Project Summary

Intelligent robots have not quite made it to the front lines of security and reconnaissance. A more prevalent approach is to remotely teleoperate a robot from a safe distance to investigate and if needed, disarm a threat. However as research continues in intelligent algorithms, we expect that operators will become a peer of the robot, allowing for a higher robot to human ratio. Unfortunately as algorithms become more intelligent, the practice of operators making in field hardware modifications to a robot will become an issue. Updates to the hardware will need to be reflected in intelligent algorithms either autonomously or by highly-skilled operators.

This proposal is about defining a specification that completely describes robot hardware devices in a way that enables three goals: 1) provide enough information for simulation and visualization of hardware and controllers, 2) declaratively specify the mechanism (both syntax and semantics) for requesting data and actuation, and 3) inform users of standard message types that can be obtained from the hardware to facilitate connection to existing frameworks. RDIS (robot device interface description) is a domain specific language that currently supports the syntax and semantics of differential drive robots and automates the creation of ROS drivers and command line programs. We will expand RDIS to accommodate a larger set of kinematic designs, sensors and enabled tools and frameworks to move towards a specification that can be used to communicate hardware configuration changes to intelligent algorithms or operators.

Intellectual Merit

The intellectual merit of the proposed work centers on creating a domain specific language to leverage the services provided on hardware devices by embedded firmware. RDIS (Robot Device Interface Specification) provides a mechanism for centralizing the knowledge of the syntax (how to access the service) and semantics (what is the meaning of the input/output of the service) to the embedded system itself through a fully descriptive, declarative language. This aspect of the proposed project is transformative; it could potentially affect the creation of new hardware platforms, reduce the complexity of device drivers (encourage new devices) and lower the learning curve to working with platforms. To accomplish a specification that is generalizable, this project will expand a preliminary domain model that includes base concepts (linkages, joints and transmission) and account for simulation parameters (collision, appearance, dynamics, etc) to a more general set of popular platforms and kinematic linkages. Although some frameworks use declarative specifications for hardware, none are complete or meet the needs of both simulation and capturing the interface syntax and semantics. It is expected that manufacturers will choose to RDIS enable their devices once there are more RDIS-enabled environments are available. RDIS also has applications to reconfigurable hardware platforms. RDIS becomes a mechanism for communicating changes. As a longer-term goal, research methods that learn the current configuration based on embedded sensors could also use the RDIS to communicate changes to physical configuration.

Broader Impacts
The broader impacts of the proposed work can be categorized as either enabling more efficient controller design or enabling new research. First, RDIS will allow for discovery of hardware services without needing to create point solutions that map each device to each framework. Second, the creation of an RDIS file for a custom platform will ease integration into existing tools and frameworks. Third, RDIS is the mechanism for allowing platforms to communicate hardware changes, allowing for the operator or intelligent algorithms to accommodate that change. These results will disseminated not only through high-quality publication venues but this project also enables collaboration with industry partners with similar concerns (NASA and RTP-Huntsville AL).

Keywords: Domain Specific languages, Robotics, Robot Architectures and Frameworks

1 Introduction

Robotic systems are an important class of CPS. The ability of robots to interact intelligently with the world rests upon embedded computation and communication, real-time control, and perception of the world around them[1]. Integration of sensors and actuators into a single cohesive robotics system often requires a composition of service-based components that either produce, consume or transform information in a loosely coupled, parallel manner.

Although robot controller developers can create a program to execute directly on a specific robot platform, more often frameworks are used as an intermediary between autonomous controllers and the hardware. Controllers access and manipulate robots via APIs that reflect a generalized model of the robot. The APIs access robot specific interfaces through device drivers. This approach not only speeds development through reuse of code (code is not hardware specific) but also facilitates the repurposing of controllers to other hardware platforms. Device drivers in this context act as a mechanism for mapping individual robot features to domain concepts. Device drivers are usually implemented as processes that manage the hardware connection for the framework while translating requests for data and action from the framework into formatted message requests and responses. Although some increases in latency are experienced with multiple levels of indirection, it is often considered trivial compared to the time investment of attempting to develop and debug remote applications or retrofitting algorithms for varying hardware platform configurations.

There are some opportunities inherent in this approach. Some would say that device driver development is complex. It is probably true that writing reasonable performing device drivers (like all embedded development) requires an understanding of hardware, timing issues, threading, process synchronization, performance tuning and the operating system of the driver host system. However, good examples exist within frameworks and development of reasonable drivers is not prohibitive in terms of skill set. A more pressing issue concerns the multiplicity involved in creating device drivers. As device drivers provide a programmed link between hardware and frameworks, custom drivers have to be created for each framework hardware pair. Although there are some robot frameworks enjoy more popularity than others, many frameworks are routinely used within the robotics research and education community. An additional challenge is that while there are groups that focus on framework development, in house driver support for existing platforms is usually limited to a few popular platforms and certainly does not include any custom platforms or configurations. Robot manufacturers typically provide device drivers for a subset of "supported" frameworks.

Addressing the issue of needing one-to-one mappings between frameworks and devices can introduce some efficiencies and additional flexibility that robotics developers do not currently have. However there is a bigger issue at stake. Although there are many innovations from software engineering that have been utilized successfully within robotics architectures, there is a notable exception when it comes to identifying the system of record for device syntax and semantics. In the context of robotics, it is this failure that results in major duplication in integrating hardware devices into autonomous controllers. Moreover the more important opportunity in making the hardware the system of record for its syntax and semantics is the ability to discover services and provide a component wrapper around devices that allows for composition of intelligent software.

RDIS (Robot Device Interface Specification) [2] is a DSL that captures the syntax and semantics of hardware devices with embedded controller managed external interfaces. This declarative description captures the manufacturer invariant interface made available by firmware in terms of transports and messages and how they map to domain specific concepts. RDIS is currently a preliminary prototype that supports a limited set of processing models and domain concepts. The work proposed as part of this effort will accomplish the following major innovations to RDIS as a contribution to the engineering

of cyber-physical systems: 1) extend the processing models to include a larger set of **execution semantics**, 2) extend the actuation model to include a larger set of **popular kinematic chains** from both manufacturing and exploration robotics, and 3) using RDIS as a **hardware discovery mechanism**, show composition of devices into a controller that is error-aware.

Hardware that can communicate its capabilities and programming API to a general programming tool would offload the knowledge of the hardware to the system of origin (the correct system of record). This aspect of the proposed project is **transformative** and directly impacts **engineering of cyber-physical systems** ; it could potentially affect the creation of new hardware platforms, reduce the complexity of device drivers (encourage new devices) and lower the learning curve to working with platforms. This project accomplishes a **breakthrough** using the intersection of software engineering and embedded systems. RDIS describes available services much like web-based services while also providing transport and timing specific details. While this breakthrough proposal concerns itself primarily with robotics devices (those that provide sensor information or actuation), it is expected that the process and principles developed here will be generalizable to sensors and actuators in other domains (i.e. car sensors, etc). Interactive design has bought new energy and rigor to fabrication and hardware prototyping. Incorporating interactive design principles into robotics software tools would encourage software prototyping and experimentation, providing learning experiences in computational thinking, computer science and robotics.

