ONBOARD AUTONOMY OF CUBESAT CLUSTERS BASED ON SMARTPHONE TECHNOLOGY

Jian Guo[(1)]**, Jing Chu**[(2)]**, and Eberhard Gill**[(3)]

[(1)] *Faculty of Aerospace Engineering, Delft University of Technology, Kluyverweg 1, 2629 HS Delft, The Netherlands, Tel: +31-15-2785990, Email: j.guo@tudelft.nl*

[(2)] *Faculty of Aerospace Engineering, Delft University of Technology, Kluyverweg 1, 2629 HS Delft, The Netherlands, Tel: +31-15-2786098, Email: j.chu@tudelft.nl*

[(3)] *Faculty of Aerospace Engineering, Delft University of Technology, Kluyverweg 1, 2629 HS Delft, The Netherlands, Tel: +31-15-2787458, Email: e.k.a.gill@tudelft.nl*

Abstract: *This paper addresses the problem of onboard autonomy of satellite cluster utilizing smartphone technology. Several attempts have been made to assess the suitability of using smartphone in space. This paper is an extension of existing attempts to distributed space systems and, more specifically, in the context of a CubeSat formation flying mission called DelFFi. The DelFFi mission and its Concept of Operations are introduced. The onboard autonomy architecture of DelFFi is presented, which is based on a group of distributed ARM processors, the Android operating system (OS), and Multi-Agent System (MAS) technology with cooperative mission planning capability. The preliminary ground experimental results of implementing the onboard autonomy architecture are presented as well.*

Keywords: *Formation Flying; CubeSat; Onboard Autonomy.*

1. Introduction

In the recent two decades, very small satellites such as CubeSats are attracting more and more attentions from academia, industries and space agencies due to their low cost, short development cycle and promising capability. Currently, most CubeSat missions are used for technology demonstrations or education, which only explore the capability of an individual satellite. However, the capability of CubeSats can be extremely enhanced by flying a cluster of satellites, which is not a simple repeat of single satellite, especially from the operational point of view.

This paper addresses the onboard autonomy problem of Cubesat clusters in the context of DelFFi mission, a Cubesat formation flying mission within the QB50 project [1],[2]. The innovation of this paper is the utilization of smartphone technology for the onboard autonomy of a cluster of very small satellites.

In order to promote the spin-in of terrestrial technologies and reduce the complexity and costs of space systems, recently several projects have been kicked-off to test smartphones onboard CubeSats [3],[4]. This paper further extends the utilization of smartphone technology to the onboard autonomy of a cluster of very small satellites in the context of the DelFFi CubeSat formation flying mission. The paper is organized into three parts. In the first part, the DelFFi mission is introduced with the Concept of Operations (CONOPS) and the requirements of autonomy. The second part

investigates the onboard autonomy experimental architecture of DelFFi. In this part, an onboard autonomous architecture with cooperative mission planning capability is grounded on a group of distributed ARM processors, the Android operating system (OS), and Multi-Agent System (MAS) technology. In the final part of the paper, preliminary ground experimental results of implementing the onboard autonomy architecture are presented. The results from both the simulation and the hardware-in-loop test indicate that the architecture is efficient and has potentials.

2. The DelFFi Mission

QB50 is a unique project establishing an international network of 50 CubeSats for multi-point, in-situ measurements in the lower thermosphere and re-entry research [2]. As part of the FP7 QB50 consortium, the Delft University of Technology (TU Delft) intends to contribute two CubeSats, named Delta and Phi, to form the space segment of the so-called DelFFi mission, which will be an integral part of the QB50. Meanwhile, the DelFFi mission is the third satellite project within the Delfi program, a development line of nanosatellites for education, technology demonstration and research development at TU Delft [5].

The objective of the DelFFi mission is to demonstrate as part of QB50 autonomous formation flying between the two CubeSats Delta and Phi. This will be achieved using innovative concepts, methodologies and technologies. The other satellites of QB50 form a network with permanently changing relative positions and velocities. In contrast, DelFFi enables the autonomous control of the relative dynamics of Delta and Phi using various guidance, navigation and control technologies.

