



Defence Research and
Development Canada

Recherche et développement
pour la défense Canada



Towards Distributed Intelligence

A High Level Definition

G. Broten, S. Monckton, J. Giesbrecht, S. Verret, J. Collier & B. Digney
Defence R&D Canada – Suffield

Technical Report
DRDC Suffield TR 2004-287
December 2004

Canada

Report Documentation Page		Form Approved OMB No. 0704-0188
Public reporting burden for the collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington VA 22202-4302. Respondents should be aware that notwithstanding any other provision of law, no person shall be subject to a penalty for failing to comply with a collection of information if it does not display a currently valid OMB control number.		
1. REPORT DATE DEC 2004	2. REPORT TYPE	3. DATES COVERED -
4. TITLE AND SUBTITLE Towards Distributed Intelligence (U)		5a. CONTRACT NUMBER
		5b. GRANT NUMBER
		5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S)	5d. PROJECT NUMBER	
	5e. TASK NUMBER	
	5f. WORK UNIT NUMBER	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Defence R&D Canada - Suffield, PO Box 4000, Medicine Hat, AB, CA, T1A 8K6		8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)		10. SPONSOR/MONITOR'S ACRONYM(S)
		11. SPONSOR/MONITOR'S REPORT NUMBER(S)
12. DISTRIBUTION/AVAILABILITY STATEMENT Approved for public release; distribution unlimited		
13. SUPPLEMENTARY NOTES The original document contains color images.		

14. ABSTRACT

Unmanned Ground Vehicle (UGV's) Research and Development within the Autonomous Land Systems (ALS) project will assist the Canadian Forces in fulfilling their future mandate. The ALS project derives its focus from the Autonomous Intelligent Systems (AIS) activity outlined by the DRDC Technology Investment Strategy (TIS). There are five anticipated classes of Unmanned Vehicles (UV): fixed or rotor wing aircraft Unmanned Air Vehicles (UAV); typically tracked, wheeled, legged Unmanned Ground Vehicles (UGV); stationary monitoring Unattended Ground Sensors (UGS); untethered, propellor or bouyancy driven, Unmanned Underwater Vehicles (UUV); and light propellor driven Unmanned Surface Vehicles (USV). The future battlespace demands compatibility between all UV classes. All UVs must have an inherent ability to share information if they are to provide the desired force multiplication factor for the future asyemtric battlespace. To effectively distribute intelligence modules within and between UVs, layered modular hardware design and portable, maintainable coding practice require an architecture that, at once, intrinsically supports and encourages distributed computing, and frees investigators to focus on the development of intelligent single and multi-vehicle control systems. An architecture founded on these elements defines, at a high level, the links between various software components that create an operational vehicle. Ideally, architectures should seamlessly transition between real vehicle control; system diagnosis through the replay of gathered data; and the control of a vehicle in a simulated world. Ideally, the investigator is then free to develop intelligence algorithms without vehicle implementation distractions. With satisfactory simulated performance, algorithms may be safely run on a physical vehicle. Conversely, historical data gathered from a real vehicle run can be replayed in a simulated environment to investigate, debug and optimize the algorithm performance. This document explores the depths of the multi-vehicle architecture problem using the past experience of other investigators, the apparent technological evolution of both hardware and software, and the demands of the future CF environment. This report overviews fundamental methods in multi-vehicle cooperation and coordination, single vehicle autonomous control, and the underlying infrastructure of real and simulated systems.

15. SUBJECT TERMS

16. SECURITY CLASSIFICATION OF:

a. REPORT

unclassified

b. ABSTRACT

unclassified

c. THIS PAGE

unclassified17. LIMITATION OF
ABSTRACT18. NUMBER
OF PAGES**92**19a. NAME OF
RESPONSIBLE PERSON

Towards Distributed Intelligence

A High Level Definition

G. Broten, S. Monckton, J. Giesbrecht, S. Verret, J. Collier & B. Digney
Defence R&D Canada – Suffield

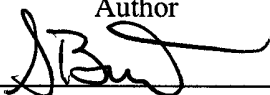
Defence R&D Canada – Suffield

Technical Report

DRDC Suffield TR 2004-287

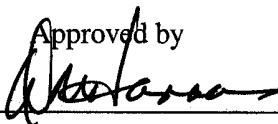
December 2004

Author



G. Broten, S. Monckton, J. Giesbrecht, S. Verret, J. Collier, B. Digney

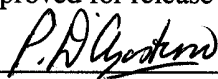
Approved by



D. Hanna

Head/Tactical Vehicle Systems Section

Approved for release by



Dr. P. D'Agostino

Chairman/Document Review Panel

Abstract

Unmanned Ground Vehicle (UGV's) Research and Development within the Autonomous Land Systems (ALS) project will assist the Canadian Forces (CF) in fulfilling their future mandate. The ALS project derives its focus from the Autonomous Intelligent Systems (AIS) activity outlined by the DRDC Technology Investment Strategy (TIS).

There are five anticipated classes of Unmanned Vehicles (UV): fixed or rotor wing aircraft Unmanned Air Vehicles (UAV); typically tracked, wheeled, legged Unmanned Ground Vehicles (UGV); stationary monitoring Unattended Ground Sensors (UGS); untethered, propellor or bouyancy driven, Unmanned Underwater Vehicles (UUV); and light propellor driven Unmanned Surface Vehicles (USV). The future battlespace demands compatibility between all UV classes. All UVs must have an inherent ability to share information if they are to provide the desired force multiplication factor for the future asymmetric battlespace.

To effectively distribute intelligence modules within and between UVs, layered modular hardware design and portable, maintainable coding practice require an architecture that, at once, intrinsically supports and encourages distributed computing, *and* frees investigators to focus on the development of intelligent single and multi-vehicle control systems. An architecture founded on these elements defines, at a high level, the links between various software components that create an operational vehicle. Ideally, architectures should seamlessly transition between real vehicle control; system diagnosis through the replay of gathered data; and the control of a vehicle in a simulated world. Ideally, the investigator is then free to develop intelligence algorithms without vehicle implementation distractions. With satisfactory simulated performance, algorithms may be safely run on a physical vehicle. Conversely, historical data gathered from a real vehicle run can be replayed in a simulated environment to investigate, debug and optimize the algorithm performance.

This document explores the depths of the multi-vehicle architecture problem using the past experience of other investigators, the apparent technological evolution of both hardware and software, and the demands of the future CF environment. This report overviews fundamental methods in multi-vehicle cooperation and coordination, single vehicle autonomous control, and the underlying infrastructure of real and simulated systems.

Résumé

La recherche et développement dans le domaine des véhicules terrestres sans pilote appartenant au projet des Systèmes terrestres autonomes (STA) aidera les Forces canadiennes (FC) à remplir leur mandat futur. Le projet STA est articulé autour de l'activité des Systèmes intelligents autonomes (SIA), soulignée par la Stratégie d'investissement technologique (TIS) de RDDC.

On prévoit cinq catégories de véhicules sans pilote : les véhicules aériens télépilotes (UAV) à ailes fixes ou à voilure tournante; les véhicules terrestres sans pilote (UGV) ordinaires à chenilles, sur roues ou sur jambes; les capteurs au sol isolés (UGS) de surveillance stationnaire ; les véhicules sous-marins sans équipage (UUV) sans ancrage, dirigés par propulsion ou flottaison ainsi que les véhicules sans pilote de surface dirigés par propulsion légère. Les futurs espaces de combat exigent que toutes les catégories de véhicules sans pilote soient compatibles. Tous ces derniers doivent posséder une capacité inhérente à mettre l'information en commun si celle-ci fournit la force de facteur de multiplication souhaitée pour le futur espace de combat asymétrique.

Pour répartir efficacement les modules d'intelligence à l'intérieur et entre les véhicules sans pilote, la conception modulaire et multidimensionnelle du matériel ainsi que la pratique de codage maintenable et portable exigent une architecture qui, sur-le-champ, soutient intrinsèquement et encourage un traitement réparti *et* qui libère *également* les chercheurs pour que ceux-ci puissent focaliser sur la mise au point de systèmes de contrôle intelligents d'un seul véhicule et d'un groupe de véhicules. Une architecture fondée sur ces éléments, définit à un haut niveau, les liens entre des composants de logiciels variés qui créent un véhicule fonctionnel. Les architectures devraient idéalement faire une transition harmonieuse à partir du contrôle d'un véhicule réel vers le diagnostic du système et la retransmission des données pour accomplir enfin l'exécution du contrôle d'un véhicule dans un contexte de simulation. Idéalement, ceci libère le chercheur qui est alors en mesure de mettre au point des algorithmes d'enregistrement sans être distrait par l'implémentation sur véhicule. Si la performance simulée est satisfaisante, les algorithmes peuvent être effectués sur un véhicule réel. Inversement, les données historiques recueillies à partir d'un véhicule réel peuvent être retransmises dans un contexte de simulation pour étudier, déboguer et optimiser la performance de l'algorithme.

Ce document explore en profondeur le problème de l'architecture d'un groupe de véhicules en tenant compte de l'expérience acquise par d'autres chercheurs, de l'évolution technologique apparente du matériel et des logiciels ainsi que des besoins du milieu auquel les FC seront exposées à l'avenir. Ce rapport donne un aperçu des méthodes fondamentales concernant la coopération et la coordination d'un groupe de véhicules, le contrôle d'un seul véhicule sans pilote et l'infrastructure sous-jacente de systèmes réels et simulés.

Executive summary

Background: Unmanned Ground Vehicle (UGV's) Research and Development within the Autonomous Land Systems (ALS) project will assist the Canadian Forces (CF) in fulfilling their future mandate. The ALS project derives its focus from the Autonomous Intelligent Systems (AIS) activity outlined by the DRDC Technology Investment Strategy (TIS). The TIS defines Autonomous Intelligent Systems as [54]

"...automated or robotic systems that operate and interact in the complex unstructured environments of the future battlespace"

There are five anticipated classes of Unmanned Vehicles (UV): fixed or rotor wing aircraft Unmanned Air Vehicles (UAV); typically tracked, wheeled, legged Unmanned Ground Vehicles (UGV); stationary monitoring Unattended Ground Sensors (UGS); untethered, propeller or buoyancy driven, Unmanned Underwater Vehicles (UUV); and light propeller driven Unmanned Surface Vehicles (USV).

This document explores the depths of the multi-vehicle architecture problem using the past experience of other investigators, the apparent technological evolution of both hardware and software, and the demands of the future CF environment. Considerable research within real and simulated systems has revealed fundamental approaches to multi-vehicle cooperation and coordination, single vehicle autonomous control, and the underlying infrastructure to make these systems possible. The military and industry have produced internet infrastructures that hold promise for future vehicle and multi-vehicle systems, while posing additional engineering design and development problems. Therefore, this document reviews multi-vehicle coordination, single vehicle control, vehicle software infrastructures, and simulation systems, each a critical element in modern multi-vehicle research.

Principal Conclusions: Unmanned Multi-vehicle systems are a relatively new concept with few clear trends in collective command and control mechanisms. Given tenuous battlefield communications, centralized schemes of task allocation appear less attractive while collaborative or negotiation based systems may be more practical. Ultimately a layered deliberative/reactive system with stigmergic and explicit communications may prove the most versatile and robust.

Mainstream research in single vehicle autonomy has swung from logical or deliberative methods through instinctive or reactive methods to settle on a mixture of long term planning and short term instinctive control schemes. Many systems recognize that no single control scheme will adequately capture the full control problem and, therefore, exploit multiple control strategies simultaneously using environmental triggers or longer term plans to engage or mix control schemes. Further, many larger systems draw on multiprocess architectures and network capabilities to achieve satisfactory performance.

Any multi-vehicle infrastructure, particularly within a research environment, must be based on highly modular and extensible software components that supports distributed code development and execution. Subsequent architectures must define, at a high level, how the various software components are linked to create an operational vehicle. Such systems must support real world operations, the recorded data playback, and integration with a simulated environment. Its modular design must encourage engineers and scientists to concentrate on developing algorithms supporting autonomy without sinking into the details of the vehicle implementation.

Military, Industrial, and academic research clearly points towards the Common Object Request Broker Architecture (CORBA) as the favoured communications middleware. The TAO implementation of CORBA implements a real-time version of CORBA boasting impressive performance capabilities. The Miro framework, perhaps the best example in vehicle control is the Miro framework, offers CORBA services tailored to autonomous vehicle applications. The use, extension, and formalization of this framework will significantly assist in the development of autonomous vehicle applications.

Simulation provides multi-vehicle C2 experimental environments, single vehicle tactical simulation and internal software design simulation capabilities to DRDC's multi-vehicle research effort. While simulation systems are generally designed for specific simulation problems, a number of generic or all-inclusive simulation systems provide some of the necessary capabilities. Given that no single simulator supports UAVs, UGVs, UGSs and UUVs, this document proposes to extend the capabilities of the Gazebo 3D simulator to support more UAV types and to include support for UUVs and UGSs. Extending Gazebo is possible due to its open source nature that encourage users to enhance it capabilities.

Significance of Results: For DRDC, any credible multi-vehicle architecture must allow information to easily flow to wherever it is required. This seamless flow of information encourages scalability and extensibility whether it be within a single UV or a multi-vehicle group. With the future battlespace largely undefined, scalability and extensibility is a key requirement to accommodate the co-evolution of technology and doctrine.

As described in this document, the MIRO framework and ACE/TAO foundations exemplify highly modular, extensible, and reusable components with direct research benefits. This component framework will ease installation across DRDC systems and, significantly, will simplify cooperation with other research institutions. Components achieve these goals by defining information sharing standards between processes. Thus the distribution details remain hidden from the researcher, allowing full research effort to focus on increasing vehicle capabilities. The network foundations of these tools are well tailored for the implementation of distributed intelligence. Information flows as easily between independent UVs as it does within a single UV. For example, allowing an UAV to share its bird's eye view of the world with an UGV located on the ground and a UGV to pass its current position to a field of UGSs.

When all of these capabilities are taken together, an open, powerful foundation for UV research and development in both real and simulated worlds becomes apparent. This foundation simplifies the creation and integration of new capabilities, encourages re-use on different platforms. With these tools and techniques, future systems will focus research on the intelligence within and between multiple autonomous unmanned vehicles by intrinsically supporting distributed operations.

G. Broten, S. Monckton, J. Giesbrecht, S. Verret, J. Collier, B. Digney. 2004. Towards Distributed Intelligence. DRDC Suffield TR 2004-287. Defence R&D Canada – Suffield.

Sommaire

Contexte : La recherche et développement au sujet des véhicules terrestres sans pilote appartenant au projet des Systèmes terrestres autonomes (STA) aidera les Forces canadiennes (FC) à remplir leur mandat futur. Le projet STA est articulé autour de l'activité des Systèmes intelligents autonomes (SIA), soulignés par la Stratégie d'investissement technologique (SIT) de RDDC. Ce SIT définit les Systèmes intelligents autonomes comme [54]

« ...systèmes automatisés ou robotisés qui opèrent et interagissent dans les milieux complexes et non structurés des futurs espaces de combats. »

On prévoit cinq catégories de véhicules sans pilote : les véhicules aériens télépilotés (UAV) à ailes fixes ou à voilure tournante; les véhicules terrestres sans pilote (UGV) ordinaires à chenilles, sur roues ou sur jambes; les capteurs au sol isolés (UGS) de surveillance stationnaire ; les véhicules sous-marins sans équipage (UUV) sans ancrage, dirigés par propulsion ou flottaison ainsi que les véhicules sans pilote de surface (USV) dirigés par propulsion légère.

Ce document explore en profondeur le problème de l'architecture d'un groupe de véhicules en tenant compte de l'expérience acquise par d'autres chercheurs, de l'évolution technologique apparente du matériel et des logiciels ainsi que des besoins du milieu auquel les FC seront exposées à l'avenir. Une recherche considérable concernant les systèmes réels et simulés a révélé des méthodes fondamentales à la coopération et à la coordination d'un groupe de véhicules, au contrôle d'un seul véhicule autonome et à l'infrastructure sous-jacente qui rend ces systèmes possibles. L'armée et l'industrie ont produit des infrastructures Internet prometteuses pour les futurs systèmes de véhicules et de groupes de véhicules mais ces infrastructures posent des problèmes additionnels en termes de conception technique et de mise au point. Ce document examine, par conséquent, la coordination d'un groupe de véhicules, le contrôle d'un seul véhicule, les infrastructures de logiciels de véhicules et des systèmes de simulation, chacun de ces éléments étant essentiel à la recherche moderne sur les groupes de véhicules.

Conclusions principales : Les systèmes de groupes de véhicules sans pilote sont un concept relativement nouveau qui évolue clairement vers la commande collective et les mécanismes de contrôle. Étant donné la fragilité des communications sur le champ de bataille, les schémas d'allocation de tâches centralisés semblent moins attirants que ceux des systèmes à base de collaboration ou de négociation plus pratiques. À la limite, un système multidimensionnel délibératif / réactif permettant des communications stigmergiques et explicites peut se montrer plus polyvalent et plus robuste.

La tendance dominante de la recherche au sujet de l'autonomie d'un seul véhicule a pivoté à partir de méthodes logiques ou de délibération vers des méthodes instinctives ou réactives pour s'établir sur un mélange de schémas de contrôle instinctifs de

planification à long et à court terme. Beaucoup de systèmes reconnaissent qu'aucun schéma de contrôle simple ne capture de manière adéquate le problème du contrôle complet et ils exploitent ainsi les stratégies de contrôle polyvalent qui utilisent simultanément les déclencheurs environnementaux ou des plans à long terme pour procéder ou qui mélangent des schémas de contrôle. De plus, pour réaliser une performance satisfaisante, beaucoup de systèmes plus importants s'articulent autour d'architectures multiprocesseurs ayant des capacités de réseau.

Toutes les infrastructures de groupes de véhicules doivent être basées, surtout dans un climat de recherche, sur des composants de logiciels hautement modulaires et extensibles qui soutiennent le développement décentralisé des codes et l'exécution de ces derniers. Les architectures qui en découlent doivent définir, à haut niveau, le lien entre les composants de logiciels variés, afin de créer un véhicule opérationnel. De tels systèmes doivent soutenir des opérations mondiales réelles, relier les données enregistrées et intégrer le tout dans un milieu simulé. Les conceptions modulaires doivent encourager les ingénieurs et les scientifiques à se concentrer sur le développement d'algorithmes qui soutiennent l'autonomie sans se perdre dans les détails de l'implémentation du véhicule.

La recherche militaire, industrielle et académique pointent clairement vers l'architecture CORBA comme le meilleur logiciel médiateur en matière de communication. L'implémentation TAO de CORBA implémente une version en temps réel de CORBA qui prétend avoir des capacités de performance impressionnantes. Le cadriciel Miro, qui est peut-être le meilleur exemple en matière de contrôle de véhicule, offre à CORBA, des services adaptés aux applications des véhicules sans pilote. L'utilisation, l'extension et l'élaboration de ce cadriciel appuieront de manière importante la mise au point des applications de véhicules sans pilote.

La simulation fournit à l'effort en recherche sur les groupes de véhicules de RDDC, un milieu expérimental pour le groupe de véhicules C2. Elle fournit aussi des capacités de simulation tactique pour un seul véhicule et des capacités internes de conception de logiciel de simulation. Alors que les systèmes de simulation sont conçus en général pour des problèmes de simulation spécifiques, un certain nombre de systèmes de simulation génériques ou globaux apportent quelques-unes des capacités nécessaires. Étant donné qu'aucun simulateur ne soutient les UAV, UGV, UGS ni les UUV, ce document propose d'étendre les capacités du simulateur Gazebo 3D pour soutenir un plus grand nombre de types de véhicules aériens télépilotes et pour inclure le soutien aux UUV et UGS. Étendre Gazebo est possible du fait de la nature de sa source qui est non secrète, ce qui encourage les utilisateurs à améliorer ses capacités.

La portée des résultats : En ce qui concerne RDDC, toute architecture crédible de groupes de véhicules doit permettre le libre flot de l'information partout où celui-ci est requis. Le flot non interrompu de l'information encourage la variabilité dimensionnelle et l'extensibilité à l'intérieur d'un seul véhicule sans pilote comme à l'intérieur d'un groupe de véhicules. L'avenir des espaces de combat étant surtout

indéfini, la variabilité dimensionnelle et l'extensibilité est un besoin clé qui permettra de réaliser simultanément l'évolution de la technologie et de la doctrine.

Tel que décrit dans ce document, le cadriceil MIRO et les fondations ACE/TAO sont l'exemple de composants hautement modulaires, extensibles et réutilisables qui bénéficient directement la recherche. Ce cadriceil de composants facilitera l'installation de systèmes à RDDC et simplifiera de manière importante la coopération entre les établissements de recherche. Les composants atteignent ces buts en définissant les normes d'échange de l'information entre les processus. Les détails de la répartition demeurent cachés au chercheur, permettant ainsi à l'effort complet de recherche de se concentrer sur l'augmentation des capacités des véhicules. L'infrastructure du réseau de ces outils est bien adaptée à l'implémentation de l'intelligence répartie. L'information est transmise aussi facilement entre les véhicules sans pilote qu'à l'intérieur d'un seul véhicule. Elle permet à un véhicule aérien télépiloté, par exemple, de partager sa vue à vol d'oiseau du monde avec un UGV placé au sol et avec un UGV qui transmettra sa position actuelle à un champ de UGS.

