Generating a ROS/JAUS Bridge for an Autonomous Ground Vehicle

Patrick Morley
The University of Akron
pjm39@zips.uakron.edu

Alex Warren
University of Arizona
amwarren@email.arizona.edu

Ethan Rabb
Washington University in St. Louis
ethanrabb@gmail.com

Sean Whitsitt
University of Arizona
whitsitt@email.arizona.edu

Matt Bunting
University of Arizona
mosfet@email.arizona.edu

Jonathan Sprinkle
University of Arizona
sprinkle@ece.arizona.edu

ABSTRACT
Robotic systems have benefitted from standardized middleware that can componentize the development of new capabilities for a robot. The popularity of these robotic middleware systems has resulted in sizable libraries of components that are now available to roboticists. However, many robotic systems (such as autonomous vehicles) must adhere to externally defined standards that do not contain a large repository of components. Due to the real-time and safety concerns that accompany the domain of unmanned systems, it is not trivial to interface these middleware systems. However, previous attempts to do so have succeeded at the cost of *ad hoc* design and implementation. This paper describes a domain-specific approach to the synthesis of a bridge between the popular Robotic Operating System (ROS) and the Joint Architecture for Unmanned Systems (JAUS). The domain-specific nature of the approach permits the bridge to be limited in scope by the application's specific messages (and their attribute mappings between JAUS/ROS), resulting in smaller code size and overhead than would be incurred by a generic solution. Our approach is validated by tests performed on an unmanned vehicle with and without the JAUS/ROS bridge.

1. INTRODUCTION
Robotics, specifically robotic ground vehicles, are a rapidly growing field of interest in both academia and industry today. These systems tend to be developed with component based architectures that work through a message passing paradigm.

The pervasive nature of TCP/IP networks has permitted roboticists to depend on traditional ethernet packets for message passing. By far the most popular robotic platform is ROS (Robot Operating System) [5] , but there are several other significant architectures in use, such as the Joint Architecture for Unmanned Systems (JAUS) [7]. JAUS in particular deserves attention, since it is a standard developed for autonomous systems in the defense industry.

As in most engineering solutions, the requirements of the project in question frequently dictate the platforms in use. JAUS compliance is more typical in the defense industry in the United States, while ROS is often preferred by the open source community. Given its open source software development and contribution model, ROS has a large library of components available to run many different sensors and simulations off the shelf, without requiring the developer to spend valuable time designing low level drivers or physics environments.

Most robotic middleware is not expressly designed to be able to communicate with other middleware, and ROS and JAUS are no exceptions. While communication is possible, the development of a generic driver or adapter that will automatically connect JAUS to ROS by rewriting sent messages is not feasible for several reasons: (i) message types can be defined by the user, and must therefore be matched to the other domain (ii) components on either side of the bridge may need to send heartbeat and other messages at a certain rate, in order to avoid timeout failures of the components; and (iii) to produce a generic component that is capable of handling the combinatorial explosion of these possibilities would be unwieldy to maintain, and would consume a significant amount of system resources at compile time and runtime.

1.1 Contribution
This paper approaches the problem of developing such a bridge through application-specific configuration: namely, using domain-specific concepts, we define a method to synthesize the ROS/JAUS bridge for the message types that are defined for a specific application. Our approach is to integrate the existing message types in both ROS and JAUS, and define their attribute mappings in order to solve issue (i) above; the generated code then takes care of the necessary boilerplate requirements of item (ii); because the messages are limited to those required by a specific application, the complexity risks of (iii) are mitigated.

To validate our approach, we consider hardware-in-the-loop verification of a sufficiently bi-similar execution of the JAUS-only, and ROS/JAUS implementations of two controllers running concurrently. We show that these two simulations are similar enough to warrant the replacement of JAUS components with ROS nodes for time critical operations.

2. BACKGROUND
The work described in this paper requires domain-specific modeling approaches to map attributes of data passed between programs. In order to understand the complexities that are inherent in this approach when the programs are used to control an autonomous vehicle, we provide some background material.

2.1 Component based systems
The complexity involved in autonomous vehicles projects makes them ideal implementation platforms for component-based systems. These projects require the integration of software that can manage sensing, control of vehicle actuation, and logging information for safety and debugging purposes. Component-based designs per-

mit a functional decomposition of the tasks involved into atomic processes allowing them to communicate via message passing, abstracting even whether a set of tasks are operating on the same machine or across a network.