2.1. Concept of Operations

The ultimate validation of the DelFFi mission objectives will only be possible in space. Thus, successful operations of the DelFFi mission play a crucial role. The operations architecture of DelFFi comprises the space segment, the control segment and the ground segment. The space segment is composed of the Delta and Phi satellites which both will have independent telecommand and telemetry capabilities to the ground segment. In addition, for some scenarios an inter-satellite link (ISL) is used based on the AFF payload, described below. The ground segment comprises the ground station at TU Delft with UHF/VHF uplink and downlink capabilities. Telecommands to Delta and Phi will be uplinked through this ground station. Telemetry of Delta and Phi, comprising housekeeping and science data, will be received through the network of ground stations provided by QB50, including the Delft ground station.

The concept of operations foresees a separation of the two spacecraft from the orbital deployer, followed by a deployment of the spacecraft antennas and solar arrays. Both spacecraft will then undergo a check-out phase. The natural drift of the spacecraft will allow a precise characterization of the relative positions and their drift and help building-up the targeted separation. Once this separation has been achieved, a drift stop maneuver will be conducted followed by the acquisition phase of the formation and the

active control of the two spacecraft in a formation flying mode. Depending on the mission sequence and the needs from science and technology stakeholders, various formation flying configurations and baselines can be demonstrated. A key limitation of QB50, and thus of DelFFi, is the low altitude orbit which constraints the mission lifetime to several weeks. Together with uncertainties and risks associated with all elements of the system and a lack of experience from students involved in operations, this poses a tremendous challenge. To cope with that challenge, the design of the entire system will strive for maximum simplicity wherever possible, intentionally disregarding complex options which may be demonstrated towards the end of the mission, if the schedule allows.

2.1.1. Launch and early operation phase

The Launch and Early Operations Phase (LEOP) consists out of the launch, orbit insertion, detumbling, and checkout of the Delta and Phi spacecraft. According to the QB50 mission, the Delta and Phi spacecraft will be launched together with other 48 Cubesats on a launcher into an orbit of around 350 km altitude. The two spacecraft are not mated during launch. Collision of spacecraft is avoided through the design of the orbital deployer. After the separation, the two spacecraft will perform a series of check-out activities for both the platforms and the payloads.

2.1.2. Continued drift phase

During LEOP, the satellites will drift apart (mainly) due to differential aerodynamic drag. Depending on the differential ballistic coefficient and on the duration of LEOP, the along-track separation of the Delta and Phi spacecraft can be several hundreds of kilometers. Since it might be too early to implement the formation acquisition immediately after LEOP, the two spacecraft will keep natural drift for several days. During this period, the QB50 scientific payloads will be switched on regularly to gather scientific data. Therefore, this phase is a scientific observation phase without orbit control, like for all other satellites within the QB50 space segment.

2.1.3. Formation acquisition phase

The formation acquisition phase will commence when the two spacecraft achieve a certain along-track distance. This phase is expected to last for a minimum of 10 days. During this period, the relative position and drift will be determined and this phase ends with drift stop maneuvers. The final along-track distance of this phase will be around 1000 km.

2.1.4. Formation keeping phase

After the formation acquisition phase, the two spacecraft will have an inter-satellite distance of around 1000 km. Then the formation keeping phase will start. In the formation keeping phase, Delta and Phi will demonstrate the ability to autonomously maintain an along track separation of 1000±10 km for a period of 20-30 days. A suitable

control window size in this case would be 100 km which is kept with a control accuracy of 10 km. The relative navigation accuracy required for such a scenario is about 1 km which is easily achievable, e.g. using differential Two-Line Elements (TLE).

During the formation keeping phase, scientific observations will still be performed. This is of interest for scientists since this enables unique collection of thermosphere data from spacecraft with fixed baselines. As the other CubeSats realize continuously changing baselines, the fixed-baseline geometry of DelFFi is expected to provide valuable additional data which enhance the science case of QB50.

