THE USE OF REAL-TIME PUBLISH-SUBSCRIBE MIDDLEWARE IN NETWORKED VEHICLE SYSTEMS

Eduardo R. B. Marques, Gil M. Gonçalves, João B. Sousa

Underwater Systems and Technology Laboratory (USTL)
Faculty of Engineering, University of Porto (FEUP)
{*edrdo,gil,jtasso*}*@fe.up.pt*

Abstract: Seaware is a publish-subscribe middleware used in multi-vehicle networked systems composed of autonomous and semi-autonomous vehicles and systems. Seaware provides a high level interface to network communications and may be deployed with a combination of heterogeneous components within a dynamic network. Seaware supports the RTPS (Real Time Publish Subscribe) protocol, underwater acoustic modems and other forms of network transport. This paper gives an overview of Seaware's implementation and its application to multi-vehicle networked systems.

Keywords: autonomous vehicles, communication networks, distributed control.

1. INTRODUCTION

The PISCIS project (Cruz *et al.*, 2003) has developed a system for the mixed initiative control and coordination of multiple underwater and surface vehicles for oceanographic and environmental data collection. The system consists of 1) autonomous underwater vehicles and autonomous surface vehicles equipped with acoustic modems (for underwater communications), radio/GPS systems (for interactions at the surface), 2) buoys equipped with transponders (for acoustic localization), 3) a sensor network and 4) computer consoles for mixed initiative interactions with human operators. The PISCIS system provides services for vehicle tele-operation, tele-programming and supervision, services for data collection from a sensor network, and services for system supervision with aggregation of information collected in real-time. Vehicles and operator consoles come and go and services and interactions are built on the fly and communication links can be of various types (radio, Ethernet, acoustic links).

Seaware is a publish-subscribe middleware that has been developed for the PISCIS project to address this networked environment. Seaware aims to provide an uniform, portable, efficient and high level interface to network communication for the heterogeneous components and communication links used in multi-vehicle networked systems of the PISCIS project. In this paper, we describe Seaware and its use in PISCIS.

The rest of this paper is structured as follows. Section 2 provides reference to related work, section 3 gives an overview of the Seaware implementation, section 4 describes its application to multi-vehicle networked systems and section 5 ends the paper with concluding remarks and discussion of future work.

2. RELATED WORK

We next provide brief references to related work covering: middleware toolkits and their use in

robotics; the use of RTPS technology ; and underwater acoustic modem communications.

A review of communication toolkits in application to robotics environments is provided in (Gowdy, 2000). Two examples of modern middleware systems used in robotics are the CORBA-based Miro (Utz *et al.*, 2002) and the publish-subscribe based Artificial EcoSystem (Sgorbissa and Zaccaria, 2004).

The Real-Time Publish-Subscribe (RTPS) protocol is a standard (IDA, 2001) for distributed real-time communication, complemented and tightly coupled in some implementations with the Data Distribution Service (DDS) standard (OMG, 2005) Seaware uses the NDDS (RTI, n.d.) implementation of RTPS/DDS from Real Time Innovations and also the open-source OCERA ORTE (Smolík *et al.*, 2003) implementation of RTPS. Discussion on the usability and performance of RTPS can be found in (Sierla, 2003; Schneider *et al.*, 1999; McCormick and Madden, 2005) and examples of applications of RTPS in robotics and automation can be found in (RTI, n.d.; Almadani, 2005).

The deployment of underwater acoustic networks is described in (Proakis *et al.*, 2001). An underwater acoustic communication framework, with similarity in several aspects to that used in the PISCIS project, is described in (Freitag *et al.*, 2005).