2 Background

2.1 Domain modeling of Robot Hardware and Model Metrics

Not too surprisingly, the Object Management Group (OMG) has taken an interest in standardizing the domain of robotics in the establishment of the Robotics Domain Task Force (RDTF) [3]. They have a small set of platform independent models (PIM) and platform specific models (PSM) [4, 5, 6] for robotics defined in terms of UML. None, however, focus on the modeling of the kinematics and dynamics of robots. The researchers in [7, 8] used Eclipse Modeling Framework to deploy application software to different device-framework pairs. One [7] used models for a P3DX robot and a QFIX-based robot to for several different platforms, including Player, Microsoft Robotics Studio, and SmartSoft [9]. Statecharts [10] were used to define robotic behavior. The other [8] proposed SMARTMARS, a meta-model derived from SMARTSOFT to use for modeling and analysis of robots. A separate researcher [11] used a similar approach to build a subsumption-based robot architecture which was deployed to a NXT Mindstorms robot. These approaches represent an alternative to working directly in a domain-specific language, instead relying on well-understood standards like UML. There are some drawbacks to this approach as at least some textual representation is still needed for the model to be serialized and passed over the network. As well, textual representations tend to simplify visual models as they do not need to include user interfaces aspects of visual models such as the position of a model element on the computer screen. However, structured visual formalisms may be compiled into a domain-specific language which could turn out to be a preferred way to model particular robots.

A secondary goal for this project will involve evaluating the proposed model in reference to other existing models and to assess the progress of the model as it grows in the long-term. Therefore, a metrics must be determined so model growth and quality may be assessed and possibly also the size in relation to other models (other models in this case being the implied models used by the aforementioned frameworks). To achieve a reference frame for how this might be accomplished, a short literature review was conducted on the current view of metrics at the model level, focusing on the metrics which appear to be useful for this project. Mohagheghi presents a survey of the state of model metrics as a

part of an ongoing research effort in model quality metrics [12]. The paper presents six general goals for measurement at the model level which should serve as the inspiration for the metrics used to evaluate a model. The ideas of Lange on model size [13] are also presented, in particular dealing with UML models. Lange proposes a specific metric for UML relative size, which could be used to measure the size of the proposed robot model against existing description models. In [14], the authors present their perception of model metrics for DSMLÕs, offer a classification of the types of effort they believe are encountered when working within a DSML, the two basic types being Development Effort and Runtime Effort. Within the context of this project, the metrics which measure Development Effort are most interesting, particularly those which measure Modeling Effort, the effort involved in developing a DSML model, and Cognitive Effort, the cognitive effort required to do so. They use a weighted sum to approximate the size of a model as a means to measure modeling effort. This metric should be fairly sufficient for calculating an inexact model size, and it should translate well between different models. Another interesting metric they use is the closeness of mapping ratio (COMR) which they used the ratio of the number of problem-level language primitives to the number of solution-level primitives which do not have mapping to the problem domain. This metric is interesting because it provides insight to the disconnect between the problem domain and the solution domain. However, to actually implement this metric would require a classification of any existing language to be benchmarked against and could be expensive time-wise. The authors also mentioned other metrics which are useful when evaluating imperative languages, however this project will focus on building a declarative language so they will not be so directly applicable. The author in [15] considers a new approach to estimating a model complexity via the analysis of the meta-model which defines it. This could solve the problem of evaluating complexity in a purely declarative model like RDIS.

2.2 Hardware descriptions in existing frameworks

Many approaches have been used to increase reusability of software artifacts. Player/Stage drivers can be used directly in ROS [16]. Microsoft Robotics Studio uses a service-based paradigm to abstract data sources. Orocos [17] leverages the component design to package related software into a reusable piece. Carmen [18], LCM [19] and ROS [16] abstract the message content from the message generation by providing a transport between modules with self-describing data. These approaches utilize fairly advanced computer science paradigms of distributed development (utilized by many Linux-based open source projects). Unfamiliarity with these paradigms may discourage many researchers, hobbyists and students from attempting to reuse software, even when it can teach domain specific concepts.

Many frameworks use a declarative description of the robots. Player/Stage [20] is both a 2D simulator and a robot control framework. Robot description files are broken into two pieces: 1) a 2D description of the robot and its sensors(Figure 1: right) and 2) a set of interfaces that abstract the data produced by hardware to a standard format. The description, used for simulating the robot, consists of a polygon-based footprint with sensor locations marked. Actuation and sensor characteristics along with parameters for simplified error models are used to complete the model of the robot. A domain-specific set of classes and message types describe what data can be obtained or how the robot can be manipulated including position (pose2d) and distance to other objects (single point is range, multipoint is laser). The classes and message types represent the interface that abstracts the robot hardware to the data that it can produce or consume. Writing software to the interfaces that a robot can utilize (rather than the specific robot) allows software to be written either for a simulated robot or a real robot, which in turns eases the transition from simulation to physical implementation.

ROS [16] targets a 3D simulation framework (Gazebo) and more sophisticated intelligent controller, which require a more rigorous description. UDRF (Uniform Robot Description Format) provides a 3D

```
driver
(
  name "create"
  provides ["position2d:0"
"power:0" "bumper:0" "ir:0" ]
  port "/dev/ttyS2"
  safe 1
)
define roomba position
(
  size [0.33 0.33 0.1]
  block( #this block approximates
the circular shape of a Roomba
    points 16
    point[0] [ 0.225 0.000 ]
    point[1] [ 0.208 0.086 ]
    point[2] [ 0.159 0.159 ]
    point[3] [ 0.086 0.208 ]
    point[4] [ 0.000 0.225 ]
    point[5] [ -0.086 0.208 ]
    point[6] [ -0.159 0.159 ]
    point[7] [ -0.208 0.086 ]
    point[8] [ -0.225 0.000 ]
    point[9] [ -0.208 -0.086 ]
    point[10] [ -0.159 -0.159 ]
    point[11] [ -0.086 -0.208 ]
    point[12] [ -0.000 -0.225 ]
    point[13] [ 0.086 -0.208 ]
    point[14] [ 0.159 -0.159 ]
    point[15] [ 0.208 -0.086 ]
  )
  bumper( bcount 2
          blength 0.33
          bpose[0] [0.12  0.12
45]
          bpose[1] [0.12 -0.12 -
45]
  )
```

```
<controller:diffdrive_plugin name="differential_drive_controller"
plugin="libdiffdrive_plugin.so">
  <alwaysOn>true</alwaysOn>
  <update>100</update>
  <updateRate>100.0</updateRate>
  <leftJoint>base_link_right_wheel_joint</leftJoint>
  <rightJoint>base_link_left_wheel_joint</rightJoint>
  <wheelSeparation>${caster_wheel_offset_y*2}</wheelSeparation>
  <wheelDiameter>${wheel_radius*2}</wheelDiameter>
  <torque>50</torque>
  <interface:position name="position_iface_0"/>
  <robotNamespace>/</robotNamespace>
  <topicName>cmd_vel</topicName>
</controller:diffdrive_plugin >
```

ROS URDF Overview for iRobot Create

Mesh/stl files attached to base link provide "skin" and collision models

JOINT: Continuous joint defines wheel actuation relative to the base

Framework specific data defines parameters for ROS/ Gazebo Simulation

LINK: Base_link

LINK: Rear_left_wheel Defines geometry (sphere), actuation type (Simple), wheel radius, friction, max velocity, damping, etc