ROS (Robot Operating System) [5] is a widely used open-source architecture in robotics, capable of interaction with a significant number of available sensors, simulators, and programs. The availability of a large selection of previously-developed code makes it ideal for code reuse in larger research projects allowing research scientists and engineers to concentrate more on the research aspects of the project rather than reimplementing trivial software packages and programs. As an open-source technology, users are free to modify the message types and their attribute specifications, as well as add as many more message types as they like.

JAUS relies on message passing for its inter-component communication, though its design is more of a top-down approach. As a standard, it provides relatively few message types, which have a fixed number of attributes. Users are free to add additional messages, though clearly without coordination between other developers, these additional messages may not be understood by other systems if they are received.

While ROS and JAUS are both component based systems with (generally) a one to one correspondence between concepts, they do not use the same terminology. For the purposes of brevity and clarity in this paper, the terms used herein will be described and linked between the two standards in this subsection. These terms can be used interchangeably, and this paper will attempt to use the correct terminology in the differing contexts. The base level task or component in a ROS system is called a “node” while in JAUS it is called a “component” In order to pass data between components in the system ROS uses the term “topic” while JAUS uses the term “message”. In ROS, “nodes” must “publish” data in order for other “nodes” to access or “subscribe” to that data. In JAUS, “components” can send single “messages” or they can initiate a “service connection” where one node sends “messages” at a determinate frequency to the requesting “component”.

2.2 Related Work

Some implementations of ROS to JAUS have already been completed. For instance, the Army Research Laboratory (ARL) developed a ROS to JAUS bridge [6]. However unlike the implementation described in this paper, the bridge developed by the ARL does not use code generation to build customized ROS to JAUS bridges. Instead it relies on human developers to adapt existing code to suit their own needs. Also, a team from Case Western Reserve University (CWRU) at the 2010 Intelligent Ground Vehicle Competition (IGVC) used both ROS and JAUS components in their vehicle’s software [8]. However, much like the ARL project, the CWRU team did not implement a code generator to build their interface between the two standards.

There are many other robotic middleware systems available, and the approach we describe in this paper can be extended to include them as necessary. However it is important to note that the work discussed in this paper is not simply an extension of bridges for software oriented architectures [4], or traditional middleware [3]. This is because, as a robotics platform, the timing requirements for many components are tight. Components are frequently defined to “time out” if they do not receive active communication from their providers, in order to degrade to a safe mode. These kinds of domain-specific requirements mean that a new approach must be taken in our code generators, to mitigate these issues.

Figure 1: The CAT Vehicle

2.3 JausML

The work described in this paper on generating ROS to JAUS bridges is meant to be integrated into JausML [9, 12]. This modeling language is currently capable of generating everything necessary to build a JAUS system, but does not include anything for generating ROS artifacts from the models. The ROS to JAUS bridge code generator will open the doorway to including those components in future work of JausML.

JausML is built using the skeleton design method (an extension of the more common template design method) [11]. Building the code generator for the ROS to JAUS bridges to be compatible with the skeleton design method will make integrating the final results with JausML a trivial matter.

2.4 The CAT Vehicle Platform

The testing platform for this project is a modified Ford Hybrid Escape named the Cognitive and Autonomous Testing Vehicle (CAT Vehicle, see Fig. 1). The vehicle utilizes JAUS as the standard interface for receiving state data from the vehicle, and sending controller inputs to the vehicle. All communication with the computers that manage the low level actuation and control of the vehicle requires the use of JAUS.

Switching from JAUS to ROS would be disadvantageous since it would require a redesign of the vehicle’s low level controls. In addition, switching operating systems is not a viable solution for systems that require JAUS compliance. Instead, the joint approach we describe could be used in order to allow the vehicle to operate with open source ROS component packages, but still maintain JAUS message passing capabilities.

Currently, the CAT Vehicle is simulated within a JAUS component with simple bicycle vehicle dynamics [1]. For most test cases where it is not necessary to move the vehicle over long distances these dynamics approach a reasonable approximation of the actual behavior of the vehicle. However, in order to maintain a high degree of accuracy another approach is necessary. For this reason, the Gazebo simulator in ROS is desirable since it offers an open environment in which to design an accurate model of the CAT Vehicle and additional parameters to specify the conditions under which the vehicle is driving (e.g. friction between the tires and the road or the

