J Intell Robot Syst (2009) 54:331–357
DOI 10.1007/s10846-008-9269-7

Distributed Simulation and Middleware for Networked UAS

Ali Haydar Göktoğan · Salah Sukkarieh

Received: 15 March 2008 / Accepted: 30 June 2008 / Published online: 14 August 2008

Abstract As a result of the advances in solid state, electronics, sensor, wireless communication technologies, as well as evolutions in the material science and manufacturing technologies, unmanned aerial vehicles (UAVs) and unmanned aerial systems (UAS) have become more accessible by civilian entities, industry, as well as academia. There is a high level of market driven standardisation in the electronics and mechanical components used on UAVs. However, the implemented software of a UAS does not exhibit the same level of standardisation and compartmentalisation. This is a major bottleneck limiting software maintenance, and software reuse across multiple UAS programs. This paper addresses the software development processes adapted by the Australian Centre for Field Robotics (ACFR) in major UAS projects. The presented process model promotes software reuse without sacrificing the reliability and safety of the networked UAS with particular emphasis on the role of distributed simulation and middleware in the development and maintenance processes.

Keywords Unmanned aerial system (UAS) · Unmanned aerial vehicle (UAV) · Software frameworks · Middleware · Simulation · RMUS · HWIL

A. H. Göktoğan (✉) · S. Sukkarieh
ARC Centre of Excellence for Autonomous Systems Australian Centre for Field Robotics School of Aerospace, Mechanical, and Mechatronic Engineering,
The University of Sydney, 2006, NSW Australia
e-mail: aligoktogan@acfr.usyd.edu.au

S. Sukkarieh
e-mail: salah@acfr.usyd.edu.au

1 Introduction

The goal of this paper is to describe the middleware oriented software development methodology used for the networked unmanned aerial system (UAS) at the Australian Centre for Field Robotics (ACFR).

The software development methodology provides the capability of transitioning, testing and validating, complex research algorithms from simulation to real-time demonstration. Particular focus is on the need to minimise re-coding of the research algorithms between the various testing phases; and for the development of the framework to provide distributed, decentralised and scalable capabilities.

Establishment of an operational infrastructure of an UAS, incorporating a dedicated fleet of UAVs in a R&D environment involves complex and demanding system development processes. Success of this endeavour highly depends on understanding the system life cycle management (SLCM) in R&D environments, and the unique characteristics of the UAS and flight operations. This paper looks at the software development side of the SLCM.

The next section discusses the market conditions of the commercial-off-the-shelf (COTS) hardware and software components for UAS. Section 3 briefly presents the software development processes that we used in developing our networked UAS. Section 4 presents the high level architecture of a multi-UAV system. Section 5 introduces a framework for distributed autonomous systems (AFDAS). The CommLibX/ServiceX middleware and its novel concept of virtual channel are covered in Section 6. Section 7 examines the real-time multi UAV simulator (RMUS). Finally, conclusion and future works are covered in Section 8.

2 COTS Components for UAS

Advances in solid state and electronics technologies fit more processing power to unit volume while reducing the cost and the electrical power consumption. Consequently, the new developments in sensor technologies offer a wider range of more reliable, faster, multi-modal sensors (MMS) in smaller packages with lower price tags. Likewise, the wireless communication industry is providing wider bandwidth over longer distances through smaller devices while radiating lower Radio Frequency (RF) energy. Furthermore, recent electrochemical fuel cells and batteries with high power density offer new possibilities both for the payload's electrical power needs, and for electrically driven propulsion systems particularly used on small UAVs.

Thanks to the evolution in material science that, stronger, lighter, fatigue resistant, and easily machineable complex composite structural materials are now widely available. Similarly, development in the manufacturing technologies, such as wide spread of computerized numerical control (CNC) machinery and rapid prototyping systems, made it possible of fabricate more complex structures in shorter time, with higher precision.

Along with the other unmanned systems, the UAVs benefit the most from these positive trends. The radio controlled (R/C) hobby gadgetry market, and the light aircraft industry offer wide range of mechanical components that can be used in construction new UAVs. Acquisition of commercial-off-the-shelf (COTS) small but powerful computers, sensors, and wireless communication devices are not beyond the reach of many academic institutions. Therefore, increasing number of academic institutions are able to utilise UAVs in their research, teaching and learning programs.

There is a high level of standardisation in the electronics and mechanical components. Highly standardised and compartmentalised, modular hardware sub-systems and components offer flexibility to UAS developers. The UAS architects and engineers can easily configure their hardware and upgrade it when needed. Because of the availability of multiple vendors of the same or similar hardware components, many UAS developers feel free in selecting their suppliers.

However, the software side of UAS does not exhibit the same level of standardisation and compartmentalisation. This is a major bottleneck limiting the software maintenance, and software reuse across multiple UAS programs. It is not easy to find many successful examples neither in academia nor in industry in which the UAS developers can acquire COTS software components, in an equally flexible manner as if they were acquiring the hardware components.

The limited availability of the COTS software components for the UAS development might be natural result of demand-and-supply relationship in the software market. That is, the demand for the COTS software solutions generated by the UAS developments may be not strong enough to create its own market, yet. Even so, there are exceptions to this generalisation where a significant standardisation works in progress in the military domain. The "Standard interfaces of UAV control system (UCS) for NATO UAV interoperability", (STANAG 4586) [1] is one of them. This particular standard has created its own market in which a number of companies provide STANAG 4586 compatible software solutions. This example shows the obvious; the UAS software market could thrive if enough demand is generated. However, STANAG requirements are too complex for many research groups in academia for their relatively smaller scale research projects.

By definition, research projects address new concepts and therefore one may argue that majority of the new UAS development may need their own implementation of software modules. Although this argument is partially valid, there is still a place for the COTS software solutions if enough demand is generated.

One of the reasons why the civilian UAS domain could not generate enough demand is because; in this domain, there are many different types of UAVs. They are ranging from micro aerial vehicles (MAVs) which can carry few grams of payload to the larger UAVs with hundreds of kilograms of Maximum Take-Off Weight (MTOW) capabilities. These large varieties of UAVs are equipped with very different types of computational units, such as tiny microcontrollers for MAVs, or a network of high performance computers for the top scale UAVs.

The need for reusable and maintainable software modules, components, frameworks, architectures, and services are not new and not specific to the UAS domain. These fundamental computer science and software engineering issues are widely addressed, and numerous techniques have already been developed and successfully used in other application domains by industry and academia. However, the same techniques are being slowly accepted by the UAS community particularly in academia. This may be due to the fact that many UAS projects are initiated by highly qualified experts in aeronautical and/or control engineering fields. Inadvertently, due to their highly focused views, they often underestimate the importance and complexities associated with maintainable software development methodologies.

Increasing involvement of computer scientist and software engineers in the UAS development life cycle is expected to change this bottleneck.

There is a variety of literature, addressing software development methodologies for aerial vehicles. Ernst et al. in [2] presents a method of designing and building UAV flight controllers using C code generated from MatLab. They also address the validation of the resultant system on a COTS simulator X-Plane. Although, this method is highly efficient way of developing flight controllers for small UAVs in which the flight controller consists of a single microcontroller, it does not address widely distributed systems. Similarly, as they acknowledge in their paper, X-Plane based simulation may provide acceptable simulation for single UAV, but, in its current form, it can not address the needs of complex mission simulations incorporating multiple UAVs and other entities such as mission sensors, and ground vehicles. Shixianjun et al. in [3] also presents a MatLab-Simulink based code generation for a UAV and its hardware-in-the-loop simulation (HWIL) tests. Their system is also based on a single UAV, and does not address distributed systems.

There are also a number of examples in the literature covering issues of multi-UAV system software. In [4] Doherty et al. explains the WITAS UAV project in which their helicopter UAV is controlled by a flight control software built around a DOS based real-time kernel “RTkernel”. The research methodology used in WITAS project acknowledged the importance of the simulation. Their flight experiments were supplemented with a great deal of simulated experiments. The simulator architecture was based on a real-time common object request broker architecture (CORBA) as the software communication infrastructure. Their intelligent vehicle control architecture (IVCA) also used the same real-time CORBA based infrastructure which allows transition of the software implementations from the simulation environment to the HWIL simulator and eventually to the actual platforms.

CORBA, from the object management group (OMG) is an extreme case of an open standard specification for general purpose, distributed middleware architecture which can be implemented by different vendors. There are a number of CORBA implementations from different vendors. However, it is widely accepted that, many implementations do not match all of CORBA’s published specification [5, 6]. For many software professionals, this may not necessarily be a serious disadvantage.

Once, (and if) a satisfying CORBA implementation is acquired, the CORBA based distributed system development (at least in theory) should be straightforward. Even so, it is acknowledged by many [5, 7–9] that using CORBA for any nontrivial application is surprisingly difficult. CORBA has a very large and complex set of application programming interface (API). Employment of a dedicated team of CORBA experts is almost a necessity for the development of successful, large scale, and maintainable CORBA based systems. For some mid-sized research organisations with limited resources, like the ACFR, this is less than a desired option. Having said that, there are also a large number of gratified CORBA users from a variety of application domains, including robotics and autonomous systems [10–13].

Flexibility and performance are considered rivals in software systems [14]. Despite its wide acceptance, particularly by moderate size information technology (IT) companies, CORBA’s alleged flexibility leads to its rather unpleasant reputation of being “too big and complex”. Surely, the technological advances in both memory and processing power of computer systems are helping CORBA to change its reputation in a positive direction. The OMG’s recently published draft specifications for "common object request broker architecture—for embedded" (CORBA/e) [15] is another positive step toward CORBA's wider acceptance in the future. The CORBA/e draft specification addresses the bottlenecks of the old/existing, enterprise-CORBA and acknowledges CORBA's implementation issues. It particularly references CORBA's resource-hungry nature and its limitations to be used in real-time applications on resource limited embedded computing platforms. This draft is yet another valuable, lengthy document which one should study to learn from CORBA's past experiences. "CORBA/e sacrifices some of the general purpose nature of CORBA in order to support the development of real-time systems." [15], this statement is a clear indication that OMG now has a very different vision for the future of CORBA.

Following section briefly presents the software development processes that we used in developing our networked UAS. It also highlights the importance of modelling and simulation.

3 Software Development Processes for UAS

A simplified version of the application software development processes that we have followed in developing software applications for the UAS is shown in Fig. 1. It emphasises that the Modeling and Simulation (M&S) is an integral part of each process.

