Towards Framework-Based UxV Software Systems: An Applied Research Perspective

Gregory S. Broten, Simon P. Monckton, Jared L. Giesbrecht and Jack A. Collier

Defence R&D Canada – Suffield, Canada `Firstname.Lastname@drdc-rddc.gc.ca`

1 Introduction

Defence R&D Canada changed research direction in 2002 from pure tele-operated land vehicles to general autonomy for land, air, and sea craft (UxV). The unique constraints of the military environment coupled with the complexity of autonomous systems drove DRDC to carefully plan a research and development infrastructure that would provide state of the art tools without restricting research scope.

DRDC's long term objectives for its autonomy program address disparate unmanned ground vehicle (UGV), unattended ground sensor (UGS), air (UAV), and subsea and surface (UUV and USV) vehicles operating together with minimal human oversight. Individually, these systems will range in complexity from simple reconnaissance mini-UAVs streaming video to sophisticated autonomous combat UGVs exploiting embedded and remote sensing. Together, these systems can provide low risk, long endurance, battlefield services assuming they can communicate and cooperate with manned and unmanned systems.

Clearly computational, structural, and control requirements will vary greatly in both complexity and scale between platforms. Yet future forces must share a common infrastructure, command and control to participate in joint operations. This chapter describes DRDC's experiences during the migration from a legacy, tele-operated system, to a modern frameworks based robotics program. To assist this migration DRDC chose a whitebox framework approach that extended the capabilities of the existing, academic based, Miro framework. By adapting an existing framework DRDC leapfrogged directly to the application development phase of porting legacy systems into modern robotic application and waived the need to develop an applicable framework. This whitebox approach saved DRDC a considerable amount of time and effort over the alternative "develop from scratch" option.

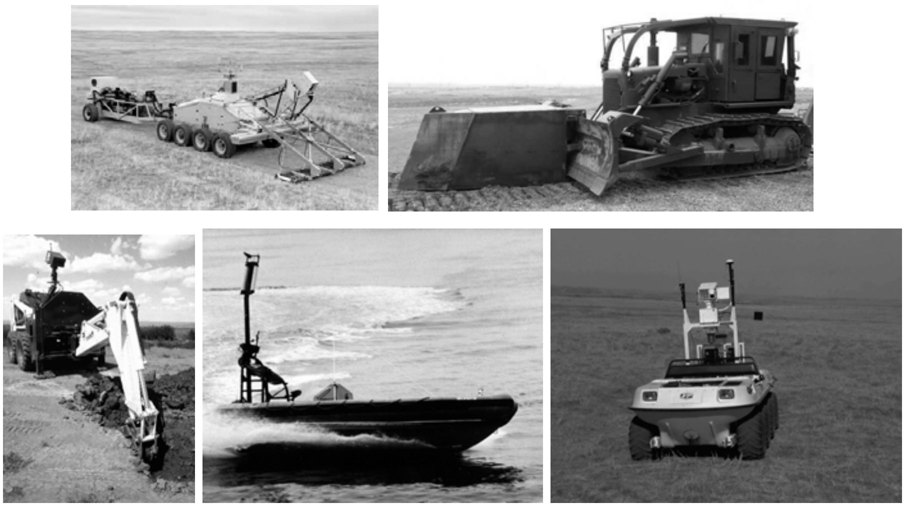

Fig. 1. Tele-Operated Vehicles

2 Background

Over the past 20 years, Defence R&D Canada developed numerous tele-operated unmanned ground vehicles, many founded on the ANCÆUS [GBV03] command and control system. First fielded in 1990, ANCÆUS solved difficult wireless networking and vehicle control problems five years or more before similar military systems appeared internationally. Further, ANCÆUS demonstrated the viability of network architectures for fieldable robotic systems and multivehicle control. With remarkable foresight, the developers decoupled the vehicle platform from both the commands and communications interface. The ANCÆUS design proved adaptable to other platforms such as the Barracuda sea target, the combat CAT, the ILDP de-mining vehicle and the tele-operated Bobcat with a backhoe, depicted in Figure 1.

Hardware and software development eventually overtook ANCÆUS, however. Substituting today's technology, ANCÆUS becomes a modest command interface similar to the Joint Architecture for Unmanned Systems (JAUS)[JAU04]. Nevertheless, DRDC learned some valuable lessons in this early program, specifically that systems become more expensive, less reliable, and time consuming with:

- the need to port, extend and maintain customized software across platforms and payloads.
- platforms and payloads that must evolve continuously to avoid obsolescence.
- platforms and payloads that are task dependent.

Despite the power and flexibility provided by human operators, tele-operation is fundamentally manpower intensive and communications links can be fragile. Thus, DRDC launched the Autonomous Land Systems (ALS) Project shifting its research focus to *autonomous* UGVs. The project objectives were to build personnel and technical capability in multivehicle robot systems targeted at reconnaissance in medium complexity terrain.

2.1 The ALS Project

To pursue the long term objectives within and beyond the ALS project, DRDC needed an R&D strategy that would permit multiple investigators to collaborate on unmanned systems. Unlike commercial system integrators on an industrial project, DRDC relies on voluntary contributions from a loosely organized team across the agency to a develop and test a research system. Only a set of long term objectives, some specific performance expectations, a sketched hybrid arbitrated control system, a target vehicle, and sensor suite constrained system development. Over a period of 15 months, DRDC-Suffield assembled two robotic systems: one founded on the SegwayRMP and the other on converted Koyker Raptor. The SegwayRMP project familiarized the team with Player/Stage[BGH03], basic sensing, and the perils of systems integration. The Raptor project would represent new development using distributed processing software, sophisticated sensing and mapping.

2.2 Requirements Analysis

Within this general context, the ALS project captured specific requirements, treating the UGVs as 'black boxes' and establishing the outside requirements of vehicle performance. The requirements document outlined, in detail, the team's expectations of a land multivehicle system in two *Autonomous Search* (AS) demonstrations: *Terrain Discovery (AS-TD)* and *Map Guided (AS-MG)*. In AS-TD, the demonstration would show a multivehicle, cooperative, silent reconnaissance task with a minimum of stop-think behaviour. Quoting from the document:

1. a *complement* of vehicles,
2. through an MMI accept a *search mission specification* .
3. *navigate* in *formation* to a search area at least 300 to 500 metres distant, *identifying* and *reporting* targets if any
4. *cooperatively search* the area, *identifying* and *reporting* targets if any
5. return to a base following a different route, *identifying* and *reporting* targets if any

AS-MG would be identical with the change that a mission plan would be developed autonomously based on a-priori elevation maps. These loose objective statements were expanded into more detailed, often numerical, requirements

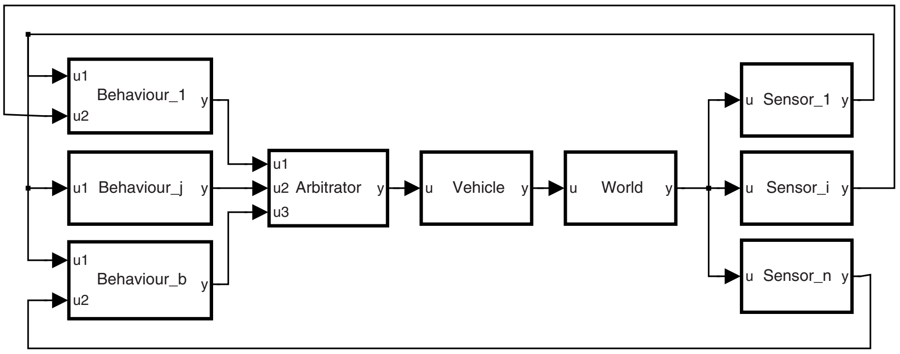

Fig. 2. An abstract control block diagram of arbitrated hybrid control.

such as ground speed and obstacle dimensions. Ultimately, though, the requirements served both as a measure of progress towards an ideal and as a means to constrain software development.