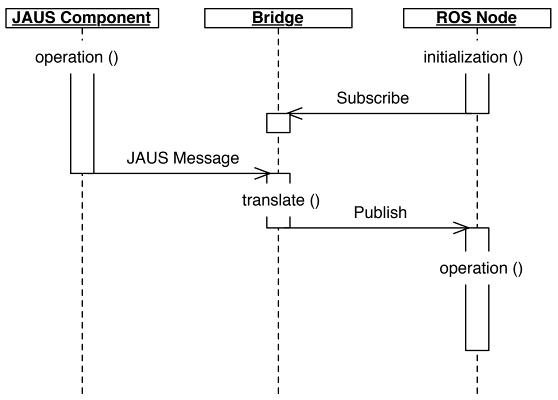


Figure 2: A single message can be sent from a JAUS component to a ROS component even though ROS has to subscribe to the content of the message beforehand.

grade of the road). Additionally, the Gazebo simulator developed for the CAT Vehicle uses a generic algorithm to adapt its parameters to the given scenario so that it performs more accurately than other simulators.

3. METHODS

Without an explicit knowledge of the messages that are passed between ROS and JAUS nodes, a bridge would need to dynamically determine how to translate the messages. Generic methods used to perform translations during runtime can be slow and have high overhead which is potentially dangerous in many real-time systems. This motivates the domain-specific approach to defining the messages passed, and the style in which those messages are passed (i.e., one-time messages, or a service connection).

It is assumed that there exists a one to one correspondence between ROS topics and JAUS messages. That is, if a ROS topic exists, then there is one and only one JAUS message that exists and contains the same data. This assumption is fairly trivial since both ROS topics and JAUS messages are data structures.

For message passing, ROS operates entirely using service connections (i.e. it uses the publish/subscribe concept to send topics). However, JAUS has the ability to send just a single message without any of the hassle required in setting up a service connection. This is advantageous in JAUS, but it creates a bit of an issue when integrating JAUS with ROS. JAUS messages that are being dynamically translated would not have a respective ROS service connection to receive the message. However, since the bridge can assume that it knows everything about the messages that will be passing through it, those service connections can be established beforehand and used to pass along the single messages that JAUS can send.

Fig. 2 shows the sequence of events to set up these connections. Upon initialization, the ROS node has to communicate with the bridge to be able to receive the information that the JAUS component will eventually send. This assumes that the bridge has already stated to the ROS portion of the network that it is capable of publishing the message that it will receive from the JAUS component.

Fig. 3 shows the process that occurs when a ROS node needs to establish a service connection to a JAUS component. Fig. 3 assumes that the JAUS component was initially set up to allow other com-

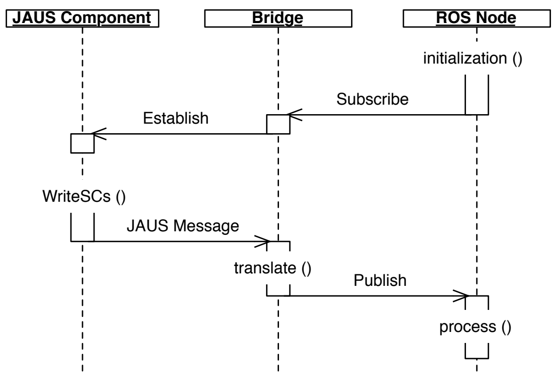


Figure 3: Service connections can be established by establishing them at the bridge and publishing from the bridge.

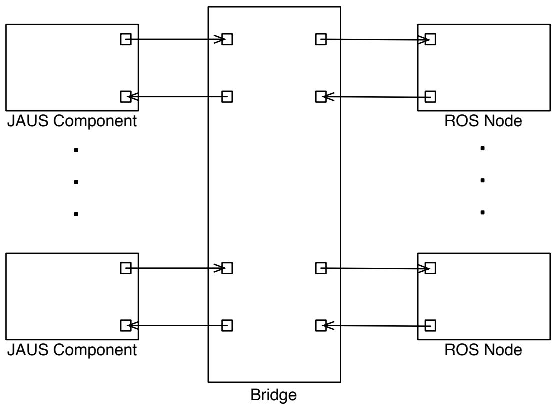


Figure 4: One bridge can service many components.

ponents to establish a service connection with it and that the bridge has similarly already told the ROS portion of the system that it can publish those messages.

Also, after establishing the service connection, the JAUS component would continue to repeat the block that writes the service connection messages. The reverse of this scenario is also possible in a similar manner. A JAUS Component could establish a service connection with the bridge which would then subscribe to content that exists on a ROS node. Assuming of course, that the bridge has already set itself up to establish service connections of those type and that the ROS node is set up to publish that information.