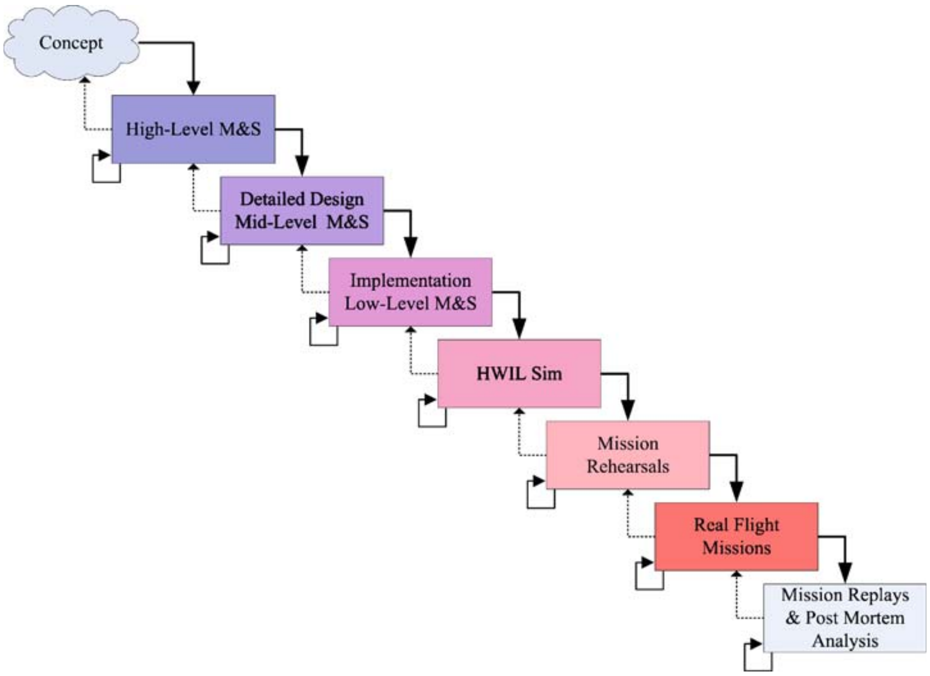
Concept
High-Level M&S
Detailed Design Mid-Level M&S
Implementation Low-Level M&S
HWIL Sim
Mission Rehearsals
Real Flight Missions
Mission Replays & Post Mortem Analysis

Fig. 1 A simplified version of the software development process flow for UAS

Due to its cascading form, the development process model in Fig. 1 is known as the waterfall model [16]. The waterfall model keeps iterations between neighbouring phases. In order to proceed from one phase to the next, often a review takes place to determine if the phase is completed successfully. The large downward solid arrows indicates the desired flow direction; from upper level phase to the lower level phase. However, sometime in practise, it become inevitable to go back to previous phases. This less desired upward flows are depicted by the dashed arrows. The small loop-back arrows on the lower left corner of the processes indicates that it is encouraged to iterate within the life-cycle phase before proceeding to the lower level phase.

The first stage in our software development process is the "high-level M&S" of the proposed system in Matlab/Simulink or similar high level simulation environments. After the *validation*[1] of the new concepts, techniques and algorithms, the class hierarchies, hardware and software components are defined in the "detailed design, mid-level M&S" phase. At this phase, unified modeling language (UML) is particularly useful for software design and documentation as well as for defining the test cases.

The "Implementation, Low-Level M&S" phase consists of coding the actual running software (often in C/C++) and testing individual modules. In these tests low-level M&S, such as simulation of software modules, interaction between software components, and sensor and control outputs are tested. Test results are used in the *verification*[2] of the developed algorithms.

All software applications, their interaction with each other, and the actual hardware that will be used in the real flight missions are tested in the "hardware-in-the-loop simulation (HWIL Sim)" phase. The HWIL Sim test results are used for the verification and *accreditation*[3] of the developed software, ie. results obtained at the high-level simulation phase are compared with the HWIL Sim test results.

Following successful completion of the Validation, Verification and Accreditation (VV&A) procedures in the HWIL Sim are assumed to be ready for the "mission rehearsals" phase. At this stage, the operation plans are prepared. These plans state the mission objective, define personnel work distribution and flight plans. The operational personnel are prepared for the mission through repeated mission rehearsals performed in the HWIL Sim environment.

After the operational personnel are reached to the required proficiency level for the proposed missions, the "real flight missions" phase can be performed in the actual flight area. During real flight missions, a number of data logging processes are performed both onboard the UAVs and on the ground stations. The logged data are used in "mission replays and post mortem analysis" phase. Soon after the flight operations, the logged data are replayed in the RMUS environment and the mission team assess if mission objectives are reached and decide if the flight mission needs to be repeated or not. The logged data is later used in the RMUS environment many times for post-mortem analysis of the missions.

[1] In the context of modelling and simulation (M&S), *validation* is the process of determining whether the selected simulation model accurately represents the system.

[2] *Verification*, answers the question of whether the selected simulation model is implemented correctly.

[3] *Accreditation* is "the official certification that a model or simulation is acceptable for use for a specific purpose" [17].

4 Multi-UAV System Architecture

This section presents the architecture of the multi-UAV system which is developed and successfully used by the ACFR. It also introduces the terminology used in this paper.

Traditional UAV applications involve relatively large UAVs with the capability of carrying multiple payload sensors in a single platform. Ever increasing availability, and accessibility of smaller UAVs to the research community encourage the development of multi-UAV applications, where the mission sensors would be distributed across a number of UAVs [18].

This is a major undertaking; it shifts the focus from the platform-centric to network-centric capabilities, and it requires the adoption of a network-centric operational philosophy; "The network-centric operational philosophy relies on the capability of shared situation awareness, through fusion of information disseminated from a wide range of networked sensors, and decentralized decision making to control the platforms and the sensors to maximize the information gain toward achieving the mission goals" [19].

Compared to the rather confined problem domain of the platform-centric capability, the network-centric capability has a much broader domain. In network-centric systems, the information dissemination often takes place across heterogenous, mobile information sources and information consumers. In such systems, it is not uncommon to have range and the bandwidth limited communication links with sporadic dropouts. However, in platform-centric systems, where all nodes are carried on the same platform, the information dissemination occurs on a much faster and more reliable local network.

The Brumby Mk III UAV (shown below) is capable of demonstrating the platform-centric system capabilities when a single-UAV is used in the flight missions [20]. More importantly, the network-centric system capabilities are also successfully demonstrated in flight missions in which, multiple Brumby Mk III UAVs are used [18, 21–25].

Brumby Mk III UAV The Brumby Mk III (Fig. 2) UAVs are the main flight platform used by the ACFR in multi-UAV experiments. They are delta wing UAVs with 2.9 m wing span, pusher type four blade propeller, and conventional tricycle undercarriage suitable for takeoff from and landing on short grass and asphalt surfaces. The Brumby Mk III UAVs are designed and manufactured at The University of Sydney as the third generation UAVs in the Brumby series. Their maximum take-off weight (MTOW) is approximately 45 kg. The replaceable nose cone of the Brumby Mk III UAVs is ideal for reconfiguration of the UAVs to carry different mission payloads in a plug-and-play manner.

Due to their delta wing design and the 16 Hp engine, the Brumby Mk III UAVs are very agile platforms. They can travel around 100 Knots (183.5 km/h) and have roll rates of approximately 180°/s. The agility of a UAV relates to the highly dynamic nature of the platform. Compared to slow, hovering vehicles like helicopters and blimps, highly dynamic, fixed wing UAVs pose additional challenges, especially in control and in communication. Although the software framework presented in this paper can be applied to a wide variety of systems, this paper is focused particularly on the multi-UAV systems of the agile, fixed wing UAVs.

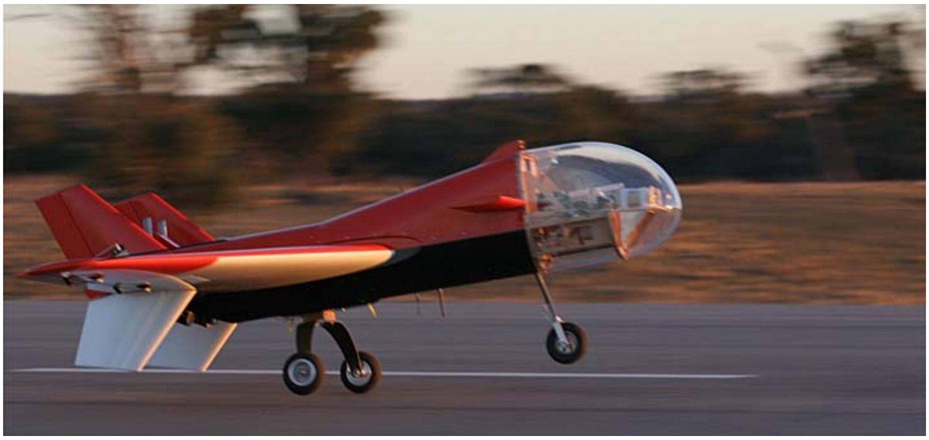

Fig. 2 The Brumby Mk III UAV as it takes off. It has delta wing with 2.9 m wing span, pusher type propulsion, conventional tricycle undercarriage, and replaceable sensor nose cone

Avionics The Brumby Mk III UAVs are equipped with relatively complex avionics (Fig. 3). In a typical configuration, a single Brumby Mk III UAV carries six PC104+ computers, all networked through an onboard local area network (LAN), a flight and mission sensors, a microcontroller based fight mode switch (FMS), an optional digital signal processing (DSP) board, a spread spectrum radio modem, a 802.11b wireless Ethernet, a data acquisition module and power subsystem with two separate buses to power flight critical and non-flight critical payloads. The FMS is one of the most critical units in the Brumby Mk III UAV avionics architecture. It consists of a spread spectrum radio modem and an embedded microcontroller board. Depending on the flight mode activation strategies defined by the system engineering and the HWIL simulations, the embedded application software running on the microcontroller drives the servo actuators either based on the remote control commands received from the ground or commands received from the flight control computer (FCC). Thus, the FMS can switch the UAV operation mode between remotely piloted vehicle (RPV) and autonomous flight modes.

The FCC is a PC104+ computer and runs the QNX Real-Time Operating System (RTOS). The Brumby Flight Control System (BFCS) is the main high level application running on the FCC. The multi-threaded BFCS application calculates the real-time navigation solution and also generates low level control signals. Through the CommLibX/ServiceX middleware, the BFCS provides real-time navigation data to other onboard computers.