2.3 Conceptual Design

With the 'black box' characteristics of the system prescribed, the team could explore plausible vehicle control architectures. Early in the project the team settled on the increasingly popular hybrid arbitrated control philosophy (e.g. [Ros95]) exemplified in Figure 2. In this structure, an 'arbitrator' combines or selects a course of action based on the input from multiple controllers. Generating a common output format, each behaviour controller regulates a subset of observed variables based on unique inputs and algorithms. For example: though obstacle avoidance and waypoint following both issue steering and speed commands, obstacle avoidance regulates the clearance distance between the vehicle and an obstacle, while waypoint following regulates steering to follow a desired track. The arbitrator combines or switches between these inputs to ensure safe travel to the objective. This platform-neutral control structure structure offers scalability and extensibility through computing parallelism, but requires sophisticated computing middleware. The following sections detail the system's sensing and vehicle components.

Platform

DRDC selected a modified Koyker Raptor as the unmanned ground vehicle (UGV) demonstrator. In each, a 25Hp gasoline engine powered a 4x4 hydrostatic drivetrain and generated 1.5 kW surplus electrical power. The vehicles' payload included power inverters, quad and single Pentium servers, ethernet and USB hubs. A SpeedLan router provided 802.11b class wireless mesh networking.

Proprioceptive Sensing

The sensing system collected raw position and orientation data from a GPS, an IMU, and odometry. The Sokkia GSR2600 GPS, combined with a Pacific Crest PDL RVR radio, supplied differentially corrected GPS positions at 4 Hz. Two Honeywell 1GT101DC hall effect sensors provided on-board software with quadrature pulse trains describing the displacement, velocity and direction of each front wheel. A Microstrain 3DM-GX1 produced orientation and angular rates with respect to gravity and magnetic north.

Exteroceptive Sensing

The Raptor system captured range data through active laser and stereo devices. DRDC and Scientific Instrumentation Ltd. developed a nodding mechanism for a 2-D SICK, creating 3-D data scanning system. Communicating through an ethernet interface, the embedded RTEMS controller nods the laser up to 90°/sec with 0.072 degree resolution and 4cm accuracy over 1 - 30 metres. DRDC adopted Point Grey's Digiclops system to provide high speed range image streams. The Digiclops develops a disparity map between three camera image streams, publishing the resulting 3D range image stream over an IEEE-1394 Firewire digital connection.

3 Managing Software Complexity

The relentless march of ever increasing microprocessor computational capabilities, as given by Moore's Law, has resulted in small, efficient and portable computers that deliver traditional "super computer" capabilities. This growth in computational processing power is an enabling technology that allows programmers to implement evermore elaborate software.

Today's UxVs, with an extensive suite of sensors and capabilities, require sophisticated algorithms that yield complicated software implementations. Such UxV software engineering challenges development teams to efficiently and effectively manage the complexity, size, and diversity of software components. Robtics researchers have seized upon two important software concepts, middleware and frameworks, in their quest to control this complexity.

3.1 Networking Middleware

Given its long term objectives, DRDC identified communications middleware as a key tool for managing software complexity[Gow00b] and evaluated a number of middleware toolkits. The evaluation process centred on mitigating the research life cycle risks found in design, development, maintenance, publication and commercialization of research software that, nevertheless, approaches industrial quality and reliability. The requirements listed below, though subjective, captured DRDC's primary concerns:

- Open Source: Toolkits with Open Source Initiative licensing encourage the rapid publication and growth of new techniques within and between institutions without licensing encumbrances[Ini05].
- Open Tools: Support and use of open source tools and operating systems ensures that developers do not become orphaned by proprietary techniques.
- Component Based: Software based on component services, rather than single object hierarchies, to encourage modular, scalable, and distributed autonomous systems.
- Messaging: Multiple messaging paradigms frees developers to adopt any messaging protocol, process distribution, and structure.
- Comprehensiveness: Additional services, such as logging or exception handling, can often simplify or improve research software.
- Portability: Multiple platforms are a certainty in DRDC's UxV program, making toolkit portability across operating systems a crucial consideration.
- Reconfiguration: Mission and payload variation makes dynamic reconfiguration highly desireable.
- Applicability: The toolkit must not restrict the application domain and must accommodate current and future robotic architectures, payloads, and missions, particularly multi-robot tasks.
- Resource Usage: With drastic variation in possible scale, memory and storage space should be conserved and/or support off-loaded computation.
- Performance: Autonomous vehicles need fast processing to move at human rates on the battlefield, making high control rates with low overhead very important.
- Usage: Popularity and maturity of the toolkit influences adoption and the number of comparative experiments.
- Ease of use: Similarly, ease of use will widen the toolkit's popularity and lower development costs.

Table 1 lists communications middleware toolkits reviewed by DRDC. The middleware attributes were subjectively ranked on a 5 point scale, with 1 denoting a low ranking. Table 2 lists the middleware name acronyms used in Table 1 and provides references.

While no consensus has emerged on the best toolkit, clearly many recognize the significance of UxV middleware. From DRDC's perspective only three middleware toolkits, ACE, TAO and ICE, ranked as possible candidates. The Carnegie Mellion University trio, IPT, RTC and IPC, were too narrow in their application domain. NML, with its close links to the synchronous NASREM/RCS architecture[Alb97], lacked flexibility. The proprietary NDDS and .NET products deny impartial software review while restricting the distribution and sharing of software. Promising the capabilities of CORBA without its complexity, the ICE middleware is such a new product that both the adoption

Table 1. Summary of Communications Middleware Toolkits and evaluation criteria

Criteria	IPT	RTC	NML	NDDS	IPC	ACE	TAO	.NET	ICE
Open Source	5	2	5	1	5	5	5	1	3
Open Tools	5	5	5	1	5	5	5	1	4
Component Based	2	2	3	4	2	5	5	5	5
Messaging	2	2	2	3	2	3	5	5	5
Comprehensive	1	1	3	3	1	3	5	5	5
Portability	3	3	4	3	3	5	5	1	5
Reconfiguration	3	3	2	4	3	4	5	5	5
Applicability	3	3	3	3	3	5	5	5	5
Resources	5	5	5	4	5	3	1	1	3
Performance	5	5	5	5	5	5	3	2	4
Usage	1	2	3	3	1	5	5	5	3
Ease of Use	4	4	3	4	4	3	2	2	3

Table 2. Middleware Acronyms

Acronym	Name
IPT	Interprocess Communications Toolkit[Gow96]
RTC	Real-Time Communications[Ped98]
NML	Neutral Messaging Language[MP00]
NDDS	Network Data Distribution System[GPCH99]
IPC	Inter-Process Communications[Sim91]
ACE	Adaptive Communications Environment[SH02, SHS04]
TAO	The ACE ORB[HV99, Bol02, SK00, DSM98]
.NET	Microsoft Web Services Strategy
ICE	Internet Communications Engine[Hen04]

and long term viability are unknown. Thus DRDC concluded ACE and TAO are the most suitable communications middleware for UxVs.

3.2 Robot Middleware

Changing focus from tele-operation to autonomy, DRDC reviewed the history of robotic architectures and examined current architecture trends. Over the last 20 years, robotics has shifted from philosophically grounded deliberative artificial intelligence and reactive subsumption-esque architectures to pragmatic make-it-work hybrid architectures. This migration to pragmatism, captured by Gowdy[Gow00a], resonates with DRDC's long experience with UGVs. Gowdy groups robotic architectures into those driven by either reference or emergent architecture philosophies. Reference architectures establish idealized guidelines for component design and data flow. The emergent philosophy eschews any single reference design, advocating software frameworks that allow architecture to emerge and evolve. In retrospect, DRDC's ANCÆUS was a reference architecture that, ultimately, could not provide the necessary flexibility, extensibility, and scalability for autonomous military robotics. Additionally, ANCÆUS was a "closed system", supported and used by only DRDC and a few selected contractors, thus making it difficult for DRDC to leverage and utilize other sources of research.

DRDC concluded from its ANCÆUS experience that open source software and collaborative research are a crucial means of capturing and communicating science, the ultimate product of DRDC's UxV research.