As an additional note: JAUS service connections operate at a given frequency (declared during the initialization process). If messages do not appear on the receiving end at that desired frequency, then that service connection may become inactive, implicitly changing the state of the component to a degraded mode. The ROS bridges described herein make certain that service connections on the JAUS side of the system stay active as long as the corresponding JAUS components and ROS nodes are alive.

3.1 Modeling the Bridge

Using the modeling syntax from JausML, Fig. 4 shows a single bridge servicing many different JAUS components and ROS nodes. While a separate bridge could be generated and deployed for each individual connection, it will likely prove more efficient to collect the different types of connections that will need to be made into one single bridge or possibly just a handful of bridges, depending on the requirements of the system.

The code generator for building these bridges is capable of building bridges with multiple connections. It should be noted that the bridge can manage multiple JAUS components subscribing to the same ROS data and similarly multiple ROS nodes establishing the same service connection. This cuts down on network traffic by preventing the publishing node/component from having to send multiple messages to the bridge to serve the same data to multiple nodes/components on the other side of the bridge.

3.2 Message transformation

In Fig. 5 an example mapping is given in the ros2jaus bridge modeling language. Although the language is not visual, the definition of this textual script permits an intermediate format that can (in future work) be automatically generated from a visual environment.

```
 struct [
   type name
 ]
 array_type [s] name

MessageName {
  JAUS jausname (
    type membername
    type secondmembername
  )
  ROS rosname (
    type membername2
    type secondmembername2
  )
  JAUS->ROS (
    membername -> membername2
    secondmembername -> secondmembername2
  )
  ROS->JAUS (
    membername2 -> membername
    secondmembername2 -> secondmembername
  )
}
```

Figure 5: An example bridge mapping in the ros2jaus mapping language.

The example shows how attribute types can be mapped to one another. Not shown in this example is that attributes can be ignored or mapped to a constant value if necessary. When invoked, the frequency of the generated bridge component can be specified.

4. RESULTS

Prior to the development of the ROS to JAUS bridges a suite of examples have been developed, which illustrate different control aspects on the CAT Vehicle both in simulation and hardware-in-the-loop. For this paper we chose an example where a dead reckoning steering controller is used to follow a trajectory for a right-hand turn, while maintaining a safe velocity for the given steering angle. This prevents the vehicle from tilting to an unsafe angle (in the extreme), but pragmatically preserves the comfort of the occupants, as described in [10].

The vehicle has a control algorithm designed to follow a preplanned trajectory with dead reckoning and another control algorithm to limit the speed of the vehicle during a turn so that it remains safe and does not flip over or slide (while driving on a flat, non-slippery surface). The steering and velocity controllers were then used to drive the vehicle along a right turn. Success in these results is indicated by a path that appears to be a right turn and a ratio between tire angle and velocity that does not exceed a predetermined value by the velocity controller. All of the paths shown for the vehicle were either recorded directly from the simulated vehicle's state or were recorded using a GPS/INS system mounted inside of the physical vehicle.

Because the controller in use is utilizing dead reckoning (and not global position information), variations are expected between the two implementations. This is because the ROS2JAUS bridge will only preserve real-time communication deadlines, and will not replicate the real-time behavior. Thus, we consider the scenario has validated our approach if the trajectory is sufficiently followed by observation.

4.1 Only JAUS

The original control software for the CAT Vehicle has been thoroughly tested in the field and in simulation. The results of running a right turn with this software in the vehicle can be seen in Fig. 6a. Fig. 6b shows a plot of the velocity of the vehicle versus the tire angle of the vehicle to show that the vehicle safely and quickly moves through the turn.

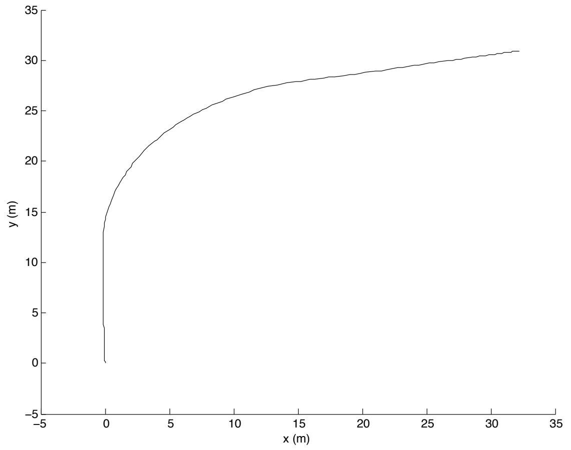
4.2 ROS Controlled CAT Vehicle

The second involves porting the control algorithms for the vehicle to a ROS node. A bridge is then set up to allow for communication between the physical vehicle and the ROS node running the control algorithm. This software is then deployed to the physical CAT Vehicle for operation. Fig. 6c shows the path taken by the vehicle as recorded by the GPS/INS system while Fig. 6d shows the tire angle versus velocity for the ROS driven vehicle.