The Mission Computer (MC) also runs the QNX RTOS. As illustrated in Fig. 3, the FCC, and MC is linked via the onboard Ethernet hub. The MC is equipped with an 802.11b wireless Ethernet card. It provides air to air and air to/from ground communication links. Compared to the FMS's radio modem, the 802.11b provides a wider bandwidth. However, the 802.11b link has much shorter communication range and it is subject to frequent dropouts. Hence, flight critical data, such as RPV control commands are not transmitted via wireless Ethernet but via a radio modem.

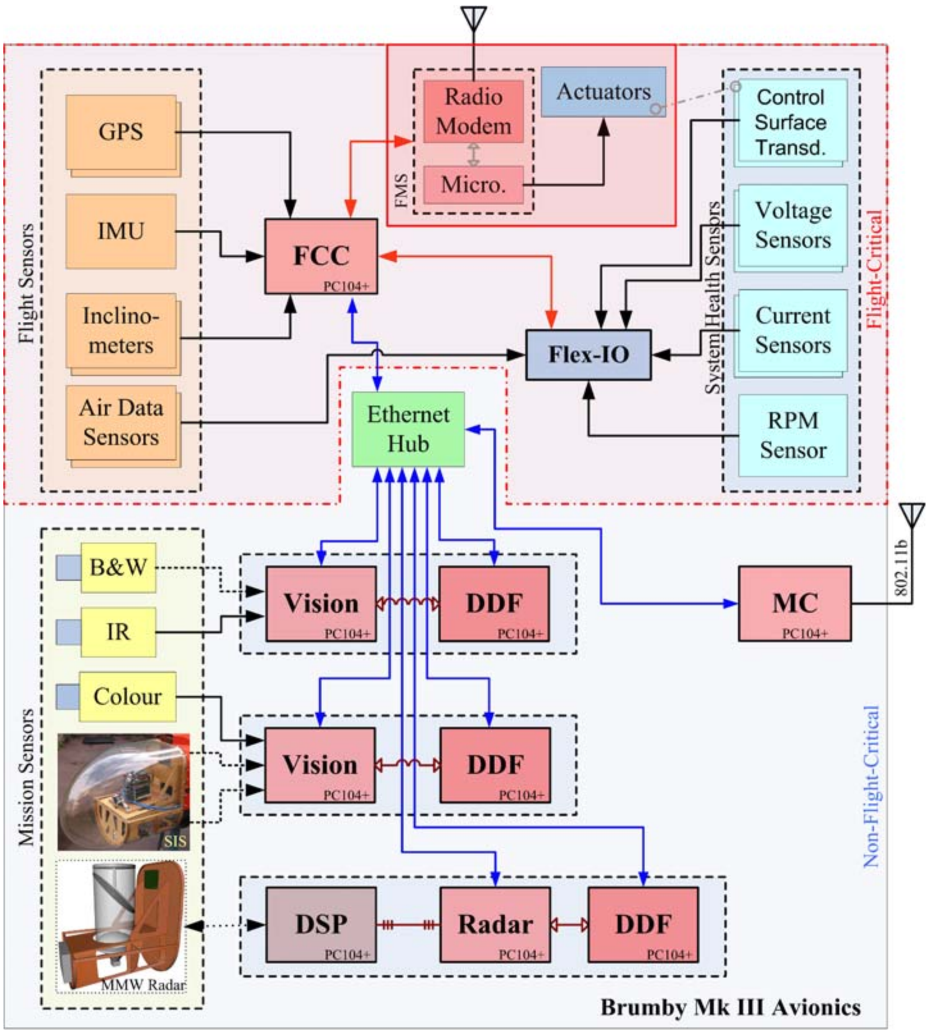

Fig. 3 A high level representation of the Brumby Mk III UAV avionics with the optional mission sensors

The mission sensor control computers run embedded Linux in soft-real-time while the decentralised data fusion (DDF) computers run QNX in hard real-time. All interact with each other via CommLibX/ServiceX middleware.

Unmanned Aerial System (UAS) With a number of interacting onboard subsystems UAVs are considered complex systems. However, a UAV is only one component of a larger whole; known as an *unmanned aerial system* (UAS), Fig. 4. This larger whole is comprised of the ground stations, communication, telemetry, control and navigation equipments, sensor payloads and all the other entities necessary to perform missions involving UAVs.

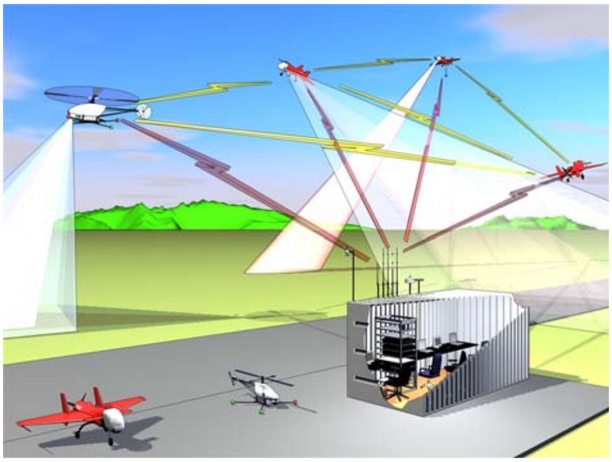

Fig. 4 Illustration of a multi-UAV system consists of heterogeneous UAVs carrying mission sensors and communicating with the ground control station as well as with each other

As illustrated in Fig. 3 and in Fig. 5, an operational multi-UAV system has a number of computing and communication nodes. Each node concurrently runs multiple processes with various levels of real-time requirements. In this context, the term *real-time*, refers to satisfaction of timeliness requirements of individual processes [26].

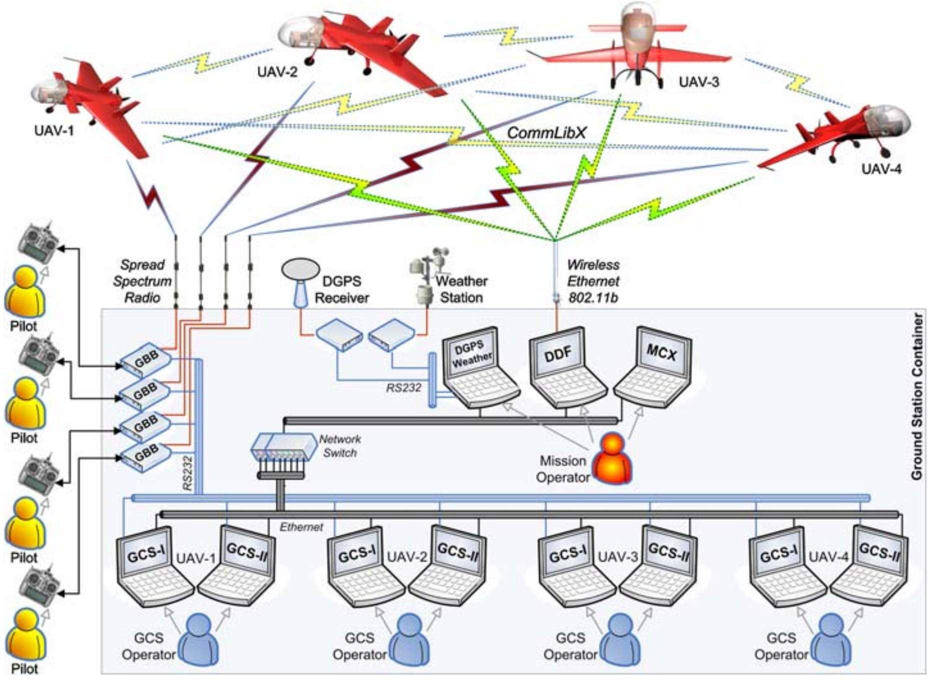

Fig. 5 An operational multi-UAV system consists of a number of computing and communication nodes onboard the UAVs and the ground station

Multi-UAV systems have a number of components both on the ground and in the air. The communication links between these *distributed* components, particularly air to air and air to/from ground links are often subject to intermittent dropouts. Therefore, in practical terms, the overall system can not be controlled efficiently by a centralized computer. The nature of multi-UAV systems dictates the adoption of *decentralised* system architectures in which no single node is central for the operation of the entire system.

At the ACFR, the multi-UAV systems are often used for the demonstration of DDF [27], Simultaneous localisation and mapping (SLAM) [28] and various cooperative control [29] techniques. The decentralised system architecture is also an algorithmic requirement of these research areas [18].

Ground Station The UAV flights are monitored and controlled from the ground station which is within an impact-proof cargo container located next to the runway as shown in Fig. 6. All the ground based computing and the communication hardware is enclosed in the ground station. The system hardware architecture of the ground stations for four UAVs is shown in Fig. 5.

As shown in Fig. 5, the main components in the ground station are the ground control stations (GCS), mission control console (MCX), weather station, uplink controllers (also known as ground black box [GBB]), differential global positioning system receiver, and a decentralized data fusion node.

Often, during the taxi, take off and landing, human pilots control the UAVs with modified hand held radio control (R/C) units. Unlike the standard R/C transmitters, these modified R/C units, send the servo control signals to the uplink controllers instead of transmitting them directly to the UAVs as modulated radio frequency (RF) signals.

An uplink controller (GBB) is a unit built around an embedded microcontroller and a spread spectrum radio modem. Its main functions are to digitise the servo pulse streams coming from the R/C unit and pack them for transmission to the UAV and to the GCS. The GBB unit also acts as a full-duplex packet router and as such, it routes the telemetry data received from the UAV to its corresponding GCS and also sends the commands issued by the GCS to the UAV (Fig. 5).