DRDC evaluated candidate robotics frameworks using the same criteria as networking middleware, with the addition that each framework should:

- support an emergent architectural design strategy.
- support modular, extensible, flexible and scalable software systems.
- use standard communications middleware toolkits.

Numerous robotics architectures were reviewed[GBD04] including Player/ Stage[BGH03], DRCS[Alb97], NASREM[JAH87], TCA[Sim94], Dervish[INB95], CLARAty[RVD01], RAP[Fir89], Xavier[SKS98], Saphira[KM98] and 3T[RBS97], Berra[MLC00], Marie[CCT04], Carmen[MMT03], OCP[LWV01], Miro[HUK02, HUS05] and Orca[ABW05]. Table 3 provides more details on a selected list of architectures. The rating given to each category ranges from a low of 1 to a high of 3 and are subjective values from the view points of the authors.

Table 3. Candidate Architectures

Architecture	Open	Tools	Emergent	Comms	Modular	Usage
Player/Stage	3	3	3	1	3	2
Carmen	3	3	2	2	2	1
Marie	3	3	3	3	2	1
Miro	3	3	3	3	3	1
Orca	3	3	3	3	3	1
CLARAty	1	?	?	?	?	1
OCP	1	3	?	3	?	1
Saphira	1	?	?	?	?	2
DRCS/NASREM	3	3	1	3	2	2

Using the above criteria the list of potential candidates was quickly reduced to Player/Stage, Carmen, Marie, Miro and Orca. Obvious candidates such as CLARAty, OCP and Saphira are either closed source or unavailable in Canada, while DRCS and NASREM define reference architectures.

The five remaining frameworks were examined for their ability to meet the needs of implementing autonomy on vehicles at DRDC. This investigation concluded that frameworks based upon the CORBA communications middleware held the most promise it terms of allowing a modular, extensible, flexible and scalable architecture to emerge. Two of the reviewed frameworks were based upon CORBA and only one of these frameworks is completed and in current use. It was concluded that Miro, while originally designed for soccer playing robots, had both the flexibility and expandability to serve as a foundation for DRDC's UxV program.

4 Miro and CORBA

4.1 Introduction

Though very powerful and robust, CORBA has a steep learning curve. As a measure of this complexity, the CORBA bible, *Advanced CORBA Programming with C++*[HV99], at a 1000 pages is surpassed by the two-volume *TAO Developer's Guide* [TAO03] at approximately 1500 pages. Fortunately, for robotics researchers, the Miro framework encapsulates much of this complexity within a small, manageable toolset[Utz03].

Implementing services under Miro requires understanding six key concepts, depicted in Figure 3:

- Interface Description Language
- Naming Service
- Servers
- Clients
- Event Channels and Polling
- Data Exchange Patterns

Figure 3 shows the relationships between these concepts, while the following sections introduce these concepts in more detail.

Interface Description Language

The Interface Description Language (IDL) describes the properties of a CORBA object in terms of data types and access methods. The IDL compiler transforms this description into client and server side code using a language binding. Thus, via the IDL language, a CORBA object represents a component with an interface that allows independent processes to exchange information in a network transparent manner. While TAO supports numerous language bindings including Java, C++, C and Fortran, Miro currently uses only the C++ language binding.

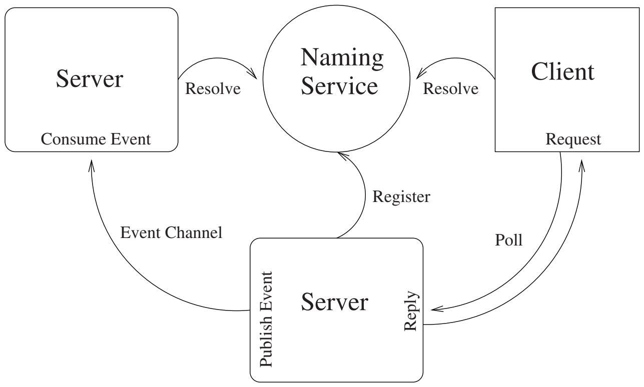

Fig. 3. Key Miro and CORBA Services

Naming Service

Much like a phone book's *white pages* that maps names to phone numbers, the CORBA Naming Service facilitates data exchange between processes by mapping object names and event channel names, both simple text strings, to object references and event objects respectively. In the Miro context, the Naming Service allows a Miro server or client to resolve an object name to an external object reference and instantiate the object reference locally, even though it exists on another Miro server. For event channels the Naming Service enables a Miro server to subscribe to an event channel using a simple text string.

Miro Server

Using the Miro server framework, both traditional client-server transactions and event driven processing can occur simultaneously. *Events* can be broadcast or *published* on an event channel to multiple *subscribers*, while the server can respond to traditional polling requests from a client [1]. The Miro server uses the server side IDL object binding to setup, maintain and allow remote (polling) access to an internal CORBA object.

Miro Client

The Miro client defines a framework, using the client side IDL object binding that allows a client process to poll a Miro server in a familiar client-server transaction.

[1] A CORBA derived hybrid server is both a server and a client.

Event Channels and Polling

Using CORBA capabilities Miro implements both the message and information paradigms, as polling and event channels respectively. Under the polling paradigm a Miro client directly requests and receives data from a Miro server. Events channels, implemented using the CORBA Notification Service, allow a Miro server to anonymously subscribe to events published by another Miro server. Figure 3 shows both of these data exchange implementations.

Data Exchanges Patterns

The Miro framework enables data exchanges that adhere to specific patterns. For a Miro server these patterns include:

- Register an event channel under a text string name.
- Register an object reference using a text string name.
- Subscribe to an event channel and receive published events.
- Resolve an object name into an object reference and polling the object interface for data.

The Miro client is limited to resolving an object name into an object reference and polling the object interface for data.

4.2 Design Patterns

The Miro server and client processes, referred to in Section 4.1, are constructed using one of three basic design patterns. All software implemented using these design patterns have defined CORBA object interfaces, or in software parlance these design patterns yield *components*.

The following sections provide details on the implementation and use of these design patterns.

Subscribe-Publish Server

The *Subscribe-Publish server* design pattern, illustrated by the collaboration diagram shown in Figure 4, allows a Miro server to receive events, process the data, and publish events. Using this design pattern the Miro server becomes an independent component with its interfaces defined by the CORBA objects that facilitate event subscription and publication as well as the CORBA objects that enable polling.

This design pattern results from the collaboration of the following classes:

- The Structured PushConsumer class responds to events published on event channels by one or more Miro servers.
- The Algorithm processes the data provided by the event.
- The Implementation class contains:

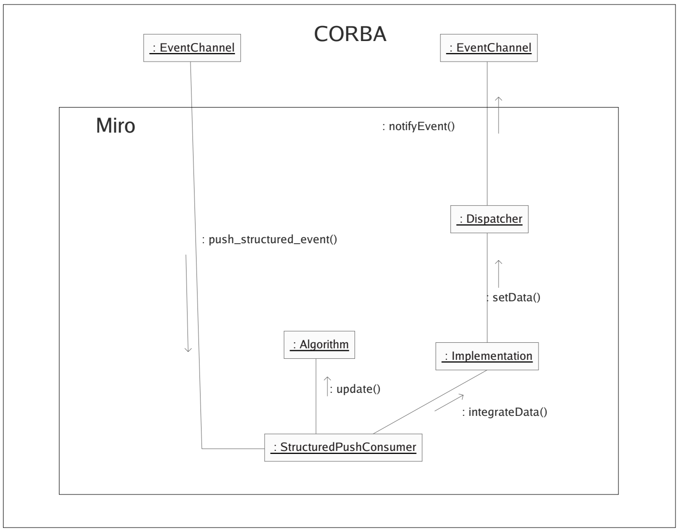

Fig. 4. Publish-Subscribe Server Design Pattern

  – Defines the the CORBA object's polling methods thus allowing the server to respond to polling requests.
  – Invokes the Dispatcher class to publish events.
- Finally the Dispatcher class is responsible for supplying the CORBA objects to the event channel where they can be distributed to other Miro servers.