3. OVERVIEW OF SEAWARE

Seaware is a middleware for network communication in dynamic and heterogeneous network environments, oriented to data-centric network computation. In spite of being designed for use in the PISCIS project, Seaware has a generic approach in its design and is therefore suitable for application in other general-purpose networked systems in automation and control. The core feature of Seaware is the use of the publish-subscribe messaging paradigm enabled by an abstract object-oriented API, with support given by the RTPS protocol (IDA, 2001) and other forms of network transport. We next provide an overview of Seaware, in terms of software framework, use of publish-subscribe messaging, network transports and core implementation.

3.1 Software framework

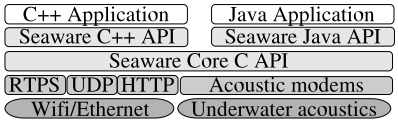

Fig. 1. Seaware software framework

The Seaware software framework architecture is shown in Figure 1. An object oriented API, with Java and C++ bindings, is available for application development. This high level API encapsulates access to a core API written using the C programming language. Further down this tier-architecture, a network interface layer encapsulates access to network transports; in the current implementation these are RTPS, acoustic modem based communications, HTTP and (raw) UDP.

The framework tries to address the following types of requirements: 1) to define a modular and abstract communications layer that encapsulates the heterogeneity of component systems in various aspects (operating system, network interfaces, hardware architecture and programming language bindings) in a simple object-oriented publish-subscribe API ; 2) to provide for a dynamic network environment, where components can enter and leave and communication patterns change over time, without disrupting the overall communications environment; 3) to attain robust and lightweight (soft) real-time performance.

The software framework works both on Linux and Windows operating systems. Besides the standard x86 port for Linux, two specific Linux ports exist: one for PC-104 based platforms and another one for Intel Xscale host architectures.

3.2 Publish-subscribe messaging

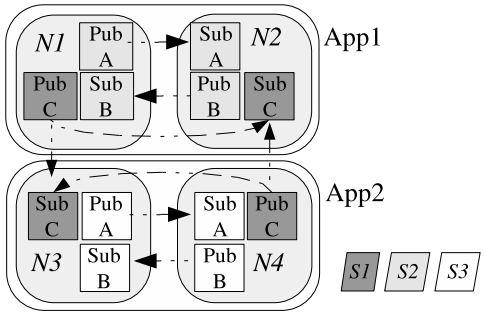

Fig. 2. Publish-subscribe model entities

Publish-subscribe messaging is defined by anonymous message exchange between components that produce data (*publishers*) and those that consume it (*subscribers*). Each message exchanged has an associated *topic* id that pairs together publishers and subscribers. Data issued by a publisher or various publishers for a certain topic is delivered to all subscribers of that topic, allowing variation of components and types of data in the network over time. Publishers and subscribers are decoupled and do not need to know who their peers are; instead the middleware is responsible for dynamically enabling peer discovery and adapting to network changes at execution time.

This form of communication is simple and inherently suitable for dynamic communication environments, with on-the-fly component integration, many-to-many communication and changeable communication patterns. Seaware embraces publish-subscribe messaging in two ways: firstly it provides a publish-subscribe data-centric model; secondly it uses the RTPS protocol as the main form of network transport.

Figure 2 illustrates the publish-subscribe messaging model implemented by Seaware. *Applications* (`App1` and `App2` in the figure) have resident *nodes* (`N1` to `N4`) which exchange *messages* for published or subscribed *topics* (`A` to `C`). *Topic namespaces* (`S1` to `S3`) are used to define distinct subsets of topics. Message exchange may occur within the same application (`N1` to `N2`, `N3` to `N4`) or over the network (`N1` to `N4`, `N3` to `N2`).

3.3 Network transports

RTPS - RTPS support has been initially deployed using the research-based open-source ORTE implementation (Smolík *et al.*, 2003), then also incorporated NDDS (RTI, n.d.), a commercial implementation of DDS/RTPS used in industrial automation/real-time applications. Although ORTE and, in a more elaborate form, NDDS, are middleware technologies on their own, the Seaware API encapsulates and hides access to them, in order to provide a simpler interface.

