Managing Reactivity</b>                           | <b>79</b> |
| 5.1      | Reactivity in robotics . . . . .                     | 79        |
| 5.1.1    | Requirements . . . . .                               | 83        |
| 5.2      | Literature review . . . . .                          | 84        |
| 5.2.1    | Expected events . . . . .                            | 85        |
| 5.2.2    | Unexpected events . . . . .                          | 90        |
| 5.2.3    | Summary . . . . .                                    | 91        |
| 5.3      | Semantics of reactivity management . . . . .         | 92        |
| 5.3.1    | Boost.Signals . . . . .                              | 94        |
| 5.3.2    | Qt signals and slots . . . . .                       | 95        |
| 5.3.3    | Adapting signals and slots for robotics . . . . .    | 98        |
| 5.4      | Reactivity in RADAR . . . . .                        | 99        |

---

|          |   |            |
|----------|---|------------|
| 5.4.1    | Event objects . . . . .                               | 100        |
| 5.4.2    | Response objects . . . . .                            | 100        |
| 5.4.3    | Connectivity statements . . . . .                     | 104        |
| 5.4.4    | Formalisation . . . . .                               | 106        |
| 5.4.5    | Shared resources . . . . .                            | 109        |
| 5.5      | Implementation . . . . .                              | 111        |
| 5.5.1    | Syntax . . . . .                                      | 111        |
| 5.5.2    | Example . . . . .                                     | 112        |
| 5.5.3    | Preprocessor . . . . .                                | 117        |
| 5.6      | Comparison with existing work . . . . .               | 121        |
| 5.7      | Discussion . . . . .                                  | 122        |
| <b>6</b> | <b>Evaluation</b>                                     | <b>125</b> |
| 6.1      | Evaluation by programming language criteria . . . . . | 126        |
| 6.1.1    | Language design and implementation criteria . . . . . | 126        |
| 6.1.2    | Human factors . . . . .                               | 135        |
| 6.1.3    | Software engineering criteria . . . . .               | 137        |
| 6.1.4    | Application domain criteria . . . . .                 | 138        |
| 6.1.5    | Summary . . . . .                                     | 138        |
| 6.2      | User study . . . . .                                  | 138        |
| 6.2.1    | Methodology . . . . .                                 | 139        |
| 6.2.2    | Results . . . . .                                     | 141        |
| 6.3      | Evaluation by goals . . . . .                         | 144        |
| 6.4      | Example application . . . . .                         | 144        |
| 6.5      | Limitations . . . . .                                 | 149        |
| 6.5.1    | Dimensional analysis . . . . .                        | 149        |
| 6.5.2    | Reactivity . . . . .                                  | 150        |
| 6.6      | Discussion . . . . .                                  | 150        |
| <b>7</b> | <b>Player Contributions</b>                           | <b>159</b> |
| 7.1      | New interfaces . . . . .                              | 160        |
| 7.1.1    | Actuator array interface . . . . .                    | 160        |
| 7.1.2    | Limb interface . . . . .                              | 163        |
| 7.1.3    | Gripper interface . . . . .                           | 164        |
| 7.2      | Data modes . . . . .                                  | 166        |
| 7.3      | Summary . . . . .                                     | 168        |

---

|          |  |            |
|----------|--|------------|
| <b>8</b> | <b>Future work and conclusions</b>             | <b>171</b> |
| 8.1      | Future work . . . . .                          | 171        |
| 8.1.1    | Improvements to robot programming . . . . .    | 171        |
| 8.1.2    | Dimensional analysis . . . . .                 | 171        |
| 8.1.3    | Reactivity support . . . . .                   | 172        |
| 8.1.4    | Other . . . . .                                | 172        |
| 8.2      | Conclusions . . . . .                          | 172        |
| <b>A</b> | <b>Task analysis</b>                           | <b>177</b> |
| A.1      | System model . . . . .                         | 181        |
| A.1.1    | Components . . . . .                           | 181        |
| A.2      | Task analysis . . . . .                        | 182        |
| A.3      | Involved components and interactions . . . . . | 190        |
| <b>B</b> | <b>User study</b>                              | <b>191</b> |
| B.1      | User study code samples . . . . .              | 191        |
| <b>C</b> | <b>Source code</b>                             | <b>211</b> |
| C.1      | Resource decorator source . . . . .            | 211        |
| C.2      | Preprocessed example . . . . .                 | 215        |
| C.3      | Unit tests . . . . .                           | 223        |