Publish Server and Reactor

The *Publish Server and Reactor* design pattern serves as the basis for handling external hardware/software entities. Its fundamental objective is to separate the physical device driver from the software that publishes events and enables polling. Figure 5 shows a collaboration diagram detailing structure of this pattern.

The following classes collaborate to implement this design pattern:

- The Connection class is responsible for managing the connection between the physical device and Miro, which includes establishing the connection and writing commands to the device.
- The interrupt driven EventHandler class receives and processes data from the Connection class when data is available. This processing includes packet synchronization and the construction of a message from the low level byte stream.
- The Message class handles the routing of completed messages to the Consumer class.

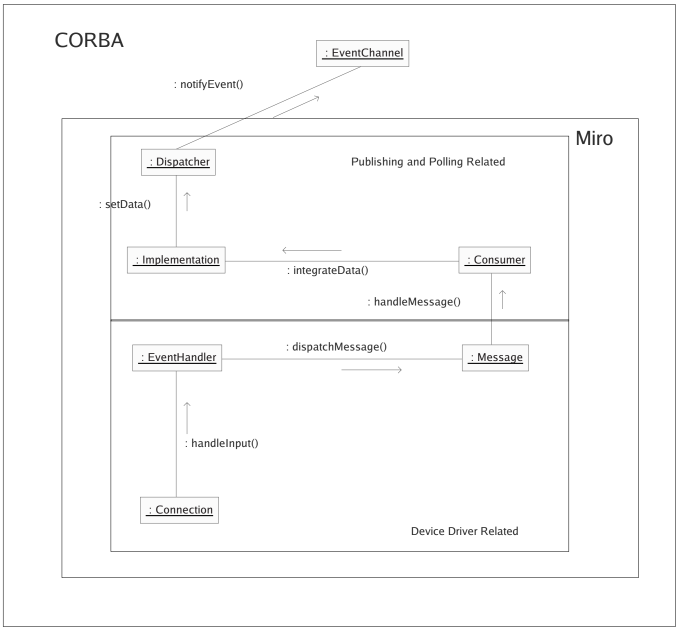

Fig. 5. Publish Server and Reactor Design Pattern

- The Consumer class is responsible for taking the device specific message and converting it into a CORBA object. It is important to note that all classes up to this point are device specific while the output from the Consumer class uses a generic CORBA object and thus different devices of the same type may reuse the same Implementation class.
- The Implementation class:
  – Defines the polling methods of the CORBA object, thus allowing the server to respond to polling requests.
  – Invokes the Dispatcher class to publish events.
- Finally the Dispatcher class is responsible for supplying the CORBA objects to the event channel where they can be distributed to other Miro servers.

Client

The client design pattern, shown in Figure 6, allows a client process to poll data from a Miro server. The client application's Implementation class allows:

- The resolution of a CORBA object reference using an object name lookup in the Naming Service.

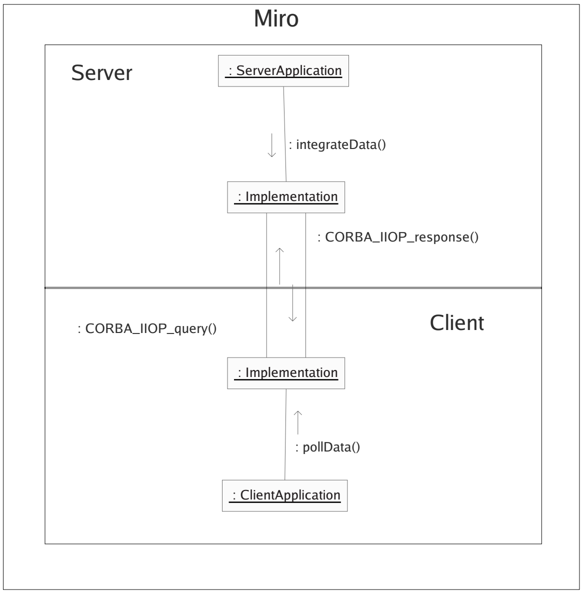

Fig. 6. Client Design Pattern

- Polling uses the CORBA Internet Inter-ORB Protocol to request query and receive data from Miro server application[2].

4.3 A High Level Example: World Representation

Introduction

Section 4.1 introduced basic Miro and CORBA concepts, while Section 4.2 gave an overview of the design patterns used to implement Miro services. This section illustrates how the Miro framework, design patterns, and CORBA services are used to implement world representation capabilities for a UxV.

World Representation

An autonomous UxV requires a representation of its world, which allows it to safely avoid obstacles while navigating. Figure 7 shows a flow diagram representation of the software that implements range data acquisition and the creation a terrain map representation of the world.

The generation of a world representation utilizes three services, the CORBA Naming Service, the Laser component and the Map component. The Naming Service is a standard CORBA service whose purpose and usage was described

[2] The identical Implementation class is instantiated on the client and server processes.

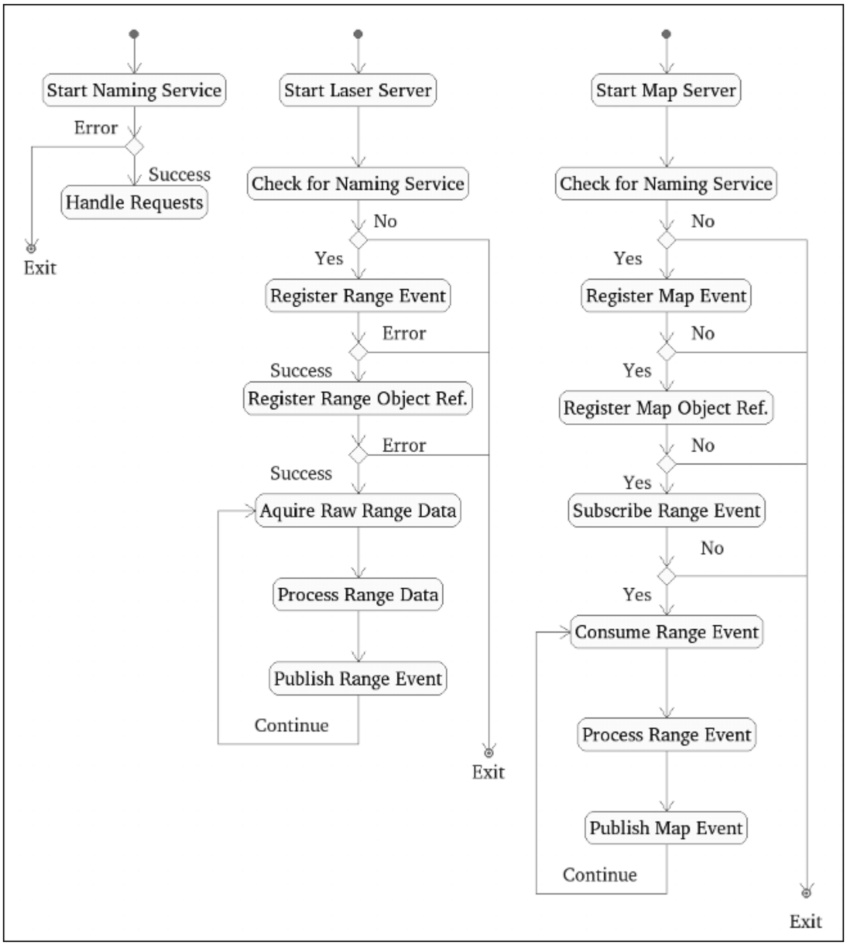

Fig. 7. Range Data and Map Generation Flow Diagram

in Section 4.1. The following sections describe the implementation of the Laser and the Map components using the Miro framework and design patterns.

Laser Component

The Laser component acquires raw range data from a laser range finder, processes this data into a 3D representation and then publishes the data as a *Range* event. It requires a physical device driver for the external nodding SICK laser data source and thus uses the *Publish Server and Reactor* design pattern described in 4.2 as a template.

Upon startup the Laser component first verifies a Naming Service is running. It then registers the *Range* event object, under the event channel named

EventChannel, with the Naming Service. It also registers the *Range* object reference with the Naming Service which, when resolved, allows a client to poll for *Range* object data.

