Software reuse in robotics: Enabling portability in the face of diversity

Robert Smith, Glenn Smith and Aster Wardani
Smart Devices Laboratory, Faculty of Information Technology
Queensland University of Technology
GPO Box 2434, Brisbane, Qld 4001, Australia
Email r2.smith@qut.edu.au

Abstract Software development for robotics applications is characterised by a high degree of specialisation. The reasons for this may centre on the diversity of robotic hardware, limitations on performance, and the need to perform complex and diverse tasks. The result of using such specialised software is an almost non-existent level of software portability.

It is proposed that the use of abstraction can enable the use of component software and bring with it the benefit of reuse. Abstraction of robotic hardware and software is difficult, and it is clear that a single robot abstraction is not practical due to the degree of diversity. It is proposed that some middle ground between specialisation and complete abstraction can be found. A second level component framework using fuzzy logic techniques is presented to illustrate how a significant degree of abstraction can be achieved, facilitating software portability, while accommodating the diversity of robotics.

I. Introduction

Mobile robots today, while varying greatly in design, often have a large number of similarities in terms of their tasks and goals. In turn, robots of similar design should be able to reuse the bulk of their controlling software.

Autonomous robotics in the past has been limited by the computational and storage capacity of the hardware used. Hence the controlling software was written to run as efficiently as possible to maximise the available power. This required very specialised solutions. Coupled with robot diversity and the problem domain, it is not surprising that the software produced offered little opportunity for reuse.

Robotics hardware has ever increasing CPU and storage capacity. This increase in capacity reduces the need for low-level software specialisation in order to gain critical speed optimisations. However it appears that robotics software engineering has not adopted modern software engineering techniques as would be expected given this increase in capacity. It is proposed that level of diversity in robot hardware needs to be addressed before advances in software engineering can take place. This paper proposes the use of higher-level abstractions of robot hardware to constrain the problems of diversity. This is achieved through the construction of a “Virtual Robot Framework” that presents a standard set of interfaces for interacting with low-level hardware. This takes the form of a component framework that allows standardised components to be plugged into. It negotiates and translates between the higher and lower level frameworks relying on fuzzy abstractions to speak the ‘same language’ to the higher level components. This is what in turn facilitates software portability.

As a working definition of portability, we are using that of Mooney [1] where ‘a software unit is portable across a class of environments to the degree that the cost to transport and adapt it to the new environment in the same class is less than the cost of redevelopment’. It is this concept we are trying to maximise.

II. Challenges of robotics software portability

The hardware limitations faced in robotics are becoming less of an issue; it no longer seems to be a limiting factor in robotics software engineering. But this change has not lead to the adoption of more modern software engineering techniques utilised in normal software engineering. The practice of regular software reuse in robotics is virtually non-existent. The use of modular software components is only starting to develop (as discussed section III).

Before considering the proposed solution in Section IV, and then discussing instances of this solution, we will first discuss the challenges in robotics that may be contributing to the current state of its software engineering.

A. Diversity of environmental interaction

The biggest problem faced by a robot is its physical presence in the real world. This means its position, heading, speed and size relative to its environment. It’s a major concern for the robot because it is this physical world that it is expected to operate in. This problem does not exist for your standard desktop PC, for instance, where it is not concerned which way it monitor is facing. In contrast a robot has large concerns over which way its camera is facing. The robot also has an interest in its size and shape and relative position to obstacles in the environment. Solutions for abstracting these physical properties aren’t yet properly used in robotics software.

B. Diversity of sensor input

The tasks that autonomous mobile robots typically perform are complex and extremely diverse. This is inevitable as robots interact with their real world environment. In general they: sense their environment, calculate a response, and then articulate that response. These actions contribute to achieving some higher level goal.
While processing of input can be equated to processing performed in normal software, the nature of the input and output is different. Input and output values are not usually discrete and require significant interpretation. In normal software systems possible input values are constrained by the input device. For example a keyboard only produces valid characters or key combinations. While the sequence of characters input by a user may need to be interpreted, and the sequence itself may be invalid, the values representing that sequence are fixed. Where robotic input devices (sensors) typically provide continuous values that may vary in accuracy. For example, the input from an infrared sensor may vary in successive readings without any distinguishable changes in the environment. A value of 0.8670 and 0.0903 may be returned in successive readings, and the distinction between these values may or may not be significant.

The very nature of robots is to interact with their environment and the non-deterministic nature of input and output is a side effect. The range and variation of values returned by robot sensors creates dependencies between controlling software and the hardware components.