JOINT: Rigid joint defines link to ground plane

LINK:Base_footprint

Links include visual, Inertial and collision information to support visualization of robots
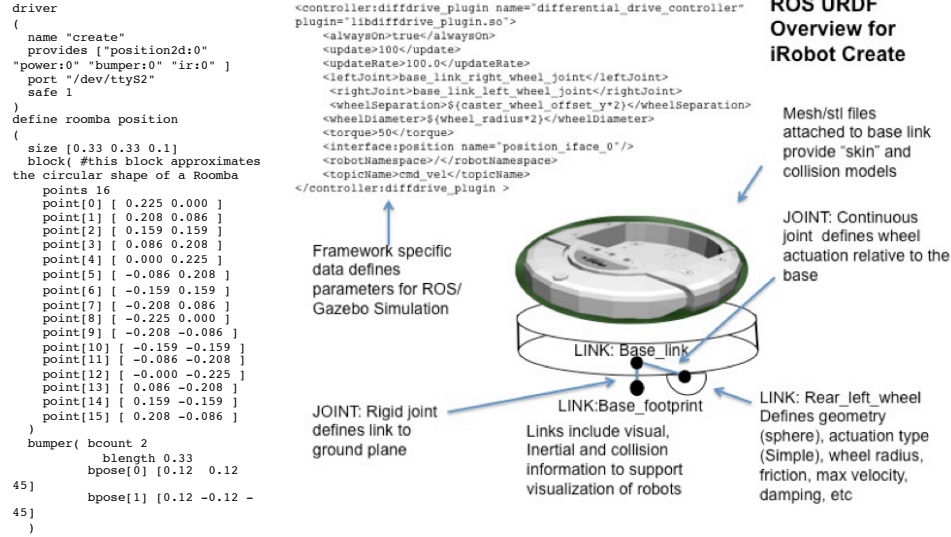
Figure 1: Sample device descriptions within frameworks.

physical description broken into links and joints to facilitate not only mobile robots but manipulators as well (Figure 1: right). Geometric bounding boxes and meshes allow for collision detection and realistic visualization. Like Player Stage, ROS utilizes a message-based model to decouple data providers from data producers. Ideally robots that provide and consume similar data types can be controlled similarly. Unlike Player Stage, URDF not only serves as a mechanism for simulating robots but also allows for the visualization of real robots in both real-time and off-line (through saved messages).

URBI [21] is a domain specific language for robot applications. It is a platform which controls robots. Its language, urbiscript, has native syntax for parallel and event-driven constructs. This gives it high expressivity for solving problems common in robots. As an example, urbiscript can easily natively express "when my left arm stops moving, start moving the right arm." However, the limitation in URBI is that it is another platform. It expects that the "UObject" needed to control the robot exists. Also, URBI focuses only on robotic control, however it can tie into simulators to be used as a control module for the simulation.

The authors in [22] present a domain specific language for the control of a novel robot. The ATRON self-configurable robot is composed of several independent modules which can be assembled in different manners to create a robot whose function changes according to how the modules are connected. They use the example of a car configuration in their paper. Their language is role-based, meaning a module can deduce its role by examining its own connectivity. The advantage to this type of description is that it allows the reuse of one behavior description for several configurations of ATRON modules. For example, a description file that represents a car may be used for four-wheel or six-wheel configurations. This is an interesting abstraction for application code, however it has the same problem as URBI, in that a platform-specific module must be created so that a robot may communicate with the framework. Although the literature reveals very few attempts at using DSLs for hardware device drivers, Thibault et al report the creation of efficient video device drivers using a novel DSL [23].

A select number of robot control frameworks move beyond visualization information and relevant interface declaration in the hardware description. PREOP, an Alice-based programming interface [24, 25, 26] for robots takes this paradigm further. Not only is 3D visualization information supplied but also the programming interface is completely specified by the selection of the robot object. This is

accomplished by linking the real-time control mechanism and exposed API available to the user within the robot object.

All robotic architectures require that the user provide information regarding the target hardware devices and the resulting data at development time in absentia of the hardware. Two problems result from this process: 1) users must understand the hardware and resulting data and 2) encode the hardware details properly in the framework of choice. A mechanism that provides the user with the details of available resources could remove this step and the associated issues with attempting to properly characterize the hardware. More importantly, hardware that can self-describe could lessen both frustration and the need for in depth hardware knowledge on the part of the user.

2.3   RDIS (Robot Device Interface Specification)

We propose a new paradigm where knowledge of the hardware mechanism is embedded in the hardware, rather than declared in software. This novel universal design called the (RDIS) Robot Device Interface Specification has three purposes: 1) provide enough information for simulation and visualization of hardware and controllers, 2) declaratively specify the mechanism for requesting data and actuation, and 3) inform users of standard message types that can be obtained from the hardware to facilitate connection to existing frameworks. The challenge in successfully defining the RDIS is in creating a model that captures the generalizable aspects of robots and provides a mechanism to specialize the aspects that vary.

RDIS (and general reuse in the community) relies upon the relatively invariant nature of mobile robots. Although some robots are built for a specific task, general use robots within education and research communities tend to leverage designs that provide closed loop inverse kinematic solutions; differential drive being part of this class of robots. In addition, many robots including the popular Mobile Robots Pioneer class, iRobot Creates, K-Team robots, Erratic ER-1, White Box Robotics Model 914, Ar.Drones, and BirdBrain Finches contain an embedded firmware controller that accepts commands via a serial, Bluetooth, WiFI or USB interface rather than require the users to download a program to onboard memory. Even robots that require a local software program to run have modes where the local software program presents an API to an external computer (i.e. Lego Mindstorms via Lejos and E-Puck). In both of these cases, the users create an autonomous controller program that communicates with the firmware to affect actuation and to obtain sensor information.

The RDIS specification can be defined as domain specific language (DSL) that is used to program robots at a higher, domain-specific level. In the application of robots, the existence of firmware (not programming hardware directly) simplifies the process. Designing a declarative specification for robot hardware requires a solid domain model of the hardware connection and services that are provided. Domain models, when designed properly, can be somewhat invariant to changes and can provide a stable basis for deciding the structure and parameters of the specification.

The preliminary artifacts generated in this approach include RDIS specifications and grammars that generate a command line program and a ROS driver. The initial scope of this research focuses on a set of domain concepts that represent popular platforms and devices and therefore are present in most frameworks as an abstraction. The current set of domain concepts include DIFFERENTIAL DRIVE and RANGE. Differential drive robots (robots with two opposing wheels on a single axis) are controllable using linear and rotational velocity. Some frameworks explicitly recognize differential drive as a kinematic design (Player) while others group all platforms under a single fully expressive interface where the user must know which dimensions are controllable. Acknowledging differential drive explicitly allows for an easier parameterization through state variables that denote odometry resolution, wheel size, offset of axis and wheel base. For example, within the ROS system, a TWIST message type

is used to provide control commands to a differential driver robot through linear and angular velocity. Mapping the TWIST message to the Koala setMotor interface involves calculating left and right velocity from linear and angular velocity and calling the setMotor interface. We map the differential drive domain concept to the SETSPEED interface (defined in the interface section. TRANSVEL and ROTVEL are parameters of the controlDIfferentialDrive abstraction that map to the interface parameters. The unit conversion (based on wheel base and encoder resolution) occurs between the interface and primitive to avoid having to encode the conversion multiple times. Once an interface is designated as fulfilling a domain concept, the mapping better the interface and the framework specific mechanism can be generated. The ROS template (discussed in more detail in [2]) maps each domain concept to the appropriate programming paradigm, which in this case is a subscription to the TWIST message and a callback handler that contains the code to map the TWIST message parameters to the interface parameters. A preliminary RDIS has been implemented for the Finch robot from Bird Brain and the K-Team Koala[2]. In this design, a JSON-based description file is used (although any syntax can be used including XML). The intermediate product is an abstract syntax tree that represents the Finch details in a domain specific model. This intermediate format can be further processed to verify conformance to the specification. End products are generated from the verified syntax tree, either in a single or multiple passes, using templates that format data based on the model.
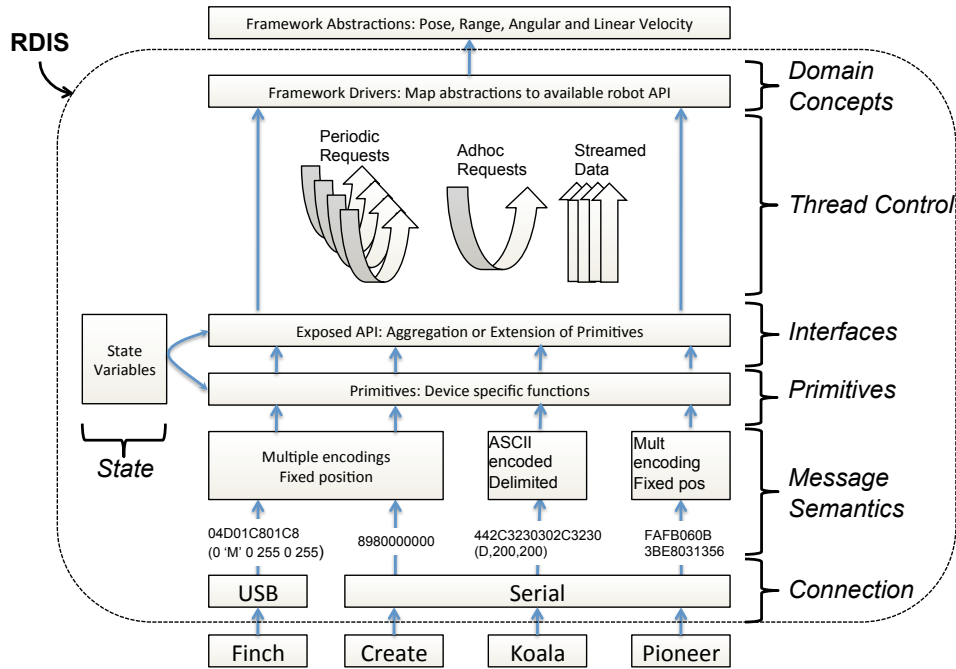
3 Research Plan