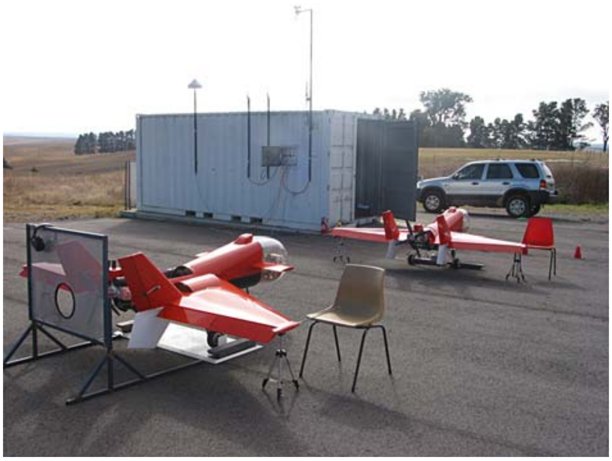
Fig. 6 Photograph of two Brumby Mk III UAVs, next to the ground station container, are being prepared for a multi-UAV mission

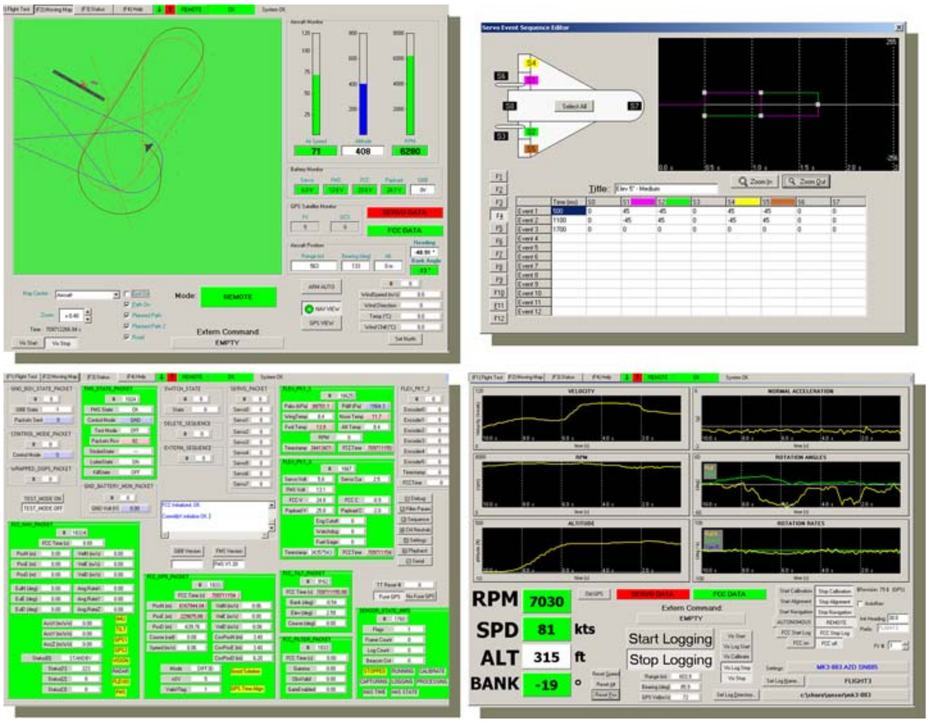

Fig. 7 Screen shots from the ground control station—I (GCS-I) application

The status and onboard instruments of each UAV is monitored in real-time by two GCS computers. GCS-I (Fig. 7) displays and logs the real-time telemetry data and GCS-II displays and logs the data related to mission and cooperative control. The telemetry data consists of more than a hundred types of data packages transmitted from the UAV to the ground at various data rates. These include position, air speed, altitude, engine RPM, and battery voltages which are also relayed by the GCS operator to the human pilot during the remote control of the UAVs. The logged telemetry data is used for post-mortem analysis of the flights. GCS-I also receives

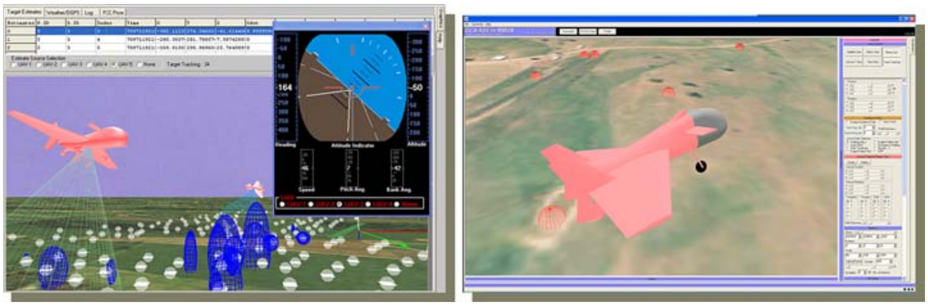

Fig. 8 Screen shots from the ground control station—II (GCS-II) application

the differential global positioning system (DGPS) messages and weather data via the local network in the ground station. It relays the DGPS messages to the UAV via the GBB. The local weather data is also displayed on GCS-I to aid in Situation Awareness (SA) [30] of the ground station operator and the pilot. As shown in Fig. 8, the GCS-II can optionally display, in real-time, the monitored UAV in a simulated 3D environment. This helps GCS operators to increase their SA.

Both the GCS-I and the GCS-II are multi-threaded applications and written for the MS-Windows operating system and they use CommLibX/ServiceX middleware.

5 A Framework for Distributed Autonomous Systems

The above mentioned multi-UAV system architecture exhibits the characteristics of highly decentralised, distributed, and networked systems. This networked system has heterogenous nodes. The nodes have a number of different hardware. They run a variety of software applications on a number of different operating systems with different levels of real-time requirements.

Typical multi-UAV missions, such as DDF, SLAM [18] and cooperative control demonstration missions [31], performed by the ACFR consists of recurring development phases with similar or identical tasks like platform, and sensor modeling, communication, data logging and replay, visualisation and time synchronisation. In order to minimise the repeated solutions, a systematic approach for design, development, deployment, and maintenance of the UAS system software is needed.

Component based software frameworks provide a suitable domain to apply systematic approach to the software development phases. The “Lego character” of components offers well defined, simple, yet powerful integration mechanisms. This leads to easily re/configurable and maintainable systems. A Framework for Distributed Autonomous Systems (AFDAS) is designed and developed with this vision in mind.

AFDAS aims to address both the low-level and high-level software requirements of a networked UAS. These include distributed, real-time simulator architectures, the ground station, and the flight critical applications of multi-UAV systems. AFDAS is leveraged by a large collection of software libraries, modules, and components that have been developed over the years and still continue to expand with refinement of the existing code as well as the new software being developed for the new UAS projects.

AFDAS uses a top-down decomposition approach to the domain of networked UAS and introduces a layered system software view. The operational and safety requirements of the networked UAS play an important role in these approaches. The domain partitioning, and framework layering increase the efficiency of managing the complex system requirements.

Control layers have different levels of task abstractions; higher control layers have more abstract tasks than the lower layers. Abstraction is also reflected to the real-time requirements. The real-time requirements ease at higher level of abstractions.

The networked UAS software system is designed in a hierarchical form. As it is illustrated in Fig. 9 and tabulated in Table 1 the system software hierarchy can be specified in four layers; actuator, subsystem, system, and system of systems (SoS) layers.

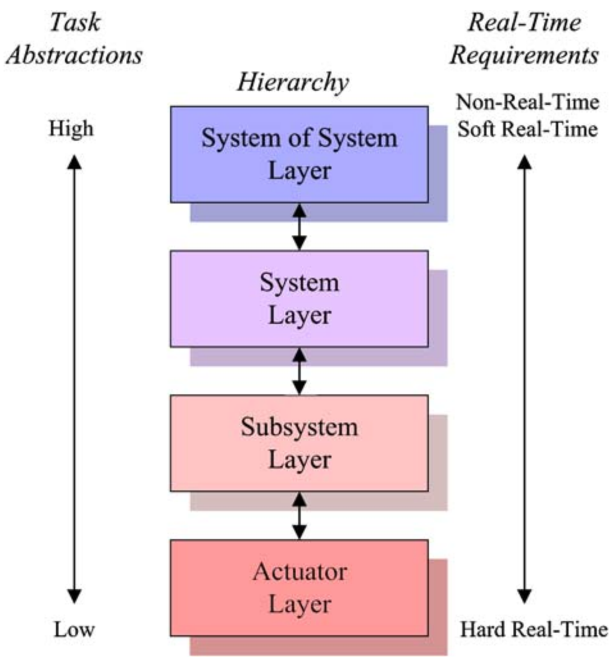

Fig. 9 Layers of the networked UAS software hierarchy

Actuator Layer The actuator control layer provides access to the actuators via driver electronics and low-level device driver software modules. It is common to control the actuators by microcontroller based electronics boards. The small microcontroller boards often programmed with proprietary C compilers, and assemblers.

Table 1 Networked UAS Software Hierarchy

Control layers	HW&SW modules	SW Tools	RT requirements
SoS layer	WAN & LAN, IPC, comms.	Matlab, Simulink, Java, C/C++	Non real-time, soft real-time, event based tasks
System layer	Computers, WAN & LAN, IPC, comms.	Matlab, Simulink, Java, C/C++	Hard real-time, soft real-time, periodic tasks, event based tasks
Subsystem layer	Embedded computers, microcontroller boards, servo drivers, sensors, IPC, comms.	C/C++	Hard real-time, periodic tasks
Actuator layer	Microcontroller boards, PAL/PLD/FPGA peripheral devices, driver electronics, device drivers	Proprietary C, CUPL, HDL, VHDL, assembler	Hard real-time

The device driver software modules for these boards are carefully written, optimised codes with hard real time requirements tailored for particular hardware.

Subsystem Layer The subsystem layer provides low-level control and fusion behaviours. Low-level sensor data fusion, such as INS-GPS fusion for navigation solutions, and low-level servo controls such as PID servo loops are typical tasks at this level. This layer is built around multiple embedded computers, or microcontroller boards, connected to servo drivers and sensors.

The tasks at the subsystem layer are often programmed in C/C++ and run at a constant rate in hard real-time. The concurrent tasks running on a single embedded computer or on multiple networked embedded computers use Inter-Process Communication (IPC) mechanisms for data exchange.

System Layer The system layer provides high level control and decision making functionalities. Some of the tasks at this layer are the system level diagnostics and error handling, path planning, way-point navigation and guidance. These tasks could be periodic tasks or event based tasks with soft real-time or hard real-time requirements. Although most of the software code for the system layer tasks are hand written in C/C++, other higher level languages and graphical tools like MatLab and Simulink can also be used to auto generate the C/C++ codes [29, 31, 32].

The system layer consists of heterogenous set of computers, networked together via wide area network (WAN) and/or local area network (LAN). The networked computers in this layer also use various IPC and high level communication mechanisms for data exchange between multiple processes.

System of Systems Layer The System of Systems (SoS) layer is the highest level of control in the control hierarchy. Compared to the lower layers, the real-time requirements of this level are eased. The tasks at the SoS layer are concentrated on the interaction and interoperability of multiple systems. Humans may also be in the control loop in the SoS layer.