# List of Figures

|     |   |     |
|-----|---|-----|
| 2.1 | Robot programming methods. . . . .  | 12  |
| 2.2 | Categories of manual programming methods. . . . .   | 13  |
| 2.3 | The KUKA programming environment and robot programming language. . . . .                        | 14  |
| 2.4 | The Lego Mindstorms programming environment. . . . .  | 20  |
| 2.5 | Categories of automatic programming methods. . . . .  | 21  |
|     |   |     |
| 3.1 | Top level tasks in the system model. . . . .  | 33  |
| 3.2 | The tasks for determining the functional requirements. . . . .                                  | 34  |
| 3.3 | The tasks required to develop the robot system. . . . .   | 35  |
| 3.4 | The tasks for maintaining the robot system. . . . .   | 36  |
| 3.5 | The tasks for training users of the robot system. . . . .                                       | 36  |
| 3.6 | Use case analysis of the implementation tasks. . . . .  | 39  |
| 3.7 | Use case analysis of the task of creating a robot program . . . . .                             | 40  |
| 3.8 | Use case analysis of the basic tasks performed when creating a robot program. . . . .           | 41  |
|     |   |     |
| 4.1 | The structures used to hold the dimensioned type, defined units and defined dimensions. . . . . | 67  |
| 4.2 | The structure of the Python interpreter. . . . .  | 70  |
| 4.3 | The configuration space parent class and some possible child classes. . . . .                   | 76  |
|     |   |     |
| 5.1 | Reactivity management methods. . . . .  | 85  |
| 5.2 | A simple three-layered robot in the Subsumption architecture. . . . .                           | 87  |
| 5.3 | An example of the objects used in a Qt GUI. . . . .   | 93  |
| 5.4 | The RADAR <i>event</i> object. . . . .  | 100 |
| 5.5 | The RADAR <i>response</i> object. . . . .   | 102 |
| 5.6 | The control flow of a <i>response</i> in RADAR. . . . .   | 103 |
| 5.7 | An example of objects used by RADAR to manage reactivity. . . . .                               | 116 |
| 5.8 | The process of checking an event and calling a slot when it occurs in RADAR. . . . .            | 119 |
|     |   |     |
| 6.1 | The relative speed of the dimensioned type and the Python float type. . . . .                   | 134 |

|     |   |     |
|-----|---|-----|
| 6.2 | The behaviour of the example shown in Listing 6.4. . . . .  | 145 |
| 6.3 | The interactions between objects in the RADAR example shown in Listing 6.4 in the case where the robot's battery goes flat. . . . . | 146 |
| 6.4 | The interactions between objects in the RADAR example shown in Listing 6.4 in the case where a blob is found. . . . .               | 147 |
| 7.1 | The 5DOF robotic arm for Pioneer 3-DX robots. . . . .   | 162 |
| A.1 | Top level tasks in the system model. . . . .  | 177 |
| A.2 | The tasks for determining the functional requirements. . . . .  | 178 |
| A.3 | The tasks required to develop the robot system. . . . .   | 179 |
| A.4 | The tasks for maintaining the robot system. . . . .   | 180 |
| A.5 | The tasks for training users of the robot system. . . . .   | 180 |

