SEAWARE: A PUBLISH-SUBSCRIBE MIDDLEWARE FOR 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: This paper describes the use of the Seaware publish-subscribe communications middleware in networked vehicle systems composed of AUVs (autonomous underwater vehicles), ROVs (remotely operated vehicles), UAVs (unmanned air vehicles) and ASVs (autonomous surface vehicles). Seaware provides a high level interface to network communication and may be deployed with a combination of heterogeneous components within a dynamic network. Seaware uses the RTPS (Real Time Publish Subscribe) protocol and other forms of network transport.

Keywords: autonomous vehicles, communication networks, distributed control.

1. INTRODUCTION

In the PISCIS project (Cruz *et al.*, 2003) a system for the mixed initiative control and coordination of multiple underwater and surface vehicles for oceanographic and environmental data collection has been developed. 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) and 3) a sensor network. PISCIS is a distributed system supported by multiple communication links (radio, Ethernet, acoustic links) with services for vehicle tele-operation, teleprogramming and supervision, services for data collection from a sensor network, and services for system supervision with aggregation of information collected in real-time. In this system, vehicles and operating modules come and go and services and interactions are built on the fly.

To address the diverse communication requirements and enhance its support in PISCIS , a communications middleware called Seaware has been developed. Seaware was deployed in vehicles and ground stations of the PISCIS system, in order to provide an uniform, portable, efficient and high level interface to network communication. Seaware embraces a publish-subscribe messaging paradigm and makes use of modern developments in that field, specifically the Real Time Publish Subscribe (RTPS) network protocol (IDA, 2001).

The rest of this paper is structured as follows. Section 2 provides an overview of the Seaware implementation. Section 3 describes its application to multi-vehicle networked systems with an example application, discussion of general operating scenarios and evaluation summary. Section 4 ends the paper with concluding remarks and highlights future work.

2. OVERVIEW OF SEAWARE

Seaware is a middleware for network communication in dynamic and heterogeneous network environments, oriented to data-centric network

computation. It has been specifically designed for integration in networked vehicle systems of the PISCIS project, which is the focus of this paper, however it has a generic approach in its design and is therefore suitable for application in other general-purpose networked systems in automation and control. The main aspects of Seaware are: the use of publish-subscribe messaging paradigm, the support of heterogeneous platforms and the provision of a high-level interface to network communications.

2.1 Requirements

The requirements imposed by a networked vehicle system which have led to the development of Seaware are as follows:

Heterogeneous support - The system components are heterogeneous and it is the role of the middleware to abstract and encapsulate the diverse nature of these components and underlying support in an usable, modular and uniform interface for communications. Several heterogeneity aspects need to be addressed, such as the diverse nature of underlying hardware, software and network support.

Dynamic network environment - The network interactions are complex and may change over time in terms of communication flow or components in the network and the middleware must adapt to communication patterns and network events on the on-the fly. Specifically, the system has to provide support for: dynamic component peer discovery, plug-in and plug-out; many-to-many changeable communication patterns; and on-the-fly component re-configurability.

Lightweight robust real-time operation - The middleware must provide lightweight robust operation, considering limited resource or operation restraints in components and underlying support. The main restrictions may be posed on the network interface (intermittent availability, limited network bandwidth) or processing power (for example, processor speed, memory availability). In regard to real-time operation, the requirements can be considered as soft real-time, with no strict deadlines, but real-time response with network latency of a few milliseconds at most is required.

Software engineering - The middleware must provide a simple and abstract model of communication of feasible use by any programmer and portable across different platforms. Specifically, an high level object-oriented design embodying familiar concepts in general programming, bindings to standard programming languages and support by diverse operating systems are highly desirable.

2.2 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.

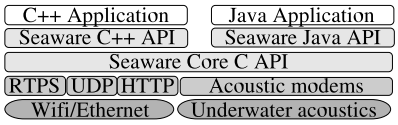

Fig. 1. Seaware software framework

This software framework works both on Linux and Windows operating systems. The use of an high level object-oriented API, with bindings to standard programming languages and operating systems, enables portable software development in heterogeneous systems.