Having successfully completed the event and object registration, this component then initializes the laser and begins to acquire raw range data. This raw range data is converted to a 3D representation and the 3D data is published to the *EventChannel* as a *Range* object event. This process is repeated on the 26.6 ms update period of the SICK laser.

Map Component

The Map component uses 3D range data to create a $2\frac{1}{2}$ D terrain map representation of the world. This component receives and publishes events on an event channel, thus the *Publish-Subscribe Server* design pattern described in 4.2 served as the template for this implementation.

As a first step the Map component verifies the existence of the Naming Service and then registers the *Map* event object against the event channel named *EventChannel*. It also registers the *Map* event object, under the *EventChannel*, with Naming Service and registers the *Map* object reference with the Naming Service.

With the registration completed the Map component then resolves and subscribes to *Range* events. The 3D Range data, provided by the *Range* object events, is delivered under the information based paradigm, as there is no direction connection between the Laser and Map components. The Laser component anonymously produces *Laser* events, while the Map component anonymously consumes these events. Thus as detailed in Section 9, the Laser component can easily be replaced by the Logging component as the producer of *Range* events.

Upon receiving a *Range* object event the Map component processes this 3D range data, creates the $2\frac{1}{2}$ D terrain map and publishes a *Map* object event.

5 Design Process

The requirements analysis and the available hardware placed necessary constraints on the development process. Working down from the requirements and up from the available hardware, an initial system architecture was constructed which detailed software components and established the dataflow between elements. Negotiation between the suppliers and consumers resulted in skeleton CORBA objects, defined via IDL interfaces.

The Miro infrastructure and CORBA object discipline provide a flexible development environment with firm deliverables. Miro decouples the algorithm from the interface implementation, limiting a given module's sensitivity to 'upstream' process failures and reducing the impact of internal failure on other downstream processes. Consuming from and publishing to established CORBA object interfaces, the developer may adopt any algorithm or technique within the process. Thus, the CORBA object based skeleton provides the freedom to implement any algorithm, the security from other poorly behaved processes, and the responsibility to deliver to agreed interfaces.

Miro and the CORBA object skeleton approach effectively seeds system development. Interfaces mark both a start and endpoint of a development effort and Miro's design patterns provide basic communication mechanics. Together, a raw skeletal image of a system can be rapidly assembled. If the algorithm development method is iterative – often the case in research– successive improvements can be incorporated to working skeleton code with the confidence that the mechanical details of interprocess communications are fully functional.

Both the skeleton and design patterns freed DRDC staff to develop algorithms within only partially working systems. Routinely, components were exercised in isolation using the minimum Miro services, while some algorithmic elements were developed in Player/Stage [BGH03] and later 'dropped' into the appropriate skeleton. By adhering faithfully to CORBA object interfaces, investigators engaged in iterative development with minimal impact on others.

Based on a rough hybrid control scheme and the requirements analysis, development began with the negotiation/establishment of the CORBA object interfaces between software components. The early establishment of these CORBA objects benefited development by ensuring that developers understood the information flow through their modules and that holes in the design could be found quickly.

Developers chose to either develop CORBA objects directly using IDL derived data types, or to develop outside of the framework using custom or third party data types. Algorithm efficiency typically drove custom type adoption, while throughput and simplicity drove the use of IDL types.

CORBA object based design guaranteed simple integration of the algorithm into established Miro design patterns using relatively simple data types. However since CORBA does not marshall and unmarshall pointer types, structures containing pointers cannot be transferred between processes. This forced IDL-based code to use simpler data types, often at the cost of algorithm efficiency.

Developers using custom data structures freely adopted the most suitable data types but ultimately had to convert to and from IDL equivalents. So while this technique favoured algorithmic efficiency, performance was sacrificed in type conversion.

Components were developed using the three design patterns described in 4.2. Low-level hardware drivers were developed using the *Publish Server and Reactor* design pattern as seen in Figure 5, while information processing and control services were developed using the *Publish-Subscribe Server* and *Client* design patterns seen in Figures 4 and 6 respectively.

A number of the low level drivers were leveraged off the Player/Stage project. The ease of integrating Player drivers into Miro is a testament to the design of both systems. Since both Player and CORBA decouple the interface from the implementation a majority of the Player source could be used directly in the Miro based driver. Even though the low-level drivers would undergo minor changes throughout the development cycle, these drivers were and utilized early in the development process. This functionality allowed developers to focus their efforts on algorithmic development.

5.1 Testing

Like development, testing benefits from the process isolation afforded by CORBA objects. Since Miro was not a monolithic system, fragments of the Raptor controller could be launched or killed with little impact. In conjunction with LogPlayer, and archive data, such partial testing could occur on the Desktop and across the network.

Testing Tools

In addition to standard testing tools and practises, DRDC used Miro's Logging toolset and Log Level facilities.

Two components comprise the Miro Logging toolset: the LogNotify and LogPlayer duo, Miro Debug Logging and visual interfaces.

LogNotify and LogPlayer

In realtime, the LogNotify component captures desired events published on an event channel. The graphical LogPlayer component, shown in Figure 8, controls the sequenced, variable rate, data playback over an event channel. Event streams could be accelerated, decelerated, paused, and reversed, permitting close inspection of process failure modes. Further, any event stream could be disabled in the player, permitting selective testing of process event handling. While most processes tolerate such input control, any process using an internal clock would only perform accurately with logged data played back at original rates.

Since downstream processes relied on raw sensor data streams, DRDC usually archived only raw sensor data from trials and not downstream process output. This strategy reduced log size while permitting comparative off-line optimization and debugging of downstream processes.

Figure 9, where the Laser Server component from 7 has been replaced by the LogPlayer component, illustrates the power of these logging tools. In this collaboration diagram the Map component consumes logged *Range* object events published by the LogPlayer and this is accomplished without changing the Map component code or configuration.

The CORBA *Range* event object enables this seamless integration between *device* data and *logged* data by providing a generic, uniform interface.

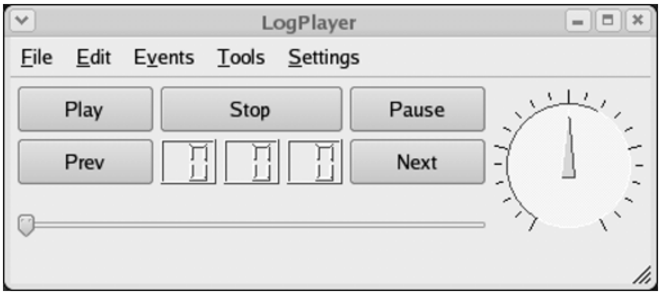

Fig. 8. Miro Log Player

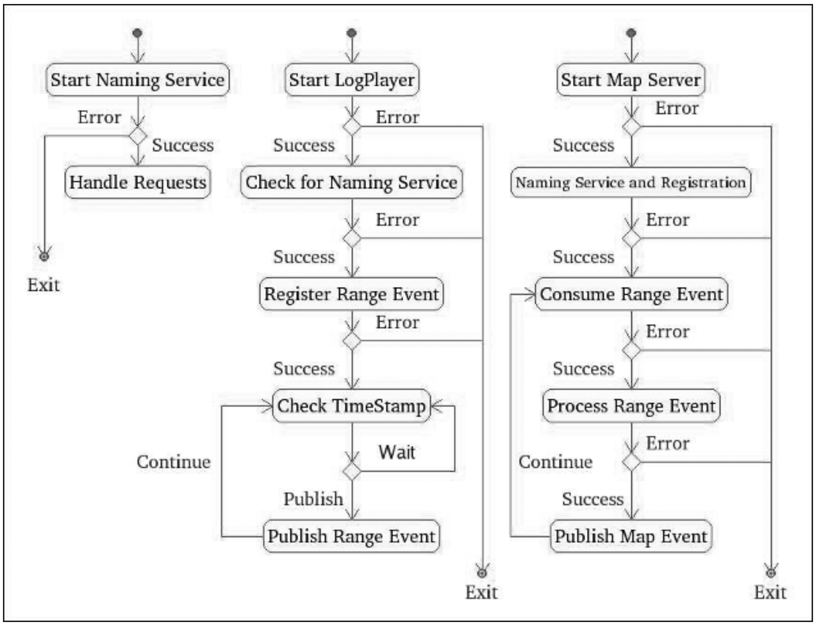

Fig. 9. Map Generation with Logged Range Data Flow Diagram

Miro Debug Logging

Debugging a multi-process, distributed environment frustrates the best development efforts. Invariably, testing methods fall to the simplest mechanism: printed debug statements. Fortunately, Miro's LogLevel utilities permitted testers to output filtered debug statements based on a unique log levels. Processes supporting Loglevels take command line switches that display or suppress debugging information without recompiling.