# List of Tables

|     |   |     |
|-----|---|-----|
| 3.1 | Components of the system model. . . . .   | 33  |
| 3.2 | The features that can benefit the task of robot programming. . . . .                              | 44  |
| 4.1 | Current methods for dealing with geometric data in some common robot programming systems. . . . . | 56  |
| 4.2 | Requirements for a robotic dimensioned data management method. . . . .                            | 58  |
| 4.3 | Options for providing dimensioned data support. . . . .   | 62  |
| 4.4 | Examples of operations on dimensioned data. . . . .   | 68  |
| 4.5 | The predefined dimensions and units in RADAR. . . . .   | 72  |
| 6.1 | The unit tests used to confirm the dimensioned data type is functional. . .                       | 127 |
| 6.1 | The unit tests used to confirm the dimensioned data type is functional. . .                       | 128 |
| 6.1 | The unit tests used to confirm the dimensioned data type is functional. . .                       | 129 |
| 6.2 | The evaluation criteria used to evaluate RADAR. . . . .   | 139 |
| 6.3 | The summarised results of user study code samples DA1 to RC7. . . . .                             | 141 |
| 6.4 | The results of user study code samples RC8 to RC12. . . . .                                       | 141 |
| 6.5 | Comments in favour of RADAR made by user study participants. . . . .                              | 142 |
| 6.6 | Comments against RADAR made by user study participants. . . . .                                   | 142 |
| 7.1 | The messages of the actuator array interface. . . . .   | 161 |
| 7.2 | The messages of the limb interface. . . . .   | 164 |
| 7.3 | The messages of the original, Pioneer-specific gripper interface. . . . .                         | 164 |
| 7.4 | The messages of the new, standardised gripper interface. . . . .                                  | 165 |
| A.1 | Components of the system. . . . .   | 182 |



# List of Listings

|      |   |     |
|------|---|-----|
| 2.1  | A wall following algorithm implemented using Yampa. . . . .   | 18  |
| 4.1  | Defining a new dimension and a new unit in RADAR. . . . .   | 72  |
| 5.1  | Managing signals and slots using Boost.Signals. . . . .   | 95  |
| 5.2  | Using a Qt button's triggered signal. . . . .   | 96  |
| 5.3  | The creation and use of a custom signal in Qt. . . . .  | 97  |
| 5.4  | RADAR reactivity syntax. . . . .  | 101 |
| 5.5  | An <i>event</i> object, after expansion by the preprocessor. . . . .                                      | 118 |
| 5.6  | A <i>response</i> object, after expansion by the preprocessor. . . . .                                    | 120 |
| 5.7  | <i>once</i> , <i>whenever</i> and <i>waitfor</i> statements, after expansion by the preprocessor. . . . . | 121 |
| 6.1  | Ambiguity in the dimensioned data type's syntax. . . . .  | 130 |
| 6.2  | Ambiguous use of <i>until</i> in the <i>start</i> statement. . . . .                                      | 131 |
| 6.3  | The code used to evaluate the efficiency of the dimensioned data type. . . . .                            | 133 |
| 6.4  | A complete robot program written using RADAR. . . . .   | 152 |
| 7.1  | Setting the client's data mode and replace rules in Player. . . . .                                       | 168 |
| B.1  | User study code sample DA1. . . . .   | 191 |
| B.2  | User study code sample DA2. . . . .   | 191 |
| B.3  | User study code sample DA3. . . . .   | 192 |
| B.4  | User study code sample DA4. . . . .   | 192 |
| B.5  | User study code sample DA5. . . . .   | 192 |
| B.6  | User study code sample RC1. . . . .   | 193 |
| B.7  | User study code sample RC2. . . . .   | 194 |
| B.8  | User study code sample RC3. . . . .   | 195 |
| B.9  | User study code sample RC4. . . . .   | 195 |
| B.10 | User study code sample RC5. . . . .   | 197 |
| B.11 | User study code sample RC6. . . . .   | 198 |
| B.12 | User study code sample RC7. . . . .   | 199 |
| B.13 | User study code sample RC8. . . . .   | 199 |
| B.14 | User study code sample RC9. . . . .   | 201 |



---

|   |     |
|---|-----|
| B.15 User study code sample RC10. . . . .   | 202 |
| B.16 User study code sample RC11. . . . .   | 203 |
| B.17 User study code sample RC12. . . . .   | 205 |
| C.1 The RADAR Resource object. . . . .  | 211 |
| C.2 The RADAR combined resource object. . . . .   | 212 |
| C.3 The RADAR priority resource object. . . . .   | 213 |
| C.4 The block collector example after preprocessing. . . . .                                  | 215 |
| C.5 The unit tests used to confirm the dimensioned data type meets the specification. . . . . | 223 |