Quand toutes ces capacités sont mises ensemble, une fondation ouverte et puissante devient apparente pour la recherche et le développement dans le domaine des contextes réels ou simulés. Cette fondation simplifie la création et l'intégration de nouvelles capacités et encourage la réutilisation de différentes plates-formes. Au moyen de ces outils et de ces techniques, les systèmes futurs se concentreront sur la recherche et l'intelligence à l'intérieur et entre les groupes de véhicules sans pilote et soutiendront intrinsèquement les opérations réparties.

G. Broten, S. Monckton, J. Giesbrecht, S. Verret, J. Collier, B. Digney. 2004. Towards Distributed Intelligence. DRDC Suffield TR 2004-287. R & D pour la défense Canada – Suffield.

Table of contents

Abstract	i
Resume	ii
Executive Summary	iii
Sommaire	vi
Table of contents	ix
List of figures	xiv
List of tables	xv
1. Introduction	1
2. Multi-Robot Architectures	2
2.1 Introduction	2
2.2 Background	3
2.2.1 Seminal Research	3
2.2.2 System Architectures	3
2.2.3 Control Strategies	6
2.2.4 Communication	7
2.3 Discussion	8
2.4 Recommendations/Conclusions	8
3. Control Architectures	9
3.1 Introduction	9
3.2 Background	10
3.2.1 Raibert's Hopping Robot	11
3.2.2 Andersson's Ping-Pong Player	12
3.2.3 Task Control Architecture (TCA)	12
3.2.4 Distributed Architecture for Mobile Navigation (DAMN)	13

3.2.5	Motor Schemas	14
3.2.6	Subsumption Architecture	14
3.2.7	Reynold's Boids	16
3.2.8	Sensor-Actuator Networks	16
3.3	Discussion	17
3.3.1	Redundant Control	17
3.3.2	Control Rate Spectrum	17
3.3.3	Model based Control	17
3.3.4	Distributed Processing	18
3.3.5	Arbitration	18
3.4	Recommendations/Conclusions	18
4.	Infrastructure Services	19
4.1	Introduction	19
4.2	Background	19
4.3	Data Flow Patterns	20
4.3.1	Message Based Communications	21
4.3.2	Information Based Communications	21
4.3.3	Conclusions	21
4.4	Communications Middleware	22
4.4.1	Introduction	22
4.4.2	Background	22
4.4.3	IPT - Interprocess Communications Toolkit	23
4.4.4	RTC - Real-Time Communications	23
4.4.5	NML - Neutral Messaging Language	23
4.4.6	NDDS - Network Data Distribution System	24

4.4.7	MPI - Message Passing Interface	24
4.4.8	CORBA - Common Object Resource Broker Architecture . . .	24
4.4.9	IPC - Inter-Process Communications	25
4.4.10	ACE - Adaptive Communications Environment	25
4.4.11	Discussion	26
4.4.12	Conclusions	27
4.5	Frameworks	28
4.5.1	Introduction	28
4.5.2	Background	28
4.5.3	OCP - Open Control Platform	29
4.5.4	MARIE - Mobile and Autonomous Robotics Integration Environment	29
4.5.5	CARMEN - The Carnegie Mellon Navigation Toolkit	30
4.5.6	Orca	30
4.5.7	MIRO	31
4.5.8	Discussion	32
4.5.9	Conclusions	33
4.6	Simulation Environments	33
4.6.1	Introduction	33
4.6.2	Background	33
4.6.3	Unmanned Ground Vehicles	35
4.6.3.1	CARMEN	35
4.6.3.2	MobotSim	35
4.6.3.3	EasyBot	36
4.6.3.4	Webots	36
4.6.3.5	Player/Stage/Gazebo	36

	4.6.3.6	Vortex	37
	4.6.4	Unmanned Air Vehicles	37
	4.6.4.1	Matlab/Simulink	38
	4.6.4.2	RT-LAB UAV Engineering Simulator	39
	4.6.4.3	MUSE	39
	4.6.4.4	CAE STRIVE	39
	4.6.5	Unmanned Underwater Vehicles	40
	4.6.5.1	CADCON	40
	4.6.5.2	DeepC System Simulator	40
	4.6.6	Discussion	40
	4.6.7	Conclusions	41
4.7		Experiences	41
	4.7.1	Introduction	41
	4.7.2	Background	42
	4.7.2.1	Carmen	42
	4.7.2.2	Player/Stage	43
	4.7.2.3	Miro	44
	4.7.3	Discussion	46
	4.7.4	Conclusions	48
4.8		Conclusions	49
5.		System Integration	49
	5.1	Introduction	49
	5.2	CORBA Integration	50
	5.2.1	CORBA Naming Service	50
	5.3	Miro Integration	51

5.3.1	System Initialization	51
5.3.2	Current Services	52
5.3.3	Future Services	53
5.4	Simulation Integration	54
5.4.1	TAO Centric	54
5.4.1.1	Data Playback	54
5.4.1.2	Simulated Environment	54
5.4.2	Player Centric	56
5.4.3	Hybrid	57
5.4.4	Discussion	57
5.4.5	Conclusions	59
5.5	Conclusions	59
6.	Conclusions	60
	References	62
	Annex	70

List of figures

Figure 1. The ALLIANCE Architecture	4
Figure 2. The LAYERED multi-vehicle architecture [96].	5
Figure 3. The CHARON architecture [37]. a) Agent hierarchy diagram. b) Robot-group agent. c) A robot agent consists of estimator, control and hardware interface agents. d) Robot modes within the Controller Top mode.	6
Figure 4. The CAMPOUT architecture [51].	7
Figure 5. Carnegie Mellon’s Task Control Architecture for the Ambler hexapod. Note the centralized planner and distributed reactive controller	13
Figure 6. Carnegie Mellon’s Distributed Architecture for Mobile Navigation for the NAVLAB series of robots. Each behaviour votes on every possible command (e.g. steering radii) and all votes are processed within the arbiter.	13
Figure 7. A subsumption network. Perception (P) drives augmented finite state machines (M _#) to output messages. Suppression nodes substitute horizontal line messages with vertical (tap) messages. Similarly Inhibition nodes disable line messages if a tap message is received.	15
Figure 8. A SAN network example. Note the interconnections sensor (S), hidden (H) nodes and Actuator (A) nodes. Each node has the structure expanded at right. The hidden nodes act as an interactor information mechanism.	16
Figure 9. Carmen System Architecture.	42
Figure 10. Simulations in the Player/Stage Environment	44
Figure 11. Player Software for Segway RMP Trials	44
Figure 12. Robot Control in Segway RMP Trials	45
Figure 13. Data flow for a Miro service	46
Figure 14. CORBA Networking	50
Figure 15. CORBA Name Service	51
Figure 16. TAO Centric using Stage and Gazebo	55
Figure 17. TAO Centric using only Gazebo	56
Figure 18. Player Centric Integration Plan	57

Figure 19. Hybrid Integration	58
---	----

List of tables

Table 1. Data Flow Patterns	21
Table 2. Middleware	22

This page intentionally left blank.

1. Introduction

Unmanned Ground Vehicle (UGV's) Research and Development within the Autonomous Land Systems (ALS) project will assist the Canadian Forces in fulfilling their future mandate. The ALS project derives its focus from the Autonomous Intelligent Systems (AIS) activity outlined by the DRDC Technology Investment Strategy (TIS). The TIS defines Autonomous Intelligent Systems as [54]

“...automated or robotic systems that operate and interact in the complex unstructured environments of the future battlespace”

Automated or robotic systems are subdivided into five classes of Unmanned Vehicles (UV):

1. Unmanned Air Vehicles (UAV): fixed or rotor wing aircraft.
2. Unmanned Ground Vehicle (UGV): typically tracked, wheeled, legged ground vehicles.
3. Unattended Ground Sensors (UGS): stationary monitoring systems.
4. Unmanned Underwater Vehicles (UUV): untethered, propeller or buoyancy driven, submarine vehicles.
5. Unmanned Surface Vehicles (USV): light propeller driven surface marine craft.

This TIS foresees that for these vehicle technologies

“cross-pollination of technology issues such as interoperability and compatibility will be an advantage”[54].

Though the UGV environment is significantly more complex than the environments in which other unmanned craft operate, airspace and open water posing few obstacles for UAVs and UUVs, all unmanned systems share the fundamental problem of interpreting and acting within their environment. Multi-vehicle operations, specifically communication and coordination, add further dimension to the natural environment and complexity to the unmanned system problem.

Effective multi-vehicle autonomy requires rich sensing and precise control, all residing within a flexible extensible software and hardware architecture. As understanding of autonomous perception and control improves, hardware and software will certainly grow and change with the addition, alteration, and removal of systems as technology evolves. This simple engineering constraint of “flexibility” demands that the vehicle architecture, the software and the hardware must be designed for modularity,

portability, and maintainability. In short, the architecture design must be largely immune to structural changes, i.e. a small change to hardware and software must not induce substantial re-engineering.

This document explores the depths of the multi-vehicle architecture problem using the past experience of other investigators, the apparent technological evolution of both hardware and software, and the demands of the future CF environment. Considerable research within real and simulated systems has revealed fundamental approaches to multi-vehicle cooperation and coordination, single vehicle autonomous control, and the underlying infrastructure to make these systems possible. The military and industry have produced internet infrastructures that hold promise for future vehicle and multi-vehicle systems, while posing additional engineering design and development problems. Therefore, this document reviews multi-vehicle coordination, single vehicle control, vehicle software infrastructures, and simulation systems, each a critical element in modern multi-vehicle research.

2. Multi-Robot Architectures

2.1 Introduction

The “ability to learn, adapt and share information between platforms ... to achieve collective intelligence” succinctly defines the necessary capabilities for effective multi-vehicle/multi-agent systems.

A multi-vehicle system must exhibit cooperation amongst its agents in order for any degree of effectiveness to be realized. Cooperation can come in many forms and whether it is stigmergic or explicit, simple or complicated, cooperative behaviours must be inherent to the multi-vehicle system. Cooperative behaviour is a subclass of collective behaviour characterised by cooperation or is a system that, through some underlying mechanism of cooperation, increases the total utility of the system [23].

Collective intelligence or collective behaviour is a layer above cooperative behaviour. Collective intelligence can be thought of as that which overcomes “groupthink” and individual cognitive bias in order to allow a relatively large number of agents to cooperate in one process - leading to reliable action. In this context, it refers to a very rigorous consensus decision making process[42]. The central issue in multi-vehicle control, particularly in the military context is coordination and while multi-vehicle team activity can be orchestrated through a central multi-vehicle controller given high bandwidth and reliable communications, coordination becomes either fragile or simply impractical with more realistic assumptions of intermittent or low bandwidth communication.

In the following sections, we will detail a non-exhaustive review of several multi-vehicle system architectures, control strategies and communication efforts. We will then discuss various methods used in multi-vehicle systems, making loose

recommendations for a system supporting distributed intelligence.

2.2 Background

Almost all research done in multi-vehicle systems has been done since the early 1990's. There are of course the notable exceptions [107, 106], but the multi-vehicle research work today seems to stem off of the seminal multi-vehicle works done in the early 1990's and the behavior-based robotics work done in the middle 1980's by Braitenberg [12], Brooks [19] and Arkin [3].

2.2.1 Seminal Research

Several different task domains have been used to demonstrate multi-vehicle systems. Some of the first research in multi-robot systems came in the foraging/sorting area by Parker [77] and Beckers [9] and was likely fueled by the bio-inspirations provided by Deneubourg [30]. Other tasks such as multi-vehicle communication [7, 67, 29, 34, 104], cooperative transport [71, 65, 64, 68, 101], landmine detection [44, 40], collective building [109, 76, 75], traffic control [23], robot formation [84, 99, 8, 41], collective mining [88] and collective exploration [5] have been researched. Many more surveys on multi-vehicle research have been done and are mentioned here [32, 23, 79, 105]. These systems all used various different architectures and control strategies and in the next two sections we'll discuss these and give a few examples.

2.2.2 System Architectures

In robotic systems there are three basic system architectures. Deliberative systems follow the sense-model-plan-act method and are based mainly on planning; reactive systems follow the behaviour-based approach and have a tight coupling between the systems sensors and actuators; hybrid systems include a mixture of the two. A more detailed discussion about architectures is given in Section 3.. These basic architectures lay the foundation for more complex architectures.

From the three basic architectures, researchers have started to develop more complex architectures that have the ability to negotiate the issues of synchronization, cooperation, coordination, task allocation, communication, etc. under a wide range of environmental conditions. At the very low level, robots must be able to act quickly to dynamic changes in the environment and perform reactive routines in order to accomplish tasks such as obstacle avoidance. These robots must be able to do this while at an intermediate level accomplishing longer term missions, such as route planning and world mapping. Finally, at higher levels robots must be able to coordinate with each other, performing asynchronous tasks like cooperative searches or highly

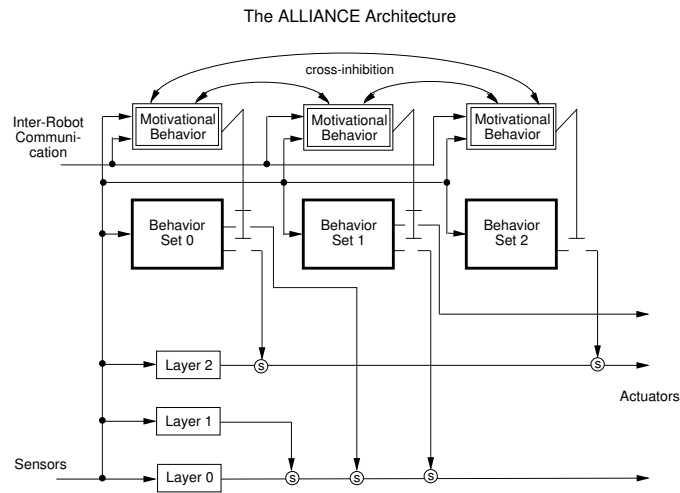


Figure 1: The ALLIANCE Architecture

synchronized tasks like cooperative transportation. Any system architecture that has multi-vehicle systems in mind must allow flexibility between the various levels of the architecture as well as arbitration amongst the various control strategies 2.2.3.

There are several examples of different multi-vehicle specific architectures that have been created. Below we briefly describe four prominent architectures:

- **ALLIANCE:** The ALLIANCE architecture by Parker [78] is a software architecture that facilitates the fault tolerant cooperative control of teams of heterogeneous mobile robots performing missions composed of loosely coupled subtasks that may have ordering dependencies. ALLIANCE is a fully distributed, behaviour-based architecture that incorporates the use of mathematically-modeled motivations [78]. “The ALLIANCE architecture, implemented on each robot in the cooperative team, delineates several behavior sets, each of which correspond to some high-level task-achieving function. The primary mechanism enabling a robot to select a high-level function to activate is the motivational behavior. [78].” A complete description can be found in [78]. Figure 1 shows a graphical representation of the ALLIANCE architecture¹.

¹Implemented on each robot in the cooperative team, delineates several behavior sets, each of which correspond to some high-level task-achieving function. The primary mechanism enabling a robot to select a high-level function to activate is the motivational behavior. The symbols that connect the output of each motivational behavior with the output of its corresponding behavior set (vertical lines with short horizontal bars) indicate that a motivational behavior either allows all or none of the outputs of its behavior set to pass through to the robot’s actuators. The non-bold, single-bordered rectangles correspond to individual layers of competence that are always active.

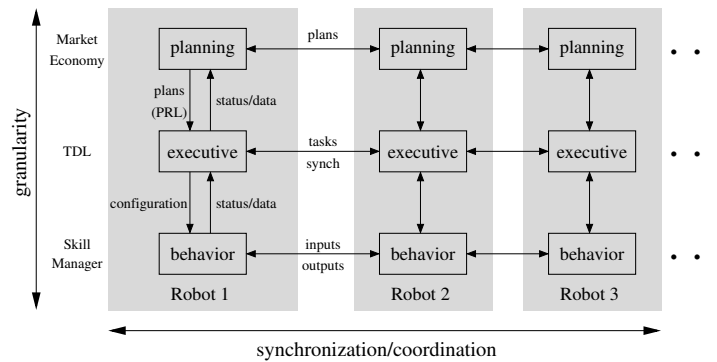


Figure 2: The LAYERED multi-vehicle architecture [96].

- LAYERED ARCHITECTURE:** The Layered Architecture for coordination of mobile robots by Simmons *et al.* [96] is an architecture that enables multiple robots to explicitly coordinate actions at multiple levels of abstraction. Their layered architecture has three layers that enables robots to interact directly at the behavioural level, the executive level and the planning level. This architecture ensures that at all levels the robots utilize coordinated behaviours, coordinated task execution and coordinated planning. Each robot essentially has these three layers and on an individual robot the layers can interchange information while on a robot-to-robot basis the synonymous layers (e.g. the executive layer) talk to each other. Figure 2 shows a graphical representation of this layered architecture.
- CHARON:** The CHARON high-level language describes multi-agent systems [37]. CHARON is a language for modular specification of interacting hybrid systems based on the notions of agents and modes. “For the hierarchical description of the system architecture, CHARON provides the operations of instantiation, hiding, and parallel composition on agents, which can be used to build a complex agent from other agents. For the hierarchical description of the behaviour of an agent, CHARON supports the operations of instantiation and nesting of modes [37].” Figure 3 a, b, c, and d give four graphical representations of the CHARON system.
- CAMPOUT:** The CAMPOUT architecture, designed by Huntsberger *et al.* [51], is an architecture that is able to autonomously adapt to the uncertainties of a dynamic environment. “CAMPOUT is a distributed control architecture based on a multi-agent behaviour-based methodology, wherein higher-level functionality is composed by coordination of more basic behaviours under the downward task decomposition of a multi-agent planner. Basically CAMPOUT provides the infrastructure,

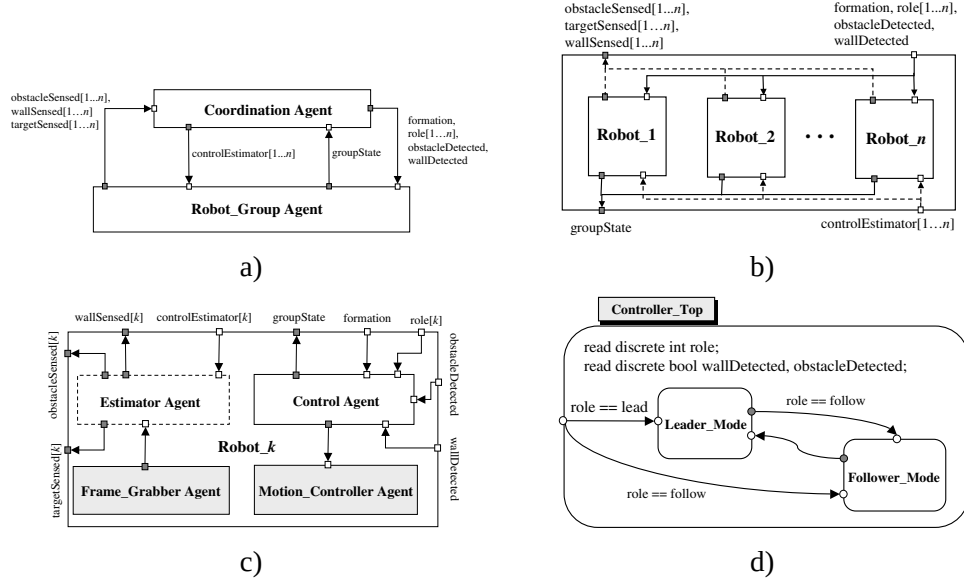


Figure 3: The CHARON architecture [37]. a) Agent hierarchy diagram. b) Robot-group agent. c) A robot agent consists of estimator, control and hardware interface agents. d) Robot modes within the Controller Top mode.

tools and guidelines that consolidate a number of diverse techniques to allow the efficient use and integration of these components for meaningful interaction and operation [51].” CAMPOUT is comprised of five different architectural mechanisms including, behaviour representation, behaviour composition, behaviour coordination, group coordination and communication behaviours. A schematic overview is shown in Figure 4.

- **OTHER:** The above architectures are but a few of the complex architectures that have been developed strictly for multi-vehicle systems. However there are many other architectures [22, 24, 6, 73, 108, 89, 97, 72, 56, 110, 61, 53] that were not reviewed for this document.

2.2.3 Control Strategies

In multi-vehicle systems there are two different high-level control strategies: centralized and decentralized. Centralized systems have one machine, agent or process that is in control of the entire system. This agent controls all of the other agents in the system. Decentralized systems do not have a central agent that monitors all the agents. There also exists hybrid strategies in which there is a centralized planning system that overlooks the decentralized agents. One can also imagine a centralized control system of centralized teams comprised of decentralized agents.

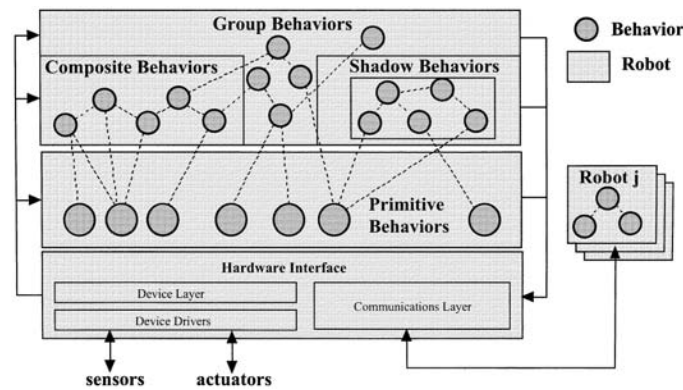


Figure 4: The CAMPOUT architecture [51].