C. Diversity of physical robot platform design

There are an enormous variety of robot hardwares available and a huge number of possible configurations. This diversity causes problems in two ways.

1) Diversity of robot composition: Robots can vary hugely in composition. The set of hardware component types from which they are constructed is extensive. The input that a robot receives can be derived from a wide variety of sensors, for example, infrared sensors, sonar, video cameras, thermostats, GPS locators, pressure sensors. For each of these kinds of sensors there are typically many variations, for example, a video camera could be colour, greyscale, monocular, binocular, fixed or moving. Different signal meaning, data types, and data rates require very particular coupling of the software to these devices. The information sent to the controlling software needs to be interpreted correctly across a very broad range. The output generated by the controlling software then needs to be formatted for use by an equally diverse range of effectors - from actuators, servos, lights and sound. This variation of input and output devices may be compared with the variation of input and output devices for desktop computers, with one major distinction, the physical configuration (location and direction) of desktop computer devices typically do not change the interpretation of the input produced. This is the next problem.

2) Diversity of robot configuration: So in addition to composition, robots can also vary in configuration. That is, the way the hardware components are set up for the robot, namely their placement in terms of position and direction. This combination of robot configurations combined with the set of possible configurations produces an extremely large matrix of possibilities. Currently controlling software is typically specialised for both aspects, and even a change in configuration of a robot can lead to software failure. Controlling software is typically not reusable even for different configurations of the same robot. The physical configuration of robot devices does change the interpretation of input. Variation in physical configuration is likely to force changes in the controlling software, in the same way changes in composition would.

D. Lack of standardisation

Some of the diversity of robot composition could be constrained by standardisation of software interfaces for hardware components. However, there is a distinct lack of interface standardisation possibly as the robotics domain is still a relatively small market. Currently the number of robots in existence is some orders of magnitude less than the number of desktop computers. This smaller market will not generate those same forces that have lead to standardisation of interfaces for desktop computer hardware. The combination of the small market and the diverse range of hardware reduce the impetus for standardisation as it is difficult for a single vendor to dominate and the advantage of doing so is limited. The earlier stated problems are also exacerbated by the lack of standardisation.

E. Diversity of the operating system

There are as many potential robotic operating systems as there are desktop operating systems and variants. Any controlling architecture must attempt to provide software portability across as large a section of these as possible.

III. RELATED WORK

There are a few robot control systems in the reviewed literature that use component-based paradigms for robotic control architectures. Some of these incorporate mechanisms for abstraction and reuse. The hardware abstraction problem has been partially addressed in some architectures ([2], [3], [4] and [5]), where hardware abstraction layers (HALs) have been designed to allow basic control functions to be ported to different robots. These control functions allow low-level abstractions such as requesting a sensor value.

However, more abstract concepts such as turn right, or is obstacle near, require different implementations on different robots even with the abstraction provided by HALs. A mechanism to provide consistent interpretation of such methods on various robots is what is needed. The implementation of these methods needs to take into account of the physical size and shape of the robot and the configuration of sensors and actuators. It is proposed that the use of fuzzy values to provide an abstraction from actual values returned from sensors and passed to effectors allows a more general definition of these methods. For example the actual distance that defines whether an obstacle is near will depend on the physical size of the robot and its task in progress.

for abstraction are achieved through diligent decoupling of implementation from interfaces and the use of the ‘object-port-connector’ [8] software pattern.

Notable proprietary software also exists. The high-profile MOBILITY architecture from iRobot [4] and the Evolution Robotics Software Platform (ERSP) from Evolution Robotics [3]. The MOBILITY package is a object-oriented, CORBA-based robotics control architecture. This product uses extensible “building blocks” and tools for construction of any style of robot control system. The ERSP architecture uses components to achieve its modularity and a hardware abstraction layer to facilitate reuse.

From the information available, it was found that none of these technologies provide a basis for true portability of software components. They achieve some modularity and flexibility through basic hardware abstraction, but not in the more sophisticated ‘virtual robot’ sense.

IV. TOWARD A SOLUTION

The diversity described earlier presents a very broad scope across which to consider portability. To address each of the associated problems we have designed techniques and systems within the overall architecture to support the software portability.

Assuming the processing capacity of autonomous robots is no longer the limiting factor in software development, we consider mechanisms for constraining diversity and promoting standardisation. Firstly, hardware abstraction constrains the effects of the diversity of robot composition and configuration. Secondly, the abstraction of values input from sensors and output to effectors constrains the diversity of environmental interaction, and also to some extent the diversity of robot composition.