Besides RTPS, other heterogeneous forms of network transport are supported by Seaware in order to deal with specific communication media or operational limitations. All forms of transport are accessed through the same uniform API that provides a generic publish-subscribe messaging interface.

Acoustic modems - Underwater communications are made possible through the use of an acoustic modems transport layer, with a simple functional design. Message integrity control is done using CRC checksum and basic message re-transmission control is done through a basic message acknowledgment mechanism provided by the acoustic modems we use. No support for more advanced features such as as time-division multiple-access (TDMA) is however provided (application-level logic must ensure collision avoidance on the half-duplex acoustic channel, for example using master-slave communication layouts).

Raw UDP transport - Some of our vehicles use a legacy software infrastructure which is not practical to change and where communication is done through raw UDP sockets. For communication with these vehicles, a Seaware enabled application must use the raw UDP transport back-end.

HTTP transport - HTTP based-transport is an experimental feature which is being tested for web-data publishing. The aim is to interconnect Seaware applications and Java applets which may be remotely displaced on any site in the Internet.

3.4 Implementation

The core aspects of the Seaware implementation are summarized in the following topics.

Application programmer interface - The Seaware application programmer interface (API), with Java or C++ bindings, has its core support provided by the API class skeletons shown in Figure 3 (only a simplified overview is shown due to space constraints): 1) a *message* that applications must extend to define message contents and network serialization format; 2) *publication and subscription* classes for topic configuration; 3) a *node* class providing the core messaging operations; 4) a *message listener* class for asynchronous message notification. This API interface is in direct correspondence with the publish-subscribe entities described before, provides uniform access both in Java and C++ and for all types of transport, thus message transport is encapsulated using only a publish-subscribe oriented access.

1) Message

```
public interface Message ... {
 public int serialize(...);
 public void unserialize(...);
}
```

2) Topic configuration

```
enum QOS { DEADLINE, RELIABILITY, SEPARATION, ...};
class TopicConfiguration{
  void setQOS(QOS qos, int value);
  void setNamespace(String ns);
}
class Publication<T extends Message>
  extends TopicConfiguration { ... }
class Subscription<T extends Message>
  extends TopicConfiguration { ... }
```

3) Node class

```
class Node {
 void defineTopic(TopicConfiguration p);
 void publish(Message m);
 Message pull(String topic,MessageInfo info);
 void setMessageListener(String topic,
      MessageListener l);
}
```

4) Message listener

```
class MessageListener{
 public void onMessage
 (Message msg,MessageInfo info);
}
```

Fig. 3. Seaware API overview

QoS (Quality of Service) settings - RTPS/DDS provides support for a number of built-in QoS settings of interest (IDA, 2001; OMG, 2005), some of which have been deployed in the implementation of Seaware: configurable *message reliability*,

on a per-topic basis, which enables/disables message reliability depending on the requirements of the message exchange *time-based separation* and *deadline* for topic subscriptions, which specify the minimum and maximum time interval for the receipt of new message issues, allowing optimization of network flow through different separation settings for different subscribers and timeout control using the deadline setting; *UDP multicast* for automatic application discovery, allowing a dynamic environment on which the network hosts need not be known in advance; use of *RTPS domains* for separation of distinct application sets; *application refresh time and expiration time* which specify how long an application acknowledges its presence on the network (even if not publishing data) and how long it may be out of reach or not sending any messages and later re-establish contact with their peers.

Since there may be other types of transport apart from RTPS, support is provided so that some of the settings may be interpreted generically for all transports, specifically those related to message timing (time-based separation and deadline) and also to some degree message reliability (excluding the legacy "raw UDP" transport).

Configuration support - The system uses a XML configuration format for the various components in the publish-subscribe model, to minimize programming work using the API. A XML configuration may be loaded to allow direct configuration in Java, or automatic code generation in C++.