2.2.4 Communication

Communication in multi-vehicle systems has several meanings. In this section we discuss how robots are going to communicate with each other? Other communication issues are concerned with how different processes “talk” to each other or how different modules on a robot share information with each other. These important issues are discussed in Sections 4.4 and 4.5.

Centralized coordination, capable of optimal or near-optimal distribution of unmanned forces, requires reliable and frequent feedback and control to ensure predictable performance. Unfortunately, neither the rate nor quality of communications can guarantee this performance on the battlefield or even in urban settings. Autonomy can compensate for this deficit by inserting greater intelligence into each unmanned system. This reduces the coordination bandwidth and monitoring requirements but at the cost of optimality and predictability. Research must seek out the minimum necessary level of communication that achieves the maximum optimality or, perhaps more realistically, predictability.

In multi-vehicle systems inter-robot communication can be either implicit (stigmergic) or explicit i.e. the robots can either explicitly communicate with each other or they can infer (stigmergic communication) what each other is doing based on other robots actions or how the environment has been changed. When considering explicit communications the major concern of the system is what data should be sent and how should it be sent. In a bandwidth-rich simulation environment living on a high-power CPU sending large data packets packed with information might be acceptable, but in an outdoor wireless environment this luxury may not be available. Instead, care must be taken in data transmissions such that only useful information is sent and that that information will be easily interpreted by its receivers. Robots in

an explicit-communication-poor environment need the ability to interpret the environment and the robots in the environment in order to gain intelligence and make smart decisions i.e. via stigmergic communication, the robot needs the ability to gather information from the dynamic environment. Therefore a need for both stigmergic and explicit communications exist. Stigmergic communications are warranted such that all information about the world doesn't need to be communicated to a robot, nor will it always be available. Also we don't want a multi-vehicle system to come to a halt if there is a temporary or even permanent loss in explicit communications. Finally, there will always be a need for explicit communications for higher level communications such as task allocation, task planning etc.

2.3 Discussion

This section has reviewed of the various systems, technologies and architectures that have been implemented as a result of research in the field of multi-vehicle control and co-operation. Many systems have been developed for different reasons, whether it be fault tolerance, layered systems, tight coupling or just simply coordination, and in the future we plan to take the lessons learned from these systems and improve upon them. One clear lesson learned is that if not properly constructed, multiple-robot systems may actually increase the complexity of an automated solution rather than simplify it[78].

2.4 Recommendations/Conclusions

In many ways, the multi-vehicle environment is reminiscent of the single robot architecture problem, but with the added complexity of intermittent, slow communication, narrow and intermittent sensing, and a vastly more complex potential for activity. The key difficulty of multi-vehicle autonomous control is achieving useful, cooperative behaviour despite these significant problems. Based on our study it is apparent that we should not be employing technologies that are at the extreme end of their technology spectra. Instead, we should use a balanced approach that selects the best ideas, disregards the impractical and implements a function system. Thus, we don't want to use a fully reactive system, nor do we want to use a fully deliberative system. The same can be said for the architecture and communication strategies. As we move towards distributed intelligence we envision a layered deliberative/reactive system with stigmergic and explicit communications. There are many ideas to research in these areas, especially the higher level arbitration areas regarding task allocation, tactics planning etc. However, the first task at hand is the development of an architecture for autonomy for individual robots. This architecture must be portable, scalable and extensible so that it also meets the requirements of multi-vehicle systems.

3. Control Architectures

3.1 Introduction

Robots are dynamic systems, rooted in the physical phenomena of motion, friction, and impact. By applying forces through motors, drive trains and surface features, robots interact with a complex changing environment to achieve prescribed objectives. Less apparent, though, are the internal dynamics of the robot's control logic.

A robot is a device that interacts with the environment in a predictable and desirable way, but only after the observable world is translated into useful action through sensors, logic, and controllers. In effect, a set of 'artificial' constraints on the robot's mechanical system relate sensing to control through a *thread*², or series of processes.

DRDC's architecture for autonomy must accommodate all types of control threads, from teleoperation to autonomous deliberative control. We must assume that there is no 'silver bullet' to autonomous control and that at some point, all control methods will be found wanting.

As discussed earlier, the primary job of a robot controller is to artificially constrain a robot to perform some desirable task. The chain of events that link sensing to action is formally known as a *control loop*³ within a *block diagram* in control theory. Practically speaking, a control thread is composed of a series of steps or, in control theoretic terms, *blocks* that traverse the gap between sensing and control. In increasing level of abstraction, these blocks are usually embodied within⁴:

Transfer-Functions: electromechanical transfer functions, or mathematical descriptions of the transformation of an input variable to an output variable.

Firmware: the low level software embedded into a digital controller that implements control functions.

Functions: one or more program procedures (or methods) compiled into a library,

Hierarchies: one or more class hierarchies, again compiled into a library,

Processes: complete processes that manage the safe initialization, maintenance, and control of an automatic system.

Consider the following approach to the problem. Suppose a robot consists of a group of sensors, s , n actuators, and some goal description (or set-point), r . Then the basic problem is to convert sensing, goals, and actuation into action or, in mathematical form:

²The term *thread* is rarely used in this context, but effectively describes the sometimes tortuous connection between sensing and control that form some SMPA cycles.

³i.e. a control uses sensor feedback to regulate a controlled variable – forming a loop

⁴All of these methods will be encountered within the evolving architecture discussed here.

$$\mathbf{u}_i = h_i(s, \mathbf{r}_i)$$

where $i : 1 \dots n$, $\mathbf{u}_i$ is the control effort (say a voltage) to an actuator from the i^{th} controller, h_i . In short, somehow, control effort is a function of a goal statement and observed state. Since the state space of the 'world' is very large (even infinite), we must limit the 'observed state' to a substantially smaller *subspace* of the observable world. Further, for $\mathbf{u}_i$ to be controllable, the controlled variable must be derivable from the observed state (conditions called *observability* and *controllability*). To be *reachable*, a goal must be described through observable and controllable variables. For example:

A vehicle camera must be pitched +20 degrees in World Coordinates.

Observability: The vehicle must measure elevation and azimuth angles of the camera in *world coordinates*. Note that local camera pan/tilt angles are *insufficient* and require the vehicle pose to produce the correct pan/tilt angles.

Controllability: The vehicle must pan and tilt the camera in world coordinates either through pan/tilt motors or vehicle motion. To command pan/tilt motors to the right positions, a geometric vehicle model is combined with simple camera motor models, possibly modeling only a gear train or pulse/ angle transformation.

Models can be built from device parameters (control variables, device geometry, etc.) to form a large device model. Hence commanding the device into a new state involves the reversal or 'inversion' of this model.⁵ Clearly, the linearity, size, and fidelity of the model will greatly influence the computational complexity and ultimately the thread's control rate. The recurring issue in robot design the determination of quality and necessity of modeling.

3.2 Background

Early AI [17] broke 'intelligence' into four distinct problems: sensing, modeling, planning, and action (SMPA). Modeling and planning, considered the most difficult, were often implemented as *monolithic* processes supported by the relatively simple sensing and action modules. Research focused on symbolic manipulation of the environment and task-level robot programming techniques⁶. These deliberative architectures assumed the environment was completely known to guarantee the robot's behavior.

⁵In vehicles, this model can be as simple as a basic geometric model (e.g. a steering model) to a sophisticated mixture of hyper-spatial (greater than 6dof) geometry of vehicle, terrain, communications environment, etc – whatever can be measured.

⁶Stanford Research Institute's 'Shakey' and STRIPS are examples of task level programming in which logic and planning were of primary interest.

Of course *unstructured environments* (UE) are intrinsically *unpredictable*, making deliberative SMPA threads appear brittle and cumbersome. With the escalation in complexity, UEs either break or slow the MP steps. Without appropriate design, simplified Model-Planners (MPs) face input starvation, lacking sufficient quality or quantity of information. Conversely detailed MPs risk becoming compute bound (lacking computing cycles), a characteristic of the *Frame Problem*. Not surprisingly, purely deliberative systems are often large with relatively slow control rates and are often expensive.

Robotics research diverged from a single deliberative SMPA thread into reactive systems, in the process reducing reliance on the modelling and planning steps (artificial intelligence[70], mobile robotics [15, 14, 16, 17, 18, 26], and high performance manipulation [60, 93, 25]). While capable of robust behavior in high complexity UEs, these systems shared a lack of predictability and provided few useful applications.

Most systems now exploit hybrid strategies composed of both deliberative and reactive control strategies, implemented as a network of complementary control threads.

Robot architecture design has evolved from a philosophical programming problem to a pragmatic systems engineering problem involving multiple computing elements of varying scales, protocols, and capabilities networked into a functional whole.

A practical robot architecture must satisfy the peculiar demands of both hybrid and reactive control methods. Pure deliberative systems are largely linear, moving step by step through the SMPA cycle while reactive systems tend to be composed of multiple, simple MP threads⁷, initially drawing from a common sensor pool and finally committing multiple results to an *arbitrator* prior to Action. A quick review of some examples is instructive:

3.2.1 Raibert's Hopping Robot

Raibert [83] and others [82] have designed robots to explore hopping and leaping dynamics. Raibert's system was controlled through a sequencer, or finite state machine, driven by data streams from pressure, inclinometer, angle and position sensors. By observing incoming data streams the sequencer could coordinate height, velocity, and attitude controllers with the timing of the machine's support and flight phases.

Raibert's hopping robot employed a double acting pneumatic cylinder connected to a pair of pneumatic 'hip' actuator joints, and the entire leg/hip assembly was attached to a large inertia balance beam. By tethering the balance beam to rigid aluminum boom, the robot was constrained to hop within a spherical surface. Two controllers cooperated in the motion of the robot. Each phase of the hopping sequence was triggered by specific sensor

⁷or μ plan , a scripted action triggered by sensing.

thresholds determined through experimentation and analysis. The modeling portion of each phase was limited to estimating the dynamics of the hopping frame, while the planning portion used the dynamic model to plan an angle/thrust response.

Raibert's robot is significant for its use of a combination of dynamic analysis, control, and finite state strategy sequencing rather than an SMPA like planning of the robots running stride. Quoting from [83]:

“The back and forth motions were not explicitly programmed, but resulted from interactions between the velocity controller that operated during flight and the attitude controller that operated during stance.”

3.2.2 Andersson's Ping-Pong Player

Andersson's Ping Pong player [2] used an 'expert controller', an hybrid between an expert system and a controller. Andersson employed a *figure of merit* system to trigger activity within the expert controller. Through an analysis of the ping pong ball dynamics, Andersson identified a set of 'free variables' upon which the ping pong task was dependent including: paddle orientation, ball velocity, manipulator settle time and others. In each control cycle these values would be run, in parallel, through a set of simple models, each generating a figure of merit. The model returning the highest figure of merit would be executed as the next task in the system. In effect, Andersson condensed the model-plan portion of the cycle into a set of parallel processes that simultaneously examined the free variable stream and developed correspondence measures.

3.2.3 Task Control Architecture (TCA)

The Task Control Architecture, shown in Figure 5 is a product of CMU's Robotic Institute, was one of the first architectures to try to unite deliberative and reactive systems.

The CMU Robotic Institute's Task Control Architecture (TCA,) in figure 5 was one of the first to unite deliberative and reactive systems. TCA acts as a planner/overseer on top of task specific reactive systems. Perhaps the most well known implementation of TCA is on the Ambler hexapod. After initially using an SMPA cycle for Carnegie Mellon's six legged robot, 'Ambler's' Task Control Architecture (TCA)[94, 95] was modified into an asynchronous reactive layer combined with traditional AI modeling and planning elements.

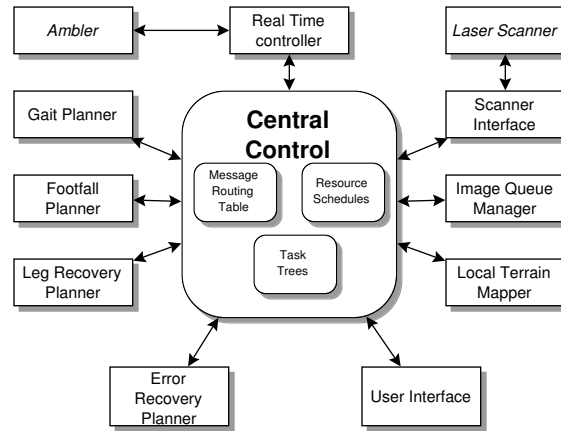


Figure 5: Carnegie Mellon's Task Control Architecture for the Ambler hexapod. Note the centralized planner and distributed reactive controller

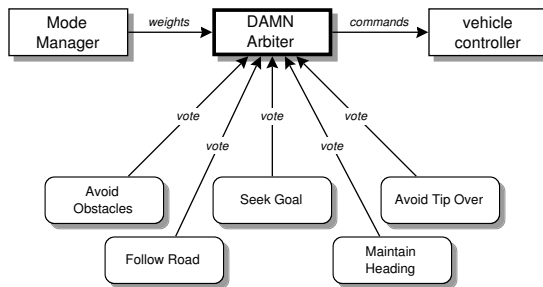


Figure 6: Carnegie Mellon's Distributed Architecture for Mobile Navigation for the NAVLAB series of robots. Each behaviour votes on every possible command (e.g. steering radii) and all votes are processed within the arbiter.

3.2.4 Distributed Architecture for Mobile Navigation (DAMN)

The DAMN architecture, depicted in Figure 6, also developed at CMU, also sought to integrate deliberative planning with reactive control. In DAMN, a discrete set of control actions on a group of actuators (e.g. pan/tilt camera, vehicle steering motors, engine speed) forms a command space in which multiple modules concurrently share robot control. By voting for or against alternatives in the command space, each module contributes to the control commands for the robot. DAMN employs an arbitrator for the resolution of the voting process on each device. In the case of an UGV project command space could be: a turn arbiter, a speed arbiter, and a 'field of regard' arbiter. To explain the arbitration process the turn arbitration procedure will be described:

Each behaviour votes between -1 and $+1$ on every member of a discretized set of radii, R_{0i} . This means that each behaviour's vote is actually a distribution over all the possible steering radii. The arbiter collects vote

distributions from all participating behaviours, performs a gaussian smoothing on each, followed by a ‘normalized weighted sum’ for each of the i radii candidates:

$$(1) \quad v_i = \frac{\sum_j w_j v_i^j}{\sum_j w_j}$$

where w_j is a behaviour weight and v_i is the vote for the j th behaviour. The radius with the highest vote $v = \max(v_i)$, is sent to the controller. ‘Field of regard’ and velocity arbiters perform similar smoothing and selection operations. This approach allows for multiple modules operating at multiple frequencies to vote on various command spaces. DAMN runs on a number of platforms [58, 86, 85]. Rosenblatt explored a number of alternatives to the turn arbiter including more elaborate path and path with prediction arbiters that exploited progressively more detailed vehicle models.

3.2.5 Motor Schemas

Motor schemas [3, 4] are small processes that correspond to primitive behaviours that, when combined with other motor schemas, yield more complex behaviour. Arkin employed two kinds of schema, perceptual schema, that observed and represented the environment through sensing and potential field models respectively and motor schema that devised responses to classes of events. A central MOVE-TO-GOAL or MOVE-AHEAD schema sums the responses and commands the robot motors.

Thus if a FIND-OBSTACLE schema detected an obstacle, an AVOID-OBSTACLE schema was instantiated that produced a velocity vector based upon a repellent potential field around the obstacle (similar to Khatib [59]). By summing the output velocity vectors from a collection of such schemas, a MOVE-ROBOT could navigate through the environment. Motor schemas continue to this day within the MissionLab software architecture (founded on IPT discussed in Section 4.4.3)

3.2.6 Subsumption Architecture

Physically, subsumption is a hierarchical network of simple sensors, controllers, and actuators that can be embedded into relatively small robots. In Ferrell’s 14 inch 3 kilogram hexapod, 19 degrees of freedom were controlled through 100 sensors, including leg mounted foil force sensors, joint angle and velocity sensors, foot contact sensors, and an inclinometer. Applications include aircraft flight and landing systems [49], heterarchical subsumption (Connell’s *Herbert*[26]), and hexapod motion (Ferrell’s *Hannibal*[36]).

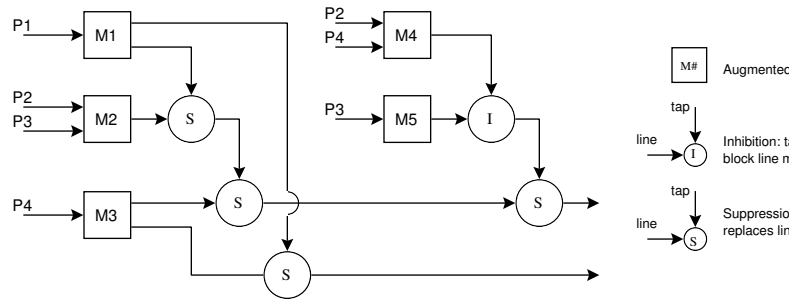


Figure 7: A subsumption network. Perception (P) drives augmented finite state machines ($M\#$) to output messages. Suppression nodes substitute horizontal line messages with vertical (tap) messages. Similarly Inhibition nodes disable line messages if a tap message is received.

Mataric's [69] *Nerds* showed that behaviour arbitration could be learned through repeated trials.

Each subsumption network node is an *augmented finite state machine* (AFSM), consisting of registers, a combinatorial network, an alarm clock, a regular finite state machine, and an output. Sensors are connected to specific registers while actuators receive commands from the output of specific AFSMs. A message arriving at a register or an expired timer can trigger the AFSM into one of three states: wait, branch, or combine register contents. Results of combinatorial operations may be sent to an input register or an output port. Since each AFSM uses an internal clock, output messages can decay over time.

AFSMs can inhibit inputs and suppress outputs of other AFSMs through inhibition and suppression 'side taps' placed on input or output connections in the network. Inhibition side taps prevent transmission of original messages along an input connection if an inhibition message has been received from an AFSM. When a suppression message is sent to a suppression side tap from an AFSM, the original output message is substituted by messages from the AFSM.

Inhibition and suppression side taps encourage layered subsumption (as in Figure 7) in which basic behaviours are embodied within a fundamental layer of AFSMs. Through judicious use of side taps, additional behaviours can be built over the basic set (e.g. 'leg down', 'walk', 'prowl'). Mataric developed a set qualitative criteria to aid in the selection of basic behaviours. Each behaviour should be: *Simple*, *Local* through local rules and sensors, *Correct* by provably attaining the desired objective, *Stable* through insensitivity to perturbations, *Repeatable*, *Robust* by tolerating bounded sensor or actuator error, and *Scalable* by scaling well with group size.

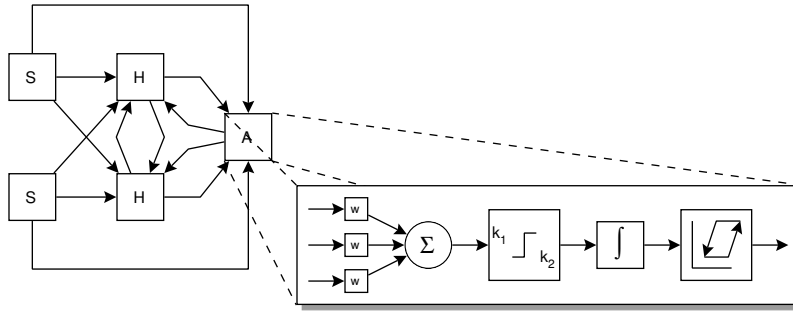


Figure 8: A SAN network example. Note the interconnections sensor (S), hidden (H) nodes and Actuator (A) nodes. Each node has the structure expanded at right. The hidden nodes act as an interactuator information mechanism.

3.2.7 Reynold's Boids

The realistic animation of large collections of entities e.g. crowds of people, schools of fish, etc. becomes time consuming and inflexible if the trajectories of each entity are specified a priori. In a novel solution, Reynolds [84] employed a set of three behaviours: collision avoidance, velocity matching, and flock centering to model formation flying within every bird-like graphical construct, bird-oids or boids. Each behaviour generated an *acceleration vector* and was placed in a priority list.

As each behaviour contributed a desired acceleration to the arbitrator, the arbitrator would accumulate both the acceleration vectors and the magnitudes of each output behaviour over time in order of priority. When the sum of the accumulated magnitudes exceeded a fixed acceleration value, the acceleration vector components would be apportioned to each behaviour in priority. Under normal flight conditions in which each behaviour is of approximately equal priority, this is equivalent to vector averaging. However, if one behaviour experiences an emergency and issues large magnitude vectors, this method effectively suppresses lower priority requests. Similar strategies were used by Terzopoulos [100] to produce realistic behaviour within more sophisticated animated fish.

3.2.8 Sensor-Actuator Networks

The realistic animation of complex running or galloping entities, like the real world equivalent, is difficult to coordinate and control. Van De Panne and Fiume tackled this problem through the creation of Sensor Actuator Networks (SAN) [103]. Given a mechanical configuration and the location of binary sensors and PD actuators on the mechanism, a generate-and-test evolution method modulates weights connecting *sensor nodes*, *hidden nodes*, and *actuator nodes* of the network. In a complex information exchange, all sensor nodes are linked to all hidden and actuator nodes, all hidden nodes to one

another and actuator nodes, and all actuators nodes to all sensors and hidden nodes. Each link is unidirectional and weighted. Within each node, a weighted sum is performed on all connections, thresholded, integrated, and filtered through a simple hysteresis function. The SAN structure is depicted in figure 8.