Many UASs have inherently distributed system architectures. In order to achieve interoperability at the SoS layer, the software associated with each individual system should meet a common set of requirements for their interfaces with each other. Often, multiple heterogeneous vehicles, which are complex systems themselves, and other systems, such as stationary sensors networks, operate together to fulfill the mission objectives in a dynamic environment. Communication plays a major role in their interactions since cooperation (often [33, 34]) requires communication between the vehicles [35, 36].

Depending on the communication requirements, layers can be linked together by using a number of different inter-process communication (IPC) methods, such as shared memory, message passing, pipes, sockets, and remote procedure calls (RPC) etc. or higher level middlewares. Middleware is a suite of software that provides the connectivity between multiple otherwise isolated applications.

The difficulties of using different IPC mechanisms and middlewares should not be underestimated. Different IPC techniques have strengths, weaknesses, and traps that the software developers should manage in order to facilitate reliable and effective communication between tasks.

Mastering the intricacies of some of the traditional IPC techniques is a challenging task that requires time and effort, and above all experience. Developers who have the expertise in control of autonomous systems may not necessarily have the sufficient level of expertise in IPC. The CommLibX/ServiceX middleware [37, 38] which is detailed in the next section, addresses this problem and offers a simple, yet powerful way of IPC for the UAS software developers.

6 CommLibX/ServiceX Middleware

Increasing complexity of autonomous systems and the adoption of network-centric operational philosophy are motivating the development of large scale distributed systems. The non-distributed/stand-alone system approach is becoming a thing of the past.

Distributed system software components are glued together with middleware. Middleware is the systems software connecting the distributed software components, applications, and simulation modules together. As illustrated in Fig. 10 the middleware system often resides between the operating system and the high level distributed applications.

Middleware hides the complexities and the heterogeneities of the underlying operating systems, network protocols, and hardware of the target computers. It enhances the efficiency and quality of the distributed system development work and simplifies the maintainability.

There are many different middlewares available in the software market today. They can be grouped according to their application domains, language support, and real-time characteristics. Some of them are designed to have target specific requirements of a particular application domain, such as distributed simulation [39], and the others are general purpose [40].

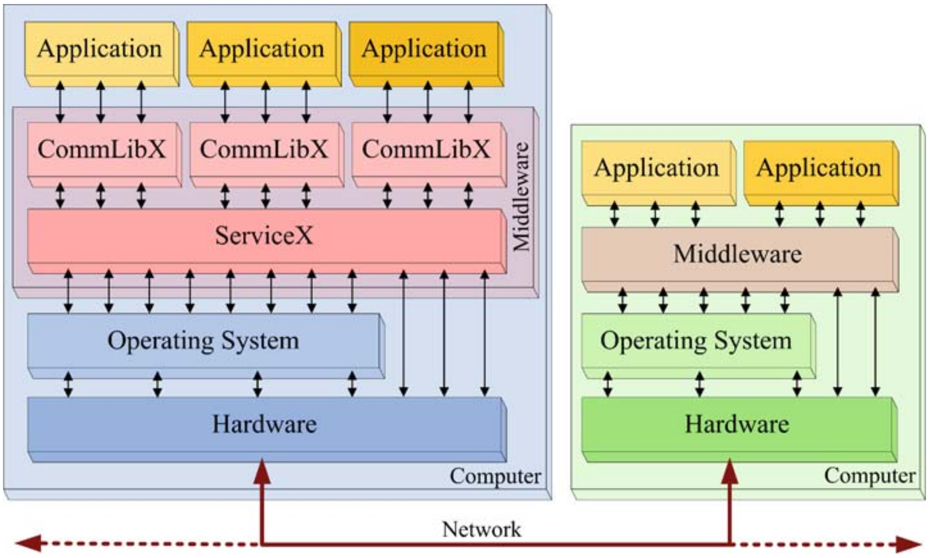

Fig. 10 CommLibX/ServiceX is the middleware, designed and developed to support the seamless network UAS research works at ACFR

The CommLibX/ServiceX is a novel, layered software framework designed as a middleware to provide hardware, operating system, and programming language abstraction for the communication, and other system services required for a networked UAS. The CommLibX/ServiceX middleware supports the seamless network UAS research works at the ACFR.

Originally, the CommLibX/ServiceX middleware is designed in UAS applications in mind. However, it is used on a number of other autonomous systems, and distributed simulation environments. The CommLibX/ServiceX middleware has very flexible and modular architecture. Its base code is small in size, and through the plug-in modules, its functionality can be tailored for the specific application domain. The CommLibX/ServiceX middleware supported on QNX, Linux, and Windows operating systems. A cut-down version of the CommLibX/ServiceX middleware is also demonstrated on low end small Atmel AVR microcontrollers with no operating system.

One of the most important objectives of the CommLibX/ServiceX middleware architecture is to enable the application developers to develop high-level distributed applications, much the same way, as if they were developing stand-alone, non-distributed applications. This assists the developers in focussing on their applications rather than the elaborately complex details of the underlaying operating systems and hardware.

The CommLibX/ServiceX middleware consists of two major parts; the CommLibX library and the ServiceX module (Fig. 10). The CommLibX is the interface between the high level distributed application and ServiceX. As its name implies ServiceX provides the middleware services including communication, task scheduling, and some limited direct hardware control. Each application initiates an instance of the CommLibX. The high level applications invoke operations on ServiceX via the CommLibX instance. And similarly, the ServiceX invokes operations on the high-level application through the same instance of the CommLibX.

The separation of the middleware into two parts as CommLibX and ServiceX has another benefit; CommLibX “isolates” ServiceX from the high-level applications. Execution of ServiceX does not depend on the state of the user application. What this means in practice is that an ill-behaved user application does not affect the performance of ServiceX. Hence, a faulty program does not jeopardize the operation of the rest of the distributed system.

In a CommLibX/ServiceX based distributed system, high-level applications may run on either one or multiple computers. Often, high-level applications are not aware if the other applications are running on the same computer or not. Regardless if they are running on the same computer or on separate computers, high-level applications do not directly interact with each other, but the interaction occurs through *virtual channels*. The virtual channel is a novel concept introduced by the CommLibX/ServiceX middleware. The virtual channels are the logical links between communicating nodes. Often they are used for the logical separation of different kinds of data packages. The virtual channels can be allocated either a single communication medium or they can be spread over all the communication mediums available to the distributed system.

Figure 11 illustrates a simplified version of a high-level information dissemination map of a networked UAS. Figure 4 and Fig. 5 would also help for the interpretation of the information dissemination process. The information flow between

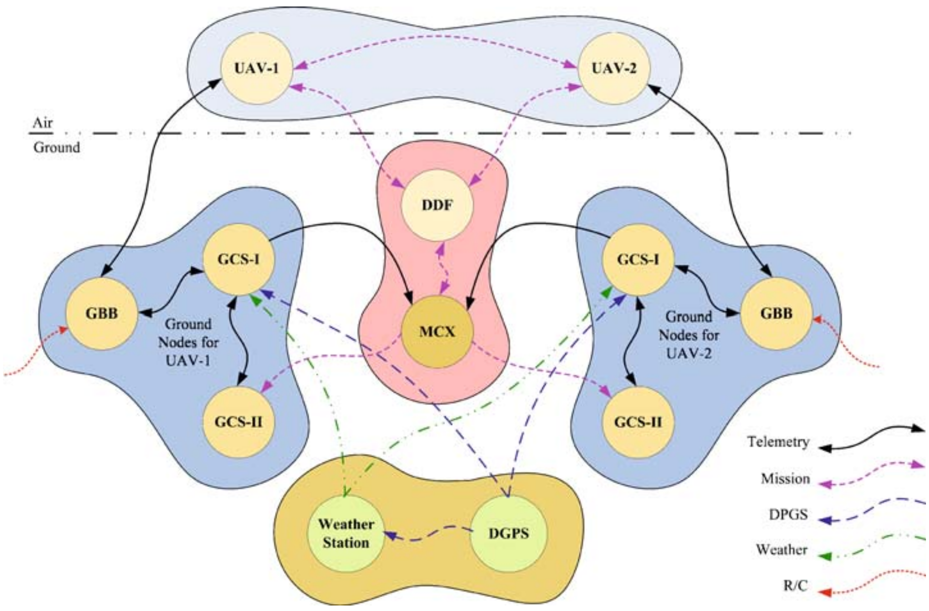

Fig. 11 Information dissemination map of a networked UAS with two UAVs

communicating entities are represented by the directed arrows. The line patterns of the arrow indicate the context of the information/data that will be communicated between nodes via virtual channels.

Intentionally, Fig. 11 does not show the physical properties of communication medium or the absolute physical locations of the applications constituting the communicating nodes. The information dissemination map is what the application developers need to know about the high level interactions between subsystems. Depending on the final physical configuration of the system, the virtual channels can be allocated and reallocated on any physical medium (cable or wireless Ethernet, Controller Area Network (CAN), shared memory etc.) available to the system. And the high-level application developers do not need to have any assumption about the physical nature of the communication channel. Furthermore, hardware configuration of the system can be changed, even at runtime, without changing the source code.

Virtual channels have event driven interfaces which ensures that upon receiving a data package/message from a virtual channel, the CommLibX invokes a user call-back function for data transfer.

As illustrated in Fig. 12, ServiceX provides middleware services and message transfer between CommLibX instances. ServiceX links the physical communication layers to the CommLibX library. ServiceX supports various networking devices including the CAN, standard serial ports (RS232/RS422/RS485), cable and wireless Ethernet, shared memory etc. It also supports multiple protocols including UDP, TCP/IP, and additional communication hardware and protocols can be interfaced to ServiceX through plug-in-modules.