The hardware abstraction separates high-level software from low-level hardware. The implementation of this abstraction we call the Virtual Robot Framework (VRF). The VRF defines a component-based framework for interacting with the low-level hardware. This allows the development of higher level hardware independent software components that can be plugged into any suitably configured VRF. Figure 1 shows the VRF in the context of the greater architecture.

To understand the function of the VRF, consider the function of the Java Virtual Machine (JVM) [9]. The JVM provides for platform independent execution of Java Bytecode. That is a program compiled to Java Bytecode can execute on any platform that has a JVM available. This is similar to the function of the VRF, that any code that uses the VRF interface can be moved between robots that implement the VRF interface. However, this comparison is not entirely accurate as VRFs are configured based on the capabilities of the robot. So there is a limitation: code can only be used on VRF configurations that support all the required capabilities. This maps directly to the capabilities of the robot, so it is the case that if the VRF cannot support the code, then the robot hardware cannot directly support the code either. The benefit lies in the space where robot capabilities overlap.

Fig. 1. The VRF connects the high-level and low-level systems and components. The high-level frameworks needs to use the standard interfaces and fuzzy abstractions provided by the VRF to be portable.

So the VRF is not intended to be the same for all robots. However, where the robots have overlapping capabilities the VRF provides standard interfaces for accessing those capabilities. Thus the VRF does not aim to support complete portability of software across all robots, merely to support portability where possible. For example, if a particular robot has a video camera, then the controlling software accesses it via the appropriate VRF interface. Thus, this piece of software will be portable to another robot that has a similar camera. However, if a robot does not have a camera then the VRF for that robot will not support the camera interface, and hence could not support that software.

V. FEATURES OF THE VIRTUAL ROBOT FRAMEWORK

The VRF is the layer that provides the appearance of a virtual robot for the high-level software components to use - concealing how commands are translated from the hardware and back. It supports portability by:

- providing standard interfaces for high-level components to access low-level hardware;
- specifying the robot con guration; and
- using fuzzy abstractions to fuzzify crisp sensor input and defuzzify output intended for effectors.

A. Standard Interfaces

In the first case, a suite of standard interfaces has been designed to allow access to low-level hardware and low-level functionality. Examples include sensors and motion control. Illustrated in Figure 2 is a portion of the interface for Motion.
These interfaces are provided across all low-level functions for a standard interaction with the hardware.

```c
/* @param speed required speed in cm per second */
* @return errNone if successful */
* @param sspn速度 = speed */
* @param step = (leftStep == step) && (rightStep == step)
* @param step = (step = true)
* @param reset = step = false */
* @param rotateAdj = the inc. speed in cm/s of radial speed */
* @return errNone if successful */
* @param True */
* @param (leftStep = leftStep' + rotatedAdj) &&
* (rightStep = rightStep' - rotatedAdj) */
* @param leftStep, rightStep = step */
* @param (leftStep = leftStep') && (rightStep = rightStep') */
* @param (leftStep = leftStep + rotatedAdj) */
* @param (rightStep = rightStep - rotatedAdj) */
* @return errNone if successful */
* @param True */
* @param (leftStep = leftStep') && (rightStep = rightStep') */
* @param (leftStep = leftStep + rotatedAdj) */
* @param (rightStep = rightStep - rotatedAdj) */
* @return errNone if successful */
* @param True */
* @param (leftStep = leftStep') && (rightStep = rightStep') */
* @param (leftStep = leftStep + rotatedAdj) */
* @param (rightStep = rightStep - rotatedAdj) */
* @return errNone if successful */
* @param True */
```

Fig. 2. Extract of the Motion interface.

The VRF uses predefined standards within the system and these are built into the interface contracts. An example of this would be the Speed measurement, which is always available in cm/s before any fuzzification. Both the crisp and fuzzy values are available to accommodate cases requiring exact values, however this will mean a more tightly coupled control component that may no longer be completely portable. Sometimes though, it may be more important to be precise than portable.

B. Robot Configuration

The VRF specifies the robot configuration such as sensor and effector positioning as well as the size and extremities of the robot. This information is loaded into the VRF from a configuration file, which is written once for each robot type and altered according to hardware changes. A configuration file using XML is very flexible, as used by [3] and [10]. The XML configuration file use to specify the robots (see Figure 3) was based on [10]. This way the VRF can be easily modified for new robots, or along with changes in an existing robot hardware configuration.

```
<Robot RobotType="Kropera">
  <Dimension Height="70"/>
  <Polygon
    NPoints="6"/>
    YPoints="-40, 40, -40, 0, 40"
    <CenterOfRotation
      XCenterOfRotation="0"
      YCenterOfRotation="0"
      <Drive
      DistanceLeftRightWheel="40"
      Wheel1 Diameter="20"
      MaxSpeed="300"
      ...
    <Robot>
```

Fig. 3. An extract of the XML configuration file defining the physical structure of a robot.