3.3 Discussion

Some lessons follow from the foregoing review of architectures

3.3.1 Redundant Control

The persistence and quality of control threads cannot be guaranteed under all circumstances. Therefore:

A single control thread will not be sufficient for all circumstances.

3.3.2 Control Rate Spectrum

Years of trial and error implies that autonomous vehicle's control system must contain high rate controls focused on vehicle survivability over short time horizons. These lightweight processes have little or no explicit model/planning steps but little flexibility. Conversely, more deliberative threads are often model dependent and slower.

One or more control threads will run within a spectrum of control rates (e.g. from 1000 to .1Hz) .

It is important to recognize that while a single control thread may have one particularly slow process, there is no necessity for the thread to adhere to that control rate. This is particularly true if a module works on the output of multiple modules (e.g. a data fusion engine that draws from a stereo camera such as a Digiclops at 30Hz and a nodding laser rangefinder such as a SICK at 0.5Hz need not function at only 0.5 Hz).

3.3.3 Model based Control

In some cases model based control is crucial and, in others, unnecessary, depending on the task. Some tasks, such as configuration for high mobility vehicles, may require either a simple geometric model or more descriptive force and torque models to ensure adequate control. Ironically, as performance requirements (such as velocity and acceleration) increase, sensing becomes harder and the necessity for dynamic models becomes unavoidable for stable control. Short range obstacle avoidance may benefit

from a modest model/path planning steps, while long range planning is utterly dependent on both models and plans.

One or more control threads may contain modeling and planning steps.

3.3.4 Distributed Processing

For many processes or even whole threads, executing remotely on a dedicated computing device may be the only means of ensuring sufficient resources to perform at the necessary control rate. At the very least, many modules will draw on remote resources such as SICK or vehicle control components over a network (TCP/IP, firewire, or CANbus). Therefore:

One or more control threads may contain modules operating on remote devices.

3.3.5 Arbitration

Experience has shown that successful control architectures exploit multiple control strategies, breaking a complex control problem into a set of more tractable parts. These control algorithms often operate simultaneously over a network-like architecture. Obviously only one strategy can be applied to the available actuators at one time, forcing a decision or arbitration process to resolve conflicts between threads.

Multiple control threads will be combined through arbitration into 'composite' outputs.

With these conclusions in mind, and the previous discussions of software modularity and integration issues, we are now in a position to discuss implementation issues surrounding DRDC's architecture for autonomy.

3.4 Recommendations/Conclusions

The steps of control, philosophy and some seminal robot controllers were reviewed and a number of key conclusions discussed. Of these the most important is the necessity of Redundant Control and Arbitration. How should this be realized? While the foregoing implies a parallel computing architecture, it is equally necessary for separate control threads to share sensing resources.

Over the last decade, the tools that support network based robot control have matured to the point that industrial interprocess communications now form a reliable, high speed mechanism for building distributed controls. The introduction of internet-based industrial networks and the desire to track and mine industrial process performance history has led to the development of net-based real-time (or near real-time) network

control technology. Clearly, both multi-vehicle and single vehicle control will benefit from these technologies if care is taken in their adoption.

It is difficult to select a 'winning' arbitration architecture since few are directly comparable. The most widely tested systems are the pure potential field variants (such as Motor Schema), while the most severely tested is arguably the DAMN architecture. The simplest arbitration strategies tend to resemble Schema-type approaches, using nonlinear gains on simple sensor returns to drive action selection. As these systems evolve they approach the complexity of the DAMN system. The DAMN approach is appealing since it clearly maps the observed world onto the vehicle's actuator space, providing a human readable condensation of all control thread input. Therefore, the current bias of the program is to follow a DAMN-like philosophy in vehicle control.

4. Infrastructure Services

4.1 Introduction

Two distinct approaches for developing autonomous vehicles have been developed. The first approach specifies a reference architecture which is an all encompassing definition of information and data flows within the vehicle. Examples of global architectures include NASREM[1], TCA[95], Dervish[81], CLARAty[53], RAP[38], Xavier[62], Saphira[63] and 3T[11].

The central thesis of the second approach is the field of robotics is so new and young that it is not feasible to define a fixed reference architecture. Instead, it ascribes to an ideology which believes that the architecture for a robot should only provide the infrastructure services. These services simplify and enable the integration of the various components required by an autonomous robot[48]. IPC[94], IPT[46], NML[55], Berra[74], Marie[27], Carmen[87], OCP[111], Miro[102] and Orca[13] are examples of infrastructure services.

4.2 Background

Under the AIS activity, initiatives will research and develop algorithms that will enable the vehicles to operate in an autonomous manner. Given the degree of research that must be conducted to create "intelligence", these algorithms will undergo a continuous cycle of change. A key component to the success of research in this area will be the ability to manage this changing landscape in an efficient manner that allows researchers to easily investigate new avenues of research. The complexity of autonomous vehicles can be minimized through a component based philosophy. Szyeperski defines components as "binary units of independent production, acquisition, and deployment that interact to form a functioning system"[98]. Thus a component is an independent entity that is capable of executing without requiring the services of a complete system and is often considered to be a separate process that runs under its own workspace. Through the use of components a system can be decoupled into its constituent

elements. This decoupling results in portable and modular software that exhibits “plug-and-play” characteristics. A component based framework also supports an approach to code development that is both extensible and scalable. Extensibility means that it is simple to integrate new software and hardware into an existing system, while a scalable system allows for the easy distribution of the processing over several hosts.

Components share information via defined protocols and interfaces. This adherence to protocols and interfaces allows components to be assembled together to create a functional system. The act of communications between components is usually referred to as “Inter-Process Communications” or IPC. A specific IPC implementation is called middleware. A middleware implementation defines a library of standard protocols and interfaces that are available for sending information from one component to another. A methodology for using middleware to implement an unmanned vehicle is called a framework. An overview of middleware for IPC is given in Section 4.4 and Section 4.5 details various robotic framework implementations.

Modular software simplifies the process of porting an algorithm between the various UV platforms. Modularity also assists the process of replacing an existing algorithm with a new or different algorithm. AIS research will encompass the work of many researchers, with most researchers concentrating on a specific area of research such as stereo vision, map creation, etc. These researchers do not need to know the details of a given vehicle implementation nor are they interested in these specific details. Modular software defines a clear interface between each component in an UV, thus minimising the effort required to integrate new components.

4.3 Data Flow Patterns

An unmanned vehicle is a complex system that contains components which must exchange information and data. The manner in which this information and data is shared is characterised by Data Flow Patterns. Researchers have identified numerous data flow patterns that are commonly used in robotic applications. These patterns simplify the development of modular software by facilitating communications between components. Gowdy[47] identifies four common patterns:

- query/response;
- broadcast
- collection
- information synthesis

OROCOS@FAW[90] identified a similar set of data flow patterns:

- send
- query

- auto-update
- event
- configuration

Even though there are numerous ideas on which data flow patterns best serve robotic applications, data flow patterns can be roughly classified into only two large groups, the message based paradigm and the information based paradigm. Table 1 shows how the data flow patterns listed above can be classified into these groups.

Message Based	Information Based
Query/Response	Broadcast
Send	Collection
Query	Information Synthesis
Configuration	Auto-update
	Event

Table 1: Data Flow Patterns

4.3.1 Message Based Communications

Under the message based paradigm data is sent directly from one module to another. This message consists of a header identifying the message type and the data of the message. The message based paradigm is well suited for applications where the modules require a high throughput rate. With this paradigm the two components directly exchange data and there is no need for an intermediary.

4.3.2 Information Based Communications

For the information based paradigm there is no direct interaction between modules. A module produces information anonymously and makes this data available using a mechanism such as a shared buffer or event queue from which the consumer(s) of the information anonymously acquire this information. This paradigm usually requires the presence of an intermediary who facilitates the distribution of the information.

4.3.3 Conclusions

Research in the field of autonomous vehicles requires infrastructure services that encourage the development of modular software using a component based approach. Infrastructure services that support a wide range of Data Flow Patterns are preferable since they allow for an implementation that is best suited for the given circumstances. The following sections describe how Data

Middleware	Usage	Data Flow Pattern
IPT	Unmanned Ground Vehicles	Message based
RTC	Unmanned Ground Vehicles	Message based
NML	NASREM partner, robotics	Information based
NDDS	Laboratory tool	Information based
MPI	Parallel Computing	Messages contain only data
CORBA	Object Oriented Communications Standard	Message and Information based
IPC	Robotics	Message based
ACE	Concurrent Communications	Message and Information based

Table 2: *Middleware*

Flow Patterns influence the development of frameworks and how Data Flow Patterns are influenced and implemented by communications middleware.

4.4 Communications Middleware

4.4.1 Introduction

Toolkits are available that provide the infrastructure for writing modular and portable software code. These toolkits, usually referred to as middleware, offer varying degrees of capabilities and are amenable to a variety of autonomy architectures. This section reviews a variety of middleware toolkits and discusses their application to unmanned vehicle architectures.

4.4.2 Background

Middleware enables the development of modular software by implementing a standard communications protocol that allows for information to be exchanged between components. These processes can be running on the same processor or on a separate processor. This class of middleware often does not directly target UGVs, UAVs and UUVs, but is generic in nature. Some middleware offerings implement additional functionality that extend the capabilities of the toolkit and further increase the modularity of the software by allowing the software to become operating system independent.

The following sections give a more detailed overview of the middleware packages listed in Table 2 and discusses some of their relative advantages and disadvantages. Verret and Broten conducted a survey of IPC toolkit for possible use by the DRDC unmanned vehicle program (to be published). Another good discussion of the relative merits and limitations of selected middleware packages was conducted by Jay Gowdy[47].

4.4.3 IPT - Interprocess Communications Toolkit

IPT middleware uses a message based paradigm to allow processes to exchange data[46]. A message is comprised of two parts, the message type and data. Messages are asynchronous in nature and are well suited for event driven processes. IPT implements a server that acts as an agent which allows processes to register services and search for registered services. Hence, a process consuming data or events can locate the process creating the data or events via a simple name search. Once the processes have been located they can then use point-to-point communications to exchange data or events directly with each other. IPT is targeted for non-realtime Unix implementation. It uses Unix domain sockets and TCP/IP sockets as the transport mechanisms.

IPT was developed at the Robotics Institute, Carnegie Mellon University, by Jay Gowdy and is used by their unmanned ground vehicle program. It is used at the Robotics Institute and is also the middleware used by the MissionLab from GeorgiaTech[66]. The source code is openly available for IPT.

4.4.4 RTC - Real-Time Communications

Real-Time Communications is a robust, flexible and reliable interprocess communications middleware that provides real-time capabilities[80]. RTC is a middleware that is quite similar to IPT in that it is message based and features a server for registering module names. The major difference between IPT and RTC is that RTC has been optimized for the VxWorks real-time operating system. RTC has the capability to use shared memory on a VME backplane as a transport mechanism, which is a high bandwidth technique for transferring data between modules.

Jorgen Pedersen developed RTC for the Field Robotics Center (FRC) and the Robotics Engineering Consortium (REC). It is used by FRC and REC and a few commercial companies. RTC ascribes to the open source vision and is freely available.

4.4.5 NML - Neutral Messaging Language

NML is a uniform applications interface (API) to communications functions. It includes many popular protocols such as: interprocess shared memory, backplane global memory, and UDP and TCP/IP networking[55]. It follows the information based paradigm. Under this paradigm there is not a direct connection between modules as there is under the message based paradigm. Instead there are information producers and consumers. The producer of information does not care who consumes the information and the consumer of the information doesn't care where the information comes from. The basic

mechanism used by NML for sharing information is a buffer. Information producers put data into a buffer, while the information consumers read data from a buffer. There is no direct method for signaling the presence of new data. This is not required since NML has been designed as a facility for the NASREM architecture which is an exclusively synchronous, time-based robotics system. Under NASREM information is produced and consumed under fixed clock cycles and hence there is no requirement to signal the arrival of new data.

NML was developed by the National Institute of Standards and Technology (NIST) and thus can be considered as an industrial standard. It has a fairly large pool of users and has been ported to a large variety of real-time and non real-time operating systems. NML is freely available from NIST.

4.4.6 NDDS - Network Data Distribution System

NDDS is commercial network middleware from RealTime Innovations that implements a publish-subscribe model[43]. The publish-subscribe model is an information based paradigm where there are information producers and consumers. NDDS provides fast, deterministic data distribution over standard IP networks. This is accomplished via the UDP transport mechanism and a protocol built on top of UDP to ensure the delivery of messages. NDDS was designed as a laboratory tool for the integration of networked instruments. As a commercial product it has been ported to numerous operating systems including VxWorks, Windows and several versions of Unix. Given NDDS' commercial background it has a well staffed support center. NDDS is a closed commercial product and the source code is not available.

4.4.7 MPI - Message Passing Interface

MPI is an interprocess communications toolkit optimized to distribute data for parallel computing problems[39]. Parallel computing utilizes multiple processors, each executing the same algorithm, to solve a single problem. MPI allows the multiple processors to efficiently share data, whether it be a multi-processor super computer or a TCP/IP connection between several workstations. In its native format MPI is not useful for robotic applications since its structure does not contain the tools to send, receive, dispatch and handle named messages.

4.4.8 CORBA - Common Object Resource Broker Architecture

CORBA is object-oriented communications middleware for developing portable distributed applications for heterogeneous systems[50, 10]. CORBA was created by and is maintained by the Object Management Group (OMG). OMG is the world's largest software consortium with the backing of over 800

members. CORBA is an open standard and many implementations of CORBA are freely available. The basic unit of information for CORBA is an object. An object encompasses more than just a message type and data since the object includes methods. The process that obtains a remote object can also invoke operations on the object. While the holder of an object invokes a method, the execution of the method is performed by the object's creator. This seamless integration allows CORBA based applications to execute code at will on other processes and external processors. Standard CORBA has been designed for general software use and therefore does not have specific support for real-time applications.

A real-time version of CORBA, known as TAO⁸, is available which is based on the Adaptive Communications Environment (ACE)[28]. TAO adheres to the Real-time CORBA standard. The RT-CORBA specification defines the standard features that support the end-to-end predictability for operations in fixed-priority CORBA applications[92]. These standard policies and mechanisms permit the specification and enforcement of end-to-end quality of service (QoS) aspects such as bandwidth, latency, jitter and dependability. This in turn allows CORBA to migrate to QoS sensitive domains such as aerospace, telecommunications, medical systems, distributed interactive simulations and robotics.

4.4.9 IPC - Inter-Process Communications

IPC is a toolkit developed for the NASA DS1 New Millennium mission by Reid Simmons[94]. IPC is similar to IPT and RTC since it also subscribes to the message based paradigm. It features the capability to route messages through the IPC server or messages can be exchanged on a peer-to-peer basis. IPC has been used on projects at CMU, NASA and DARPA including: Carmen, DIRA, Skyworker and 3T. The source code for IPC is freely available.

4.4.10 ACE - Adaptive Communications Environment

The Adaptive Communications Environment is a widely-used, open-source, object-oriented middleware written in C++ that implements core concurrency and networking patterns[91] for communications software[52]. ACE is targeted for developers of high-performance, real-time communication services and applications. Like CORBA, ACE is more than just a communications toolkit. It provides a host of capabilities including:

- Concurrency and synchronization
- Interprocess communications

⁸The ACE ORB

- Memory management
- Timers
- Signals
- File system management
- Thread management
- Event demultiplexing and handler dispatching
- Connection establishment and service initialization
- Static and dynamic configuration and reconfiguration of software
- Layered protocol construction and stream-based frameworks
- Distributed communications services such as naming, logging, time synchronization, event routing and network locking etc.

These extra capabilities allow code written using the ACE toolkit to become operating system independent. Presently, ACE supports 22 different operating systems including numerous real-time operating systems. ACE is suitable for real-time operating systems due to its small footprint. The complete ACE library is approximately 800K bytes in size⁹ which could be a significant burden for some real-time systems. Consequently ACE can be broken into its component libraries, each of which implements a specific capability of ACE. These sub-libraries range in size from 10K bytes to 100K bytes.

While ACE is an open source, university based project, it also has the support of numerous other institutions, companies, and funding agencies¹⁰.

4.4.11 Discussion

A high level overview of a selection of communications middleware has been conducted. The complexity of the available middleware toolkits ranges from the simplistic to the complex. All of the reviewed toolkits support numerous non-real time operating systems including Linux.

The IPT, RTC and IPC middleware toolkits have been specifically written to assist in the distribution of data between processes for robotic applications. These middleware applications tend to be small, lightweight and fast and do not have a negative impact on the system performance¹¹, but limit the designer in the manner in which an architecture can be constructed. IPT, RTC

⁹The ACE library size varies with the operating system and compiler used.

¹⁰Funding agencies include DARPA and the NSF. Over 20 companies have supported ACE.

¹¹In terms of memory requirements and real-time applicability.

and IPC only support the message passing paradigm and thus any robotic architecture using these middleware applications must reflect this message passing philosophy. Additionally, these packages are maintained and used by a small group of developers and thus their long term viability of these toolkits could be questionable.

Other middleware toolkits come from a industrial heritage such as NML and NDDS. These middleware toolkits, along with the MPI toolkit, support only the information based paradigm and thus architectures designed around them must adhere to the consumer/producer philosophy. MPI is not useful for robotic applications in its current state. NDDS is a commercial, closed source implementation which is contrary to the open source philosophy of the ALS research.

ACE and CORBA are generic open source middleware applications. These two middleware toolkits support both the message passing and information based paradigm. This gives the user the flexibility in the architecture implementation, since he/she can choose one or the other, or even use both paradigms at the same time. While CORBA and ACE have similar communications capabilities, CORBA incorporates other high level capabilities that ACE does not support. The disadvantages of using generic CORBA include: slow performance, large memory requirements and complexity. The strengths of CORBA are its flexibility and its support for modular and extensible software . The TAO implementation of CORBA addresses the slow performance characteristics of generic CORBA by adding support for real-time operations.

From the philosophical point of view researchers in the field of unmanned vehicles should focus on the problem of implementing autonomy on unmanned vehicles. Their focus should not be on developing middleware for the distribution of data that facilitates the development of modular, flexible and extensible code development. There are already experts in this field who have examined this problem and who have created innovative solutions. Researchers should build upon this existing body of knowledge and expertise to expedite the development of unmanned vehicles.

4.4.12 Conclusions

The unmanned vehicles community has recognized the importance of middleware for the development of autonomous vehicles. Historically, unmanned vehicle researchers have handcrafted all components of their vehicles. The driving force behind this approach was the lack of computing power required to execute the algorithms required by the vehicle. Researchers were keen to harness all the available processing power as efficiently as possible and thus created specialized software. This process is akin to the

early day of computing¹² where algorithms were handcrafted in assembly language in order create highly optimized software. With the advent of faster processors, programmers began to use higher level languages such as C and eventually C++. These same forces are shaping the research field of unmanned vehicles. The availability of more powerful processors and faster networking capabilities has allowed the UV researcher to benefit from the use of middleware toolkits. Just as the use of high level languages allowed software programmers to implement more complex applications, the use of middleware for unmanned vehicles will allow researchers to construct unmanned vehicles that are more complex and thus more capable.

While middleware has been identified as a key component to a successful architecture, a consensus has yet to emerge on specific middleware toolkits. Recent trends in this area have promoted the use of TAO as generic modularity and communications middleware. The TAO combination encourages the development of modular, portable and extensible code, and offers the developer flexibility in architecture implementations. TAO offers more capabilities than currently required by unmanned vehicles, and these extra capabilities should allow them to meet the future demands. These extra capabilities have disadvantages such as large code sizes, poorer performance and a steep learning curve. But it is believed that the advantages of TAO more than offset the disadvantages of its use.

This investigation concludes that the CORBA implementation known as TAO is the most suitable communications middleware for autonomous unmanned vehicle applications.

4.5 Frameworks

4.5.1 Introduction

Frameworks are reusable designs of all or part of a software system described by a set of abstract classes and the way instances of those classes collaborate[57]. It is a reusable design for all or part of a software system and by definition a framework is an object-oriented design. It doesn't have to be implemented in an object-oriented language, though it usually is. The use of a framework simplifies the development of future software since the basic design already exists and it is amenable for reuse.

4.5.2 Background

A variety of frameworks have been specifically developed for intelligent systems and autonomous vehicles. These frameworks are constructed using communications middleware and implement templates for commonly used

¹²Or software written for any computationally limited processor or microprocessor.

communications patterns. These templates do not define the architecture but simplify the implementation of robotic software by hiding the mundane details of implementing data flow patterns. Intelligent systems and autonomous vehicles frameworks are relatively new developments and no single framework has emerged as the dominant choice. The following sections give an overview of frameworks that are available.

4.5.3 OCP - Open Control Platform

The Open Control Platform was developed to implement complex control systems for autonomous vehicles that integrate a variety of different component technologies and resources[111]. It specifically targets extreme performance UAVs, such as the Yamaha R-50/R-max helicopter, the X-cell VTOL UAV, and Boeing's J-UCAS T-33 flying test bed. OCP defines a hierarchy of control system layers that helps control the complexity of the differing time scales among components. This hierarchy defines low level control algorithms, mid-level control components and high-level control components. It used Real-Time CORBA[92] and NDDS[43] as communications middleware to achieve seamless distributed communications between components running on different processors. Two separate communications middleware applications are used since RT-CORBA excels at generality and flexibility, while NDDS specializes in high performance, high bandwidth situations. Work on OCP was initiated at the Georgia Institute of Technology, but has subsequently migrated to Boeing and it does not seem to be freely available for use by other researchers.