Figure 2: Preliminary domain model for mapping devices to frameworks.

This preliminary result supports the idea that general robot devices can be described declaratively in a manner that supports discovery and that links to the backend processes. The RDIS must be expanded to be useful in a larger context. The work proposed as part of this effort will accomplish the following major innovations to RDIS as a contribution to the engineering of cyber-physical systems:

1. Extend the processing models to include a larger set of execution semantics,

2. Extend the actuation model to include a larger set of popular kinematic chains from both manufacturing and exploration robotics, and

3. Create a supporting set of tools and frameworks that supports prototyping, evaluation and adoption.

3.1 Expansion of execution semantics

The ability to generalize device to framework mapping requires documenting the concepts in existing mappings. The domain model captures the invariant features of the device to framework mapping drives the contents of RDIS. In Figure 2, the domain is broken into seven related concepts: 1) connections, 2) state, 3) message semantics, 4) primitives, 5) interfaces, 6) thread control, and 7) robotics domain concepts. CONNECTION refers to the transport used to communicate from the device driver to the framework. Information needed to establish and maintain the connection is defined within this concept. The STATE definition serves two purposes: 1) define constants relevant to other sections and 2) allow for state to be saved and retrieved as part of adhoc and periodic requests. The MESSAGE SEMANTICS refers to the structure of the data which is actually sent to the robot over the connection. This specification is decided by the manufacturer and is generally uniform between primitives. PRIMITIVES focuses on describing the firmware interface in order to document the invariant features of the device. The INTERFACE declaration, acting as a logical view of the PRIMITIVES, specifies functions that are available to a developer to control a robot. DOMAIN CONCEPTS map interfaces to domain specific concepts in a framework agnostic manner. These concepts are discussed in detail in [2].
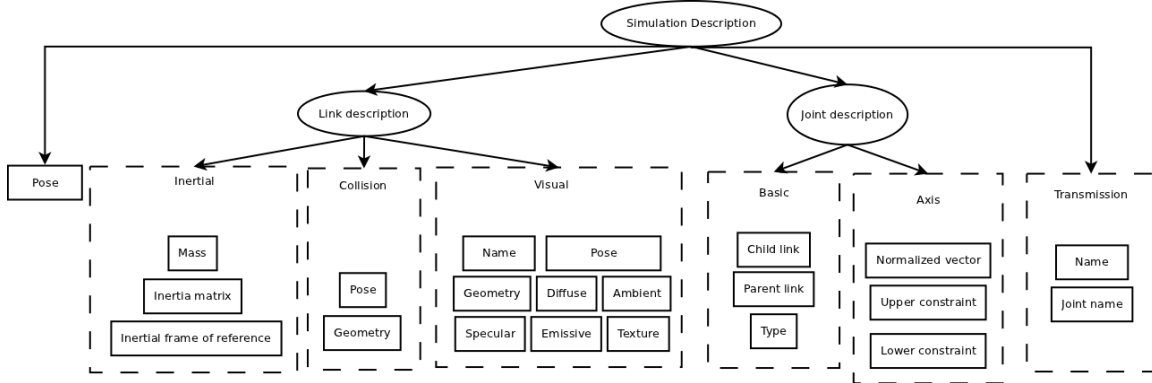

Figure 3: General model for describing kinematics, appearance, collision and dynamics parameters sufficient for simulation on any simulation platform. Currently targeted platforms include Gazebo, Webots, and ROS.

In this research, we expand the basic implementation of each concept to accommodate a larger set of embedded firmware controllers. These tasks include:

1. Addition of a complete kinematic, visual and collision description consistent with existing simulators and frameworks (see Figure 3). Although initial focus will be upon the initial kinematic design (differential drive), the proposed model is designed to accommodate other kinematic designs.

2. Standard mechanism for error handling and notification at both the communication and primitive levels. Error handling examples include re-establishment of connection or actions to take based on output parameters. This is particularly relevant to platforms that require a heartbeat request periodically.

3. Expansion of the initial single-threaded implementation to include dual and multi-threading models. A dual model accommodates different input and output threads while a multithreaded model support different frequencies for periodically submitted requests.

4. Refinement of the state concept and how it matches to primitives and interfaces would provide a scripting language for transformation of data. The state variable concept is instrumental in implementing periodic requests asynchronously from the client request/reply system.

5. Management of sensor and actuator error models consistent with existing frameworks will allow for manufacturer provided error models to be propagated to the framework or controller. An example would be a transformation of encoder error to pose error. Although all sources of error (systematic and non-systematic cannot be accounted for in this approach, it is currently better than existing approaches that abstract out specific device errors.

3.2 Expansion of domain concepts to additional kinematic chains

Domain concepts map interfaces to domain specific concepts in a framework agnostic manner. Each framework has a notion of a general set of abstract data types generated from device drivers such as pose estimates, pose relative to an identified landmark, control via linear and angular velocity along a dimension or axis, and point and fields of open distance. The DOMAIN CONCEPTS definition identifies interfaces that map to the abstract concepts allowing for templates to map the abstract concepts to the existing interfaces available on a device.

From the previous sections it is apparent that the robot kinematic modeling languages are missing necessary components for a generalized description for use in simulation. Some description formats do well in defining the geometries of the robot, but make categorical assumptions about the kinematics of the robot, offloading the actual kinematic description to the framework itself. For example. As an example, Stage has a drive "diff" statement to label a position element as having differential-drive steering, and Webots has a DifferentialWheels node which acts as the base node of a robot description having differential steering. URDF solves this by defining a relationship between transmissions and joints, but its approach is specific to the PR2 robot.

Current models of kinematics are adept at modeling the construction of the robot in a way that is almost a standard. However, what is missing between all of them is how to model controllable joints. If a solid mapping can be defined between programming interfaces and resultant efforts at joints, this would benefit the robotics community greatly as there is not currently a description format which does so without the aid of a companion plugin or hardware description. RDIS would benefit much from this component as well as it would define the bridge between the control model introduced in earlier work [1], [2] and the kinematic model which it currently lacks. Thus, the work that needs to be done is to define RDIS's kinematic model according to the trends seen from other frameworks and then provide a mapping from interfaces in the control model to transmission efforts at joints in the kinematic model. It is clear that a viable robot kinematic model will somehow take heed of some of the existing description formats to ensure there is sufficient compatibility with the model. The problem lies in deciding which parameters are essential to a proper kinematic modeling and which are parameters which lose meaning when carried to other frameworks. An additional goal is to find a concise set of these parameters such that no one parameter may be defined in terms of other parameters. It is necessary for this robot model to follow the link-joint model because it is a well understood and well-established concept within the robotics community from which much of its theory is based [6]. With the adoption of this model it is clear that the major components in the model are LINK and JOINT. These two main components will be defined and the necessary problems associated with each will be reviewed. As well, the possibility

of modeling a TRANSMISSION component will be discussed.