2.3 Publish-subscribe messaging

Publish-subscribe is a communications paradigm that has been adopted in recent years in network communication middleware systems. It 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 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. It can also easily emulate more traditional forms of communication such as peer-to-peer, client-server or broadcast.

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.

Publish-subscribe data-centric model - 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`).

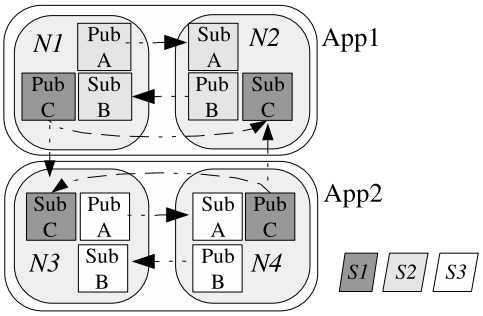

Fig. 2. Publish-subscribe model entities

Use of RTPS - Publish-subscribe messaging has been adopted in distributed real-time communication standards as the Real-Time Publish-Subscribe (RTPS) network protocol. RTPS is a standard protocol for communicating over lightweight unreliable network transport (typically UDP/IP), complemented in some implementations with the Data Distribution Service (DDS) architecture (OMG, 2005) for high level operation. We have initially integrated RTPS support 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.

2.4 Other network transports

RTPS allows applications to operate on a Wifi or Ethernet environment, but other heterogeneous transports are supported by Seaware in order to deal with specific communication media or operational limitations.

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

Raw UDP transport - Some of our vehicles use a legacy software infrastructure which is unpractical 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 backend.

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 remotely displaced on any site in the Internet.

2.5 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 requirements): a *message* (i) that applications must extend to define message contents and network serialization format; *publication and subscription* for topic configuration (ii); *node* (iii) providing the core messaging operations; a *message listener* (iv) 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, and encapsulates message transport using only a publish-subscribe oriented access.

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 - application logic may tolerate some message loss in some exchanges, or it may require reliability for strict in-order message delivery; *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, use of network bandwidth: message delivery rate yields a maximum throughout of 3000 messages per second over Ethernet/Wifi; peer connection time is normally less than 5 milliseconds; network payload is sustainable (50 bytes overhead due to RTPS message header per message issue).