4.5.4 MARIE - Mobile and Autonomous Robotics Integration Environment

The goal of MARIE is to create an integrated and coherent framework that facilitates the reuse of applications, tools and programming environments[27]. It uses the mediator design pattern[33] for distributed systems which implements a centralized mediator. The mediator interacts with each application independently, thus facilitating global interactions between independent applications. To facilitate these interactions the mediator acts as a translator between applications and thus must have knowledge of the operation and communications format of each application. This centralized mediator can be viewed as a server through which all interactions between applications (clients) must pass. The core MARIE functional components and communications framework are based upon the Adaptive Communications Environment (ACE)[91, 52]. MARIE is a relatively new project with the first version (0.1) being released in June, 2004¹³. It currently has application adaptors for Player/Stage, CARMEN and RobotFlow/FlowDesigner.

¹³MARIE is available from marie.sourceforge.net and is GPL'd software.

4.5.5 CARMEN - The Carnegie Mellon Navigation Toolkit

CARMEN is an open source collection of robot control software designed to provide a consistent interface and a basic set of primitives for robotics research on a wide variety of commercial robot platforms[87]. It has a modular software design that is influenced by the three-tier architecture popularized by Bonasso et al. [11]. The design goals of CARMEN were ease of use, extensibility, and robustness. These design goals were achieved using the following programming principles:

- Modularity: Software is developed as modules (components) that communicate using IPC[94].
- Simple core modules: The basic primitives of CARMEN include base control, localization, tracking and path planning.
- Separation of control and display: No core CARMEN modules have embedded graphical displays, all information is communicated using standard communications protocols. This helps ensure all information that could be useful is available to all modules.
- Abstract communications: All communications functionality is encapsulated by abstract interfaces, thus allowing for the underlying communications mechanisms to be easily changed.
- Abstract hardware interfaces: CARMEN supports a standard set of interface functions to ensure uniformity of operation across platforms. Thus higher level code can seamlessly command all platform bases in the same manner.
- Standardized coordinates and unit: All units in CARMEN use the International System of Units (SI), with degrees being measured in radians. There are three co-ordinate frames: robot frame of reference, global frame of reference and a map frame of reference.
- Centralized model repository: CARMEN, via IPC, supports a central repository that distributes a consistent set of parameters to all processes.

CARMEN supports a variety of commercial platforms including the Nomadic Technologies' Scout and XR4000, ActivMedia Pioneers, and iRobot's b21 and ATRV series. It also support the following high-level functions: navigation, localization and tracking in 2-D worlds.

4.5.6 Orca

Orca is an open-source framework for developing component-based robotic systems. It is based upon the research conducted by the OROCOS@KTH¹⁴

¹⁴The Centre for Autonomous Systems (Cas) is a research centre at the Kungliga Tekniska Skolan in Stockholm.

project whose mandate was to build a set of communications patterns to allow communications between distributed objects. Orca provides the means for defining and developing the building-blocks which can be pieced together to form arbitrarily complex robotic systems, from single vehicles to distributed sensor networks[13]. The Orca project plans to achieve these goals via the decoupling of components. The decoupling of components is achieved by using CORBA as middleware[10]. The specific CORBA implementation used by the Orca project is ACE/TAO[52]. Orca is new open source project, hosted on Sourceforge¹⁵, that has been in existence since June, 2004. About a dozen or so Orca component modules currently exist, with a majority being at the beta status, while others are alpha or in the planning stage.

4.5.7 MIRO

Miro is a distributed object oriented framework for mobile robot control[31, 102, 35]. It is middleware for robots that reduces software development times and costs by providing often-needed data structures, functionalities, communications protocols and synchronization mechanisms for robots.

A key aspect of Miro is the adherence to object oriented design principles such as information hiding, name spaces, exception handling, type polymorphism and inheritance. The Miro framework focuses on:

- **Open Architecture**
All Miro software is freely available under GPL and thus developers are free to read, change and contribute to the software.
- **Hardware and Operating System Abstraction**
Sensor and actuator subsystems along with other low level services are abstracted via objects and thus provide a uniform interface to these subsystems. This allows higher level code to seamlessly interface with all subsystems.
- **Multi-Platform Support and Interoperability**
The Miro software is built upon the Adaptive Communications Environment (ACE)[52] which support numerous platforms and operating systems.
- **Communications Support and Interoperability**
Communications support is provided by CORBA [10] middleware. CORBA allows for the development of portable distributed applications using object oriented communications facilities. To ensure the

¹⁵<http://orca-robotics.sourceforge.net/>

responsiveness of the complete system a real-time implementation of CORBA, known as TAO [28], is used.

- **Client/Server Systems Design**

To encourage portability Miro encourages the use of the client/server paradigm where modules provide services to other modules via defined interfaces.

When implemented together these features create robot middleware that is scalable, flexible and extensible. The extensibility and flexibility allows it to easily support other robotic platforms beside the currently supported B21, Pioneer 1 and Sparrow platforms. Miro currently provides the following robot middleware services that assist in the distribution of data gathered from devices: motionService, laserService, sonarService, irService, tactileService and imageService. Miro is currently used on soccer playing robots and is freely available from the University of Ulm.

4.5.8 Discussion

Frameworks for intelligent systems and unmanned vehicles are just beginning to emerge and have not yet gained widespread acceptance in the unmanned vehicle community. At present there is not a dominant framework offering for unmanned vehicles. An underlying problem confronting UV frameworks is the lack of agreement in the unmanned vehicle community over the type of communications middleware that is most applicable to unmanned vehicles. While a UV framework standard has yet to emerge, recent years have seen a migration away from the specialized, lightweight robotic middleware such as IPT, IPC and RTC, towards more generic middleware applications of ACE and CORBA.

OCP, is based upon real-time CORBA and has been implemented and tested in UAVs but is not openly available. Although the OCP framework is a proprietary system, its use of CORBA implies it should be possible for any UV based upon Miro to communicate with an OCP based UV. The MARIE open source project, which implements a client/server model using ACE, has only recently been started and it may or may not gain momentum. The client/server implementation of MARIE could lead to bottlenecks that may limit its usefulness for applications that require large amounts of data passing and processing power. CARMEN has been available for over two years and has yet to see a wide scale acceptance. Orca is a new open source framework for component-based robotic systems. It is based upon the use of ACE/TAO (CORBA). The use of the ACE/TAO combination will give the users of Orca tremendous flexibility in implementing unmanned vehicle architectures. Unfortunately Orca is only in the preliminary stages of design and does not yet have much to offer. Miro is an open source framework that is also based

upon ACE/TAO. Thus like Orca, Miro has tremendous flexibility. Unlike Orca, the core Miro software is complete and it has been successfully used on robotic platforms since 1999. While the core of Miro is stable, other parts of Miro are still at the developmental stage.

4.5.9 Conclusions

Frameworks offer significant benefits to developers of UV systems. While it is not guaranteed that the use of frameworks will gain widespread acceptance in the UV community and that standards will emerge, the benefits of using one of the currently available frameworks more than offset the risk of choosing a framework that may not emerge as a dominant standard.

Frameworks based upon the CORBA communications middleware seem to hold the most promise. The review of communications middleware in Section 4.4 concluded CORBA was the best toolkit available and thus frameworks based upon CORBA are viewed as superior to the other non CORBA based frameworks. Three of the reviewed UV frameworks were based upon CORBA and only one of these frameworks supported unmanned ground vehicles.

The ALS program will adapt the Miro framework for use in its current generation of unmanned vehicles.

Additionally, a close watch will be kept on the progress of the Orca project to see if it gains more community acceptance. Given that both Miro and Orca are CORBA based it should be feasible to migrate from one to the other if so desired.

4.6 Simulation Environments

4.6.1 Introduction

In the field of autonomous robotics research, there are cases where real world experiments are not feasible. Operating a group of 100 robots, for instance, is a scenario beyond the means of most laboratories. Testing individual sub-components in noise-free conditions with prescribed input variables and total access to experimental parameters are the major reasons why simulators are employed in autonomous robotic research. Simulated environments help engineers build, test, and understand robot systems in the real world. While simulators do not guarantee success in the real world, they reveal situations where a system may fail.

4.6.2 Background

There are three general simulation environments used in research at this time:

1. 1D Simulations (“playback data”)
2. 2D Simulations
3. 3D Simulations

Simulations not covered in this section are those conducted with single devices, for example, in a testing capacity. While these device simulations test individual devices or algorithms, this discussion of simulation environments will include only environments designed for complete robot system simulation.

1D simulations describe situations where the data from a real world experiment or a theoretical data set is fed back into an algorithm or subcomponent of a program and the output is observed. This level of simulation is meant to observe the response of the program in question without the interaction of other components. This isolated type of experiment can underline problems that may be obscured by other components.

2D simulations refer to a configuration space having x-axis and y-axis components but no z-axis and are generally referred to as Cartesian coordinates. The pose of any robot in this 3-space can be described by the vector below. This vector represents the dimensionality of the configuration space.

$$[x \ y \ \theta]^T$$

In effect, 2D simulations take the world and slice through real space parallel to the ground and assume all objects have the same height. Objects in this 2D world either exist on this plane or do not, denoting the difference between obstacle space and free space. These simulations do not have a high degree of fidelity in comparison to the 3D real world. These simulations do allow for more robots to co-exist in the environment on a given computer since the amount of data space held by each is smaller than a 3D simulated robot. These models can simulate all dynamics in the Cartesian plane but cannot emulate 3D devices.

3D simulations refer to a configuration space having x-axis, y-axis and z-axis components as well as a rotation around each axis. The pose of any robot in this 6-space can be described by the vector below. This vector represents the dimensionality of the configuration space

$$[x \ y \ z \ \phi \ \rho \ \gamma]^T$$

where ϕ , ρ and γ correspond to rotations about the x-axis, y-axis, and z-axis. 3D simulations allow objects to exist at any point in the 6-space as they do in the real world, and allow for a real world model of the environment to be used to simulate every detail of interaction down to viscous friction.

The ALS project has a requirement for all 3 types of simulation environments. The 2D simulators are much less complex, and tend to have no inputs to change the configurable characteristics of the robot such as mass, shape, etc. This simplicity makes 2D simulations well suited to studying group robot behaviours, or for simulation of some indoor environments in which the elevations or overhangs are not a concern. 3D simulators provide a much more adaptable environment for exploring vehicle dynamics, or for perception of and interaction with more realistic environments. This added complexity is computationally intensive and this is not always appropriate for certain types of research.

4.6.3 Unmanned Ground Vehicles

4.6.3.1 CARMEN

CARMEN (Carnegie Mellon Navigation) Toolkit has an organic 2D simulator built into its software package. This simulator uses a virtual robot which operates in a digital map generated from a previous real world test run. The virtual robot can be controlled either by the user with a keyboard/mouse/joystick, or by intelligent autonomous software (ie. CARMEN). While the virtual robot navigates around the pre-loaded map, simulated data is generated and passed to the autonomous software, or fed back to the user as a picture of what the robot sees. As an added feature simulated people can randomly wandering the mapped environment. This simulator has a number of limitations such as being only able to generate one type of data (range to obstacles), having few configurable robot characteristics, simulating only one robot at a time and having no method to control the simulated environment (only simulated pre-existing maps taken from the real world are allowed).

4.6.3.2 MobotSim

MOBOTSIM (Mobile Robot Simulator) is a software package for 2D simulation of differential drive mobile robots. This shareware package provides a graphical interface simplifying the creation and modification of robots and objects. MOBOTSIM has a BASIC Editor to create macros that call specific functions such as: get information about mobots coordinates; get sensor data; and set speed etc. MOBOTSIM supports an unlimited number of mobots

and obstacles, including several obstacle shapes (line, rectangle, round rectangle, arc, ellipse, sector, chord) and free-hand drawings. It also offers simulated range sensors, typically ultrasonic, with the ability to set configuration parameters like radiation cone, range, and misreading percentage. For the MOBOTSIM platform the configurable parameters include platform diameter, wheel diameter, distance between wheels, number of ranging sensors, and angle between sensors. Vehicle dynamics are limited to two wheel differential drive robots.

4.6.3.3 *EasyBot*

Easybot is an universal robot simulator produced at the University of Applied Sciences, Dresden. It operates as a plug-in module for the LightVision3D (LV3D) 3D-modeller. LV3D can create simulated 3D objects and build arbitrary forms of robots with multiple sensors. Dynamic link libraries (DLL's) written in C++ control each robot. For each simulation step Easybot calculates the distance to the nearest object in front of its sensors, and provides these distances and object properties to the controller. Easybot can not detect collisions or model physics. However, if desired, these capabilities can be implemented within user created controller DLLs. For simulated interaction with other dynamic objects, such as a ball in a robot soccer simulation, the user creates these objects as a separate robot with its own controller DLL.

4.6.3.4 *Webots*

Webots is a proprietary 3D simulation program. It contains a rapid prototyping tool for the creation of 3D virtual worlds with physical properties such as mass, joints, friction coefficients, etc. It supports both simple inert objects and active objects (mobile robots). It supports numerous robotic locomotion schemes such as wheeled robots, legged robots, and flying robots. Moreover, the robots can be equipped with a number of sensor and actuator devices, such as distance sensors, motor wheels, cameras, servos, touch sensors, grippers, emitters, receivers, etc. Finally each individual robot can be programmed to exhibit desired behaviors. Webots also contains a number of interfaces to real mobiles robots, so the behaviors developed under simulations can be used to control a real robot.

4.6.3.5 *Player/Stage/Gazebo*

Player/Stage/Gazebo is a package containing a controller and 2D or 3D simulation environments in which playback data may be

incorporated into the simulation if desired. The 2D simulation environment, called Stage, targets low fidelity simulation of robots. Stage simulates a population of mobile robots, sensors, and objects in a two-dimensional bit-mapped environment. Stage was designed with multi-agent systems in mind so it provides fairly simple computationally cheap models of many devices. Stage's simple environment enables the creation and operation of much larger groups than would a similar 3D simulation. Stage supports most devices developed under the Player server, supporting many popular research robots such as the Pioneer II and the Segway.

Gazebo is the 3D simulation environment of the Player/Stage package. Gazebo models robots and 3D terrain using the Open Dynamics Engine, providing higher fidelity robots, albeit in smaller numbers, than Stage. While generating both realistic sensor feedback and physically plausible interactions between objects, Gazebo supports a smaller subset of the devices than Player. However, considerable developmental effort will reduce this deficit over time. Gazebo supports mostly ground vehicles and has support for a single type of air vehicle, a helicopter.

4.6.3.6 Vortex

Vortex is a software package with much more real-world fidelity than the aforementioned simulation systems. Vortex focusses on real-time simulation that accurately models the physics of 3D motion and interaction. It includes rigid body dynamics, accurate collision detection, and comprehensive vehicle dynamics. While the simulation includes motorized joints, springs, and suspension and traction models of wheeled and tracked vehicles, there is no large pre-defined library of existing robots or sensors. In Vortex, sensor simulations will need to be written from scratch.

4.6.4 Unmanned Air Vehicles

There are a number of UAV simulators available, which test a wide spectrum of UAV operations:

- Air vehicle simulation - Air vehicle respond to the air column ,to control surface motion, and power changes.
- Meteorological / Turbulence models - Vehicle response to the air column alone.
- Actuator models - Control surface response.

- Engine / thrust model - Thrust response.
- Payload pointing model - Payload orientation response as a function of a/v dynamics and control inputs.
- Payload optical model - Optical distortion modelling as function of motion and lens/imager characteristics.
- Autopilot model - Control outputs resulting from the time series of control and sensor inputs.
- Attitude sensor models - Sensor response given dynamic motion sequences.
- Payload control model - Payload viewpoint control given dynamic aircraft behaviour.
- Mission management model - Payload and autopilot command modelling for “results” control.
- Scene generators - Scene modelling of payload view given a/v position and payload pointing, including image distortion in the payload optical model.
- Data link models - Error, latency, and distortion models of data link channels.

Most UAV simulations combine the above simulations elements. Many focus on aircraft design issues, largely irrelevant for autonomous control at this point. If the intent is mission rehearsal and operational effectiveness, the required simulator will be different from what is needed to develop an autopilot. Similarly, the development of high level automation strategies will require a simulator with different attributes. Below are a few examples of the types of simulators available:

4.6.4.1 *Matlab/Simulink*

Usually employed in simulating control subsystems for engineering studies, Matlab/Simulink has been used to simulate both complete aircraft systems and simpler UAVs. A significant investment in time would be required to develop a model of the UAV using Matlab/Simulink, and this development time would not contribute to the problem of creating UAV autonomy. However, BAE Systems has used this package to provide simulations for developing a flight control system for a UAV.

4.6.4.2 RT-LAB UAV Engineering Simulator

This simulator is intended to help researchers develop controllers, intelligence or deployment strategies and is based on Simulink. It has software libraries, which contain many components common to UAVs, such as navigation algorithms, propulsion models, flight dynamics, atmospheric models, controls and sensors. Application software uses these libraries to generate C code which can be run as a simulation on a PC. Under this mode it can be used as a stand alone aircraft simulator. It is also available with the RT-LAB hardware simulator, which uses either a pilot operator station, or a hardware in the loop controller via a D/A interface board.

4.6.4.3 MUSE

The Multiple Unified Simulation Environment, developed by the US Joint Technology Center, is an air vehicle and data-link simulation that simulates a tactical control station and a wide variety of manned and unmanned air vehicles using a generic six degree of freedom model to simulate air vehicle performance in autopilot flight mode. It also includes a payload scene generator and payload video control. It offers real-time operator-in-the-loop simulation of multiple systems for the purpose of creating a realistic operational environment. It was assembled from off-the-shelf products. The Multigen Open Flight Database generates the 3D payload scene and the Real-Time Advanced Graphics Environment, from GreyStone Technology, is used for visualization. It can currently simulate such vehicles as the Predator, Hunter, Shadow and Pioneer UAVs.

4.6.4.4 CAE STRIVE

The CAE STRIVE UAV simulator consists of two components. Firstly, a commercial off-the-shelf simulation development environment, with packages for weather, terrain, weapons, sonar, radar, optical sensors such as FLIR and CCD, and communications. The second component is a package for computer generated military forces to simulate a real-time virtual battlefield. The battlefield forces have user-defined behaviours that can interact realistically with the manned or automated systems under simulation. CAE STRIVE uses a C++ API to interface with the libraries. The simulation can be distributed over a network if so required. This simulator is used by DRDC - Ottawa to evaluate the potential applications for UAVs, and was further developed there

into the UAV Research Test Bed, which extended the simulation to model specific characteristics of candidate UAVs and sensors.

4.6.5 Unmanned Underwater Vehicles

There are fewer simulation packages available for UUVs than for other types of unmanned vehicles, because this field is less researched. Below are two examples of AUV simulators:

4.6.5.1 CADCON

The Cooperative UUV Development Concept provides an open and flexible simulation environment. CADCON is a distributed system that allows for complex inter-participant activities and is based on TCP/IP and the Client/Server model. Visualization for this simulator is based on OpenGL, and it can model shapes like torpedoes, open frames, fixed moorings, and acoustic messages. It does not have fidelity to model the vehicle with respect to hydrodynamics, but rather simulates the situation model of the UUVs. It can also simulate teams of UUVs working together. CADCON also contains a topographical sea floor as well as ice cover, thermocline, salinity, water current and inanimate objects.

4.6.5.2 DeepC System Simulator

This simulation system provides simulation of many UUV parameters such as mission planning and monitoring, vehicle data like position, speed and depth. It can simulate the geometric shape of the vehicle, its dynamic underwater behaviour, as well as environmental factors like underwater landscape, flora and fauna, and currents and layers. It also includes a sonar system, and can generate strategies of computerized learning such as the training of neural networks. This system uses a CORBA interface for communications.

4.6.6 Discussion

The most important factor in determining the value of simulations is how well the simulated results compare to real world experiments. It is a general rule that simulations, regardless of fidelity, cannot guarantee comparable real world results. Particularly in the case of mobile robotics research, simulations are not sufficient to reproduce realistic interactions between all objects. This is especially evident during nonlinear dynamics events such as friction and contact/impact, both of which are poorly understood at best and highly dependent on material property estimates. Simulators cannot guarantee

success in the real world, but failure to perform as expected in simulation strongly suggests that the system will fail in the full complexity of reality.

4.6.7 Conclusions

Unfortunately, for the current state of the art in simulation environments, there is no single simulator that simultaneously supports UAVs, UGVs, UGSs and UUVs. While it is possible to use a unique simulator for each unmanned vehicle class, different unmanned vehicle classes operating in different simulated worlds would impose considerable complexity on simulation such as inter-world consistency maintenance, timing, and interaction. However, for existing high fidelity simulations this may be an appropriate solution, as evidenced by the Synthetic Theatre of War (STOW) High Level Architecture (HLA). The United States Office of the Secretary of Defense gave HLA the mandate that new systems as well as several existing modeling and simulation systems adhere to the HLA standards.

A solution may be to extend the capabilities of one simulator to support multiple unmanned vehicle classes. Classes sharing the same simulated world would substantially reduce the implementation complexity. This simulator extension work could be accomplished as a component of the “Unified Approach to Control and Coordination of Unmanned Vehicles Teams in Complex Environments” research¹⁶ into the coordination UAVs, UGVs and UUV in complex environments.

The Gazebo simulator, part of the Player/Stage/Gazebo system, is a possible candidate for this extension since it is an Open Source project. Currently supporting ground and air vehicle platforms, the source code for an Open Source project is freely available and can be extended and modified as required. Player/Stage/Gazebo has a 2D and 3D simulation environment and both simulation environments are controlled by the same application. It can also accept canned data from a file to use with a specific device. These are very powerful features, allowing the experimentation with the same code for 1D, 2D, 3D, and real world simulations. These are features that are desirable attributes for all unmanned simulators, as discussed in Section 4.6.6.