2.2. Requirements of Autonomy

According to the CONOPS of the DelFFi mission and the natural limitation of CubeSats, following requirements are applied to the onboard autonomy of the DelFFi mission:

- **Low-cost**: Cost is now a very important issue, sometimes the driver of space projects. For the DelFFi mission, the budget is extremely limited. Therefore the OnBoard Computer (OBC) and other subsystems that support onboard autonomy shall utilize low-cost components, and the development cost of onboard software should also be reduced.
- **Availability**: Low lifecycle cost and tight development schedule both indicate that the availability of resources shall be guaranteed. The resources include hardware, software, development tools, and relevant documents. For example, there should not be ITAR or other export restrictions, Commercial-Off-The-Shelf (COTS) products are preferred, etc.
- **Performance**: Autonomy brings much higher computational workload for OBC since the tasks like mission planning are very complicated. Processors with higher clock frequency, sometimes even multi-core processors are necessary. However, power consumption is also a concerning for CubeSats. Therefore power efficiency also shall be utilized as a performance measurement.
- **Real-time**: The staggered CONOPS allows relatively poor or even no direct inter-satellite link between the two satellites. Considering this together with relatively large inter-satellite distance, there will not be strict real-time requirement on autonomy. However on the other side this implies that the autonomy system onboard one satellite shall be able to decide its own activity without knowing the real-time information of the other satellite.
- **Reliability**: High reliability is an unforgettable requirement for the robustness of space systems. For the DelFFi mission, the reliability should not reply on rad-hard components but to be realized by innovative approaches, such as function redundancy, use of silicon technologies, the maturity and stability of the design and manufacturing process, and internal error detection and correction.

3. Onboard Autonomy

In space domain there have already been several attempts of autonomous operation, such as Deep Space One (DS1) mission [6], and Earth Observing One (EO1) mission

[7]. In this section, a brief review of onboard autonomy is provided, which is the theoretical fundamental of the autonomy architecture developed for DelFFi.

3.1. Autonomy of Single Spacecraft

Remote Agent (RA) onboard DS1 is the first autonomous control system to control a spacecraft without human supervision. RA has an onboard mission manager interpreting mission plans as high-level goals, a planning and scheduling engine generating a set of actives to achieve high-level goals, executive expanding actives to a sequence of commands to issue directly to the proper destination on the spacecraft, as well as executive monitors and a mode identification and reconfiguration engine to ensure the whole procedure goes correctly. The autonomy software onboard EO1 was developed to improve science return via the satellite's autonomous detecting and responding to science events occurring on the Earth.

RA successfully demonstrated the ability to plan onboard activities and correctly diagnose and respond to simulated faults in spacecraft components via its built in REPL environment. Autonomous control will enable future spacecraft to operate at greater distances from Earth, and to carry out more sophisticated science-gathering activities in deep space. Components of the RA software have been used to support other NASA missions. Major components of RA were a robust planner (EUROPA), a plan execution system (EXEC) and a model-based diagnostic system (Livingstone). EUROPA was used as a ground-based planner for the Mars exploration rovers. EUROPA II was used to support the Phoenix Mars Lander and the Mars Science Laboratory. Livingstone2 was flown as an experiment onboard EO1, and an F-18 at NASA Dryden Flight Research Center [8].

3.2. Autonomy of Distributed Space Systems

Apart from RA onboard DS1 and EO1 mission, the development of agents capable of autonomous planning, scheduling, execution and cooperation with other agents has received a lot of interest. While aforementioned research is regarding to the autonomy management of a single spacecraft, applications of MAS for Distributed Space System (DSS) are highly wide as well. A MAS approach to multiple satellite autonomy was presented in [9], and several organizations of satellites in a constellation were proposed. In another paper, the cousin of [9], those different organizations were compared [10]. Another application of MAS in the domain of space systems is the software implementation of estimation and control algorithms for DSS, such as ObjectAgent and TeamAgent [11].

Compared to single spacecraft missions, autonomous operations of DSS are more complicated. First, it's for a group of spacecraft flying in a cluster rather than just one single satellite, which would impose new constraints on each spacecraft such as collision avoidance, balance of propellant consumption, maintenance of wireless link and so on. Second, in the accomplishment of complex goals each spacecraft should cooperate with others to form the virtual spacecraft with required functionality. To

achieve that, spacecraft in the DSS must communicate with each other, which may suffer from propagation delay or even totally blocking out for extended period of time.