Because of the strength of the model, LINK and JOINT have already been examined extensively from many different viewpoints, resulting in a mature description of each of these components in just URDF or SDF. It is the author's opinion that these components are described well by these formats and there is not much to discover save for minute details. For these components a description is offered of how SDF and URDF accomplish their description.

3.2.1   Link

The LINK is an atomic rigid body in the robot's structure which constitutes the majority of the robot's composure by mass. A link is a primary parameter to the physics simulation as it contains the parameters necessary to simulate the physics of a robot. Additionally, a simulator typically has a visual representation of the robot to present to the user and a LINK will represent the majority of the visualization, so there will be parameters which are to be used by the simulation renderer. Finally, as the LINK is the only material component in the robot model, it will serve as the basis for the collision model, so the parameters to the collision engine must also be defined.

1. The PHYSICS component of the description of LINK defines the parameters needed for physics simulation. This will include the robot mass, rotational inertia matrix, and the inertial frame of reference. This portion of the model is consistent in frameworks which simulate the dynamics of robots. SDF introduces a linear and angular velocity damping in this component, however it was alone in this specification. As the implementation proceeds, it will be apparent if this component is missing any parameters.

2. The COLLISION component describes the parameters needed by the collision engine. Generally in most simulation description formats, the main components were the pose of the collision model and the simple geometry which defines it. This is to be differentiated from the visual geometry, as it is not necessarily true that a collision geometry may be used as a visual geometry or vice-versa. For instance, Webots uses an IndexedFaceSet type to represent complex or high-fidelity geometries for the visual component of a robot, however this complex type of geometry would be a strain for the collision engine so it is not available for use as a collision geometry. A challenge in defining this component is deciding which collision geometries will effectively represent most robots.

3. The VISUAL component describes the parameters provided to the scene rendering engine. Visual representations are most easily characterized by some sort of geometry, which tended to be the observed pattern within most simulation frameworks. It should be noted that not all frameworks separate the collision model and the visual model. In this case, it usually marks that the simulator is simplistic and does not provide a high definition visual representation of the simulation. In these cases, one geometry is defined to serve as both the collision and visual geometry. In the context of our model, the collision geometry should be selected as it is assumed to be the most simplistic. Solving the collision geometry problem should also determine at least a subset of visual geometries. An additional challenge to defining the visual component will be handling mesh types. Between Webots and Gazebo, there are mesh types ranging from the standard triangle mesh to polygon meshes to height maps. As well, if meshes are supported they are best referenced as they will cloud the description file.

3.2.2 Joint

The JOINT component of the model characterizes a set of constraints imposed on two joints. In this relationship, there is strictly one related parent LINK and one child LINK. There are six degrees of freedom between two unconstrained links, and a joint restrains one or many of them. The standard degrees of freedom are X, Y, Z, roll, pitch, and yaw. From the examined frameworks, three main types of joints reappear between frameworks, all of which JOINT generalizes a prismatic joint constrains all degrees of freedom except for Z. This allows child link to translate freely in one dimension with respect to the parent link's frame. By convention, this is normally the Z-axis in joint space. This type of joint might be used to model a gripper for a robot arm.

A hinge joint constrains all degrees of freedom except for pitch. This allows a link to rotate in one dimension about a single point of rotation with respect to its parent link's frame. This type of joint might be used to model a fixed wheel on a differential drive robot if no bounds were posed on the joint rotation. A ball-and-socket joint constrains X, Y, and Z. This allows for rotation of a link about any of its axes with respect to its parent link. This type of joint might be used to model a spherical caster wheel for a differential drive robot.

3.2.3 Transmission

A TRANSMISSION component is not yet well-defined in the current state-of-the-art in robot modeling, therefore it may contain some parameters which are to be discovered. From the previously discussed frameworks, URDF was the only description format which included this component, which the documentation notes was not intended to be used for anything but the PR2 robot. Generally speaking, it is known that a transmission is supposed to define how a motor applies effort to a joint or several joints. Therefore, it can be concluded that a transmission must have an association with at least one joint.

Additionally, transmissions mark a point of control for the robot application. If an application wishes to command a robot to move in some manner, then the application must utilize the appropriate interfaces, which in turn triggers one or more robot control primitives which alter the state of the joints. Therefore, a primitive must somehow relate to some transmission and subsequently the parameters provided to the primitive. For example, it could be possible to translate a motor speed to the effort applied at that joint declaratively, and this might require a modification to the previously mentioned control model at the PRIMITIVE component.

With the major components of the model identified via a survey of existing frameworks, what remains is to discover which components are required to produce a full implementation of the RDIS model. To that end, further research is required. At this point, the relationship between the kinematic model and the control model have been established. The specifics of what is contained in the kinematic and control packages is within the scope of this project. The control model is expected to change slightly to provide support for manipulation of the kinematic model during simulation, while the kinematic model will be wholly new to RDIS while the bindings between the control model and the kinematic model will be clearly defined as a contribution to the robotics modeling community.

3.3 Prototyping, Evaluation and Adoption: RDIS-enabled Tools and Frameworks

This research will generate a set of artifacts and tools to support the prototyping and evaluation of the domain model and its accompanying language specification. These artifacts encompass three categories: 1) Language definition through language parsers and generators, 2) Framework and language

bridge tools and 3) Model metrics and evaluation artifacts. The existing grammar for RDIS will be updated and maintained in a open SVN repository (see Data Management Plan).

Model/RDIS Evaluation Criteria

- RDIS should be extended to be a model which contains the association between the exposed control interfaces and the behavior which is actuated in simulation.
- RDIS should be general enough to model more than one kinematic category of robot and more than one robot within a category of kinematic design.
- RDIS should be descriptive enough so that the model is rigorously simulatable.
- There should be a mapping between the components and their associations from RDIS to at least one of the major simulation descriptions (Stage, URDF, SDF, and Webots are worthy targets).
- RDIS is concise (not redundant) as measured via model metrics.
- RDIS generated development modules should be sufficiently easy to use as evaluated based on user study and evaluation.

4   Curriculum Development Activities

Dr Anderson teaches an intelligent robotics course to senior-level undergraduates and first-year graduate students. This course will be augmented to include software engineering in robotics and embedded systems. The prevailing architectures will be presented in terms of their use of standard distributed software engineering paradigms (transparency, cohesion, synchronous vs asynchronous processing, and real-time issues at both the operating system and the applications level), use of standard tool chains to enable reuse and code generation within RDIS and frameworks. Students will learn to evaluate performance in terms of jitter, latency, and maintainability.