DRDC in conjunction with the open source community will extend and modify the Gazebo simulation environment as required.

4.7 Experiences

4.7.1 Introduction

From our initial experiences with robot architectures, we have been able to gain a large amount of invaluable experience. So far, we have used three

¹⁶This research is funded by a DRDC Technology Investment Fund.

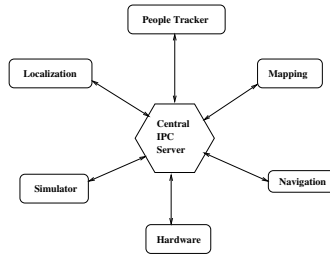


Figure 9: Carmen System Architecture.

toolkits for robot autonomy. Each toolkit has strengths and weaknesses, and from using these toolkits we learned about the problems of creating an architecture for autonomy. This section offers descriptions of the software packages we have used so far, the capacities in which they were used, and the lessons learned from their application.

4.7.2 Background

4.7.2.1 Carmen

Carmen (The Carnegie Mellon Navigation Toolkit) is an open-source collection of robot control software for obstacle avoidance, localization, path planning, people tracking and mapping from 2D laser data. It includes modular C++ programs for robot control, data logging, navigation functions, and also includes a 2D simulator. It is based upon IPC, an inter-process communication toolkit, also from Carnegie Mellon University. Software modularity through the use of IPC is one of its key design principles. IPC also makes the system flexible and robust, by spreading the computation load across a number of distributed processes. Messages are passed between processes in a Publish/Subscribe model, using the IPC server. The overall system reliability is increased through the use of multiple, co-operating processes. If a single module fails it will not halt the system since the rest of the processes are still functional. Carmen is extensible in that it is much easier to add new components when each one is a self contained module. It also increases flexibility in that only the modules which are needed can be loaded. At DRDC - Suffield, we have done some basic simulations with autonomous goal seeking and laser range finder capabilities. A investigation of porting CARMEN to a small 4-wheeled vehicle, the eXtreme Test Bed (XTB) was conducted and this investigation concluded that the effort required to conduct the port was too great.

4.7.2.2 *Player/Stage*

The Player/Stage project is another toolkit for robotic development. It has many similarities to the Carmen project. It is open-source, distributed over TCP/IP, and modular. It uses a central server called Player, and the simulator is called Stage. The Player package consists of three main parts:

1. The server runs on a processor directly connected to the robot. It provides control for the robot and acts as a communications hub for all the devices and controllers in the system.
2. One or more Player clients which connect to the Player server either from the same processor, or from a remote station. The Player client uses a TCP/IP link to issue commands to the robot and retrieve data. A client may be a user interface for tele-operation, software which retrieves data or software that creates autonomous behaviours.
3. Drivers are loaded at run time and provide the interface between specific devices¹⁷ and the Player server. The Player/Stage project also contains drivers for higher level autonomous behaviours. Unlike Player Clients, or Carmen modules, Player Drivers are tied intimately to the server. They are threads of the server process, and must be run on the same processor.

The Player project contains a number of pre-existing software modules such as: a tele-operation programs; obstacle avoidance algorithms; and drivers for controlling physical hardware like the Segway RMP and the SICK laser scanner.

The Player/Stage environment was used in two main capacities at DRDC - Suffield. The first application involved simulating a complete navigation system in the Stage 2D simulator. This included the use of obstacle avoidance algorithms (Vector Field Histogram), global path planning (Wavefront planning), and localization (Adaptive Monte Carlo Localization). These were combined to provide a global “go to” for a simulated mobile robot in a complex building environment. The system structure is shown in Figure 10.

The second application of Player/Stage was in real world trials conducted with the Segway RMP robot. These experiments included: dead-reckoning tests; GPS waypoint following; and

¹⁷Such as the Segway, the SICK laser and the GPS unit.

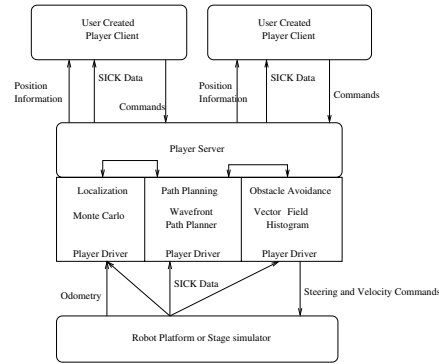


Figure 10: Simulations in the Player/Stage Environment

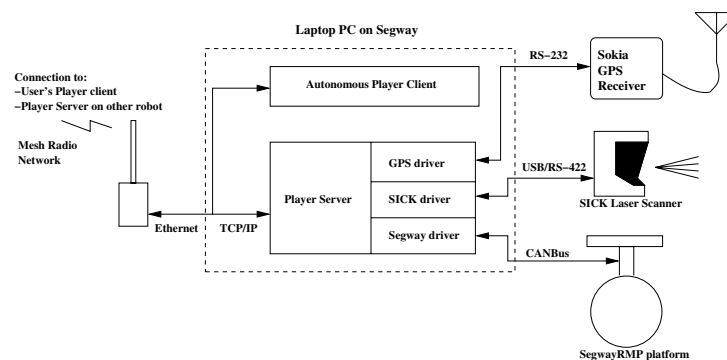


Figure 11: Player Software for Segway RMP Trials

obstacle avoidance of both static and dynamic obstacles. A SICK laser scanner was used for obstacle avoidance. In addition, multi-robot cooperation was attained by allowing the robots to share GPS position information to create leader/follower behaviour. The software control system used on each robot, as well as to provide multi-robot communication was based entirely around Player/Stage and TCP/IP protocols, as shown in Figure 11[45]. The large base of software available in the Player project allowed these demonstrations to be developed rapidly and only minor modifications to the Player drivers were required. The DRDC staff were then able to focus efforts on writing Player client software to create the desired autonomous behaviours.

4.7.2.3 Miro

Miro (Middleware for Robots) is an open source CORBA based object-oriented framework initially developed for RoboCup soccer robots. It has since been expanded to run a number of different

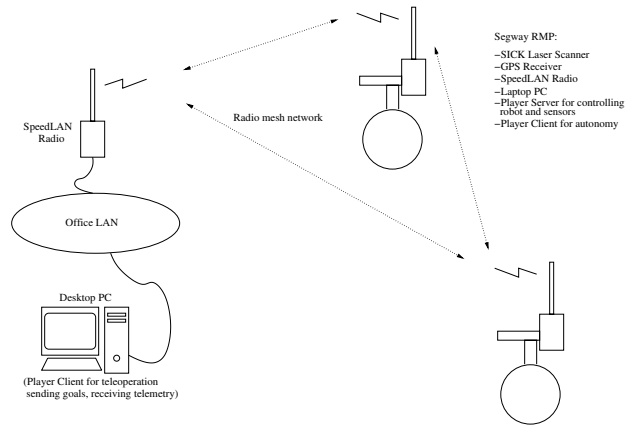


Figure 12: Robot Control in Segway RMP Trials

platforms and support a number of sensor types. In contrast to the Player and Carmen approaches, Miro doesn't push all device data through a centralized server, but instead uses a distributed approach that allows data to flow directly from the devices to the consuming applications. Miro allows multiple client programs to communicate with multiple service programs running on a TCP/IP network through an event driven process. Network name and service resolution is handled via the CORBA "Naming Service" which acts as a telephone book for remote objects. This allows routing of services to clients without the clients explicit knowledge of the network structure. Figure 13 shows the data flow for a low-level hardware device. A device communicates to Miro through its device connection "devConnection". When the device sends information over the "devConnection" an interrupt is generated. This interrupt is sent to a callback routine in "devEventHandler" which is responsible for reading in the message. Once the message has been read in it is dispatched to the "devConsumer" object whose responsibility it is to parse the message and integrate it into the devices interface implementation, "InterfaceImpl". The "InterfaceImpl" then generates a CORBA Event and the data is pushed onto the "Event Channel" where it is distributed to the clients which have subscribed to the "Event Channel". The client programs themselves can operate in polled, event based, or mixed mode. It is important to note the separation between device and interface. This allows enhanced portability by allowing sensors of the same type to use the same interface while shielding the specific details of device communication from the client.

While there are a number of pre-existing software services which

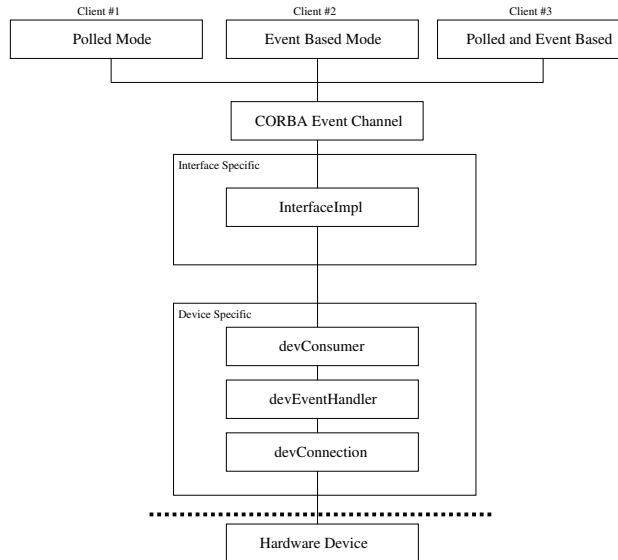


Figure 13: Data flow for a Miro service

exist under Miro, many will need to be written to suit the unique needs of the ALS program. However, when the Miro framework is used, new software consistency and compatibility between software modules is ensured. The Miro framework presents a steep learning curve for the developer, but once it is understood the development of software is straightforward. To date, DRDC-Suffield has developed Miro compliant device drivers for devices such as: an Inertial Measurement Unit (Microstrain 3-DMG); a custom nodding SICK laser scanner with an Ethernet interface; and a Garmin GPS unit. Where applicable, we were able to port code from the Player package into Miro with little effort. The results thus far have been very encouraging. The data throughput rates under Miro are greater than with the Player package. As well, the distributed nature of Miro together with the CORBA Naming Service will allow for a robust system to be developed.

4.7.3 Discussion

We learned a number of lessons about Carmen, Player/Stage, Miro and robot architectures in general, while undertaking these experiments. The most important lesson learned was that the distributed client/server model is very powerful. The model can have many different applications simultaneously connecting to the robot, with each application accessing information or writing the commands. The distributed model makes it easy to communicate across a network and move processes around to where it is most convenient or

where processing power is available. It should be noted that the Carmen central server is only a message passing mechanism, and does not have any robot functionality tied to it. Player is a less flexible design where the device drivers are tied directly to the Player server and this arrangement makes it harder to distribute tasks. In addition, all of the Player drivers must be started at the same time that the server starts, whereas under Carmen processes can be started asynchronously. To further clarify the architectural difference between Player and Carmen, IPC under Carmen uses a Publish/Subscribe model to deliver messages and Player uses a Client/Server model. Under IPC processes communicate directly with each other and only use the server for acquiring connection information. Player uses a Client/Server model in which the central server contains all data, and controls the frequency of data available. This makes the Carmen system more flexible and robust. A work around for the single Player server issue exists that involves starting multiple servers, but this is not an efficient use of resources.

The second lesson learned is that modularity is extremely important, especially during the development process. For example, during our experiments with the Segway RMP, the developers created one software program which would correct the robot odometry based on GPS information. A second piece of software was then created which provided a series of waypoints from the user to the robot. Then a third program was created which would allow one robot to interrogate another robot to get its position. All of these programs were implemented in a modular format and thus can be run concurrently to provide the combined functionality. Additionally, each of the modular software pieces can be optionally loaded at run time to create different robot configurations. For example, it is easy to start the robot with or without GPS localization, or with or without high level path planning.

A third lesson garnered from the experiences with these robotic toolkits is centralized configuration and setup can greatly increase system usability. Centralized configuration means having all of the adjustable parameters for the robot being stored and distributed from one place. Both the Player and Carmen projects use this approach. Carmen has what is called a parameter server which is a separate process that distributes settings to Carmen modules. The parameter server is a repository or registry of values to be used during operation of the robot, such as maximum allowed velocity, robot size, etc. In Player, each time a Player server is loaded, the user must supply a configuration file which tells the server which drivers it needs to load and the specific settings for each of them. For example, the configuration file controls the sensitivity of the obstacle avoidance algorithm and serial data rate for the laser scanner. Using either of these methods a centralized configuration means that once a robot is set up and confirmed operational, it is possible to retain this configuration for future use. It also means that a user need not have any knowledge of the code structure in order to adjust the system performance or

to change its functionality. This run-time loading of parameters can greatly reduce the number of software compiles needed as well. The advantage of the Carmen parameter server system over the Player system is that parameters like maximum speed can be changed during the operation of the robot.

The final lesson learned is that standardization is important. It is vital to create well defined interfaces. An interface simplifies the process of writing software and allows other processes to easily access the data from this new software. This is one area where the Player project is very strong. For example, the position commands sent to the robot are identical, no matter which platform is being used. From the point of view of the autonomous control it is completely transparent as to which type of robot is being controlled, or even if it is a simulated robot. As another example, from the point of view of the rest of the system, every GPS sensor provides the same interface. It becomes almost effortless to move control software from a simulation, to one type of robot, and then to another type of robot. On the other hand, the Carmen project does not have nearly the same degree of standardization in interfaces as does Player. With the number of different hardware devices on a robot, this can become very burdensome. Under Carmen it is difficult to add a new hardware device, or to port the system to a new robot platform. It is necessary to dig through the source code for each separate device in order to see how it was implemented before the device can be controlled.

4.7.4 Conclusions

With both Player and Carmen, there are system design issues which would hamper long term autonomous robot development. Both are steeped in the two dimensional, indoor robot mindset, although, in this aspect Player is less restrictive than Carmen. However, the Player system has no easy mechanism for passing messages between clients whereas Carmen is strong at interprocess communications. Although the pre-defined interfaces available in Player are useful for quickly developing software, they could limit the proposed unmanned vehicle research at DRDC - Suffield. One other problem with Player is that the server execution time can restrict device operation speed. As one example, we have found a practical limit to the data rate available from a SICK laser scanner.

Finally, both packages lack a command arbitration structure. They do not prioritize the commands to the hardware, and no ability to deal with competing priorities. Carmen simply lumps all the navigation functions in one process. Player/Stage uses a hierarchy between the drivers (ie. a planner passes position goals to obstacle avoidance, which passes speed and direction to the robot platform). Both of these leave little room for flexibility in system design.

The experience gained using Carmen and Player has added further

justification for selecting the MIRO framework and its underlying CORBA middleware as infrastructure services for the architecture for autonomy. Miro/CORBA provides the flexibility and message passing capabilities of Carmen/IPC, without the constraints of Player interfaces and the tight server-driver coupling. of Carmen

4.8 Conclusions

The AIS activity at DRDC does not view autonomous vehicles as standard commodities, but instead they are viewed as evolving platforms that are in a state of flux. Thus, the architecture for implementing autonomy must possess the characteristic that will support this unique research and development model. This architecture must be, for any single vehicle architecture, consistent with both the UGV Electronic Hardware Architecture and the UGV Software Development Environment and, further, must not be limited to UGV operations. The infrastructure services described in this section allow for the implementation of an architecture that is inherently modular and distributed. It allows information to easily flow to wherever it is required. This seamless flow of information leads to scalability and extensibility architecture whether it be within a single UV or a multi-vehicle group.

The architecture for autonomy will achieve the desired modularity, scalability and extensibility through the use of the ACE/TAO middleware and the Miro framework. Simulations, at all levels, will be an intrinsic capability for this architecture.

The architecture for autonomy will allow UVs to evolve with changes in their environment, thus meeting the requirements of the future battlespace. The distributed nature of the architecture for autonomy is well tailored for the implementation of distributed intelligence and multi-vehicle operations. Information moves as easily between independent UVs as it does within a single UV. This allows an UAV to share its bird's eye view of the world with an UGV located on the ground and a UGV passed its current position to a group UGSs. This architecture is expected to meet the needs UV research for the foreseeable future.

5. System Integration

5.1 Introduction

This section describes the approach to be used to meet the autonomy requirements that were detailed in the previous sections of this report. Section 1. outlined the architectural requirements for researching, developing and implementing autonomy on unmanned vehicles. It proposed a component based architecture that is highly modular and extensible and supports real world operations, the playback of acquired data and

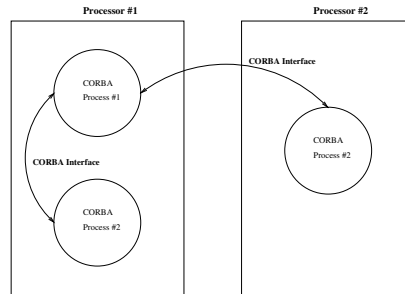


Figure 14: CORBA Networking

simulation environments. Section 4. detailed the infrastructure services required to meet these architectural requirements. These services included a communications middleware toolkit, a framework for developing autonomous vehicles and a simulation environment.

This section details how each of these services is integrated into the overall system to create the architecture for autonomy. An in depth explanation of the role played by each service is given in the following sections.

5.2 CORBA Integration

CORBA was selected as the communications middleware of the architecture for autonomy. It encourages the development of modular, portable and extensible software, thus giving developers flexibility in implementing architectures. A key aspect of CORBA is its ability to make networked operations transparent and seamless. Through the use of the IDL language and compiler, a CORBA object and its data and methods are inherently network aware. Thus any process, whether it is local to the native processor or on an external processor, can acquire a CORBA object and access its data and methods. Figure 14 illustrates CORBA's extensibility.

In the example shown in Figure 14 Process #1, located on processor #1, provides a CORBA compliant interface. When process #2 accesses the CORBA interface of process #1, it is irrelevant whether process #2 resides on processor #1 or processor #2. This ability to move code from processor to processor without changes is one of the great advantages of using CORBA as communications middleware. It gives UV system integrators tremendous flexibility in expanding or contracting the computing power available, as it is required by individual UV platforms.

5.2.1 CORBA Naming Service

CORBA compliant processes, using the TAO implementation, can be moved from processor to processor without the need to recompile code. Additionally, CORBA processes do not require configuration files or the passing of

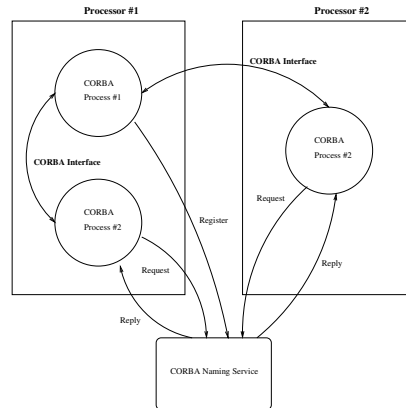


Figure 15: CORBA Name Service

command line arguments. The CORBA Naming Service plays the critical role in enabling this flexibility. Figure 15 shows the operation of the CORBA Naming Service.

A CORBA compliant process that wants to share data with other processes registers its interface with the CORBA Naming Service as an object with a text name. Other processes can access this CORBA object by using the text name to request the object. Once the object has been retrieved it can be used as if it is a local object.

5.3 Miro Integration

Miro is a robotics framework based upon the TAO implementation of CORBA. This framework defines interfaces that are commonly required by robotic applications. The interfaces defined by Miro include: Odometry, Motion, RangeSensor, Stall, Video, PanTilt, Button, Speech and others.

A key aspect of all Miro interfaces is their ability to transform data into abstract information. This transformation is best illuminated via an example. Odometry represents the position of the robot as x, y co-ordinates and heading, regardless of the source of the odometry data. Thus a robot that uses an encoder for odometry, or a robot that uses a GPS would represent position using the identical format. Similarly the RangeSensor interface represents range in a uniform manner whether the range information originates from sonar, infrared, tactile or a laser ranging device.

5.3.1 System Initialization

The MIRO system provides a 'naming service' for each robot. Within a given namespace, the system can provide a number of services that are free to come and go over time. MIRO *services* are typically, but not necessarily, provided

by independent processes residing within the vehicle network. When initialized, each service publishes its existence within the local naming service.

5.3.2 Current Services

MIRO provides a number of object-oriented interfaces that are frequently offered as services or form the foundation of derived (in the OOP sense) services:

- **Odometry:** The odometry service propagates the robots current position and velocities and encapsulates the odometry (dead reckoning) sensor.
- **Rangesensor:** The range sensor interface is the general abstraction of all range sensor devices, such as sonar, laser range finders etc. It provides a method for querying the sensors physical configuration as well as the latest sensor reading.
- **Battery:** A service for interrogating battery level.
- **Sonar:** A service for sonar based devices with crosstalk protection.
- **Infrared:** A service for infrared range sensors.
- **Tactile:** A contact or 'bumper' sensor service.
- **Stall:** Similar to the tactile interface the stall detection monitors the robots motion and detects when the robot is stuck in its movement.
- **Video:** The video service provides camera images at a rate of up to 25 images per second captured by frame grabber cards or IEEE 1394 cameras.
- **Buttons:** For simple user interaction, some robot types provide push buttons.
- **PanTilt:** Cameras often are mounted on top of a pan-tilt unit, allowing the robot to look sideways while moving in another direction.
- **Speech:** To provide a more natural way of communication, some robots are equipped with speech synthesizer cards.
- **Motion:** A generic service providing steering command and control. Often provides the foundation for other Motion classes.