In the context of autonomous DSS rather than single spacecraft, it is worth mentioning the other extreme, NASA's ambitious swarm-based projects such as ANTS (Autonomous Nano-Technology Swarm) that consists of 1000 cooperative spacecraft to explore the asteroid belt [12]. In swarms there is no central controller directing the whole system and no one member has a global view. Crucial to achievements of the mission is the autonomic properties and the ability to autonomously change operations to adapt to the dynamic nature of the mission [13].

4. Multi-Agent System based Architecture

As aforementioned, autonomy is recognized as one of the primary requirements for DSS. To implement autonomous distributed systems in space, agent technology is a very promising candidate. During last decade Multi-Agent Systems (MAS) have been used to tackle problems that are difficult or impossible for an individual agent or a monolithic system to achieve. In this section autonomous distributed space systems are compared to MAS from the characteristic standpoint. After the comparison, the MAS based architecture of autonomous distributed space systems is presented.

4.1. MAS and Autonomous Distributed Space Systems

An agent can be characterized by autonomy, social ability, reactivity and proactivity. Specifically, each agent in MAS is autonomous, because it operates without the direct intervention of humans and has control over its actions and internal states; an agent is social, because it communicates with others to achieve its goals; an agent is reactive, because it perceives the environment and responds to the changes occurring in that environment; and also an agent is proactive, because besides simply responding to the environmental changes it would also take initiative to respond to those changes. Correspondingly, spacecraft in distributed space systems should possess the same characteristics as those of agents in MAS. The spacecraft is autonomous to operate properly under certain situations without direct intervention from ground station; it is social to cooperate with others via communication; it is reactive to handle uncertainties such as communication interruption, communication delay or others; and also it is proactive to perform the expected operations such as reconfiguration for function updating.

When a group of agents behaves by following simple strategies and at the same time sharing knowledge with each other, a multi-agent system is formed to pursue a desired global behaviour. MAS systems can be characterized as self-organization as well as self-steering and other control paradigms. Typically, MAS systems tend to find the best global solution to their problems without intervention. When it comes to the domain of Autonomous Distributed Space Systems (ADSS), it is often the case that the system maintains a configuration or reconfigures in orbit to perform designed operations. Therefore, a variety of global objectives, such as minimizing and balancing propellant

consumption, optimizing network topology, maximizing operational time and so on, should be taken into account. Apart from that, ADSS should relieve the workload of ground stations and reduce operation cost with multiple satellites. Once MAS technology is adopted by ADSS, the system is able to self-adapt, self-optimize and self-configure to dynamic changes in local environmental conditions.

4.2. MAS-based architecture of ADSS

The term architecture refers to an abstract description of a system, where components of the system and their interconnections are identified. Meanwhile, this description is presented in the framework of onboard software (OBSW) static architecture, where the top level building blocks of an OBSW are divided into, from bottom to top, operating system (OS) and hardware interface layer, data handling service layer and application layer [14]. In this paper the architecture of ADSS is based on MAS and smartphone technology. An ARM processor is selected as the onboard computer; Android is chosen as the onboard OS; the spacecraft is modelled as an agent and the MAS run-time is incorporated into the data handing service layer; and applications in the application layer are implemented as behaviours of the spacecraft agent. It should be pointed out that the MAS architecture is intrinsically peer to peer, as any agent is able to sponsor communication with any other agent or be the receiver of an incoming communication at any time. It should be also mentioned that not only the spacecraft but also applications in the application layer can be modelled as agents. Figure 1 presents top level building blocks of software onboard each spacecraft agent. Figure 2 shows an autonomous distributed space system composed by multiple spacecraft agents. In the ADSS, the abstract high-level goals are decomposed autonomously into sequences of cooperative tasks by following certain rules, which is performed by the mission planner. And then there is a distributed allocator algorithm to allocate those cooperative tasks to each spacecraft agent by virtue of sharing knowledge. In the end the local controller onboard each agent makes the spacecraft to achieve the allocated tasks cooperatively. The mission planner, allocator and local controller belong to the high-level layer, the middleware and the layer underneath of a distributed system, respectively.