The XML configuration file also gives the VRF the appropriate hardware placements of sensors and cameras etc. (see Figure 4). Sensors can be grouped together to form zones of measuring (each group having its own membership function as to its direction from the robot). Configuration details can be made available to the components on initialisation to check prerequisites as well as at run-time when it can respond to high-level queries about the robots current state.

```
<Sensor/>
  <Label>1</Label>
  <Infrared Type="IMRARK"/>
  <SensorGroup1=SensorsGroup>
    <ScanRange>55</ScanRange>
    <SensorDistance>0</SensorDistance>
  </Infrared>
  <Position
    XPosition="17"
    YPosition="15"
    ZPosition="8"
    <Rotation
      XRotation="45"/>
  </Sensor>
```

Fig. 4. An extract from the XML configuration file showing the definition of an infrared sensor.

C. Fuzzy Abstractions

In general, a fuzzy system [11] has three parts. The first is a fuzzification section. This section is responsible for taking real input data (also referred to as crisp data), and converting it to data that has meaning to the fuzzy system. This is done in the VRF. Next, the fuzzified data is applied to the fuzzy-rule base section. The rule base is applied and result calculated within the high-level components. The result of the rule-base section is fuzzified output data. This fuzzified output data is converted back to real or crisp data by the VRF through a defuzzification process.

The use of data fuzzification provides a solution to another facet of the portability problem. Firstly, fuzzy systems are tolerant of imprecise data. They allow a smooth progression over insignificant variations in sensor readings. Secondly, it is used to group values that have similar meaning, but have very distinct concrete representations. Consider a simple example of determining if the path in a particular direction is clear or blocked implemented by an isBlocked method presented by the VRF. One robot may have a video camera facing forward and use optical flow to determine free space, another robot may have three infrared sensors pointing forward. Very different methods of interpretation are required of these sensors. However, these values can be translated to a fuzzy value with the direction and the range of any obstacle being queried. The appropriate range can then be returned by the VRF satisfying any component that uses that VRF method. Obviously the implementation of the isblocked method will be very different for the VRF for each robot.

However, most importantly, the fuzzy membership functions can be easily decoupled from their crisp meanings. Decoupling here is the key so that the meaning of the linguistic terms can be modified easily to meet the description relative to different robots. This is what provides portability in this aspect of the
problem. The input membership function profiles used are illustrated in Figure 5.

These fuzzy values can then be used to describe hardware placement and orientation, the direction of obstacles or targets, speeds of travel and distances to the robot. All fuzzy terms are relative. By this we mean robot-centric. So an obstacle that is near to a large robot may only be far to a small robot. The VRF is configured with the fuzzy set membership profiles accordingly to make the correct interpretations. Presently, the fuzzy membership functions have a predetermined number of sets and profile shape (triangular). A negotiable number of sets and profile shape is being considered so as to provide more flexibility for the components. This would be limited by the robot hardware and possible options it can provide.

VI. IMPLEMENTATION

Prototypes of the VRF have been implemented on multiple robot systems to test its support of portability. The following robots are quite varied in features and have been used as they provide a good sample of mobile robots in general:

- **PalmBot [12]** - a triangle shaped, three wheeled, omnidirectional robot. Sensing using three infrared sensors and a camera.
- **Khepera [13]** - a small circular shaped robot approximately 4cm in diameter. Two wheeled locomotion and eight infrared sensors, light sensors and a linear camera.
- **Koala [13]** - a six wheeled robot with multiple infrared sensors and a colour camera.
- **ALBO [14]** - from Sony, a four legged walking dog inspired robot.

A VRF has been constructed for each of these robots that can support high-level components. For instance an Obstacle Avoidance (OA) component exists that operates unchanged on each of the robots by interacting with the robot’s VRF. The configuration of the sensors on each of the robots is quite varied so the VRF must interpret requests to read the sensors and return meaningful results back to the component. Similarly, navigation is directed at a high level by the OA component and interpreted by the VRF to steer the robot in a requested direction.