Commonly, many of these interfaces are combined within a single process. However, some of these services become sufficiently complex to warrant an independent process¹⁸. MIRO provides a natural consolidation of these

¹⁸This process could be located on a different processor.

services with a *Base*, a collection of services running within a single process that embodies a single system, typically a vehicle.

5.3.3 Future Services

- GPS: The GPS service propagates an instantaneous DGPS position and heading.
- INS: The INS service propagates an instantaneous orientation, magnetic north, and cartesian accelerations.
- Pose: The pose service propagates a current filtered position and orientation.
- Engine: The engine service provides basic operations and monitoring services for an internal combustion engine.
- Emergency: An alarm service reporting an emergency condition.
- Ackerman: A motion subclass designed to provide an interface to ackerman steered vehicles.
- Map: An abstract superclass used for both local and global mapping services.
- Perception: A fusion service that generates a fused 2.5D map.
- Avoidance: A map-based obstacle avoidance service.
- Routing: A map-based planning service.
- Arbitration: An action-decision service.
- Additional services may include a set of duplicate Gazebo services to provide simulated data feeds from sensor sources.
 - application initialization including application parameter/configuration files
 - interprocess communications
 - error logging for standardized error reporting and retrieval
 - data logging for uniform data formats.
 - watchdog services for module recovery/reinitialization.

5.4 Simulation Integration

A key aspect of the architecture for autonomy is its support for multiple operating environments. An UV must transition between operations: in the real world, on stored playback data, and in a simulated environment. These transitions must occur in a seamless manner without requiring the user to change or recompile existing software. This capability to operate under multiple environments is invaluable for the timely research and development of unmanned vehicles.

The current status of simulation environments was reviewed in Section 4.6. This review has concluded the most capable UGV simulation environment is the Player/Stage/Gazebo simulation toolset.

Three separate approaches have been identified that would facilitate the integration of simulation into the architecture for autonomy. These approaches to integration are referred to as, TAO Centric, Player Centric, and Hybrid. They are differentiated by how external devices are interfaced.

5.4.1 TAO Centric

Figure 17 shows an example of how a selected subset of UGV capabilities could be integrated into the architecture for autonomy using the TAO middleware. Under this configuration all communications within the robot and to external sources are accomplished via defined CORBA interfaces. Intelligent devices such as the SIL SICK Laser could implement CORBA interfaces directly. Devices without intelligent capabilities would be controlled by an interface process which would gather the data from the device using the device's native driver and then make this data available to other modules via a CORBA interface.

5.4.1.1 Data Playback

A common practice that aids in the verification or trouble shooting of an algorithm is to log the raw sensor data to a file. This logged data can then be run back through the algorithm in order to recreate conditions that caused the algorithm problems or verify the correct performance of the algorithm. The playback logged data is applicable to any source of sensor data.

5.4.1.2 Simulated Environment

The use of a simulation environment is useful when algorithms are in the developmental stage. It allows the developer to interactively track down bugs and tune the performance of an algorithm without the need of a complete UV platform. Given that there are a limited number of UV platforms available and many team participants,

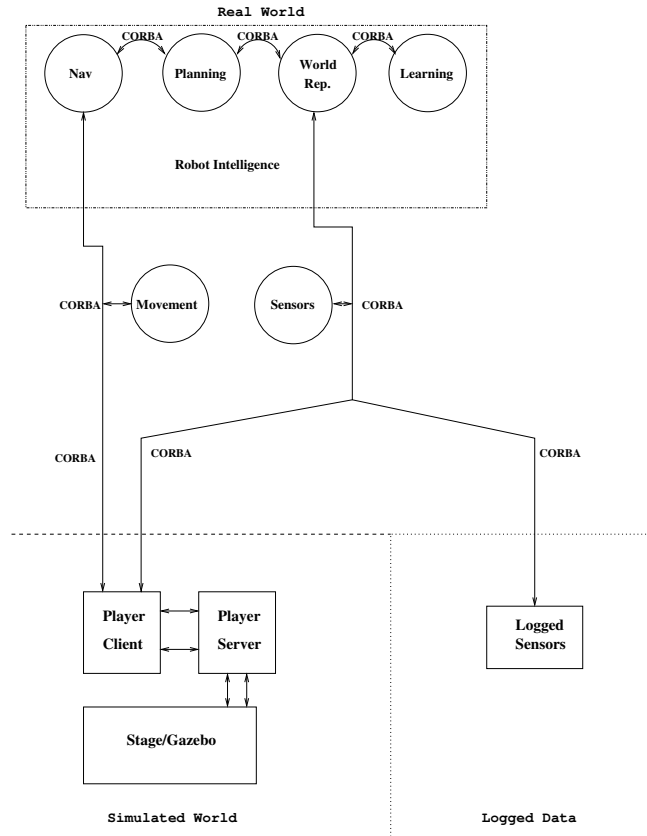


Figure 16: TAO Centric using Stage and Gazebo

this capability is crucial for allowing development to progress at a reasonable rate.

Player/Stage can be integrated into the system by constructing a Player Client that presents CORBA interfaces to UV intelligence modules. The Player Client uses standard Player library calls to acquire simulation data from the Player Server and to command movement within the simulation world. The Player Client registers the appropriate interfaces with the CORBA Naming Service and thus any higher level intelligence module requiring these services can dynamically find and bind to the required interface.

If only 3D simulations are required the TAO Centric approach could also interface directly with the Gazebo. The integration with Gazebo is shown in Figure 17.

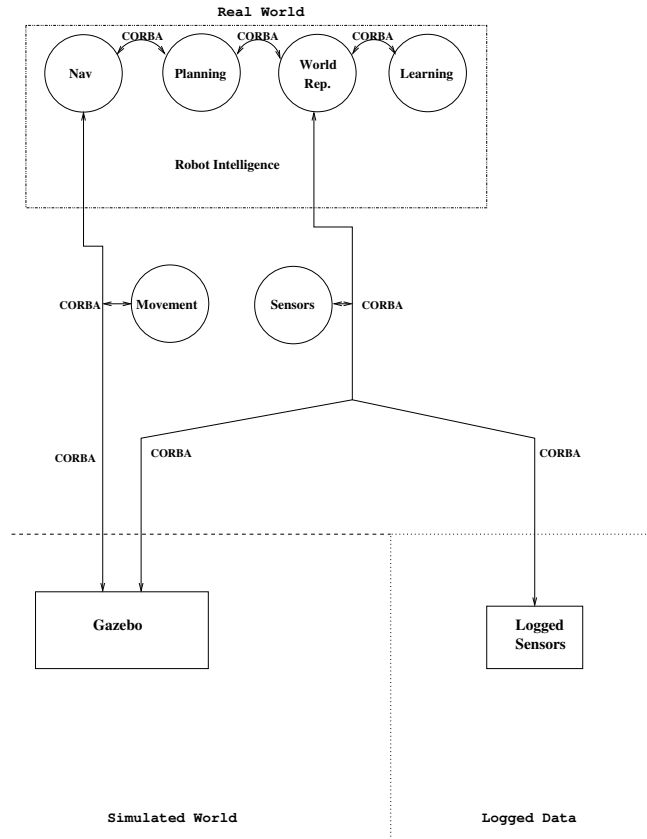


Figure 17: TAO Centric using only Gazebo

5.4.2 Player Centric

A second configuration is called the Player Centric approach. This approach uses the capabilities of the Player/Stage simulation environment as the key tool for integrating external devices into the UV platform. Figure 18 shows the Player Centric configuration. This integration strategy uses the Player Client/Server combination to control and acquire data from devices. These devices could be represented by: a physical device, logged data or a device in a simulated environment. When the Player Server is invoked, a configuration file defines the device mapping to be used. By tailoring the device mapping to suit the desired operation, the Player Server can be instructed to match one of the defined operating modes.

The Player Centric approach has the advantage of being able to leverage the existing device drivers provided by Player/Stage and to use new device drivers as they are developed.

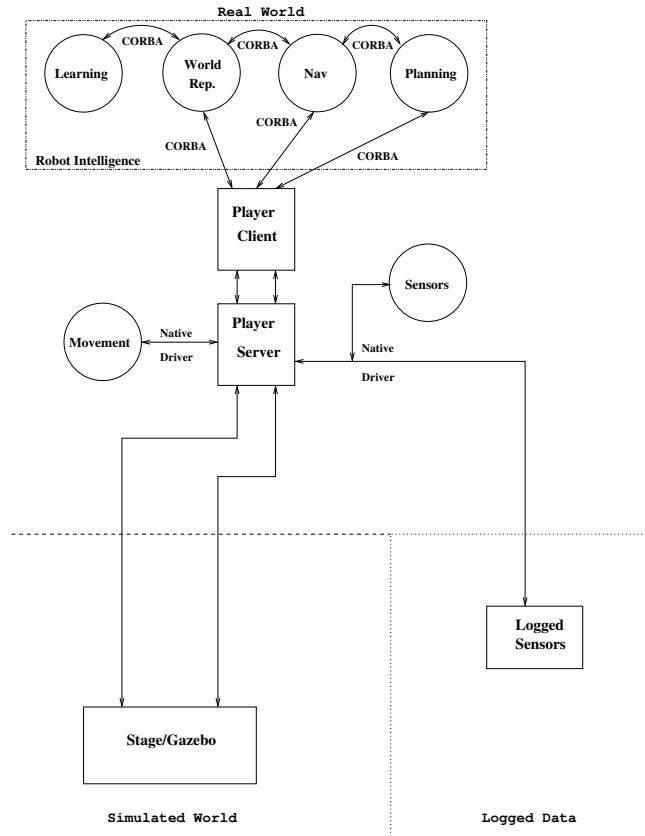


Figure 18: Player Centric Integration Plan

5.4.3 Hybrid

The Hybrid integration strategy is a combination of the TAO Centric and Player Centric approaches. Under the Hybrid approach high throughput devices are not routed through the Player server, but instead have their own separate CORBA interface. Thus devices such as cameras do not negatively impact the performance of the Player server. The interface for low throughput devices remains with the Player Server. This Hybrid approach attempts to use the best parts of the two alternative approaches. With this configuration the ability to use a majority of the existing Player/Stage device drivers is maintained without a performance penalty being applied to devices with high bandwidth requirements. Figure 19 shows a block diagram of the hybrid integration approach.

5.4.4 Discussion

Each of the three integration strategies discussed in the previous section have advantages and disadvantages. Each approach requires the writing or

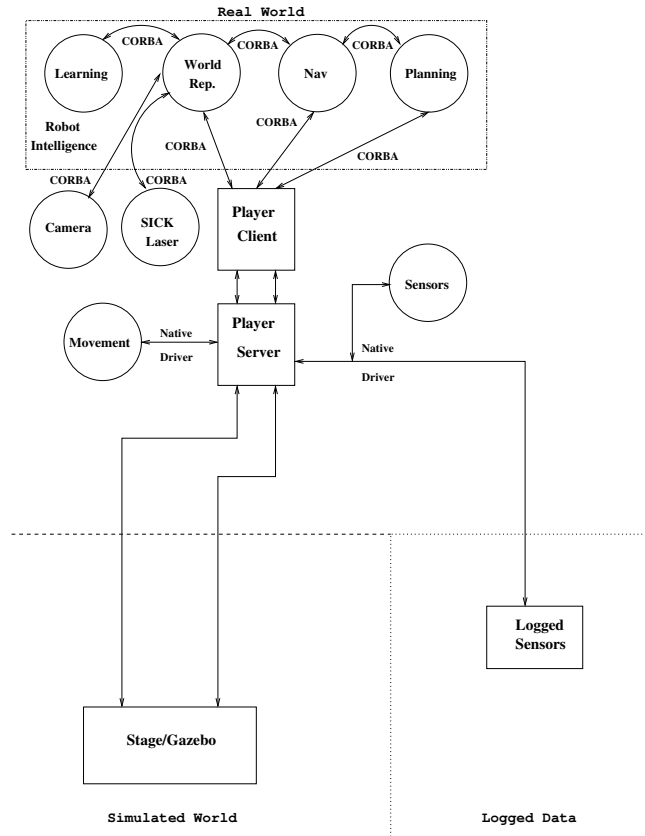


Figure 19: Hybrid Integration

adaptation of drivers and interfaces.

The Player Centric plan builds upon a large set of existing devices drivers. For this approach most of the required drivers already exist. Devices added to Player/Stage by its development community are portable since all drivers follow a common standard. This approach simplifies the integration with the simulated environment since the same TAO to Player interface is used for all devices.

On the other hand, the TAO Centric configuration is better adapted for operations on a real UV. This integration strategy allows individual modules to communicate directly with devices on an “as required” basis. The Player Centric approach forces all device interactions to be routed through the Player server. This centralized routing is a potential bottleneck that could slow the operation of the entire UV and inhibits the assigning of operating priorities to devices. The TAO Centric approach, which allows for peer-to-peer communications, does not suffer from these centralized routing issues. Using the TAO Centric paradigm the software on the UV can be optimized for

real-time operation and thus efficiently use the available processing power. While the TAO Centric approach is optimal for real-time operations, it lacks support for numerous device drivers.

The Hybrid configuration is an attempt to combine the ease of implementation of the Player Centric approach with performance characteristics of the TAO centric approach. Most devices are routed through the Player Server with the exception of high bandwidth devices that use TAO interfaces.

5.4.5 Conclusions

The three integration plans, proposed in the previous sections, have distinct advantages and disadvantages. While it tempting to use the simplistic Player centric approach, the performance penalties of routing all data through a single Player server are unacceptable. The TAO centric approach excels under real-time conditions, which it a critical attribute for UVs operating in the real world, but even with the use of the Miro framework much effort would have to be expended in developing device drivers. The Hybrid approach is a pragmatic implementation. It satisfies the seamless switching between operating environment, enables developers to tap into the large pool of Player device drivers, while helping assure that the robot will be capable of handling real-time, real world conditions.

5.5 Conclusions

The Miro framework is based upon CORBA capabilities and services; thus they are already tightly integrated. The integration with 1D, 2D and 3D simulations is more problematic. As detailed in Section 4.6, current simulators do not support all UV classes. The most suitable simulator is Player/Stage/Gazebo, which supports UGVs and a single UAV platform. The Simulation Environments Section concluded a project might have to be initiated that extends the capability of Player/Stage/Gazebo to support more UAV models and to include support for both UUVs and UGSs. This approach would require adding UV classes to both the Stage 2D simulator and the Gazebo 3D simulator. The modification of both the 2D and 3D simulators could imply a significant work effort. To minimise the effort required an extension to only the Gazebo 3D simulator could be considered. This would entail sacrificing of the 2D simulation capabilities for the simplification of integrating the simulation environment into the architecture for autonomy. The trade offs between eliminating the 2D simulation environment and the effort required to implement this environment must be further investigated before a plan of action can be determined.

6. Conclusions

The report presents a detailed investigation into the issues surrounding distributed intelligence and autonomy in the context of multi-vehicle control, providing a guide to the necessary technologies supporting distributed intelligence. The report has reviewed multi-robot architectures, control architectures for robots, the infrastructures services required by autonomous unmanned vehicles and techniques for integrating all the required services on to an autonomous vehicle. Multi-vehicle control and the distribution of intelligence and capabilities within a vehicle team depend on the ability to easily and transparently communicate between physically remote platforms. Similarly, the internal multi-threaded control architecture of a single autonomous vehicle requires that physically separate computing, sensing, and control elements seamlessly communicate with one another. These two major design constraints were illustrated and clearly show the necessity for an organized industrial strength common standard to both inter and intra-vehicle communication and computing.

To effectively distribute intelligence modules within and between UVs a layered modular hardware design[20] and portable, maintainable coding practice[21] require an architecture that, at once, inherently supports and encourages distributed computing, *and* frees investigators to focus on the development of intelligent single and multi-vehicle control systems. An architecture founded on these elements defines, at a high level, the links between various software components that create an operational vehicle. Ideally, architectures should seamlessly transition between real vehicle control, system diagnosis through the replay of gathered data and the control of a vehicle in a simulated world. Ideally, the investigator is then free to develop intelligence algorithms without vehicle implementation distractions. When satisfied with the simulated performance, the investigator can safely execute algorithms on a physical vehicle. Conversely, with the vehicle operating, data from the environment can be gathered, archived, and replayed within a simulated environment to investigate, debug and optimize the performance of an algorithm.

As described in this document, the MIRO framework and ACE/TAO foundations exemplify highly modular, extensible, and reusable components that offer direct research benefits. Extensible, scalable, distributed, and modular components will ease installation across DRDC systems and, significantly, will simplify cooperation with other research institutions. Components achieve these goals by using defined interfaces to share information between processes¹⁹. These critical interfaces permit sharing of information with other components, inherently allowing information distribution. The distribution details remain hidden from the researcher, allowing full research effort to focus on increasing vehicle capabilities.

Despite the clear advantages of the proposed framework, the introduction of simulation remains problematic. A simulation system that supports all UV classes, including UAVs, UGVs, UGSs and UUVs is desirable, but at this time no single simulator has all

¹⁹Player/Stage/Gazebo exemplifies the research utility of well defined interfaces for lab-scale systems.

these capabilities. However, one solution to this problem is to extend the capabilities of the Gazebo 3D simulator to support more vehicle classes. Currently supporting only ground and air vehicles, Gazebo appears to be extensible to UUVs, USVs, and UGSs.

When all of these capabilities are taken together, an open, powerful foundation for UV research and development in both real and simulated worlds becomes apparent. This foundation simplifies the creation and integration of new capabilities, and encourages software re-use on heterogeneous platforms. With these tools and techniques, future work will inherently support distributed operations that focus research on the intelligence within and between multiple autonomous unmanned vehicles.

References

1. J.S. Albus, R. Lumia, and H. McCain. Hierarchical control of intelligent machines applied to space station telerobots, 1987.
2. R.L. Andersson. Understanding and applying a robot ping-pong players expert controller. In *Proceedings 1989 IEEE Int. Conf. on Robotics and Automation*, pages 1284–1289, 1989.
3. R.C. Arkin. Motor schema based mobile robot navigation. *International Journal of Robotics Research*, 8(4), August 1987.
4. R.C. Arkin. Reactive control as a substrate for telerobotic systems. *IEEE Transactions on Robotics and Automation*, 8(4), June 1991.
5. Ronald C. Arkin and Johnathan Diaz. Line-of-sight constrained exploration for reactive multiagent robotic teams.
6. H. Asama, K. Ozaki, A. Matsumoto, Y. Ishida, and I. Endo. Development of task assignment system using communication for multiple autonomous robots. *Journal of Robotics and Mechatronics*, 4(2):122–127, 1992.
7. Tucker Balch and Ronald C. Arkin. Communication in reactive multiagent robotic systems. *Autonomous Robots*, 1(1):27–52, 1994.
8. Tucker Balch and Ronald C. Arkin. Behavior-based formation control for multi-robot teams. *IEEE Transactions on Robotics and Automation*, 1999.
9. R. Beckers, O.E. Holland, and J.L. Deneubourg. From local actions to global tasks: Stigmergy and collective robotics. In R. Brooks and P. Maes, editors, *Artificial Life IV*. MIT Press, 1994.
10. F. Bolton. *Pure CORBA: A code intensive premium reference*. SAMS, 2002.
11. R. Bonasso, R. Firby, E. Gat, D. Kortenkamp, D. Miller, and M. Slack. Experiences with and architecture for intelligent, reactive agents. *Journal of Experimental and Theoretical Artificial Intelligence*, 9(2):237–256, 1997.
12. V. Braitenberg. *Vehicles: Experiments in synthetic psychology*. MIT Press, 1984.
13. A. Brooks, T. Kaupp, A. Makarenko, and A. Oreback. Orca. <http://orca-robotics.sourceforge.net/index.php>, 2004.
14. R.A. Brooks. *A Robot that Walks: Emergent Behaviours from a Carefully Evolved Network*, chapter 24, pages 28–39. Artificial Intelligence at MIT. The MIT Press, 1989.
15. R.A. Brooks. *Robotic Science*, chapter 11, The Whole Iguana, pages 432–456. The MIT Press, 1989.

16. R.A. Brooks. *A Robust Layered Control System for a Mobile Robot*, chapter 24, pages 2–27. Artificial Intelligence at MIT. MIT Press, 1989.
17. R.A. Brooks. *Artificial Intelligence Memo No. 1293: Intelligence without Reason*. Massachusetts Institute of Technology, 1991.
18. R.A. Brooks. Intelligence without representation. *Artificial Intelligence*, 47:139–159, 1991.
19. Rodney A. Brooks. A robust layered control system for a mobile robot. *IEEE Journal of Robotics and Automation*, RA-2(1):14–23, March 1986.
20. G. Broten and S. Monckton. Unmanned ground vehicle electronic hardware architecture. Technical memorandum, DRDC, 2004. Suffield TM 2004-122.
21. G. Broten, S. Verret, and B. Digney. Unmanned ground vehicle software development environment. Technical Memorandum TM 2004-060, Defence R&D Canada - Suffield, February 2004.
22. Philippe Caloud, Wonyun Choi, Jean-Claude Latombe, Claude Le Pape, and Mark Yim. Indoor automation with man mobile robots. In *IEEE International Workshop on Intelligent Robots and Systems*, pages 67–72, Tsuchiura, Japan, 1990.
23. Y. Uny Cao, Alex S. Fukunaga, and Andrew B. Kahng. Cooperative mobile robotics: Antecedents and directions. *Autonomous Robots*, 4(1):7–23, March 1997.
24. Paul Cohen, Michael Greenberg, David Hart, and Adele Howe. Real-time problem solving in the phoenix environment. COINS Technical Report 90-28, U. of Mass., Amherst, 1990.
25. R. Colbaugh, H. Seraji, and K.L. Glass. Obstacle avoidance for redundant robots using configuration control. *Journal of Robotic Systems*, 6(6):722–744, 1989.
26. J.H. Connell. *Minimalist Mobile Robotics*. Academic Press, 1990.
27. C. Cote, D. Letourneau, F. Michaud and J-M. Valin, Y. Brosseau, C. Raievsky, M. Lemay, and V. Tran. Code reusability tools for programming mobile robots. In *IROS. IEEE/RSJ International Conference on Intelligent Robots and Sstems*, 2004.
28. D. Levine D. Schmidt and S. Mungee. The design and performance of real-time object request brokers. *Computer Communications*, 21:294–324, April 1998.
29. Torbjorn S. Dahl, Maja J. Matarić, and Gaurav S. Sukhatme. Adaptive spatio-temporal organization in groups of robots. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS'02)*, pages 1044–1049, Lausanne, Switzerland, September 2002.
30. J.L. Deneubourg, S. Goss, N. Franks, A. Sedova-Franks, C. Detrain, and L. Chretien. The dynamics of collective sorting: Robot-like ants and ant-like robots. In J.A. Meyer and S.W. Wilson, editors, *Simulation of Adaptive Behavior, Animals to Animats*, 1991.