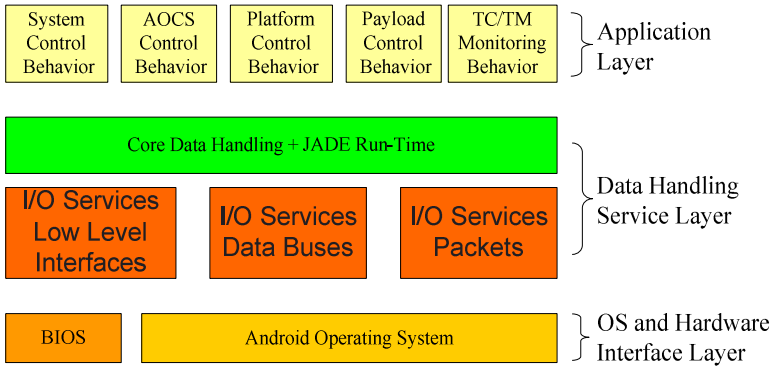

Fig.1. Top level building blocks of software onboard spacecraft agent

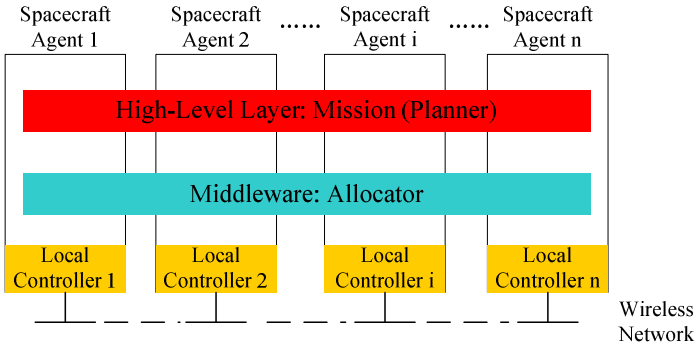

Fig.2. An autonomous distributed space system

In the rest part of this subsection the details of three special elements, i.e. the ARM processors, the Android OS and the JADE run-time, are provided. For details of other layers and their components, please refer to [14].

4.2.1. ARM Processors

ARM, short for Advanced RISC Machine, is a Reduced Instruction Set Computer (RISC) instruction set architecture developed by ARM Holdings. RISC systems are based on a very compact low level instruction set, where each instruction can be executed in one clock cycle and the clock frequencies are high. Due to the simple instruction set and thus low number of transistors, the RISC systems are robust to radiation and electromagnetic effects. Therefore, today's onboard computers are almost all based on the RISC chip architectures. A good example of onboard processors that have been used for DSS missions is the LEON3FT, which was flight on the PRISMA formation flying mission [15].

As a special branch of RISC, ARM architecture processors have been widely used in embedded systems including smartphones. ARM processors are typically deployed as Systems-On-Chip (SoC) to reduce space, power consumption and cost. An ARM-based SOC is essentially a miniaturized version of the processor layout found in normal PC, where there is typically a motherboard that supports a processor, which is further complemented by a graphics card. Although the usage of ARM processors in space is sometimes reported, it is much limited to embedded systems. On the other side, the computational power of ARM processors is very well represented by e.g. smartphones. For example, the Nexus One integrated 1GHz ARM processor with other devices in a very tiny space and is capable of multimedia, games and other complex applications [3]. In addition, it costs only several hundred Euros, with low mass, small size and low power consumption. Therefore recently several CubeSat missions using ARM-based smartphones have been proposed. For example, the Nexus One smartphone will be flight onboard the NASA PhoneSat as well as the Surrey's STRaND-1 [3],[4].