i. Message

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

ii. 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 { ... }
```

iii. Node class

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

iv. Message listener

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

Fig. 3. Seaware API overview

3. APPLICATION TO NETWORKED VEHICLE SYSTEMS

3.1 Example application

An evaluation of Seaware in the context of networked vehicle systems operational scenarios was made during tests conducted in Portugal, at the Montemor-o-Velho canoeing race track and Portuguese Navy facilities in Lisbon and Troia. Two vehicles - the *Isurus* AUV and the *Roaz* ASV, depicted in Figure 4 - and the *Neptus* framework application suite were used for our tests:

The *Neptus* framework (Dias *et al.*, 2005) 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. In the field tests, the Neptus framework was used to plan a mission for autonomous execution by the Isurus AUV and allowed remote human tele-operation of the Roaz ASV.

Fig. 4. Isurus AUV and Roaz ASV

Isurus 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 localization 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.

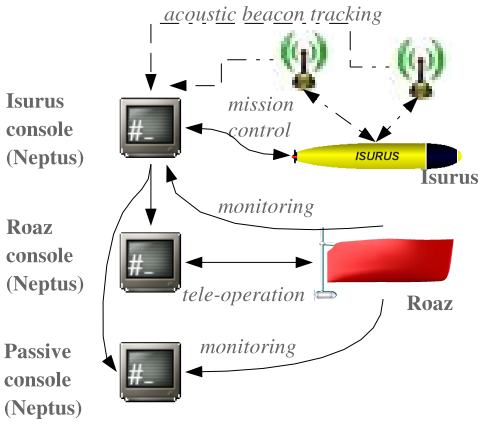

Fig. 5. Networked vehicle system

Roaz (Ferreira *et al.*, 2006) is a multi-purpose ASV with a docking station for AUVs. It is targeted to flexible and coordinated operations with long durations. Roaz is equipped with plug-and-play sensors and Wifi network support. In the test setup, Roaz was configured in tele-operation mode, allowing remote control by a Neptus console, with vehicle tracking done with on-board GPS and video camera and side-scan sonar devices operating independently from Neptus communication. 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.

The publish-subscribe setup deployed in our field mission tests is depicted in Figure 5, that shows how components are connected from an high level viewpoint, corresponding to our previous component architecture. Figure 6 provides more fine grained detail on the most elaborate publish-subscribe setup, that concerning Neptus-Roaz ASV communication, in terms of message exchange. Four topics are used for communication: `EstimatedState` for vehicle positioning data,collected using the vehicles on-board GPS receiver; `Joystick` for the tele-operation command through Neptus; `Motor` for on-board motor monitoring data, as for example rotations per minute and power consumption data; and `Heartbeat` used for mutual acknowledgment issued with a periodic 1 Hz rate.

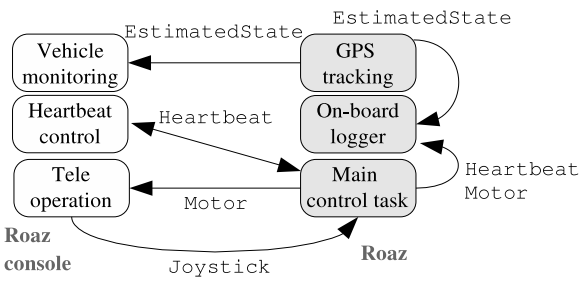

Fig. 6. Detailed setup for Roaz ASV

The global communication setup was experimented with good results. Overall, a robust operation has been observed, running applications could enter and leave the system on the fly without disrupting the communications environment and its configuration could be adjusted with little effort. 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.

3.2 General application

The usability of the Seaware system, from the perspective of network communication and control can be discussed considering generic classes of typical operational scenarios in networked vehicle systems. We address some of the typical scenarios (Sousa and Sengupta, 2001; Girard *et al.*, 2004) from an high level network control infrastructure perspective.

Centralized ground-to-vehicle control - A ground control console and a vehicle communicate, allowing remote operation and monitoring of the vehicle. The scheme may be generalized to more than one vehicle, but keeping a centralized control approach, where a single console controls more than one vehicle. This is the basic operational scenario for which Seaware provides support, on which other more elaborate scenarios build up.

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 ground control tasks for shared semi-autonomous control. Besides the described integration of Seaware in the example scenario described, on-going developments are integrating Seaware in a new generation of vehicles for multi-vehicle operation scenarios (AUVs, UAVs, ROVs and ASVs).

Dynamic many-to-many vehicle control - The control environment is highly distributed between ground control and vehicles in the environment, with many-to-many communication. 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 (the field test setting we described is a particular example). Richer operational scenarios may be considered where main control of one or more vehicles is effectively shared between several nodes and may change over time with complex interaction, as it happens with layered control architectures for multi-vehicle coordinated missions.

3.3 Evaluation

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. Centralized vehicle control was the bare support we had, with only one static ground control and one or more vehicles connected to it or connected to non-communicating ground controls; a vehicle was also only prepared to communicate with a single ground control. 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. Finally, 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. All these limitations are addressed in the implementation of Seaware.

4. CONCLUDING REMARKS

We have provided an overview of the implementation of the Seaware publish-subscribe middleware and its application to networked vehicle systems with heterogeneous components.

Future work in Seaware will mainly be driven by other projects which have already integrated Seaware, for example the ASASF multiple UAV control project (Almeida *et al.*, 2006) and a new generation of other vehicles being developed in our research group (AUVs, ASVs and ROVs). 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.

REFERENCES

Almeida, P., G. Gonçalves and J. Sousa (2006). Multi-UAV platform for integration in mixed-initiative coordinated missions. In: *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. In: *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. In: *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. In: *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. In: *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. In: *ROBOTICA 2006, Portuguese National Robotics Meeting.*

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

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

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

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

Smolík, P., Z. Sebek and Z. Hanzálek (2003). ORTE-open source implementation of Real-Time Publish-Subcribe protocol. In: *2nd International Workshop on Real-Time lans in the Internet Age.* pp. 68–72.

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