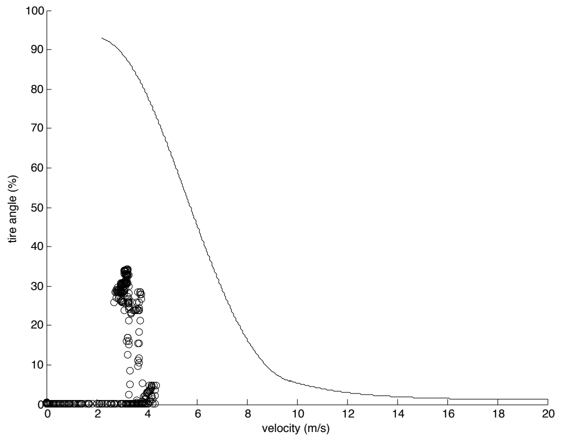
In this test some accuracy issues can be seen as the GPS/INS system locks onto the car. More noticeably from Fig. 6d, the ROS driven vehicle is reacting more wildly to the turn than the JAUS only vehicle (the reader will note from the figure that it still remains within the safe operating region). This is likely due to the additional delay caused by the bridge. It is likely that further improvements to the performance of the bridge would fix this issue. However, the vehicle is still capable of safely following the given trajectory. Also, it is possible that some of the issue may lay with the ROS implementation of the control algorithms and not with the bridge.

5. FUTURE WORK

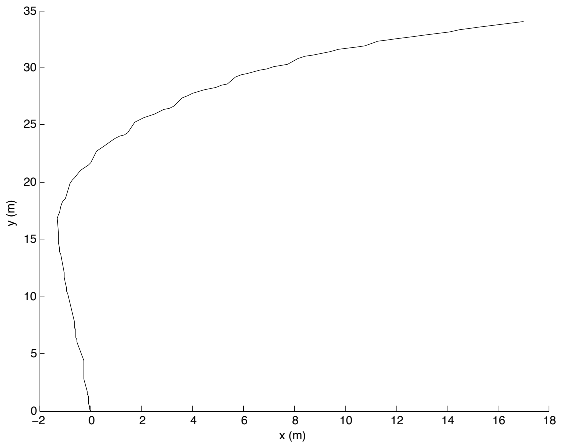
As illustrated by the examples above, there are some improvements that can be made in the performance of the generated JAUS to ROS bridge. Most importantly for the discussion of this paper, the examples show that the bridge needs to be capable of quickly passing messages along. The first step in this process will be determining how much of a bottleneck the bridge actually is for the hybrid ROS/JAUS system. The second step will be to take the necessary steps to cut down the latency between when a message is sent on one side of the system and when it is received on the other. It is likely that the issues seen in the results of these generated bridges are due to performance issues with the ROS or JAUS code rather than the interaction of the two.



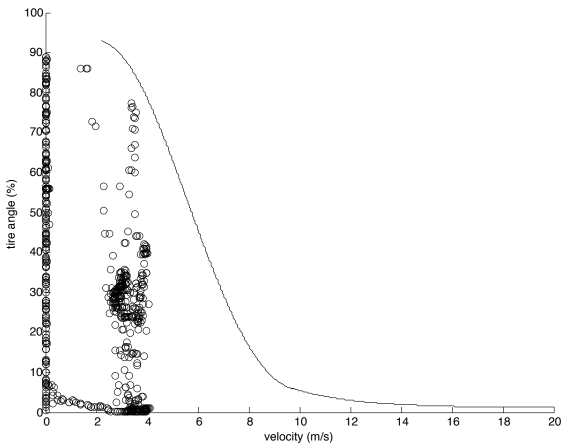
(a)



(b)



(c)



(d)

Figure 6: (a) The actual vehicle's attempts at following the prescribed right turn path with JAUS only. (b) Velocity plotted against tire angle for the JAUS only right turn. The solid line indicates the division between safe and unsafe maneuvers. (c) The physical vehicle's attempts at following the prescribed right turn path using the ROS2JAUS bridge. (d) Velocity plotted against tire angle for the ROS2JAUS right turn. The solid line indicates the division between safe and unsafe maneuvers.