4.2.2. Android OS

Android is a Linux-based operating system primarily designed for mobile devices utilizing ARM processors. This light weight OS is also suitable for embedded systems

such as networking equipment [16]. During last two years a few projects have been planned to use Android OS in space. For example, NASA uses Nexus S phones running Android OS to upgrade SPHERES by performing tasks such as recording sensor data and capturing video footage. Android is very attractive to space applications, because it's an open source platform, which makes it easy to customize the software to meet the specifications required to fly in space. One step further, the developers of applications for smartphones could feasibly create apps for satellites thanks to the open platform. Therefore, it opens up a lot of new technologies to a variety of people and companies who usually can't afford it.

4.2.3. JADE

JADE (Java Agent DEvelopment framework) is a distributed agent-oriented middleware that provides domain-independent infrastructure for developing agent-based applications, such as how to allow agents to communicate [17]. JADE creates a run-time environment to implement the life-cycle support required by agents, the core logic of agents and a rich suit of graphical tools. In order to run JADE on the Android OS, an ad-hoc version of JADE has been created to meet Android requirements and specificities. Basically, the JADE run-time is wrapped by an Android service. In practice, JADE run-time creates containers to host agents. In a JADE-based distributed system a main container is required to form a platform, with which all other containers register. However, till now main container can't be generated on Android OS. Therefore, for a system involving Android OS, the main container should be created somewhere else such as a PC with Windows OS, and then normal containers created on Android OS can register with the main container, which is shown in the following figure. In Fig.3, the main container is created on a PC and two containers are hosted on two Android smartphones separately. The forth container is running on an Android emulator.

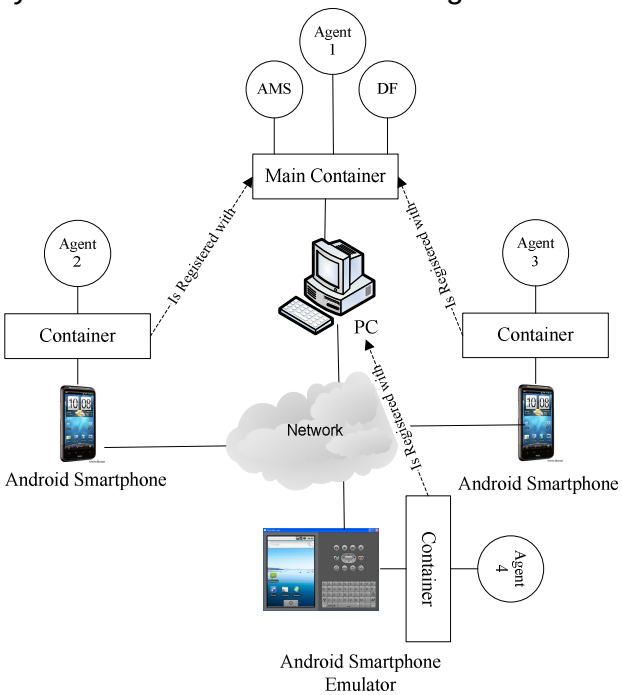

Fig.3. Relationship between Main Container and Container

However, there are a lot of issues need to be solved before using smartphone for space. For example, some functionalities of the smartphone are not useful for space missions and therefore have to be removed if they will occur power consumption or interference other functionalities or devices; the operating system of smartphone, e.g. Android, may be too heavy for space missions and should be tailed more or less; also the real-time performance of Java-based environment needs to be further investigated. But the extraordinary performances, low cost and abundant software resources make these issues less important and will not stop the attempting of using smartphone technologies for low-cost space missions.

5. Preliminary Experiment

To implement autonomous operations of distributed space systems, the complex interactions of distributed mission planning, task allocation, control and inter-spacecraft communication must be addressed. As in the development of any technology, aforementioned interactions are first verified by means of analysis and simulation. Then hardware-based demonstration is designed to further verify and validate the developed system. To keep consistent with this philosophy, a four-step verification plan is made. The first step is to perform MAS simulations only on PCs with software testbed, where the simulations cover the satellite motion, the developed distributed algorithms for cooperative planning & allocating, as well as the control strategy and control law for each spacecraft. The second step is a Hardware-in-Loop test, which utilizes smartphones with ARM processors, Android OS and JADE as the MAS testbed to verify the MAS operations. The third step is to integrate the MAS testbed with a formation flying testbed that is under development in TU Delft. Finally the onboard autonomy system developed in this paper will be validated in orbit through the DelFFi mission.