For the applications running on the same computer, ServiceX maps virtual channels to the shared memory for maximum throughput. Virtual channels can be

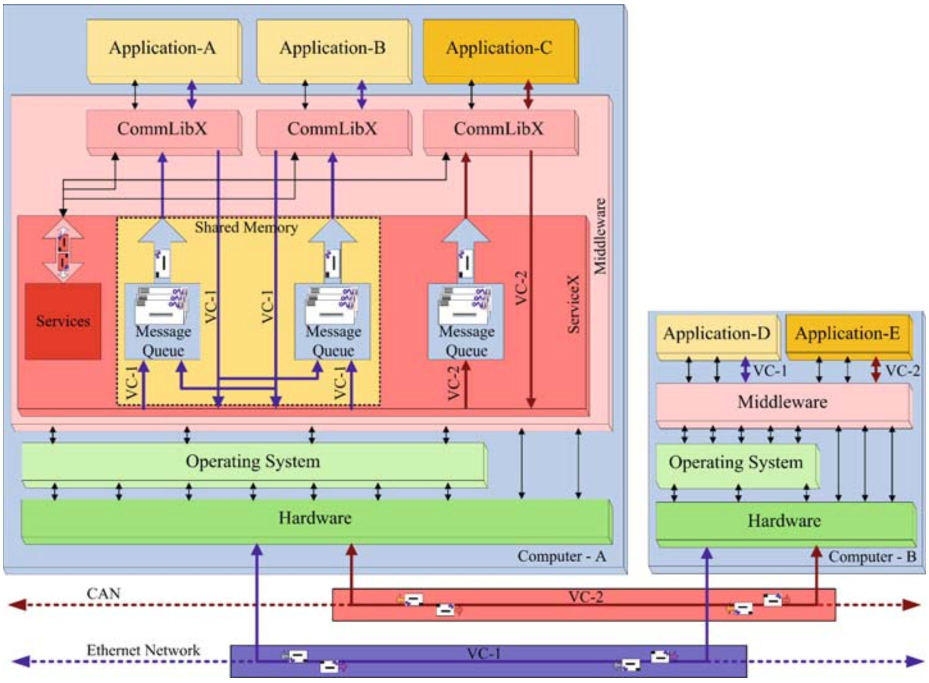

Fig. 12 Virtual channels and message exchange in CommLibX/ServiceX middleware

allocated on the same or different communication devices for applications running on separate networked computers.

The mapping of virtual channels to different physical mediums can be better explained by an example. Figure 12, shows two computers: “Computer-A”, and “Computer-B”. The first computer runs three high-level applications (A, B, C) and the other computer runs two high-level applications (D, E). Applications A,B and D communicate with each other over the virtual channel “VC-1”. Similarly, applications C and E communicate via virtual channel “VC-2”.

Computer-A, and Computer-B share two physical communication mediums; a CAN bus, and an Ethernet network. VC-1 is mapped to the Ethernet network while VC-2 is mapped to the CAN bus. Since application A, and B are running on the same computer (Computer-A), and using the same virtual channel (VC-1), then, regardless of the external communication mediums, the ServiceX links the application A and B over the shared memory. However, as shown in Fig. 12, VC-1 is also mapped to the Ethernet so that application D can also communicate with applications A and B.

Figure 13 shows ServiceX’s multi-layered, hierarchical architecture. The layered architecture of ServiceX simplifies its port to different operating systems and different hardware. It consists of following three layers;

OS and HW Adaptation Layer This is the lowest level and provides interfaces to the operating system and to the hardware. A significant portion of this layer should be re-written when ServiceX needs to be ported to a new operating system or to a new computing hardware.

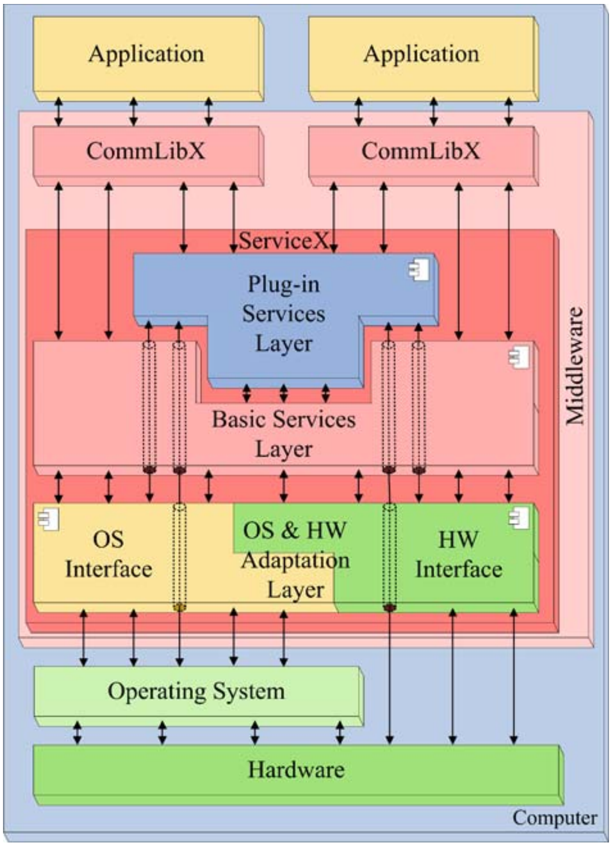

Fig. 13 ServiceX has multi-layered architecture, and allows its features to be extended through plug-in services

Basic Services Layer The basic services layer resides on top of the adaptation layer. As its name implies, the basic services layer provides bare minimum services for CommLibX/ServiceX middleware. These include, creation and utilisation of virtual channels, data logging, message packing/unpacking, basic communication statistics, support for basic communication hardware including Ethernet, CAN, standard serial ports and shared memory.

Plug-in Services Layer The plug-in services layer resides on top of the basic services layer. The features of ServiceX, hence the features of the overall CommLibX/ServiceX middleware, can be extended by incorporating plug-in services.

All the distributed software applications running on board the UAVs and on the ground station use CommLibX/ServiceX middleware. We also used CommLibX/ServiceX in our real-time mission simulator, and HWIL simulator. The next section gives an overview of our distributed real-time simulator architecture.

7 Real-Time Multi-UAV Simulator

In UAV experiments there are many avenues of failure: in software, hardware, and conceptual understanding and failures in algorithm development are just a few of many probable failure points. These pose significant burdens on the progress of the project as each flight test poses a risk to the survivability. Hence, any new hardware or software module has to be thoroughly tested before used on real missions. A *real time multi-UAV simulator* (RMUS) system has been developed, at the earlier phase of ACFR's UAV research programs to address the problems associated with real life experiment [37].

The RMUS has been implemented as a testing and validation mechanism for our networked UAS. These mechanisms include off-line simulation of complex scenarios, HWIL tests, mission rehearsals, on-line mission control for real UAS demonstrations, and validation of real test results. Prior to real flight operations, all algorithms, software implementations, their interaction with each other, and the actual hardware that will be used in the flight missions should be extensively tested in simulation [41]. The software and hardware, once validated, are then ported directly to the UAV platforms ready for real flight tests.

The RMUS has been extensively used in the various phases of the development of all single-UAV and multi-UAV flight operations conducted by the ACFR [18, 22, 24, 27, 31, 32, 38, 42–45]. These real flight operations include the demonstrations of various DDF, SLAM, and cooperative control techniques. The RMUS has been also used in a large number of other simulation-only UAV experiments [35, 46–48].

The RMUS architecture encompasses distinct architectural styles; it is distributed, it is multi-layered and it is event-based with message-passing foundations. The RMUS's architecture promotes component-based, modular, distributed simulation development where the simulation modules interact with each other with message passing mechanisms provided by the CommLibX/ServiceX middleware.

The key terminology and concepts comprising the RMUS architecture are as follows;

Simulator Cluster A simulator cluster is a group of distributed simulation modules, and other hardware or software resources connected together that act as a single system to provide a wide span of resources and high processing power for complex simulations. The size and content of the cluster can be changed as the simulation requirements may vary.

Figure 14 illustrates a typical architecture of a RMUS cluster. More than one simulator cluster can be linked with each other via CommLibX/ServiceX middleware. Figure 15 illustrates two linked RMUS clusters. If a RMUS cluster should be interfaced with another simulation environment with a different protocol standard, it may require a "bridge" application on each ends of the clusters to convert from one protocol standard to the other.

Simulation Modules Simulation modules are composed of either software applications or physical devices comprising the simulation cluster. A simulation module may be an implementation of an algorithm such as the generation of Voronoi diagrams, the DDF algorithm, SLAM code, or a flight control code, or a physical device such as GPS, R/C unit, radar electronics etc. interfaced to the simulation cluster. In Fig. 14 and in Fig. 15, major simulation modules are represented as square shaped blocks. The functionality of simulation modules can be enhanced through *Plug-in-Modules*. The rectangle blocks in Fig. 14 and in Fig. 15 show the plug-in-modules. The plug-in-modules are often created as dynamic libraries.

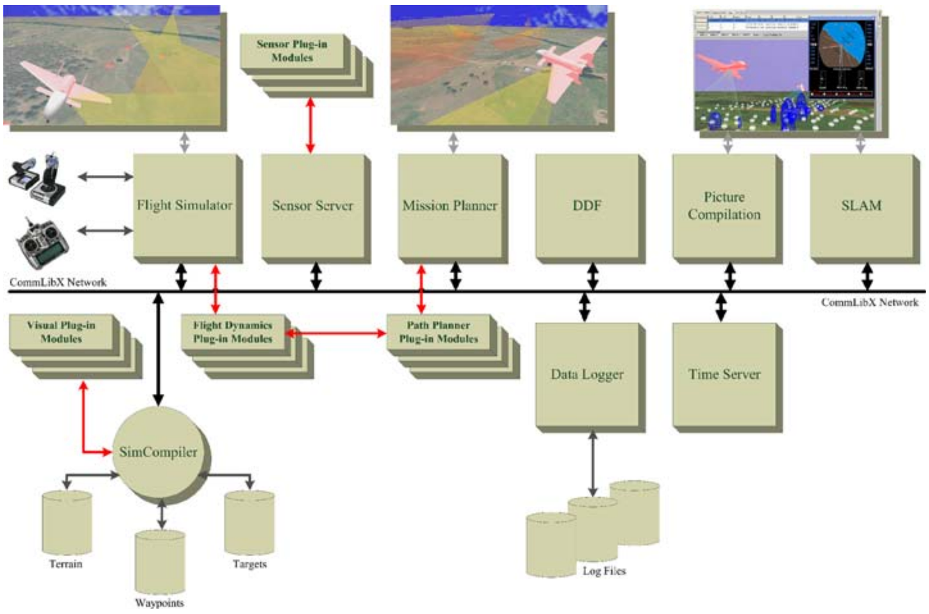

Fig. 14 An architectural view of a typical real time multi-UAV simulator (RMUS) cluster