Visual Interfaces

DRDC also developed a number of visual testing aides, such as the traversability display depicted in Figure 12. These utilities graphically display event channel and polled data in an intuitive format.

Module and Integration Testing

Under tight time constraints, the ALS program tried to avoid integration problems common to large, complex software systems by integrating software onto the Raptor as early as possible. DRDC leveraged the sensor drivers by logging data from field trials and using the LogPlayer to perform desktop testing. Together these strategies maximized the time available for algorithm development and testing.

In the final two weeks before demonstration day, system integration testing incrementally added subsystems to the control network to assess whole-system performance and stability. While unusual for projects of this scope, this short test period sprung from DRDC's arbitrated control design and Miro's run-time properties that, together, permitted extensive preintegration subsystem testing. In general, preintegration testing established stability, correctness, and performance of individual control threads. Final integration testing brought multiple threads together to expose system vulnerabilities and reveal total system performance. In most cases, final system improvements were reduced to parameter tweaking to achieve a desired vehicle behaviour.

This approach lends itself well to a research environment where algorithm content and stability fluctuate constantly. Though desireable, a more rigorous approach would not be possible given the time constraints and scope of the system.

6 Implementation

Using the software facilities discussed in Section 3.1 and design methodology detailed in Section 4 DRDC-Suffield implemented a flexible and modular control system for autonomous UGVs. Trials completed in Fall 2005 demonstrated this software on the Koyker Raptor vehicle and sensor compliment described in Section 2.3, as shown in Figure 10.

The vehicle control system, based upon separate reusable components and reusable interfaces is portable to a variety of autonomous vehicles. These components, derived from CORBA object interfaces and event delivery mechanisms, impart a number of desirable attributes:

- Extendability, allows changes in vehicle platform, sensing or algorithms.
- Flexibility to distribute processes to where computing power is available.
- Reactivity to incoming sensor data.

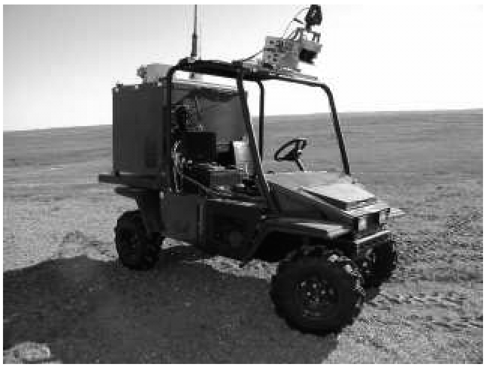

Fig. 10. One of two modified Koyker Raptors used in DRDC's ALS Project. Each Raptor used one or more roof mounted SICK lasers and stereo cameras; Differential GPS and IMU; and wireless mesh networking routers.

- Scalability to varyings levels of vehicle capability and autonomy.

This section describes the software control system, its components and interfaces, and usage.

6.1 System Overview

The vehicle controller consists of many components, each a process, possibly co-located on one processor, or distributed over a network. They can be organized into four broad categories:

1. Hardware Interfaces - Components that interface directly with hardware, providing data services from GPS, IMU, Laser and Stereo Range sensors, as well as managing the vehicle platform.
2. Information Processing Services - Components providing world modelling and data fusion from the raw sensor data, including Terrain and Traversability Mapping, and pose estimation via the Model Server
3. Vehicle Control Services - Those components that produce vehicle behaviour, such as Tele-operation, Obstacle Avoidance and Waypoint Following, Path Planning and Hazard Detection.
4. Decision Making Services - The Arc Arbiter combines the output of the reactive and deliberative control threads to provide vehicle commands.

These components and their interconnections are shown in Figure 11. The CORBA event channel facilitates the delivery of CORBA event objects between information suppliers and consumers, either local or remote, and is indicated by the data flow connectors in Figure 11. In general, information flows from the Hardware Interfaces, through the Information Processing Services, to the Vehicle Control Services. The use of components, with their defined interfaces, allows for exceptions such as the Hazard Detection algorithm that directly uses laser data.

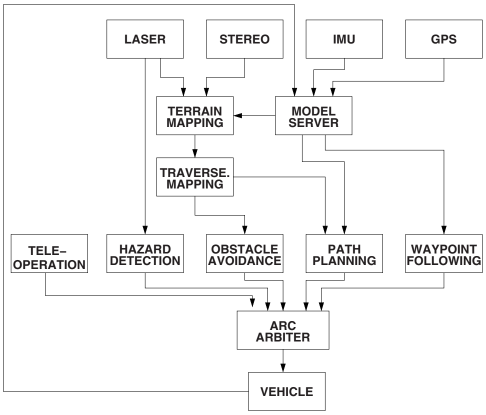

Fig. 11. ALS architecture flow diagram. Each box represents a component that can reside anywhere on a network and each connector represents data flows accessible to any subscriber component.

6.2 Hardware Interfaces

The GPS, IMU, Laser, Stereo and Vehicle components interface with the physical hardware, acquiring data and publishing CORBA event objects. As data abstractions, CORBA objects divorce physical devices from their data representations.

6.3 Information Processing Services

In general, information services such as pose estimation and world representation draw from the hardware interfaces and feed into vehicle control services.

Pose Estimation

Accurate sensing, world modelling and control on large outdoor vehicles requires accurate estimates of both vehicle and sensor locations in local and global coordinates. As shown in Figure 11, Model Server provides these services. With every generated *GPS*, *IMU*, and *WheelOdometry* event object, Model Server updates a direct state Kalman filter of vehicle pose and publishes a CORBA *Pose* event object containing the location of the vehicle in global coordinates.

Model Server maintains an internal geometry database of the system's rigid body assemblies. By polling Model Server through the CORBA *Platform* object reference, clients can interogate Model Server for either a *body* list, a given body's *frame* list, or transformations between any frames in the system. A client process can combine static internal transformations with filtered *pose* events to establish the global coordinates of any vehicle body frame.

World Representation

Intelligent navigation requires a world representation. For the ALS project DRDC represented the world through a Terrain Map and its companion, the Traversability Map.

Terrain Map

The Terrain Map component subscribes to CORBA *Range3d* event object published by the laser and stereo camera exteroceptive sensors. The service fuses range data into a grid map, a rectangular array of regions, to build a 2.5D or digital elevation map[Kel97, HK93, KK92]. The Terrain Map estimates the height of the world at each cell in the 20cm x 20cm grid, as well as the elevation variance. This map is local to the area directly in front of the vehicle, scrolling and wrapping as the vehicle moves in both the X and Y directions. With the new range data fused into the terrain map a CORBA *MapEvent* event object is published.

Traversability Map

The Obstacle Avoidance and Path Planning components of the Vehicle Control Services do not subscribe directly to the Terrain Map, but to a simpler Traversability Map service, depicted in Figure 12. The traversability analysis transforms the terrain map into a traversability map, containing a *Goodness* rating the terrain hazards in a given area, similar to the Gestalt system[SGM02]. For this map the grid cells are typically 50cm x 50cm. An analysis of step and slope hazards in the corresponding Terrain Map grid generates a traversability element's Goodness rating. The Traversability Map component publishes a CORBA *TravMap* event object with each map update.

6.4 Vehicle Control Services

Asynchronous multiple behaviour components produce a sliding scale of autonomy. Each independent behaviour publishes CORBA *ArcVote* event objects. The Arc Arbiter consumes these vote objects and directs the Raptor's movement. The following describes the five behaviour components implemented thus far.