Performance evaluation - Benchmarks and field test results indicate good performance in terms of message delivery rate, connectivity response time and memory usage: message delivery rate yields a maximum throughout of 3000 messages per second over Ethernet/Wifi; peer connection time is normally less than 5 milliseconds; applications can work with less than 1 Mb of heap memory (most of which is taken by NDDS).

Fig. 4. Roaz ASV (left) and Isurus AUV (right)

4. APPLICATION TO NETWORKED VEHICLE SYSTEMS

We next discuss the application of Seaware to networked vehicle systems considering general scenarios and giving concrete examples of each of them. The discussion is made from the perspective of high level network control and considers some of the scenarios described in (Sousa and Sengupta, 2001; Girard *et al.*, 2004). The concrete examples presented refer to field tests conducted in Portugal, at the Montemor-O-Velho canoing race track, at Portuguese Navy facilities in Lisbon and Tróia and at the Leixões harbor.

4.1 Centralized ground-to-vehicle control

The simplest scenario we consider is communication between a control console and a vehicle for remote operation and monitoring of the vehicle. This is the basic operational scenario for which Seaware provides support, on which other more elaborate scenarios build up. An example of this setting is that involving a *Neptus* (Dias *et al.*, 2005) control console and the *Roaz* ASV (autonomous surface vehicle) (Ferreira *et al.*, 2006), depicted in Figure 4.

The Neptus framework is an environment to support network centric operation of heterogeneous teams of autonomous and semi-autonomous vehicles and systems. Neptus is programmed in Java and Seaware has been integrated into it, replacing previous network support for vehicle communications. In the context of the Neptus framework, a wide variety of possible interactions take place between the pilot (human or automated) and the vehicles.

Roaz (Ferreira *et al.*, 2006) is a multi-purpose ASV with a docking station for AUVs (autonomous underwater vehicles), equipped with plug-and-play sensors and Wifi network support. In the test setup, Roaz was configured in teleoperation mode, allowing remote control by a Neptus console, with vehicle tracking done with on-board GPS device and video camera and side-scan sonar devices operating independently Neptus. The core Roaz control tasks for vehicle guidance, GPS tracking, and on-board logging, are programmed in C++ and use Seaware for Wifi communication with Neptus consoles.

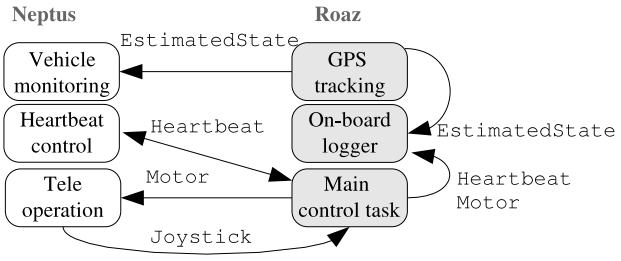

Fig. 5. Neptus-Roaz communication

Figure 5 provides detail on the publish-subscribe setup concerning Neptus-Roaz ASV communi-

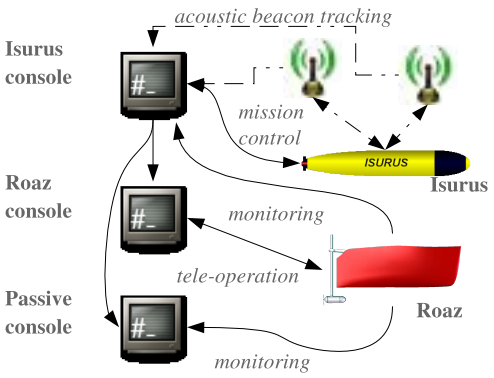

Fig. 6. Networked vehicle system

cation. Four topics are used for communication: `EstimatedState` for vehicle GPS positioning data; `Joystick` for the tele-operation command through Neptus; `Motor` to on-board motor real-time data such as rotations per minute and power consumption; and `Heartbeat` used for mutual acknowledgment issued with a periodic 1 Hz rate.