5   Project methodology, management, and dissemination plan

5.1   Team Member Roles, Responsibilities and Expertise

need something here about how a domain modeler is important to this research and what tools and expertise is helpful

The success of the proposed research requires both breadth and depth in robotics hardware and software engineering. The PIs main research area centers on distributed robot systems. However, her twelve years as a software engineer (culminating in a IT Architect position at IBM) has made the shortcomings of existing tools apparent. She designed the circuit board and the software for the eROSI, a novel robot platform [27]. This platform featured an ASCII message-based API available via Bluetooth and images were served wirelessly via UHF channels. These choices directly affected the platform usability for research students such as the undergraduate team at Berea College [28]. In addition, the PI has either programmed or modified drivers or firmware for robots and devices including the Koala, Finch, Create, Ar.Drone, Lego Mindstorms, Calliope (Dynamixel-based 4 DOF arm), Handyboard, GPS modules, lasers, analog and digital IR and ultrasonic sensors, and custom boards using ATMega chip and serial Bluetooth ICs to interface with Player/Stage, Alice/PREOP, ROS, Carmen, Webots and custom firmware applications. In addition, the PI has studied open source implementations of other drivers and frameworks including PSOS (Pioneer robots), Tekkotsu, and LCM. She was the organizer of the 2010 AAAI Robotics Workshop titled "Enabling Intelligence through Middleware" an effort to increase awareness of tools within the community and tool needs outside the community [29]. The PI

is in a unique position given her expertise, experience and enthusiasm to affect accessibility of robotics controller development within the larger community.

5.2 Dissemination Plan

Broad dissemination plans include both the academic and non-academic communities. Traditional academic dissemination plans include publications and conference presentations in both the educational and robotics software engineering communities. Future plans include workshops at conference venues as well as at partner campuses will provide hands-on opportunities for those that wish to use the tool for teaching or design. A website will be maintained and all software will be open source. However, primary means of dissemination will utilize partnerships with manufacturers or tool developers as we have seen this method is effective at making hardware widely available. The PI will also hold hands-on workshops both to inform and to gather feedback from the community.

Additional kinematic designs are motivated by partnerships with NASA and RTP. NASA is migrating a robotic platform from VxWorks to Ubuntu with a future update to include RTLinux. This custom kinematic chain involves manipulators with haptic sensors on a MLVDS bus (a new transport). RTP (Huntsville Research Technology Park) is an initiative to by the governor of Alabama to create a National Robotics Technology Development and Training Center at Calhoun Community College. Many robot platforms have been purchased and donated for training and are available for evaluation as candidate platforms for RDIS inclusion. Both opportunities expands the platforms that can modeled and contribute to the domain model that can be generalized across other platforms.

5.3 Methodology

The research approach consists of iterative design and integrated usability checks. Rather than attempt to create the research tools in a single iteration, it is better to start with a core of features and iteratively add to those features. This approach allows for the usability testing within the first 18 months as an input into the next iteration. The schedule details the work for two graduate students for the three years: one student to focus on analysis of platforms and the evolution of RDIS and one student to focus on the graphical tool for building and incorporating the RDIS into tools and frameworks. Undergraduate students will assist with coding, testing and support activities. The project team will meet weekly to discuss progress on features and use cases and any design issues. Ongoing activities include dissemination through web sites, conferences and site visits. Target tools and languages include Eclipse, C, C++ and Python. Target frameworks include ROS, Player/Stage, PREOP and URBI as license allows. Target differential drive platforms include Pioneer, Koala, ER-1, custom based on android platform and RC-car base (skid-steer). Other kinematic chains will include the Calliope and Chaira, two mobile manipulation platforms created at CMU. Additional platforms will be evaluated based on partnerships with manufacturers and industrial labs.

5.4 Tasks and timeline

6 Intellectual Merit

RDIS provides a mechanism for centralizing the knowledge of the syntax (how to access the service) and semantics (what is the meaning of the input/output of the service) to the embedded system itself through a fully descriptive, declarative language. To accomplish this, we propose an iterative approach that builds a domain model that will be modified to support an increasing larger set of popular sensors

Quarter	Task
Q4 2012	Definition of new kinematic model (joints, links and transmissions)
	Retrofit to differential drive
	Update accompanying RDIS generators and links to frameworks;
Q1 2013	Expand semantics expressiveness (threading, errors, etc)
Q2 2013	Evaluate current iteration of RDIS and tool against target differential drive robotics
	Identify collaborators that manufacturer sensors, actuators and platforms using differential drive
Q3 2013	Modify model, RDIS and tools to accommodate skid-steer
Q4 2013	Evaluate current iteration of RDIS and tool against target skid-steer
Q1 2014	Modify model, RDIS and tools to accommodate mobile manipulation platforms
Q2 2014	Evaluate current iteration of RDIS and tool against target mobile manipulation platforms
	Identify collaborators that discover kinematic parameters
Q3 2014	Modify model, RDIS and tools to accommodate bus transports (MLVDS) and RTLinux
	Summer partnership with NASA
Q4 2014	Evaluate current iteration of RDIS and tool against target platforms running under RTLinux
Q1 2015	Modify model, RDIS and tools to accommodate gantry robots and x-y-z robots
Q2 2015	Evaluate current iteration of RDIS and tool against target manufacturing platforms
	Identify opportunities for incorporation into manufacturing robot software platforms
Q3 2015	Publish final version RDIS, tools and adapters

Table 1: Timeline of proposed tasks.

and kinematic designs. This domain model will drive the evolution of the RDIS and accompanying toolsets to greatly improve the ability of developers and researchers to incorporate new platforms and sensors in a wider range of frameworks. This aspect of the proposed project is transformative; it could potentially affect the creation of new hardware platforms, reduce the complexity of device drivers (encourage new devices) and lower the learning curve to working with platforms.

Although there are many benefits to utilizing the RDIS outside of the hardwareâ the ultimate goal is best met through embedding the robot device descriptions within the device. Discovery occurs when the design environment queries the device for its supported services (or APIs). The initial approach for platforms that support onboard reconfigurable firmware is to augment the firmware to support a single command that communicates the RDIS. The information provided by the RDIS can be used by any RDIS-enabled development environment. It is expected that manufacturers will choose to RDIS enable their devices once there are more RDIS-enabled environments are available. The Finch designer has already agreed to update the firmware of a couple of select models and has committed to including the command once it is more stable and is being adopted. In the "Sketching in Hardware" community, many of the robotic devices are custom built by connecting actuators and sensors to netbooks or single board computers (SBCs). Although the RDIS cannot completely specify all possible interactions with every sensor and actuator, the communication to the computer and to the peripheral connection mechanisms can be queried and described.

Reconfigurable hardware provides a unique challenge. Rather than a static description, the RDIS becomes a mechanism for communicating changes. As a longer-term goal, research methods that learn the current configuration based on embedded sensors could be appropriately applied here. Either building blocks or configurable linkages would embed proprioceptive sensors to establish kinematic parameters needed for programming. This kinematic learning can also be accomplished by augmenting existing systems with small standalone, networked sensors that are placed at joints. Although initial iterations will focus on static hardware designs, the application to reconfigurable robots could enable

"on-the-fly" physical reconfiguration with reasonable ability to program new capabilities in the field.

7 Broader Impact

The broader impacts of the proposed work can be categorized as either enabling more efficient controller design or enabling new research. RDIS existence for a given platform will make available a wider range RDIS-enabled frameworks and tools. Development environments such as Eclipse will be augmented with plug ins that read the RDIS (from the device or from an online repository) that will present the developer with visual representations of the services and will allow for software creation through component composition for those not interested in using frameworks. Developers of custom environments can create an RDIS and will have access to standardized tools without needing to write device specific plugins for easy framework. Developers can use the error information propagated from the RDIS as a starting point for modeling error covariances.

The proposed work also enables new research. The RDIS will change the components in robotics kits so that they can be reconfigured. This requires additional research in kinematic discovery with low cost components [30]. New instruments for measuring self-efficacy in robotics skills will enable the validation of new tools and techniques in robotics tool development. Automatic component discovery can be leveraged in other domains where the interfaces are fairly static and there are many data providers and many candidate algorithms. This research enables a new model of reuse within the research community that could advance validation of published results through sharing of software artifacts.