Tele-Operation

This component allows direct joystick control of the vehicle with either a direct or network connection.

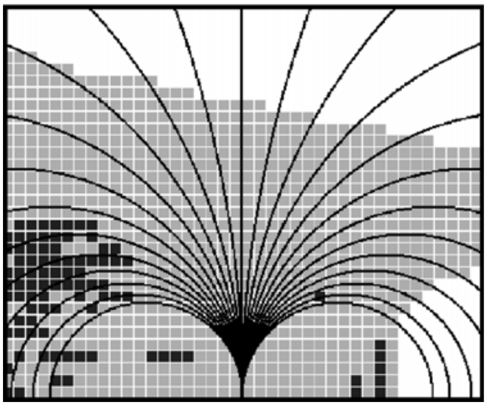

Fig. 12. A traversability map overlaid with 25 candidate arcs. Obstacles are dark cells and traversable areas are light cells.

Hazard Detection

Similar to other obstacle detection systems [HK], this component inspects raw range sensor data from a laser range finder or stereo vision camera. Received as a CORBA *Range3dEvent* event object, this component detects obstacles higher or lower than a user defined safety range. The detection of a qualifying obstacle will halt the vehicle by publishing a veto within a CORBA *ArcVote* event object.

Obstacle Avoidance

This component examines local terrain to establish safe candidate steering trajectories, similar to the Gestalt system [SGM02], and also provides maximum speed recommendations. Subscribing to the CORBA *TravMap* event object, the Obstacle Avoidance component estimates the vehicle's stability and safety over candidate steering arcs using Traversability Map data to construct a CORBA *ArcVote.*

Waypoint Following

The Waypoint Following component use the Pure Pursuit algorithm [Cou92] to follow a path of user defined waypoints. It subscribes to the CORBA *Pose* event objects published by the Model Server, executes the Pure Pursuit algorithm and publishes a CORBA *ArcVote* event object.

Path Planning

This component provides high level path planning and goal directed behaviour based upon a global map of accumulated local Traversability Maps. It accounts for both desirability of terrain as well as goal directed behaviour in making its decisions using the D* Lite algorithm [KL02]. The data for the D* Lite algorithm arrives as CORBA *TravMap* and *Pose* event objects, the algorithm executes, and a CORBA *ArcVote* event object is published.

6.5 Decision Making Services

In order to combine the outputs of all the Vehicle Control Service's behaviours, a vote based arbitration scheme was implemented in the Arc Arbiter component.

All Vehicle Control modules publish their desired behavior using the CORBA *ArcVote* event object. The Arc Arbiter component subscribes to the *ArcVote* event and fuses the individual behaviours into a global action using an arbitration scheme. DRDC expanded a DAMN-like [Ros95] arbitration scheme. Each behaviour votes for each arc in a set of arc candidates, for example as shown in Figure 12. The behaviour gives a desirability, a certainty, a maximum speed and, if unacceptable, a veto to each arc. The IDL code listing shown below illustrates the implementation of CORBA *ArcVote* event object.

```
struct ArcVoteIDL     // Defines a behaviour’s vote for a single
arc {
    float curvature;  // Curvature (1/meters)
    float desire;     // Desireability: between 0 (worst) and 1 (best)
    float certainty;  // Confidence: between 0 (least) and 1 (most)
    float max_speed;  // Maximum acceptable speed (meters/second)
    boolean veto;     // "true" vetos this arc
};

// Define the available voting behaviours
enum Voting_Behaviour{HAZARDDETECT, OBSAVOID, PATHPLANNER,
WAYPOINT};

// Define the number of candidate arcs in the system
const long NUM_CAND_ARCS = 25;

struct ArcVoteEventIDL               // The CORBA ArcVote event
object {
  ArcVoteIDL VoteSet[NUM_CAND_ARCS]; // The array of votes
  TimeIDL time;                      // The time that the vote was generated
  Voting_Behaviour votingBehaviour;  // Identify voting behaviour
};
```

The Arc Arbiter component's event driven design gives the voting components the flexibility to run asynchronously. Further, behaviours may implement as much of the *ArcVote* object as they need. For example, the Hazard Avoidance module uses only the veto field, while the Obstacle Avoidance and Path Planning modules use all fields. Nevertheless, the arbiter combines all available outputs into one coherent decision packaged in a CORBA *Motion* object reference.

The vote based Arbitration scheme and the event based delivery of Vote events creates a sliding scale of autonomy, through a subset of available components. For example, to run teleoperation control, the system needs only three modules: Teleoperation, Arc Arbiter and Vehicle. For autonomous operations in simple environments, Waypoint Following replaces Teleoperation. As the environment increases in complexity, Hazard Detection, Obstacle Avoidance, and Path Planning can be added incrementally to provide greater autonomous capabilities.

7 Conclusions

This chapter summarised DRDC-Suffields's experiences during the migration from a tele-operation focused, to a general autonomy based program.

As part of this migration DRDC-Suffield conducted an in-depth literature review of network middleware, robot middleware, and robot control architectures. This review concluded the most suitable communications middleware is TAO. TAO is a real-time CORBA implementation based on the ACE communications middleware. Fortunately DRDC-Suffield discovered open source, CORBA based Miro framework prior to developing in-house CORBA tools. Though originally developed for Robosoccer robots, Miro possessed a modular, flexible, scalable and extensible design well suited to more complex robot control problems.

DRDC modified Miro for an outdoor UGV, revealing both the advantages and disadvantages of both middleware toolkits and resulting design process. After experience with ANCÆUS tele-operated platforms, DRDC sought a platform-neutral control structure offering scalability and extensibility through computing parallelism. Miro achieved this goal by providing well designed CORBA object interfaces and, therefore, transparent, network-enabled software.

CORBA IDL interfaces decouple the control system both from the sensors and the vehicle. Creating well defined interfaces creates flexibility by removing dependence on specific command sets, for example: the specific range sensor has little impact on terrain mapping. Similarly, vehicle commands (translational and rotational velocity) remain the same regardless of vehicle. With different platform or sensor configurations, integrators need change only the drivers. More importantly for DRDC, the internal control algorithms can be easily changed, allowing direct comparison of their performance.

The CORBA event based delivery mechanisms provided two key benefits. Processes could be distributed over network nodes as needed. Secondly, processing and control modules run on the receipt of new data, using only the most recent data delivered if a recipient is too slow. In this way, no explicit timing is necessary anywhere in the system. Since sensors and algorithms are free to run at any rate, the hybrid design can react to sensory information over any time scale.

CORBA's network transparency allows components to be distributed across processors and networks as required, without the need to change code or configurations. The use of CORBA events or polling has the added benefit of decoupling the execution threads of components, thus allowing components to execute at independent rates.

This modular approach has some shortcomings, however. The complexity and reconfigurability of the system requires significant run-time management. Depending on the sensors and level of autonomy, the user may have to configure, launch, and monitor up to 20 different processes. To date, these tasks have been imperfectly managed through a single XML configuration system for all processes. For monitoring, DRDC has developed a process *watchdog* component that monitors and reports on the status of each component in the system.

CORBA's services and capabilities represented a significant, but worthwhile, learning curve DRDC. The Miro framework served as a crucial aid to understanding and using CORBA by providing templates and examples from which new components could be constructed. Despite CORBA's apparent complexity, robot subsystems remained responsive, often faster than Player-based equivalents. CORBA provides DRDC with industrial strength, platform neutral interprocess communication capabilities, and a plausible military software environment. Miro provides a much needed, accessible toolset focussed on robot systems, greatly easing DRDC's introduction to CORBA.

References

[ABW05] A. Makarenko A. Oreback A. Brooks, T. Kaupp and S. Williams, *Towards component-based robotics*, IEEE/RSJ International Conference on Intelligent Robots and Systems, August 2005.

[Alb97] J. Albus, *Drcs: A reference model architecture for demo iii*, Tech. report, 5994, National Institute of Standards and Technology, Gaithersburg, MD., 1997.