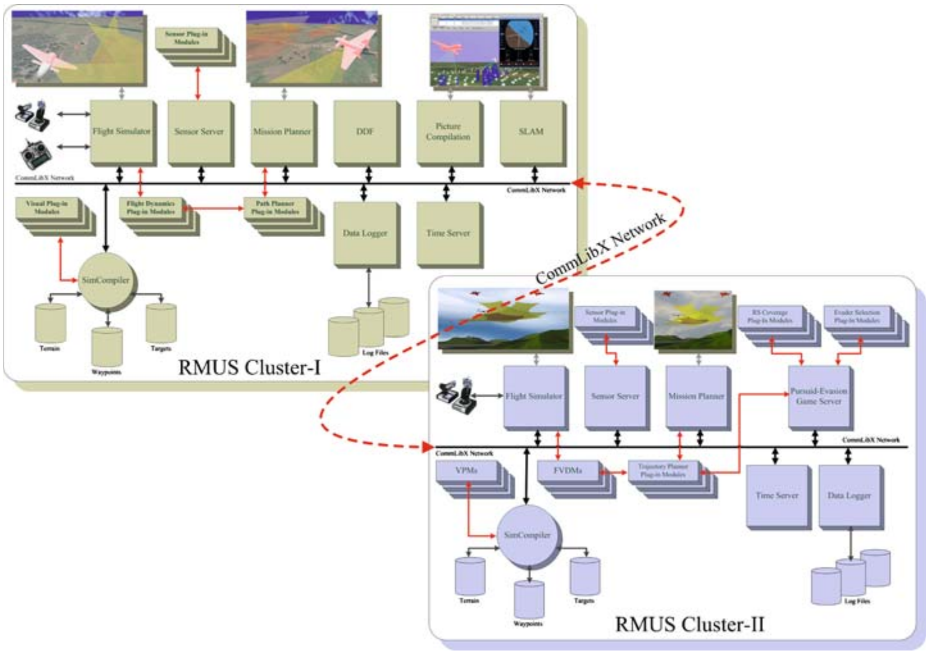

Fig. 15 Two RMUS clusters with different simulation modules, running different simulation sessions are linked with each other via CommLibX network

Simulation Objects Simulation objects are instances of simulation entities. The simulation objects interact with each other as well as interact with the physical devices, via message passing mechanism. Simulation objects may have 3D graphic appearances, or they may be non-visual. For example, in a RMUS simulation session, a simulated radar (a non-visual simulation-object) carried on a simulated UAV (visual simulation-object) may detect a simulated land vehicle (another visual simulation-object) on a terrain (yet another visual simulation-object) as a Feature of Interest (FoI). The 3D appearance of the simulation objects are defined by Visual-Plug-in-Modules (VPMs).

Simulation objects may be associated with pre-defined behaviors or their behavior can be (re)defined either through plug-in-modules or through the interaction with the other simulation objects. Consider a simple scenario; a UAV of a particular type may have a predefined flight dynamics model, and its flight can be simulated by using this model. The same UAV simulation object may receive a “kill the engine”, and a “deploy parachute” pseudo commands from the GCS object. In such circumstances, the UAV object may load separate plug-in modules defining the flight dynamics of the deployed parachute. During the rest of the simulation, the flight path of the UAV can be calculated based on the flight dynamics model of the parachute rather than the flight dynamics model of the UAV.

Simulation Session Simulation session is the participation of simulation modules and execution of the simulation to achieve the aimed simulation tasks. Depending on the complexity and processing power requirements of the simulation, simulation sessions can be initiated on a single computer or on multiple networked computers.

Simulation session may be of the type HWIL, “human-in-the-loop” (HIL), or mission rehearsals etc. The physical devices are directly utilized in HWIL simulation sessions. Direct human interaction with the simulation is also possible. A typical example of HWIL and HIL simulation session would involve a human pilot using a real R/C unit to fly a simulated UAV object over a simulated terrain to capture aerial images of the FoIs.

The HWIL simulation environment used at the ACFR is constructed around the RMUS. Hence, both the single-UAV, and multi-UAV missions can be simulated in real-time.

Figure 16 shows the HWIL Sim system configuration for two Brumby Mk III UAVs. In order to reflect the real operational system configuration at a maximum level, the distributed HWIL Sim system is configured in such a way that it has almost identical architecture with the system hardware used for the real flight missions. By comparing Figure 16 with Figure 5 one can see the similarities and differences between the HWIL Sim architecture and the architecture of the ground station used in real flight trials. (The former figure shows a HWIL Sim for two UAVs and the latter figure shows the system architecture for four UAVs, hence comparison should be made by per-UAV basis.)

During the HWIL simulations, the real UAVs remain stationary. Therefore, the flight sensors can not produce flight data. The “HWIL-I” and “HWIL-II” computers concurrently run flight vehicle dynamics models to simulate the UAV motions. The sensor data (GPS, IMU, air data, RPM etc.) are generated in real-time based on these models. Synchronous operation capability is often required for the HWIL computers. The system wide synchronousity is achieved through the Time Management Services (TMS) of the CommLibX/ServiceX middleware.

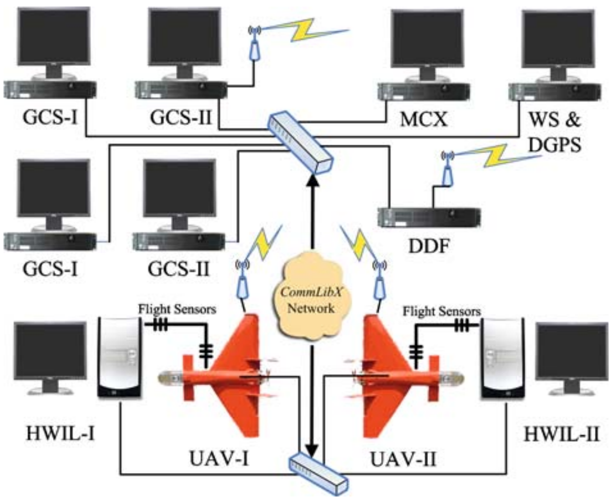

Fig. 16 The hardware-in-the-loop simulation (HWIL Sim) system configuration for two Brumby Mk III UAVs

The software applications for the HWIL simulation environment has been developed based on AFDAS design drivers. Therefore, the HWIL Sim environment accommodates software applications running on multiple operating systems (QNX, Linux, and Windows) developed in multiple programming languages (MatLab, C/C++, and Delphi) and provide support for a number of communication hardware and software mediums (RS232, RS422, CAN, cable and wireless Ethernet, spread spectrum radio modem).

Figure 17 is the photograph of the HWIL Sim set-up for two Brumby Mk III UAVs. Note that the wings, the propeller, and the nose cone covers are removed from the UAVs and they are elevated on the custom made stands for the easy access to internals. The cables connected to the UAVs are providing the electrical power to the payload electronics, network connection, and flight sensor connections.

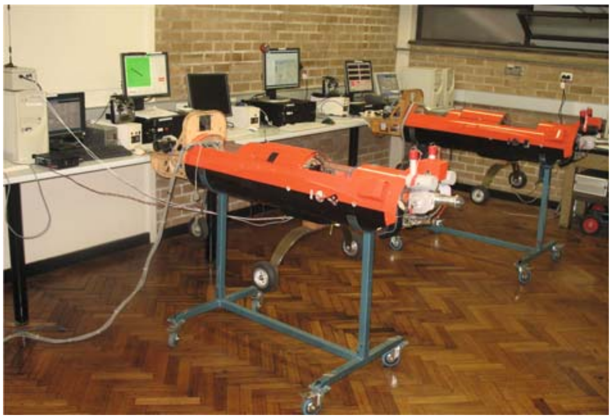

Fig. 17 The HWIL simulation set-up for two Brumby Mk III UAVs

8 Conclusion and Future Works

Due to the advancements in electronics, sensor, and communication technologies and evolutions in material science, and manufacturing technologies, UAVs are becoming more accessible by academic R&D institutions. The UAS developers can easily acquire highly standardised COTS modular hardware components from various suppliers. However, it is difficult to find well established standardised software components particularly for widely distributed, networked UASs.

This paper presented a software development methodology for a networked UAS. The methodology is strongly based on the use of real-time multi-UAV simulator (RMUS), distributed HWIL simulator, and CommLibX/ServiceX middleware. The HWIL simulator and the RMUS have been demonstrated on a large number of pre-mission planning tests as well as on the post mortem analysis of the flight missions by replaying the logged mission data, and comparing it with the simulation results for the re-evaluation of the system models.

The CommLibX/ServiceX is a novel middleware which provides hardware, operating system, and programming language abstraction for the communication, and other system services required for a networked UAS. The CommLibX/ServiceX middleware enables the application developers to develop high-level distributed applications, much the same way, as if they were developing stand-alone, non-distributed applications. Hence, the developers can focus on their applications rather than the low-level details of the underlaying operating systems and hardware.

All these software are based on A framework for distributed autonomous systems (AFDAS). AFDAS uses domain decomposition approach to the domain of networked UAS and introduce a layered system software view. It examines the UAS software applications in four layers; actuator, subsystem, system, and System of Systems (SoS) layers.

The presented software development methodology successfully has been used for many years on widely distributed, network UAS, consisting of multiple Brumby Mk III UAVs, a comprehensive ground station, containing network of computers. The Brumby Mk III UAV avionics is also comprised of a network of computers, sensors, and communication subsystems.

Both the RMUS and the CommLibX/ServiceX middleware are open to expansion. Development of additional new simulation modules for the RMUS is an ongoing process as more researchers use the RMUS in their research. Current activities include the building hovering UAVs (HUAVs) to our existing multi-UAV system to build widely distributed, and decentralised, networked UAS with heterogenous UAVs.