Additional broader impacts relate directly to the activities of the PI. The PI serves as an active mentor within her university and research community. She advises senior project groups in computer science and electrical engineering. As a PI in the ARTSI alliance (Advancing Robotics Technology for Societal Impact), she has fostered productive relationships with HBCUs (historically black colleges and universities) that have resulted in talks, workshops, sharing of technical expertise and REU student placement (seven students to date). Other REUs have been hosted from the Software Engineering REU site at UA and the DREU program. All REUs/undergraduate/graduate advisees receive active mentoring including introductions to graduate faculty at other schools where appropriate, review of application materials, recommendation letters, job reference letters and publication, speaking and attendance opportunities at conferences such as Grace Hopper, Richard Tapia, IJCAI, AAAI, ICRA and IROS.

8 Results from Prior NSF Support

1. Monica Anderson - PI for the NSF Grant: CCLI: Enhancing student motivation in the first year computing science curriculum (DUE- 0736789), 04/01/2008-08/31/2011. This is an ongoing project that addresses retention issues due to motivation through specially designed modules that use PREOP in a CS0 or CS1 laboratory setting. We provided PREOP workshops to teachers at the ARTSI Faculty Meeting, demonstrated PREOP at the Google education summit and added PREOP to Alice instructor training (July 2011). Three papers have been published including two conference papers and two journal papers [31, 32, 33, 26].

2. Monica Anderson - PI for the Grant: ARTSI-Advancing Robotic Technology for Societal Impact (BPC-A- 0742123), 09/01/07-12/31/2012. This is an ongoing broadening participation grant that increases the number of students interested in computer science and robotics through both

research and creative experiences. Robotics programs have been established or enhanced at 14 HBCUs. The PI has provided each school with both onsite and remote instructional support ranging from hands-on workshops on open-source robotics development environments and controller development to hardware assembly instructions. Seven summer research assistants have been hosted at UA, with one student contributing to an accepted paper [25] and resulting in numerous poster presentations.

3. Monica Anderson - PI for the Grant: SGER: Impact of Intermittent Networks on Team-based Coverage (IIS-0846976), 09/01/2008-08/31/2009. This project investigated both surveillance in constrained environments using limited resources and methods for coordinating robot teams with limited communications capability. The novel approach of using ALUL and a QOS metric allowed for balanced target acquisition and tracking [34, 35, 36, 37].

4. Monica Anderson - Co-PI for the Grant: EMT: Collaborative Research: Primate-inspired Heterogeneous Mobile and Static Sensor Networks (CNS: 0829827) 09/01/2008-08/31/2011. This project investigates bio-inspired approaches for multirobot coordination. We show that given a greedy approach to selecting the next point to search, observation information provides more relevant input to the next action selected. Experimental results show that using observation to infer state and intent to disperse nodes performs better than non-communicative and potential field based cooperation and in some cases direct communications. [38, 39, 40, 41, 42].

5. Monica Anderson - PI for the Grant: University of Alabama Participation of EPSCOR Institution in SSR-RC: QOS service metrics for unmanned surveillance (IIP: 1026528). 06/15/2010-06/14/2012. We consider ALUL as a metric that indicates the target acquisition ability of a particular system configuration. When used in conjunction with temporal constraints, we can intelligently automate camera coverage to improve target acquisition and tracking performance of the system. The results from the experiments confirm that the coverage in constrained environments when the existing camera configuration cannot view large portions of the region of our interest improves when ALUL is considered. To date, one conference paper and one journal article have been accepted [43, 44].

6. Eugene Syriani - new investigator and has no prior NSF support.

References

[1] Trinkle, J. and Krogh, B. (2008). CPS Virtual Organization.

[2] Anderson, M., Kilgo, P., and Bowman, J. (2012) RDIS: Generalizing domain concepts to specify device to framework mappings. *International Conference on Robotics and Automation*, Saint Paul MN.

[3] DTF, R. Object Management Group.

[4] Robotics Domain Task Force. Robotic Technology Component (RTC) 1.0. Technical report, Object Management Group. Technical Report.

[5] Robotics Domain Task Force. Super Distributed Object (SDO) 1.1. Technical report, Object Management Group,. Technical Report.

[6] Robotics Domain Task Force. Robot Localization Service (RLS) 1.0. Technical report, Object Management Group. , **?**

[7] Schlegel, C., Haß ler, T., Lotz, A., and Steck, A. (2009) Robotic software systems: From code-driven to model-driven designs. *Advanced Robotics, 2009. ICAR 2009. International Conference on*, pp. 1–8. IEEE.

[8] Steck, A. and Schlegel, C. (2010) Towards quality of service and resource aware robotic systems through model-driven software development. *Arxiv preprint arXiv:1009.4877* , **?**

[9] Schlegel., C. Smartsoft.

[10] Harel, D. (1988) On visual formalisms. *Communications of the ACM*, **31**, 514–530.

[11] Trojanek, P. (2011) Model-driven engineering approach to design and implementation of robot control system. *2nd International Workshop on Domain-Specific Languages and models for ROBotic Systems*.

[12] Mohagheghi, P. and Dehlen, V. (2009) Existing model metrics and relations to model quality. *Software Quality, 2009. WOSQ'09. ICSE Workshop on*, pp. 39–45. IEEE.

[13] Lange, C. (2007) Model size matters. *Models in Software Engineering* , **?**, 211–216.

[14] Wu, Y., Hernandez, F., Ortega, F., Clarke, P., and France, R. (2010) Measuring the effort for creating and using domain-specific models. *Proceedings of the 10th Workshop on Domain-Specific Modeling* 14. ACM.

[15] Sprinkle, J. (2010) Analysis of a metamodel to estimate complexity of using a domain-specific language. *Proceedings of the 10th Workshop on Domain-Specific Modeling* 13. ACM.

[16] Quigley, M., Gerkey, B., Conley, K., Faust, J., Foote, T., Leibs, J., Berger, E., Wheeler, R., and Ng, A. (2009) ROS: an open-source Robot Operating System. *International Conference on Robotics and Automation* , **?**

[17] Bruyninckx, H. (2001) Open robot control software: the OROCOS project. *Proceedings 2001 ICRA. IEEE International Conference on Robotics and Automation (Cat. No.01CH37164)* , **?**, 2523—-2528.

[18] Montemerlo, M., Roy, N., and Thrun, S. (2003) Perspectives on standardization in mobile robot programming: the Carnegie Mellon Navigation (CARMEN) Toolkit. *Intelligent Robots and Systems, 2003. (IROS 2003). Proceedings. 2003 IEEE/RSJ International Conference on.*

[19] Huang, A., Olson, E., and Moore, D. (2009) Computer Science and Artificial Intelligence Laboratory Technical Report Lightweight Communications and Marshalling for Low-Latency Interprocess Communications. Technical report. MIT.

[20] Vaughan, R. and Gerkey, B. (2007) Really Reusable Robot Code and the Player / Stage Project. *Engineering for Experimental Robotics,* , **?**

[21] Baillie, J.-C. (2005) URBI: towards a universal robotic low-level programming language. *2005 IEEE/RSJ International Conference on Intelligent Robots and Systems* , **?**, 820–825.

[22] Schultz, U., Christensen, D., and Stoy, K. (2007) A domain-specific language for programming self-reconfigurable robots. *Proc. Wksh. on Automatic Program Generation for Embedded Systems (APGES 2007), Salzburg, Austria.*

[23] Thibault, S. A., Marlet, R., and Consel, C. (1999) Domain-specific languages: from design to implementation application to video device drivers generation. *Software Engineering, IEEE Transactions on*, **25**, 363–377.

[24] Cooper, S., Dann, W., and Pausch, R. (2000) {Alice: a 3-D tool for introductory programming concepts}. *Proceedings of the fifth annual CCSC northeastern conference on The journal of computing in small colleges* , **?**, 107—-116.