[BGH03] R. T. Vaughan B. Gerkey and A. Howard, *The player/stage project: Tools for multi-robot and distributed sensor systems*, Proceedings of the 11th International Conference on Advanced Robotics, 2003, pp. 317–323.

[Bol02] F. Bolton, *Pure corba: A code intensive premium reference*, Tech. report, SAMS, 2002.

[CCT04] F. M. J.-M. Valin Y. Brosseau C. Raievsky M. Lemay C. Cote, D. Letourneau and V. Tran, *Code reusability tools for programming mobile robots*, IEEE/RSJ International Conference on Intelligent Robots and Systems, 2004.

[Cou92] R. C. Coulter, *Implementation of the pure pursuit path tracking algorithm*, Tech. report, Tech Report CMU-RI-TR-92-01, Carnegie Mellon University, 1992.

[DSM98] D. Levine D. Schmidt and S. Mungee, *The design and performance of real-time object request brokers*, Computer Communications **21** (April 1998), 294–324.

[Fir89] R. Firby, *Adaptive execution in complex dynamic worlds*, Tech. report, 1989.

[GBD04] J. Giesbrecht S. Verret J. Collier G. Broten, S. Monckton and B. Digney, *Towards distributed intelligence*, Tech. report, Technical Report TR 2004-287, Defence Research and Development Canada - Suffield, December 2004.

[GBV03] J. Giesbrecht S. Monckton G. Broten, D. Erickson and S. Verret, *Engineering review of ancaeus/avatar an enabling technology for the autonomous land systems program*, Tech. report, tech. rep., DRDC Suffield, Dec. 2003.

[Gow96] J. Gowdy, *Ipt: An object oriented toolkit for interprocess communication*, Tech. report, Tech. Rep. CMU-RI-TR-96-07, Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, March 1996.

[Gow00a] J. Gowdy, *Emergent architectures: A case study for outdoor mobile robots*, Tech. report, PhD thesis, Carnegie Mellon University, 2000.

[Gow00b] J. Gowdy, *A qualatative comparision of interprocess communications toolkits for robotics*, Tech. report, Tech. Rep. CMU-RI-TR-00-16, Carnegie Mellon University, June 2000.

[GPCH99] S. Schneider G. Pardo-Castellote and M. Hamilton, *Ndds: The real-time publish-subscribe middleware*, Tech. report, Real-Time Innovations, Inc., 1999.

[Hen04] M. Henning, *A new approach to object-oriented middleware*, IEEE Computer Society **8** (January-Febuary 2004), 66–75.

[HK] L. Henriksen and E. Krotkov, *Natural terrain hazard detection with a laser rangefinder*, IEEE Int. Conf. On Robotics and Automation.

[HK93] M. Herbert and E. Krotkov, *Local perception for mobile robot navigation in natural terrain: Two approaches*, Workshop on Computer Vision for Space Applications, Sept. 1993, pp. 24–31.

[HUK02] S. Enderle H. Utz, S. Sablatnog and G. Kraetzschmar, *Miro - middleware for mobile robot applications*, IEEE Transactions on Robotics and Automation (June 2002).

[HUS05] S. Enderle H. Utz and S. Sablatnoeg, *Miro - middleware for robots*, Tech. report, http://smart.informatik.uni-ulm.de/MIRO/content.html Accessed, 2005.

[HV99] M. Henning and S. Vinoski, *Advanced corba programming with c++*, Addison-Wesley, 1999.

[INB95] R. Powers I. Nourbakhsk and S. Birchfield, *Dervish: An office navigation robot*, AI Magazine **16-2** (1995), 53–60.

[Ini05] The Open Source Initiative, *The open source definition.*, Tech. report, http://www.opensource.org/docs/definition.php, 2005.

[JAH87] R. Lumia J. Albus and H.McCain, *Hierarchical control of intelligent machines applied to space station telerobots*, Tech. report, 1987.

[JAU04] JAUS, *The joint architecture for unmanned systems, reference architecture specification ra v3.2 parts 1-3*, Tech. report, http://www.jauswg.org/baseline/refach.html, August 2004.

[Kel97] A. J. Kelly, *An approach to rough terrain autonomous mobility*, International Conference on Mobile Planetary Robots, January 1997.

[KK92] S. Kweon and T. Kanade, *High-resolution terrain map from multiple sensor data*, IEEE Transactions on Pattern Analysis and Machine Vision **14** (Feb. 1992), 278–292.

[KL02] S. Koenig and M. Likhachev, *D* lite*, Proceedings of the National Conference on Artificial Intelligence, 2002, pp. 476–483.

[KM98] K. Konolige and K. Myers, *'the saphira architecture for autonomous mobile robots*, Artificial Intelligence and Mobile Robots: Case Studies of Successful Robot Systems (1998).

[LWV01] S. Sander M. Guler B. Heck J. Prasad D. Schrage L. Wills, S. Kannan and G. Vachtsevanos, *An open platform for reconfigurable control*, IEEE Control Systems Magazine (June 2001).

[MLC00] A. Oreback M. Lindstrom and H. Christensen, *Berra: A research architecture for service robots*, International Conference on Robotics and Automation, 2000.

[MMT03] N. Roy M. Montemerlo and S. Thrun, *Perspectives on standardization in mobile robot programming: The carnegie mellon navigation (CARMEN) toolkit*, Proceedings of the Conference on Intelligent Robots and Systems, 2003.

[MP00] J. Michaloski and W. S. F. Proctor, *The neutral message language: A model and method for message passing in heterogeneous environments*, Proceedings of the World Automation Conference, (Maui, Hawaii), June 2000.

[Ped98] J. . Pedersen, *Robust communications for high bandwidth real-time systems*, Tech. report, Tech. Rep. CMU-RI-TR-98-13, Carnegie Mellon University, 1998.

[RBS97] E. Gat D. Kortenkamp D. Miller R. Bonasso, R. Firby and M. Slack, *Experiences with and architecture for intelligent, reactive agents*, Journal of Experimental and Theoretical Artificial Intelligence **9-2** (1997), 237–256.

[Ros95] J. Rosenblatt, *DAMN: A distributed architecture for mobile navigation.*, Proceedings of the 1995 AAAI Spring Symposium on Lessons Learned from Implemented Software Architectures for Physical Agents , H. Hexmoor and D. Kortenkamp (Eds.) AAAI Press, Menlo Park, CA., March 1995, pp. 317–323.

[RVD01] T. Estlin D. Mutz R. Petras R. Volpe, I. Nesnas and H. Das, *The claraty architecture for robotic autonomy*, IEEE Aerospace Conference, March 2001.

[SGM02] M. Maimone S. Goldberg and L. Matthies, *Stereo vision and rover navigation software for planetary exploration*, IEEE Aerospace Conference Proceedings, 2002.

[SH02] D. Schmidt and S. Huston, *C++ network programming volume 1*, Addison-Wesley, 2002.

[SHS04] J. Johnson S. Huston and U. Syyid, *The ace programmer's guide*, Addison-Wesley, 2004.

[Sim91] R. Simmons, *The inter-process communications (ipc) system*, Tech. report, http://www-2.cs.cmu.edu/afs/cs/project/TCA/www/ipc/ipc.html, 1991.

[Sim94] R. Simmons, *Structured control for autonomous robots*, IEEE Transactions on Robotics and Automation **10** (February 1994).

[SK00] D. Schmidt and F. Kuhns, *An overview of the real-time corba specification*, IEEE Computer special issue on Object-Oriented Real-time Distrubuted Computing (2000).

[SKS98] R. Goodwin S. Koenig and R. Simmons, *Xavier: A robot architecture based on partially observable markov decision process models*, Artificial Intelligence and Mobile Robots: Case Studies of Successful Robot Systems (1998).

[TAO03] TAO, *Developer's guide*, Tech. report, vol. 1 and 2, Object Computing Inc., 12140 Woodcrest Executive Drive, Suite 250, St. Louis, MO, 63141, 2003.

[Utz03] H. Utz, *Miro manual*, Tech. report, University of Ulm, Department of Computer Science, November 2003.