5.1 Integration in JausML

At this time we have tested a single bridge with multiple components, but multiple bridges should also be feasible. In a visual modeling environment the user will be able to determine how many bridges the system should have and exactly what connections each bridge should manage. Alternatively, this process will be managed by a heuristic. The code generator described in this paper will be integrated into the JausML project in order to incorporate open source ROS products in the CAT Vehicle. The integration process should be fairly straightforward since both the bridge code generator and JausML use the skeleton design method.

5.2 Integration with Simulators

Prior to integration with ROS, the CAT Vehicle has been simulated in a limited JAUS simulator. With the existence of the ROS2JAUS bridge, we can now utilize the ROS based simulator Gazebo [2] to more accurately describe the movements of the physical vehicle and its environment.

More work is needed to fully integrate the Gazebo simulator with the CAT Vehicle project through the ROS to JAUS bridge, though we have (outside the scope of this paper) demonstrated the proof of concept of this interconnection. As of now, the limitations are the real-time nature of the communication.

There are two possible routes that can be explored in fixing the issues in communicating in real time with Gazebo. First, it should be possible to limit the rate at which the JAUS components send information to the simulator based on the ratio between real time and simulation time. However, this step would require rebuilding the logic behind the JAUS components and the rate at which they process data and send new messages. Second, the Gazebo simulator could be ported to more appropriate hardware or simplified to improve performance. However, this second scenario takes the performance of the simulator out of the developers' hands which is undesirable.

6. CONCLUSION

In this paper we demonstrated our ability to generate a bridge between two robotic middleware architectures. We showed how our textual domain-specific modeling language can be used to map the attributes of the messages passed by these two architectures. The generated bridge maps only the required messages, as specified in the domain-specific language, and therefore does not consume excessive system resources at compile or runtime. We validated our approach by porting some of our existing JAUS components to ROS, and then executing those components on ROS using the ROS2JAUS bridge, and comparing the results of the hardware-in-the-loop system behaviors.

7. ACKNOWLEDGMENTS

This opportunity is supported by the National Science Foundation and the Air Force Office of Scientific Research, under IIS-1262960 and CNS-1253334. We are thankful for the other participants of the CAT Vehicle REU, and to Dr. Haris Volos and Dr. Tamal Bose for their support.

8. REFERENCES

[1] K. J. Åström, R. E. Klein, and A. Lennartsson. Bicycle dynamics and control. *IEEE Control Systems Magazine*, 25(4):26–47, August 2005.

[2] D. Coleman. Gazebo ROS API, June 2013.

[3] A. Gokhale, D. Schmidt, T. Lu, and B. Natarajan. CoSMIC: An MDA generative tool for distributed real-time and embedded applications. In *International Conference on Distributed Systems Platforms and Open Distributed Processing/Open Distributed Processing - Middleware(ODP)*, pages 300–306, 2003.

[4] G. Hemingway, H. Neema, H. Nine, J. Sztipanovits, and G. Karsai. Rapid synthesis of high-level architecture-based heterogeneous simulation: a model-based integration approach. *Simulation*, pages 1–16, January 10 2012.

[5] M. Quigley, K. Conley, B. P. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng. ROS: an open-source robot operating system. In *ICRA Workshop on Open Source Software*, 2009.

[6] L. Sadlera, C. Rao, J. Rogers, and H. Nguyen. *ROStoJAUSBridge Manual*. Army Research Laboratory, March 2012.

[7] SAE AS-4:2010. *JAUS Standard*. SAE International, Warrendale, PA, June 2010.

[8] D. Thorndike, E. Perko, B. Ballard, K. Levine, C. Rockey, and M. Klein. Harlie. Project description, Case Western Reserve University, 2010.

[9] S. Whitsitt and J. Sprinkle. Message modeling for the Joint Architecture for Unmanned Systems (JAUS). In *Proceedings of the 8th IEEE Workshop on Model-Based Development for Computer-Based Systems*, pages 251–259, April 2011.

[10] S. Whitsitt and J. Sprinkle. A passenger comfort controller for an autonomous ground vehicle. In *51st IEEE Conference on Decision and Control*, pages 3380–3385, 2012.

[11] S. Whitsitt and J. Sprinkle. Model based development with the skeleton design method. In *20th IEEE International Conference and Workshops on the Engineering of Computer Based Systems*, page (in press), 2013.

[12] S. Whitsitt and J. Sprinkle. Modeling autonomous systems. *AIAA Journal of Aerospace Information Systems*, pages (in press, accepted in final form), 2013.