References

1. NATO, STANAG 4586 (Standard Interfaces of UAV Control System (UCS) for NATO UAV Interoperability), NATO Standardization Agency (NSA) (2004)
2. Ernst, D., Valavanis, K., Garcia, R., Craighead, J.: Unmanned vehicle controller design, evaluation and implementation: from MATLAB to printed circuit board. J. Intell. Robot. Syst. **49**, 85–108 (2007)
3. Song, S.J., Liu, H.: Hardware-in-the-loop simulation framework design for a UAV embedded control system. In: Control Conference CCC 2006, pp. 1890–1894 (2006)

4. Gösta, P.D., Kuchcinski, G.K., Sandewall, E., Nordberg, K., Skarman, E., Wiklund, J.: The WITAS unmanned aerial vehicle project. In: ECAI 2000. Proceedings of the 14th European Conference on Artificial Intelligence, pp. 747–755. Berlin (2000)
5. Henning, M.: The rise and fall of CORBA. ACM Queue **4**, 28–34 (2006)
6. Maffeis, S., Schmidt, D.C.: Constructing reliable distributed communication systems with CORBA. Commun. Mag., IEEE **35**, 56–60 (1997)
7. Henning, M., Vinoski, S.: Advanced CORBA programming with C++: Addison-Wesley (1999)
8. Brooks, A., Kaupp, T., Makarenko, A., Williams, S., Orebaeck, A.: Orca: a component model and repository. In: Brugali, D. (ed.) Software Engineering for Experimental Robotics, vol. 30 of STAR, pp. 231–251. Springer (2007)
9. Corke, P., Sikka, P., Roberts, J., Duff, E.: DDX: a distributed software architecture for robotic systems. In: Australasian Conference on Robotics and Automation. Canberra Australia (2004)
10. Paunicka, J.L., Corman, D.E., Mendel, B.R.: A CORBA-based middleware solution for UAVs. In: Fourth IEEE International Symposium on Object-oriented Real-time Distributed Computing, ISORC-2001, pp. 261–267. Magdeburg Germany (2001)
11. Jangy, J.S., Tomlinz, C.J.: Design and implementation of a low cost, hierarchical and modular avionics architecture for the DragonFly UAVs. In: Proceedings of the AIAA Guidance, Navigation, and Control Conference. Monterey (2002)
12. Wills, L., Kannan, S., Heck, B., Vachtsevanos, G., Restrepo, C., Sander, S., Schrage, D., Prasad, J.V.R.: An open software infrastructure for reconfigurable control systems. In: Proceedings of the American Control Conference, pp. 2799–2803 (2000)
13. Kuo, Y.-h., MacDonald, B.A.: Designing a distributed real-time software framework for robotics. In: Australasian Conference on Robotics and Automation (ACRA). Canberra (2004)
14. Croak, T.J.: Factors to consider when selecting CORBA implementations, CrossTalk. J. Def. Softw. Eng. **14**, 17–21 (2001)
15. Group, O.M.: Common object request broker architecture—for embedded, Draft Adopted Specification. http://www.omg.org/docs/ptc/06-05-01.pdf (2006)
16. Boehm, B.W.: Seven basic principles of software engineering. J. Syst. Software **3**, 3–24 (1983)
17. Department of Defense, D.5000.61, DoD modeling and simulation verification, validation, and accreditation. http://www.cotf.navy.mil/files/ms/DoDI500061{_}29Apr96.pdf (2003)
18. Sukkarieh, S., Nettleton, E., Kim, J.-H., Ridley, M., Göktoğan, A.H., Durrant-Whyte, H.: The ANSER Project: data fusion across multiple uninhabited air vehicles. Internat. J. Robot. Research **22**, 505–539 (2003)
19. Göktoğan, A.H.: A software framework for seamless R&D of a networked UAS. In: PhD Thesis, Australian Centre for Field Robotics, School of Aerospace, Mechanical and Mechatronic Engineering Sydney. The University of Sydney (2007)
20. Kim, J., Sukkarieh, S.: Autonomous airborne navigation in unknown terrain environments. IEEE Trans. Aeros. Electron. Syst. **40**, 1031–1045 (2004)
21. Nettleton, E.: ANSER past and present—multi UAV experimental research. In: The IEE Forum on Autonomous Systems, (Ref. No. 2005/11271), p. 9 (2005)
22. Nettleton, E., Ridley, M., Sukkarieh, S., Göktoğan, A.H., Durrant-Whyte, H.: Implementation of a decentralised sensing network aboard multiple UAVs. Telecommun. Syst. **26**(2–4), 253–284 (2004)
23. Ridley, M., Nettleton, E., Göktoğan, A.H., Brooker, G., Sukkarieh, S., Durrant-Whyte, H.F.: Decentralised ground target tracking with heterogeneous sensing nodes on multiple UAVs. In: The 2nd International Workshop on Information Processing in Sensor Networks (IPSN'03), pp. 545–565. Palo Alto, California, USA (2003)
24. Sukkarieh, S., Göktoğan, A.H., Kim, J.-H., Nettleton, E., Randle, J., Ridley, M., Wishart, S., Durrant-Whyte, H.: Cooperative data fusion and control amongst multiple uninhabited air vehicles. In: ISER'02, 8th International Symposium on Experimental Robotics. Sant'Angelo d'Ischia, Italy (2002)
25. Bryson, M., Sukkarieh, S.: Toward the real-time implementation of inertial SLAM using bearing-only observations. In: Journal of Field Robotics (2006)
26. Buttazzo, G.C.: Hard real-time computing systems: predictable scheduling algorithms and applications. In: 2nd ed. p. 425. New York, Springer (2005)
27. Nettleton, E.: Decentralised architectures for tracking and navigation with multiple flight vehicles. In: PhD Thesis, Department of Aerospace, Mechanical and Mechatronic Engineering Sydney. The University of Sydney (2003)
28. Kim, J.-H.: Autonomous navigation for airborne applications. PhD Thesis, Australian Centre for Field Robotics, School of Aerospace, Mechanical and Mechatronic Engineering Sydney. The University of Sydney, p. 237 (2004)

29. Cole, D.T., Göktoğan, A.H., Sukkarieh, S.: The implementation of a cooperative control architecture for UAV teams. In: 10th International Symposium on Experimental Robotics (ISER'06). Rio de Janeiro, Brazil (2006)
30. Endsley, M.R., Garland, D.J.: Theoretical underpinning of situation awareness: a critical review. In: NJ, M., Erlbaum, L. (eds.) Situation Awareness Analysis and Measurement: Analysis and Measurement, pp. 3–33 (2000)
31. Cole, D.T., Sukkarieh, S., Göktoğan, A.H.: System development and demonstration of a UAV control architecture for information gathering missions. J. Field Robot. **23**, 417–440 (2006)
32. Cole, D.T., Sukkarieh, S., Göktoğan, A.H., Hardwick-Jones, R.: The development of a real-time modular architecture for the control of UAV teams. In: Field and Service Robotics, FSR'05, pp. 321–332. Port Douglas, Australia (2005)
33. Michael, R.G., Matthew, L.G., Jeffrey, S.R.: Cooperation without communication. In: Distributed Artificial Intelligence, Morgan Kaufmann Publishers Inc., pp. 220–226 (1988)
34. Otanez, P.G., Campbell, M.E.: Hybrid cooperative reconnaissance without communication. In: 44th IEEE Conference on Decision and Control'05 and European Control Conference'05. CDC-ECC '05, pp. 3541–3546 (2005)
35. Chung, C.F., Göktoğan, A.H., Cheang, K., Furukawa, T.: Distributed simulation of forward reachable set-based control for multiple pursuer UAVs. In: SimTecT 2006 Conference Proceedings, pp. 171–177. Melbourne, Australia (2006)
36. Speranzon, A., Johansson, K.H.: On some communication schemes for distributed pursuit-evasion games. In: Proceedings of 42nd IEEE Conference on Decision and Control, pp. 1023–1028 (2003)
37. Göktoğan, A.H., Nettleton, E., Ridley, M., Sukkarieh, S.: Real time Multi-UAV simulator. In: IEEE International Conference on Robotics and Automation (ICRA'03), pp. 2720–2726. Taipei, Taiwan (2003)
38. Göktoğan, A.H., Sukkarieh, S., Işıkyıldız, G., Nettleton, E., Ridley, M., Kim, J.-H., Randle, J., Wishart, S.: The real-time development and deployment of a cooperative multi-UAV system. In: ISCIS03 XVIII - Eighteenth International Symposium on Computer and Information Sciences, Antalya—Türkiye, pp. 576–583 (2003)
39. DMSO: High level architecture for simulation interface specification, defence modelling and simulation office (DMSO). https://www.dmso.mil/public/ (1996)
40. O.M.G. (OMG): CORBA: Common Object Request Broker Architecture. http://www.corba.org
41. Göktoğan, A.H., Sukkarieh, S.: Simulation of multi-UAV missions in a real-time distributed hardware-in-the-loop simulator. In: Proceeding of the 4th International Symposium on Mechatronics and its Applications (ISMA07). Sharjah, UAE (2007)
42. Göktoğan, A.H., Brooker, G., Sukkarieh, S.: A compact millimeter wave radar sensor for unmanned air vehicles. In: Preprints of the 4th International Conference on Field and Service Robotics, pp. 101–106. Lake Yamanaka, Yamanashi, Japan (2003)
43. Nettleton, E.W., Durrant-Whyte, H.F., Gibbens, P.W., Göktoğan, A.H.: Multiple platform localisation and map building. In: Sensor Fusion and Decentralised Control in Robotic Stystems III, pp. 337–347. Boston, USA (2000)
44. Sukkarieh, S., Yelland, B., Durrant-Whyte, H., Belton, B., Dawkins, R., Riseborough, P., Stuart, O., Sutcliffe, J., Vethecan, J., Wishart, S., Gibbens, P., Göktoğan, A.H., Grocholsky, B., Koch, R., Nettleton, E., Randle, J., Willis, K., Wong, E.: Decentralised data fusion using multiple UAVs - The ANSER Project. In: FSR 2001, 3rd International Conference on Field and Service Robotics, Finland, pp. 193–200 (2001)
45. Göktoğan, A.H., Sukkarieh, S.: Role of modelling and simulation in complex UAV R&D projects. In: 1st National Defense Applications, Modeling and Simulation Conference, (USMOS'05), pp. 157–166. Ankara-Türkiye (2005)
46. Göktoğan, A.H., Sukkarieh, S., Cole, D.T., Thompson, P.: Airborne vision sensor detection performance simulation. In: The Interservice/Industry Training, Simulation and Education Conference (I/ITSEC'05), pp. 1682–1687. Orlando, FL, USA (2005)
47. Bourgault, F., Göktoğann, A.H., Furukawa, T., Durrant-Whyte, H.: Coordinated search for a lost target in a Bayesian world. In: Advanced Robotics, Special Issue on Selected Papers from IROS 2003, vol. 18, pp. 979–1000 (2004)
48. Furukawa, T., Bourgault, F., Durrant-Whyte, H.F., Dissanayake, G.: Dynamic allocation and control of coordinated UAVs to engage multiple targets in a time-optimal manner. In: IEEE International Conference Proceedings on Robotics and Automation, pp. 2353–2358 (2004)