In the rest of this section, some preliminary experiments implementing the first two steps of aforementioned verification plan will be introduced. First, the setup of the experiments, based on both software testbed and Hardware-in-Loop MAS testbed, are presented. Then experimental results for implementing MAS behaviors are provided. Since experimental scenarios for both testbeds are same, experimental results will be presented in a mixed manner. At the end, a case study of autonomous reconfiguration is presented. In addition, this paper only focuses on the establishment of spacecraft agents and their basic behaviours, such as orbit propagation behaviour, orbit transfer behaviour, relative motion behaviour and communication behaviour. Other high-level behaviours can be developed based on those basic behaviours.

5.1. Experiment Setup

In this subsection the setup of the PC-based simulation on software testbed and the Hardware-in-Loop test on MAS testbed are presented, respectively.

5.1.1. Setup of PC-based simulation

The PC-based software testbed utilizes Android emulators. The Android emulator is an implementation of the Dalvik virtual machine, which makes it as a valid platform to run Android applications as any Android phone. It is preferred to test and debug the developed applications firstly on the emulators, since they are decoupled from any particular hardware. However, the emulator simulation should only be regarded as the baseline for testing the developed applications. Hardware-based tests are required to verify and validate the performance of the applications by taking the computation capability of the processor and the real world communication into account.

The setup of the emulator-based simulation is shown in Fig.4. There are three agents in total and accordingly three containers. As mentioned before, the main container is hosted by a PC with Windows OS rather than by the Android emulator. The other two containers are hosted by two emulators separately. The agent residing in the main container is responsible for keeping track of all the agents connecting to it. It should be pointed out that the main container is not a bottleneck for the distributed system as each container is managed locally. However, the JADE platform does suffer from a single point of failure of the main container. But some technology, such as fault tolerance, has been developed to relieve that problem. It should also be clarified that the scalability of the system is high; namely, in theory there is no limit of the number of the containers to register with the main container.

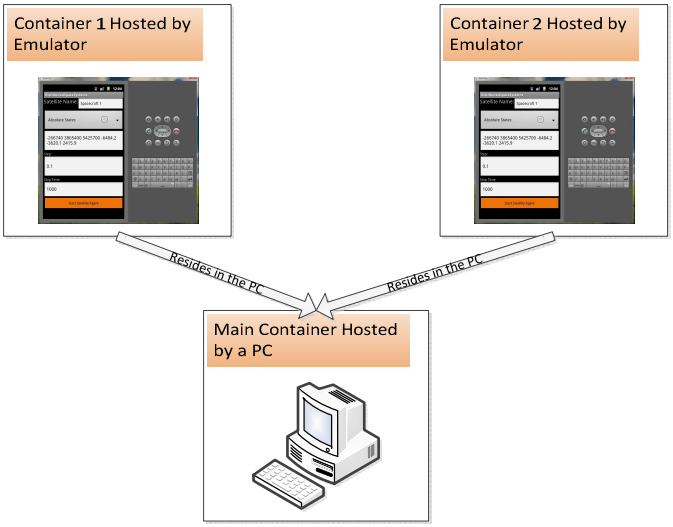

Fig.4. Setup of the Emulator-Based Simulation

5.1.2. Setup of Hardware-in-Loop test

The setup of the hardware-in-loop test is shown in Fig.5. Instead of Android emulators, two smartphones host normal containers and, therefore, two spacecraft agents separately. Wi-Fi technology is used to establish the wireless network. As shown in Fig.5, the manager agent is created first in the main container (hosted in a laptop) to get ready for the registration of agents running in the smartphones. Two identical HTC Desire S smartphones are used in the hardware-in-loop test. The HTC Desire S has a 1 GHz second-generation Snapdragon processor, 768 MB of RAM and 1.1 GB of internal storage for apps. And it runs Android 2.3.3 Gingerbread operating system.