The OA component functions completely unaltered on each of the robots. It has to be guided by the VRF to realise each robots size, speed, sensors position & signal meanings, servo positions, and speed setting. Using a component design allows a different avoidance algorithm to be easily deployed to replace this existing one at the high-level but still interact with the same VRF.

A simple illustration of the operation of the VRF will help show the usage of the abstractions. Take the pseudo-code for a Bratienberg styled obstacle avoidance algorithm as shown in Figure 6.

```plaintext
if (isBlocked(front, veryNear)) then
  if (isClear(right, veryFar))
    then rotate(slow, right);
  else rotate(slow, left);
else moveForward(midSpeed);
```

Even this simple algorithm requires the robotic framework to know the answers to questions such as:

- Which way is ‘front’?
- What does the ‘midSpeed’ value mean to this robot?
- How do I ‘move forward’?
- How do I rotate?
- Where are the infrared sensors?
- What do the infrared sensor readings mean?

Concepts such as ‘front’ and ‘midSpeed’ are specified when configuring the VRF and its fuzzy membership sets. Concepts such as ‘move forward’ and ‘rotate’ are available as standard method calls presented by an interface use in the VRF and passed onto low-level ‘Motion’ component implementing the same interface. The configuration of the infrared sensors is given to the VRF by the XML configuration file. Interpretation of the readings is handled by an appropriate sensor software component.
Also shown back in Figure 1, are five high-level component frameworks (Navigation, Deliberation etc.) that have been considered and largely implemented. Central to these is the Data Exchange (DX) framework. These are topics of future work discussed more in Section VII. The components within these frameworks are written to support high-level robotics tasks ranging from the obstacle avoidance example to navigation using vision.

Software components (as described by [15]) are used because they provide a flexible solution of varying configurations while allowing a modular and independent implementation. Software is still tailor made to fit a solution but it is configured quickly using (and reusing) the components. This way, software components can be added or removed from the system to match hardware components - addressing the problem of diversity of composition. In our prototype implementation, the robotic framework and the components are written in Java. The component model used is the Sun JavaBeans model [9]. This allows them to operate on any platform for which a suitable JVM is available. JVMs are run on any Windows or Linux OS based robot and can also be run on the Palm OS and there are even ports for the Motorola 68k series.

VII. FURTHER WORK

The next stage in this work will be to develop further the architecture and system infrastructure. This infrastructure will mediate and liaise between the component framework for Vision, Detection, Navigation, Deliberation, and Behaviour. Each of these component frameworks would be at the same level as the VRF component framework. These subsystems all communicate via the system Data Exchange (DX).

The DX framework is essentially a typical blackboard framework [16] combined with an Java style information bus [9]. It is another topic for future work and will be discussed in the context of the overall architecture and its parts.

It is the topic of future work to specify the breadth of the scope over which a single component is expected to be portable (as in to operate unchanged) and to define some measures and techniques to clearly define this breadth. This way we can effectively measure a component's portability and compare against a set of benchmarks.

VIII. DISCUSSION

The prototype implementations have demonstrated the viability of these abstractions. Current experience with the development of the VRF has yielded promising results. The VRF of each robot is successful in supporting unchanged obstacle avoidance code and in delivering camera frames (where available) to the Vision framework for further analysis. The overhead measured in inherent time so far in the experiments has been less than 3% of the total system usage. More complicated experiments will still be performed and also measured.

The advantage of using component frameworks is that composition is dynamic; hence the functionality of the system can be changed at runtime. This provides for a very flexible solution that can manage the relationship between each component. The two-tier structure also offers a scalable way to increase the complexity of the system. The benefits discussed in this paper should also apply to these higher-level frameworks and components. For example it should be possible to reuse the Navigation components across all robots supporting the appropriate Navigation component frameworks. Thus further supporting portability and reuse.

After considering the difficulties presented by robotics software engineering it is not surprising that portability and reuse have been poorly addressed. The improvements in processor capabilities have alleviated the previous limitations that required the production of very specialised software. However the diversity of the environment in which robots work and the diversity in robot composition and configuration still cause major difficulties in robotics software engineering. The work presented here has demonstrated how this diversity can be constrained to provide a much-improved level of portability and reuse of robotics software. This has been achieved primarily through the abstraction of the hardware components, using a Virtual Robot Framework that presents standard interfaces for accessing hardware functionality. The use of fuzzy functions to abstract away from non-discrete values provided by sensors and required by effectors has allowed the diversity of real world input values to also be constrained. The combination of these two abstractions provides for some exciting opportunities to enable portability across robots with different compositions.

REFERENCES