[25] Wellman, B. L., Downing, S., Moore, G., and Anderson, M. (2009) Observation-based Cooperation in Mobile Sensor Networks : A Bio-Inspired Approach for Fault Tolerant Coverage. *ISCA First International Conference on Sensor Networks and Applications (SNA-2009)*, pp. 26—-30.

[26] Anderson, M., Vrbsky, S., Brown, M., Wellman, B. L., and Mckenzie, A. Affecting Attitudes in First-year Computer Science using Syntax-free Robotics Programming. *ACM InRoads* , **?**, 111—-123.

[27] Walter, M., Anderson, M., Burt, I., and Papanikolopoulos, N. (2007) The design and evolution of the eROSI robot. *2007 IEEE International Conference on Robotics and Automation*, pp. 2984—-2989.

[28] Isaacs, M., Anderson, M., Ashworth, S., Blackburn-Lynch, J., Bynum, B., Pearce, J., and Pemberton, C. (2006) Breaking Up Is Hard To Do - Dispersion: From Design to Implementation. *Proc. of the IEEE Intl. Conf. on Robotics and Automation (ICRA), Orlando FL.*

[29] Monica Anderson, A. L. T. (2011) Enabling Intelligence through Middleware: Report of the AAAI 2010 Workshop. *AAAI Magazine*, **32**.

[30] Croxell, J. R., Weinberg, J. B., Krauss, R. W., and Smith, S. R. (2008) Robotic Limb Calibration : Accelerometer Based Discovery of Kinematic Constants. *Measurement* , **?**, 1—-4.

[31] Davis, J., Wellman, B., Anderson, M., and Raines, M. (2009) Providing Robotic Experiences Through Object-Based Programming (PREOP). *The 2009 Alice Symposium, Duke University*, pp. 2—-6.

[32] Wellman, B. L., Davis, J., and Anderson, M. (2009) Alice and robotics in introductory CS courses. *The Fifth Richard Tapia Celebration of Diversity in Computing Conference: Intellect, Initiatives, Insight, and Innovations*, pp. 98—-102. ACM.

[33] Wellman, B., Anderson, M., and Vrbsky, S. (2009) PREOP as a tool to increase student retention in CS. *Journal of Computing Sciences in College*, **25**, 167—-175.

[34] Veluchamy, A. and Anderson, M. (2010) Target Acquisition in Resource Constrained Stationary Camera Systems. *International Conference on Robots and Systems*.

[35] Alexander, Q., Hockett, H., Veluchamy, A., and Anderson, M. (2009) Acquisition of 2-D ground truth data in multirobot experiments. *2009 Robotics: Science and Systems Workshop on Good Experimental Methodology in Robotics*.

[36] Mckenzie, A., Gay, D., Nori, R., Davis, J., and Anderson, M. (2010) Comparing Temporally Aware Mobile Robot Controllers Built with Sun's Java Real-Time System, OROCOS' Real-Time Toolkit and Player. *International Conference on Robotics and Systems*.

[37] McKenzie, A., Dawson, S., and Alexander, Q. (2009) Using real-time awareness to manage performance of Java clients on mobile robots. *Intelligent Robots and Systems, 2009. IROS 2009. IEEE/RSJ International Conference on* , **?**, 3422—-3428.

[38] Wellman, B. L., Downing, S., Moore, G., and Anderson, M. (2009) Observation-based Cooperation in Mobile Sensor Networks. *Sensor Networks and Applications*, pp. 26—-30.

[39] Dawson, S. (2011) Identification of Issues in Predicting Multi-Robot Performance through Model-Based Simulations. *Intelligent Control and Automation*, **02**, 133–143.

[40] Dawson, S., Wellman, B. L., and Anderson, M. (2010) Using Simulation to Predict Multi-robot Performance on Coverage Tasks. *International Conference on Robots and Systems*.

[41] Dawson, S., Wellman, B., and Anderson, M. (2011) Categorizing interference in real robot experiments. *IEEE International Conference on Systems, Man and Cybrnetics,*.

[42] Wellman, B., Davis, J., and Anderson, M. (2009) Cooperative Multirobot Systems. *Cyber Physical Systems Luncheon and Senate Briefing*.

[43] Veluchamy, A. and Anderson, M. D. (2011) Balancing target acquisition and target tracking in existing resource constrained stationary camera systems. *Intelligent Service Robotics*, **4**, 181–190.

[44] Dukeman, A. and Anderson, M. (2011) A CSP Solution to Multi-Camera Surveillance and Target Tracking. *IEEE Systems, Man and Cybernetics 2011*.

Data Management Plan

Types of data produced by this project include: SOURCE CODE

- source code for programs that create, validate and use the RDIS data

- grammars and programs that will be used to generate source code for creating, validating and using the RDIS data

- source code for framework integration including ROS nodes, packages and stacks

- source code for eclipse plug ins

- sample programs, documentation and build files

- source code for RDS-enabling the Roadshow-in-a-Box web-based application

CURRICULUM

- curriculum modules for the addition of software to robotics class

HUMAN SUBJECT IDENTIFIABLE DATA

- completed surveys and interview notes and logs gathered from user studies of RDIS and RDIS-enabled tools

Standards to be used for data and metadata format and content SOURCE CODE: The practice of making code available via source code repository is deemed sufficient. Standard file naming, organization and coding style practices will apply based on the language and platform being targeted.

CURRICULUM: Documents will be made available in PDF, Word or Latex formats.

HUMAN SUBJECT IDENTIFIABLE DATA:Survey instruments and interview questions will be made available as part of publications.

Policies for Access and sharing SOURCE CODE: Source code will be available via publicly accessible, web-enabled source control as its results are published.

CURRICULUM: Curriculum modules will be available on the project website.

HUMAN SUBJECT IDENTIFIABLE DATA: Since the data will be from human subjects, appropriate protocols will be instituted to protect individualÕs privacy. Evaluation activities will be approved by the University of Alabama Institutional Review Board. Confidentiality of participant information will be placed as a high priority in all evaluation activities. Hence, data accessible to the project and to other researchers will not be identifiable with an individual. All data will be stored using code number identifiers in encrypted files. Access to files will be through passwords provided to appropriate project staff. Data will be stored in locked filed cabinets in a secure site and on portable hard drives by dedicated university encrypted computers at a secure location readily accessible to the project PI and personnel. Raw data will be accessible only to appropriate project staff designated by the PI.

Policies for Re-Use SOURCE CODE AND CURRICULUM MATERIALS: We will release all software artifacts developed in association with this project under the *BSD 3-Clause License* (the "New BSD" license) and will make them available via an appropriate outlet such as Google Project Hosting.

HUMAN SUBJECT DATA will only be available in aggregated form for the protection of the participants. Detailed aggregate findings will be available as technical reports and will also be published.

Plans for Archiving Data and for Preservation of Access PERSONALLY IDENTIFIABLE DATA will
be stored in locked file cabinets in a secure site and on portable hard drives by dedicated encrypted
computers for 9 years from the beginning of the project and destroyed in a secure manner at the
end of that period. Raw data will be accessible only to appropriate project staff designated by the
PI. Long term access will be available from secure storage at the University of Alabama upon
agreement to a written request sent to the project PI or, if the PI is not available, to the University
of AlabamaÕs Office of Sponsored Programs. Payment to the university of costs incurred for the
copying and delivery of data will be required.

Other Notes: We will publish our results in leading conferences and journals. As is typical in the
software engineering community, we will rigorously define our experimental procedures. For example,
for each case study we will provide the definition and context, including subject software systems,
benchmarks, metrics, and the study setting. In addition, we will document our research questions and
hypotheses, our data collection and analysis procedures, results, and threats to validity. We will provide
access to our data files via online appendices.

We will distribute annual reports via our web site.