4.2 Dynamic many-to-many vehicle control

The control environment in networked vehicle systems may be highly distributed between multiple control consoles and vehicles, with many-to-many communications. There are several possible layouts for these operational scenarios. In a single vehicle scenario one of the consoles may act as the active vehicle controller whereas others merely execute passive tasks, such as vehicle state monitoring or data logging. In a multi-vehicle scenario various consoles may be used, each acting as master control for a vehicle while at the same time being able to monitor the other vehicles in the environment.

Figure 6 illustrates an example of this type of scenarios, involving multiple Neptus consoles, the Roaz ASV and the *Isurus* AUV. The Neptus-Roaz setup is as described before. *Isurus*, depicted in Figure 4, is a AUV with programmable mission execution within the Neptus framework (Dias *et al.*, 2006), acoustic beacon based localization (Cruz *et al.*, 2001), Wifi surface connectivity, underwater acoustic modem communications, and multiple sensory devices. During mission execution the location of Isurus may be tracked by a Neptus console with an attached serial device interface to the acoustic beacon system. This data is then re-transmitted over RTPS using Seaware so that the vehicle's position may also be tracked by other peers. Mission control commands and state monitoring data can be sent or received through acoustic modems (with master-slave application level control for acoustic channel collision handling) when the vehicle is underwater or over Wifi "raw UDP" when the vehicle is at the surface.

In the test setting, both vehicles could be controlled and monitored simultaneously, nodes could enter and leave the system without disrupting the global communication environment. Interaction between Roaz and Neptus consoles allowed the surface vehicle to be guided within the bounds allowed by the on-site wireless communications infrastructure (up to 1000 meters). In regard to Isurus, the position tracking re-publication scheme and Wifi UDP and acoustic modem interfaces were successfully validated, allowing more than one ground console to monitor the vehicle's position and mission control to be done at surface level or underwater.

4.3 Multi-vehicle coordinated missions

In multi-vehicle coordinated missions, vehicles may communicate with each other in order to accomplish a task and also cooperate with control consoles for shared semi-autonomous control. An example of application of Seaware to this type of scenarios is the communications infrastructure for the ASASF multiple UAV (unmanned air vehicle) control project (Almeida *et al.*, 2006), where multiple UAVs operate in coordinated missions, implying vehicle-to-vehicle and ground control-to-vehicle communication through Wifi and spread-spectrum radio-links.

5. CONCLUDING REMARKS

Seaware greatly enhances the scope and flexibility of the networked vehicle systems communication infrastructure in a number of important aspects, considering the particular field test settings described and the generic operational scenarios discussed, by comparison with the previous networking support we used in our projects.

The previous communications support was heterogeneous and low-level in terms of systems programming which made it difficult to have an uniform and model-based implementation of communication and had a strong impact in the software development cycle. There was also no built-in support for distributed many-to-many communication, dynamic on-the fly integration of components or varying network communication patterns.

In a more high level view, considering the operational scenarios networked vehicle systems, centralized ground-to-vehicle control was the bare support we had and this has been extended to allow abstract layouts of many-to-many dynamic communication.

Future work in Seaware will be driven by a new generation of vehicles being developed in our research group (AUVs, ASVs, ROVs and UAVs). Some specific items for work in the near future are as follows: support of spread-spectrum radio based networks, to be deployed in ASVs and UAVs for long-range communications; more advanced acoustic modem communications support to cope with more elaborate communication requirements, exploiting TDMA and store-and-forward message routing schemes; development of tools for run-time network state monitoring and configuration.

ACKNOWLEDGEMENTS

This work has been partially funded by Agência de Inovação through the PISCIS project. We also wish to thank cooperation from the Autonomous Systems Laboratory at Instituto Superior de Engenharia do Porto, the Portuguese Navy, the Leixões harbor administration and all USTL team members.

REFERENCES

Almadani, B. (2005). RTPS middleware for real-time distributed industrial vision systems. *11th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA'05)*.