31. University of Ulm Department of Computer Science. *Miro*. University of Ulm, 0.9.4 edition, November 2003.
32. Gregory Dudek, Michael R. M. Jenkin, Evangelos Milios, and David Wilkes. A taxonomy for multi-agent robotics. *Autonomous Robots*, 3:375–397, 1996.
33. R. Johnson E. Gamma, R. Helm and J. Vlissides. *Design Patterns: Elements of reusable Object-Oriented Software*. Addison-Wesley, 1994.
34. Kjerstin I. Easton and Alcherio Martinoli. Efficiency and optimization of explicit and implicit communication schemes in collaborative robotics experiments. In *IEEE Conference on Intelligent Robots and Systems IROS-02*, pages 2795–2800, Lausanne, Switzerland, September 2002. IEEE Press.
35. S. Enderle, H. Utz, S. Sablatnog, S. Simon, G. Kraetzschmar, and G. Palm. Miro - middleware for autonomous mobile robots. *International Federation of Automatic Control*, 2001.
36. C. Ferrell. *Robust Agent Control of an Autonomous Robot with Many Sensors and Actuators*. PhD thesis, Dept. of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, 1993.
37. Rafael Fierro, Aveek Das, John, Spletzer, Joel Esposito, Vijay Kumar, James P. Ostrowski, George Pappas, Camillo J. Taylor, Yerang Hur, Rajeev Alur, Insup Lee, Greg Grudic, and Ben Southall. A framework and architecture for multi-robot coordination. *The International Journal of Robotics Research*, 21(10-11):977–995, October-November 2002.
38. R. Firby. *Adaptive Execution in Complex Dynamic Worlds*. PhD thesis, Yale University, Dept. of Computer Science, 1989.
39. Message Passing Interface Forum. *Mpi: A message-passing interface standard*. Technical report, University of Tennessee, Knoxville, 1995.
40. Donald E. Franklin, Andrew B. Kahng, and M. Anthony Lewis. Distributed sensing and probing with multiple search agents: toward system-level landmine detection solutions. In *Detective Technologies for Mines and Minelike Targets, Proceedings of SPIE*, volume 2496, pages 698–709, 1995.
41. Jakob Fredslund and Maja J Matarić. Robot formations using only local sensing and control. In *IEEE International Symposium on computational Intelligence for Robotics and Automation (CIRA-2001)*, Banff, Canada, July 2001.
42. Wikipedia: The free encyclopedia. Collective intelligence. http://en.wikipedia.org/wiki/Collective_intelligence, 2004.
43. S. Schneider G. Pardo-Castellote and M. Hamilton. Ndds: The real-time publish-subscribe middleware. *Real-Time Innovations, Inc.*, 1999.

44. Douglas W. Gage. Command control for many-robot systems. *Unmanned Systems*, 10(4):28–34, Fall 1992.
45. J. Giesbrecht, J. Collier, and S. Monckton. Staged experiments in mobile vehicle autonomy. Technical memorandum, DRDC, 2004. Suffield TM 2004-288.
46. J. Gowdy. Ipt: An object oriented toolkit for interprocess communication. Technical Report CMU-RI-TR-96-07, Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, March 1996.
47. J. Gowdy. A qualatative comparision of interprocess communications toolkits for robotics. Technical Report CMU-RI-TR-00-16, Carnegie Mellon University, June 2000.
48. Jay Gowdy. *Emergent Architectures: A Case Study for Outdoor Mobile Robots*. PhD thesis, Carnegie Mellon University, 2000.
49. R. Hartley and F. Pipitone. Experiments with the subsumption architecture. In *Proc. 1991 IEEE Int. Conf. on Robotics and Automation*, pages 1652–1658, April 1991.
50. M. Henning and S. Vinoski. *Advanced CORBA Programming with C++*. Addison-Wesley, 1999.
51. Terry Huntsberger, Paolo Pirjanian, Ashitey Trebi-Ollennu, Hari Das Nayar, Hrand Aghazarian, Anthony J. Ganino, Mike Garrett, Sanjay S. Joshi, and Paul S. Schenker. Campout: A control architecture for tightly coupled coordination of multirobot systems for planetary surface exploration. *IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans*, 33(5), September 2003.
52. S. Huston, J. Johnson, and U. Syid. *The ACE Programmer's Guide*. Addison-Wesley, 2004.
53. IEEE. *The CLARAty Architecture for Robotic Autonomy*, Big Sky, MT, March 2001.
54. Leggat J. *Technology Investement Strategy For the Next two Decades*. Defence R&D Canada, 2001.
55. W. Shackleford J. Michaloski, F. Proctor. The neutral message language: A model and method for message passing in heterogeneous environments. In *Proceedings of the World Automation Conference*, Maui, Hawaii, June 2000.
56. J. Jennings and C. Kirkwood-Watts. Distributed mobile robotics by the method of dynamic teams. In *Conference on Distributed Autonomous Robot Systems*, 1998.
57. R. Johnson. Frameworks.
<http://st-www.cs.uiuc.edu/users/johnson/frameworks.html#Papers>, 2004. Defintion from his website.

58. M.B. Leahy Jr. and G.N. Saridis. A behaviour based system for off road navigation. *IEEE Transactions on Robotics and Automation*, 10(6):776–782, December 1994.
59. O. Khatib. Real time obstacle avoidance for manipulators and mobile robots. In *Proceedings IEEE Int. Conf. on Robotics and Automation*, 1985.
60. O. Khatib. A unified approach for motion and force control of robot manipulators: The operational space formulation. *IEEE Transactions on Robotics and Automation*, RA-3(1):43–53, February 1987.
61. E. Klavins and D. Koditschek. A formalism for the composition of concurrent robot behaviours. In *IEEE International Conference on Robotics and Automation*, pages 3395–3402, San Fransisco, CA, 2000.
62. S. Koenig, R. Goodwin, 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.
63. K. Konolige and K. Myers. The saphira architecture for autonomous mobile robots. *Artificial Intelligence and Mobile Robots: Case Studies of Successful Robot Systems*, 1998.
64. C. Ronald Kube and Eric Bonabeau. Cooperative transport by ants and robots. *Robotics and Autonomous Systems*, 30(1/2):85–101, 2000. ISSN: 0921-8890.
65. C. Ronald Kube and Hong Zhang. Collective robotics: From social insects to robots. *Adaptive Behavior*, 2(2):189–218, 1994.
66. D.C. MacKenzie, J.M. Cameron, and R.C. Arkin. Specification and execution of multiagent missions. In *Proceedings 1995 IROS Conference*, 1995.
67. Maja J. Matarić. Using communication to reduce locality in distributed multi-agent learning. *Journal of Experimental and Theoretical Artificial Intelligence*, 10(3):357–369, Jul-Sep 1998. special issue on Learning in DAI Systems, Gerhard Weiss, ed.
68. Maja J. Matarić, Martin Nilsson, and Kristian T. Simsarian. Cooperative multi-robot box-pushing. In *IEEE/RSJ IROS*, pages 556–561, 1995.
69. M.J. Mataric. *Interaction and Intelligent Behaviour*. PhD thesis, Dept. of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, May 1994.
70. M. Minsky. *Excerpts from The Society of Mind*, volume 1, chapter 10, pages 245–269. The MIT Press, 1990.
71. Mark W. Moffet. Cooperative food transport by an asiatic ant. *National Geographic Research*, 4(3):386–394, 1988.

72. J. Morrow and P. Khosla. Manipulation task primitives for composin robot skills. In *IEEE International Conference on Robotics and Automation*, pages 3354–3359, 1997.
73. Fabrice R. Noreils. Toward a robot architecture integrating cooperation between mobile robots: Application to indoor environment. *The International Journal of Robotics Research*, 12(1):79–98, February 1993.
74. M. Lindstrom A. Oreback and H. Christensen. Berra: A research architecture for service robots. In *International Conference on Robotics and Automation*, 2000.
75. Chris A. C. Parker, Hong Zhang, and C. Ronald Kube. Blind bulldozing: Multiple robot nest construction. In *IEEE Conference on Intelligent Robots and Systems IROS-03*, 2003.
76. Christopher Parker and Hong Zhang. Robot collective construction by blind bulldozing. In *IEEE Conference on Systems Cybernetics and Man*, 2002.
77. L.E. Parker. An experiment in robotic cooperation. In *Proceedings of the ASCE Specialty Conference on Robotics and Automation for Challenging Environments*, February 1994.
78. Lynne E. Parker. Alliance: An architecture for fault tolerant multi-robot cooperation. *IEEE Transactions on Robotics and Automation*, 14(2):220–240, April 1998.
79. Lynne E. Parker. *Distributed Autonomous Robotic Systems 4*, chapter Current State of the Art in Distributed Robot Systems, pages 3–12. Springer, 2000.
80. J . Pedersen. Robust communications for high bandwidth real-time systems. Technical Report CMU-RI-TR-98-13, Carnegie Mellon University, 1998.
81. I. Nourbakhsk R. Powers and S. Birchfield. Dervish: An office navigation robot. *AI Magazine*, 16(2):53–60, 1995.
82. J.E. Pratt. Virtual model of a biped walking robot. Master’s thesis, Dept. of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, August 1995.
83. M.H. Raibert and H.B. Brown Jr. Experiments in balance with a 2d one legged hopping machine. *Transactions of the ASME, Journal of Dynamic Systems and Control*, 106:75–81, March 1984.
84. Craig W. Reynolds. Flocks, herds, and schools: A distributed behavioral model. *Computer Graphics*, 21(4):25–34, July 1987.
85. J.K. Rosenblatt. DAMN: A distributed architecture for mobile navigation. In *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.

86. J.K. Rosenblatt and C.Thorpe. Combining multiple goals in a behavior-based architecture. In *Proceedings of 1995 International Conference on Intelligent Robots and Systems (IROS)*, Pittsburgh, PA, August 1995.
87. M. Montemerlo N. Roy and S. Thrun. Perspectives on standardization in mobile robot programming: The carnegie mellon navigation (CARMEN) toolkit. In *Proceedings of the Conference on Intelligent Robots and Systems (IROS)*, 2003.
88. Dunbar W. S. and Klein B. Mining, mineral processing, and mini-machines. *CIM Bulletin*, 95(1057), January 2002.
89. T. Sandholm and V. Lesser. Issues in automated negotiation and electronic commerce: Extending the contract net framework. In *International Conference on Multiagent Systems*, pages 328–335, 1995.
90. C. Schlegel. *Communication Patterns for OROCOS Hints, Remarks and Specification*. Research Institute for Applied Knowledge Processing (FAW), Helmholtzstrasse 16, Ulm Germany, 0.12 edition, Feb. 2002.
91. D. Schmidt and S. Huston. *C++ Network Programming Volume 1*. Addison-Wesley, 2002.
92. D. Schmidt and F. Kuhns. An overview of the real-time corba specification. *IEEE Computer special issue on Object-Oriented Real-time Distributed Computing*, 2000.
93. H. Seraji, R. Steele, and R. Ivlev. Sensor based collision avoidance: Theory and experiments. *Journal of Robotic Systems*, 13(9):571–586, 1996.
94. R. Simmons. The inter-process communications (ipc) system, 1991.
<http://www-2.cs.cmu.edu/afs/cs/project/TCA/www/ipc/ipc.html>.
95. R. Simmons. Structured control for autonomous robots. *IEEE Transactions on Robotics and Automation*, 10(1), February 1994.
96. Reid Simmons, Trey Smith, M. Bernardine Dias, Dani Goldberg, David Hershberger, Anthony Stentz, and Robert Zlot. A layered architecture for coordination of mobile robots. In *Multi-Robot Systems: From Swarms to Intelligent Automata, Proceedings from the 2002 NRL Workshop on Multi-Robot Systems*. Kluwer Academic Publishers, May 2002.
97. K. Sycara and D. Zeng. Coordination of multiple intelligent software agents. *International Journal of Cooperative Information Systems*, 5(2-3), 1996.
98. C. Szyperski. *Component Software: Beyond Object-Oriented Programming*. Addison-Wesley, Reading, MA, 1998.
99. W. Tang and H. Zhang. Decentralized control of robot formation marching. In *Sixth International Symposium on Robotics and Manufacturing*, Montpellier, France, May 27-30 1996.

- 100.D. Terzopoulos, X. Tu, and R. Grzeszczuk. Artificial fishes with autonomous locomotion, perception, behavior, and learning in a simulated physical world. In *Proceedings of Artificial Life IV Workshop*, July 1994.
- 101.Ashitey Trebi-Ollennu, Hari Das Nayar, Hrand Aghazarian, Anthony Ganino, Paolo Pirijanian, Brett Kennedy, Terry Huntsberger, and Paul Schenker. Mars rover pair cooperatively transporting a long payload. In *IEEE International Conf. on Robotics and Automation (ICRA)*, pages 3136–3141, Washington, DC, May 2002.
- 102.H. Utz, S. Sablatnog, S. Enderle, and G. Kraetzschmar. Miro - middleware for mobile robot applications. *IEEE Transactions on Robotics and Automation*, June 2002.
- 103.M. van de Panne and E. Fiume. A controller for the dynamic walk of a biped across variable terrain. In *Proceedings of the 31st IEEE conference on Decision and Control*, 1992.
- 104.Sean Verret, Hong Zhang, and Max Q.-H. Meng. Collective sorting with local communication. In *Proc. IROS'04, IEEE/RSJ International Conference on Intelligent Robots and Systems (in press)*, Japan, September 2004.
- 105.Sean R. Verret. Perception and communication – their relationship in collective sorting. Master's thesis, University of Alberta, Edmonton, Alberta, September 2004.
- 106.W.G. Walter. An imitation of life. *Scientific American*, 182(5):42–45, May 1950.
- 107.W.G. Walter. A machine that learns. *Scientific American*, 185(2):60–63, August 1950.
- 108.Jing Wang. Drs operating primitives based on distributed mutual exclusion. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1085–1090, Yokohama, Japan, 1993.
- 109.Jens Wawerla, Gaurav S. Sukhatme, and Maja J. Matarić. Collective construction with multiple robots. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2696–2701, Lausanne, Switzerland, 2002.
- 110.M. Wellman and P. Wurman. Market-aware agents for a multi-agent world. *Robotics and Autonomous Systems*, pages 115–125, 1998.
- 111.L. Wills, S. Kannan, S. Sander, M. Guler, B. Heck, J.V.R. Prasad, D. Schrage, and G. Vachtsevanos. An open platform for reconfigurable control. *IEEE Control Systems Magazine*, June 2001.

Unclassified

DOCUMENT CONTROL DATA		
(Security classification of title, body of abstract and indexing annotation must be entered when document is classified)		
1. ORIGINATOR (the name and address of the organization preparing the document. Organizations for whom the document was prepared, e.g. Centre sponsoring a contractor's report, or tasking agency, are entered in section 8.) Defence R&D Canada – Suffield PO Box 4000, Medicine Hat, AB, Canada T1A 8K6	2. SECURITY CLASSIFICATION (overall security classification of the document including special warning terms if applicable). UNCLASSIFIED	
3. TITLE (the complete document title as indicated on the title page. Its classification should be indicated by the appropriate abbreviation (S,C,R or U) in parentheses after the title). Towards Distributed Intelligence (U)		
4. AUTHORS (Last name, first name, middle initial. If military, show rank, e.g. Doe, Maj. John E.) Brotten, G. ; Monckton, S. ; Giesbrecht, J. ; Verret, S. ; Collier, J. ; Digney, B.		
5. DATE OF PUBLICATION (month and year of publication of document) December 2004	6a. NO. OF PAGES (total containing information. Include Annexes, Appendices, etc). 86	6b. NO. OF REFS (total cited in document) 111
7. DESCRIPTIVE NOTES (the category of the document, e.g. technical report, technical note or memorandum. If appropriate, enter the type of report, e.g. interim, progress, summary, annual or final. Give the inclusive dates when a specific reporting period is covered). Technical Report		
8. SPONSORING ACTIVITY (the name of the department project office or laboratory sponsoring the research and development. Include address). Defence R&D Canada – Suffield PO Box 4000, Medicine Hat, AB, Canada T1A 8K6		
9a. PROJECT OR GRANT NO. (if appropriate, the applicable research and development project or grant number under which the document was written. Specify whether project or grant). 42zz78	9b. CONTRACT NO. (if appropriate, the applicable number under which the document was written).	
10a. ORIGINATOR'S DOCUMENT NUMBER (the official document number by which the document is identified by the originating activity. This number must be unique.) DRDC Suffield TR 2004-287	10b. OTHER DOCUMENT NOS. (Any other numbers which may be assigned this document either by the originator or by the sponsor.)	
11. DOCUMENT AVAILABILITY (any limitations on further dissemination of the document, other than those imposed by security classification) (X) Unlimited distribution () Defence departments and defence contractors; further distribution only as approved () Defence departments and Canadian defence contractors; further distribution only as approved () Government departments and agencies; further distribution only as approved () Defence departments; further distribution only as approved () Other (please specify):		
12. DOCUMENT ANNOUNCEMENT (any limitation to the bibliographic announcement of this document. This will normally correspond to the Document Availability (11). However, where further distribution beyond the audience specified in (11) is possible, a wider announcement audience may be selected).		

Unclassified

Unclassified

13. **ABSTRACT** (a brief and factual summary of the document. It may also appear elsewhere in the body of the document itself. It is highly desirable that the abstract of classified documents be unclassified. Each paragraph of the abstract shall begin with an indication of the security classification of the information in the paragraph (unless the document itself is unclassified) represented as (S), (C), (R), or (U). It is not necessary to include here abstracts in both official languages unless the text is bilingual).

Unmanned Ground Vehicle (UGV's) Research and Development within the Autonomous Land Systems (ALS) project will assist the Canadian Forces in fulfilling their future mandate. The ALS project derives its focus from the Autonomous Intelligent Systems (AIS) activity outlined by the DRDC Technology Investment Strategy (TIS).

There are five anticipated classes of Unmanned Vehicles (UV): fixed or rotor wing aircraft Unmanned Air Vehicles (UAV); typically tracked, wheeled, legged Unmanned Ground Vehicles (UGV); stationary monitoring Unattended Ground Sensors (UGS); untethered, propellor or bouyancy driven, Unmanned Underwater Vehicles (UUV); and light propellor driven Unmanned Surface Vehicles (USV). The future battlespace demands compatibility between all UV classes. All UVs must have an inherent ability to share information if they are to provide the desired force multiplication factor for the future asymmetric battlespace.

To effectively distribute intelligence modules within and between UVs, layered modular hardware design and portable, maintainable coding practice require an architecture that, at once, intrinsically supports and encourages distributed computing, and frees investigators to focus on the development of intelligent single and multi-vehicle control systems. An architecture founded on these elements defines, at a high level, the links between various software components that create an operational vehicle. Ideally, architectures should seamlessly transition between real vehicle control; system diagnosis through the replay of gathered data; and the control of a vehicle in a simulated world. Ideally, the investigator is then free to develop intelligence algorithms without vehicle implementation distractions. With satisfactory simulated performance, algorithms may be safely run on a physical vehicle. Conversely, historical data gathered from a real vehicle run can be replayed in a simulated environment to investigate, debug and optimize the algorithm performance.

This document explores the depths of the multi-vehicle architecture problem using the past experience of other investigators, the apparent technological evolution of both hardware and software, and the demands of the future CF environment. This report overviews fundamental methods in multi-vehicle cooperation and coordination, single vehicle autonomous control, and the underlying infrastructure of real and simulated systems.

14. **KEYWORDS, DESCRIPTORS or IDENTIFIERS** (technically meaningful terms or short phrases that characterize a document and could be helpful in cataloguing the document. They should be selected so that no security classification is required. Identifiers, such as equipment model designation, trade name, military project code name, geographic location may also be included. If possible keywords should be selected from a published thesaurus. e.g. Thesaurus of Engineering and Scientific Terms (TEST) and that thesaurus-identified. If it not possible to select indexing terms which are Unclassified, the classification of each should be indicated as with the title).

multi-robot, intelligence, collective, distributed, autonomy, control, simulation, modularity, frameworks, components, inter-process communications, middleware, CORBA, ACE, TAO, Miro

Unclassified