Almeida, P., G. Gonçalves and J. Sousa (2006). Multi-UAV platform for integration in mixed-initiative coordinated missions. *1st IFAC Workshop on Multi-Vehicle Systems (MVS'06)*. Accepted for publication.

Cruz, N., J. B. Sousa, F. L. Pereira, J. E. Silva, J. Coimbra and E. B. Dias (2003). Operations with multiple autonomous underwater vehicles: the PISCIS project. *Second Annual Symposium on Autonomous Intelligent Networks and Systems AINS 2003*.

Cruz, N., L. Madureira, A. Matos and F. L. Pereira (2001). A versatile acoustic beacon for navigation and remote tracking of multiple underwater vehicles. *MTS/IEEE International Conference Oceans 2001*.

Dias, P. S., R. Gomes, J. Pinto, G. M. Gonçalves, J. B. Sousa and F. L. Pereira (2006). Mission planning and specification in the Neptus framework. *Humanitarian Robotics, ICRA 2006 IEEE International Conference on Robotics and Automation*.

Dias, P. S., R. Gomes, J. Pinto, S. L. Fraga, G. M. Gonçalves, J. B. Sousa and F. Lobo Pereira (2005). Neptus a framework to support multiple vehicle operation. *MTS/IEEE International Conference Oceans 2005*.

Ferreira, H., A. Martins, A. Dias, C. Almeida, J. M. Almeida and E. P. Silva (2006). Roaz autonomous surface vehicle design and implementation. *ROBOTICA 2006, Portuguese National Robotics Meeting*.

Freitag, L., M. Grund, C. von Alt, R. Stokey and T. Austin (2005). A shallow water acoustic network for mine countermeasures operations with autonomous underwater vehicles. *Underwater Defense Technology (UDT) 2005*.

Girard, A. R., J. Sousa and J. K. Hedrick (2004). A selection of recent advances in networked vehicle systems. *Proc. of the Institution of Mechanical Engineers (IMECHE), Part I*.

Gowdy, J. (2000). A qualitative comparison of interprocess communications toolkits for robotics. Technical Report CMU-RI-TR-00-16. Robotics Institute, Carnegie Mellon University.

IDA (2001). Real Time Publish Subscribe (RTPS), Wire Protocol Specification 1.0. Interface for Distributed Automation group.

McCormick, B. and L. Madden (2005). Open architecture publish subscribe benchmarking. OMG Real-Time and Embedded Systems Workshop 2005.

OMG, Object Management Group (2005). Data Distribution Service for Real-time Systems Specification, v1.1.

Proakis, J., E. Sozer, J.Rice and M.Stojanovic (2001). Shallow water acoustic networks. *IEEE Communications Magazine* **39**(11), 114–119.

RTI, Real Time Innovations Inc (n.d.). NDDS. http://rti.com/products_ndds.html.

Schneider, S., G.P. Castellote and M. Hamilton (1999). Can Ethernet be real-time?. *Real Time Innovations Inc*.

Sgorbissa, A. and R. Zaccaria (2004). The Artificial Ecosystem: a distributed approach to service robotics. *IEEE International Conference on Robotics and Automation*.

Sierla, S. (2003). *Middleware solutions for automation applications: case RTPS, Diploma Thesis*. Helsinki University of Technology.

Smolík, P., Z. Sebek and Z. Hanzálek (2003). ORTE-open source implementation of Real-Time Publish-Subcribe protocol. pp. 68–72.

Sousa, J. B. and R. Sengupta (2001). Networked multi-vehicle systems. *Tutorial session for the IFAC Decision and Control Conference*.

Utz, H., S. Sablatnog, S. Enderle and G. Kraetzschmar (2002). Miro - Middleware for Mobile Robot Applications. *IEEE Transactions on Robotics and Automation, Special Issue on Object-Oriented Distributed Control Architectures* **18**(4), 493–497.