Fig.5. Setup of Hardware-in-Loop test

5.2. Initiating Satellite Agents and the Distributed System

After setting the connecting host and port for the agents hosted by normal containers (shown in the left of Fig.6), the spacecraft agents Spacecraft 1 and Spacecraft 2 are created and registered with the main container, as shown in the right of Fig.6. The manager agent is hosted by the main container to keep track of Spacecraft 1 and Spacecraft 2. Afterwards, the distributed system is created.

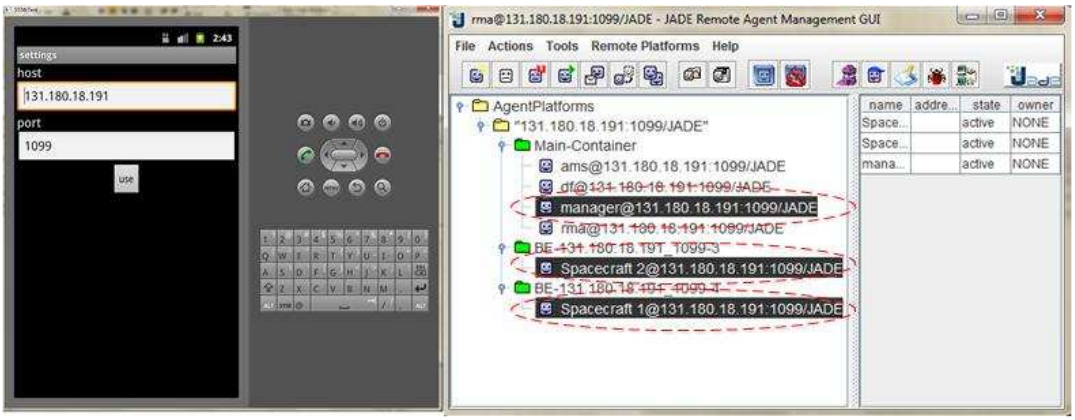

Fig.6. Create spacecraft agents

5.3. Behaviors of Satellite Agents

As aforementioned, the behaviors of satellite agents considered here only consist of orbiting behavior, communication behavior, relative motion behavior, and relative orbit transfer behavior. The orbiting behavior is based on the theory of Keplerian orbit. However, perturbations such as non-sphere gravity, atmospheric drag and so on can be integrated as well. For the communication behavior, right now the performative, content description, language and ontology of the communication are used inherently from JADE. However, those important ingredients of communication can be tailored for a specific case. At this moment the relative motion behavior is designed based on CW equations and the analytical solutions, which are shown in Eq. (1), where x, y, z are defined in the direction of radial, along-track and cross-track, respectively. Each satellite

agent maintains and updates the five parameters, i.e. p, ω, l, s and α, which describe the relative motion. Besides, each agent communicates with its neighbors to get those parameters of their relative motions, maintains them and updates them.

$$\begin{cases} x = -p\cos(\omega t + \theta) \\ y = 2p\sin(\omega t + \theta) + l \\ z = s\sin(\omega t + \theta - \alpha) \end{cases} \tag{1}$$

The relative orbit transfer behavior is designed based on the two-impulse maneuver method. The relative state transition matrix is shown in Eq. (2). Given the initial relative position vector $\mathbf{r}_0$, the initial relative velocity vector $\mathbf{v}_0$, the expected final relative position vector $\mathbf{r}_f$ and the expected relative velocity vector $\mathbf{v}_f$, the two impulse is derived as shown in Eq. (3) based on Eq. (2). The relative state transition matrix is determined by the assigned transfer time.

$$\begin{bmatrix} \mathbf{r} \\ \mathbf{v} \end{bmatrix} = \begin{bmatrix} \Phi_{\mathbf{rr}_0} & \Phi_{\mathbf{rv}_0} \\ \Phi_{\mathbf{vr}_0} & \Phi_{\mathbf{vv}_0} \end{bmatrix} \begin{bmatrix} \mathbf{r}_0 \\ \mathbf{v}_0 \end{bmatrix} \tag{2}$$

$$\begin{cases} \Delta\mathbf{v}_1 = \Phi_{\mathbf{rv}_0}^{-1}(\mathbf{r}_f - \Phi_{\mathbf{rr}_0}\mathbf{r}_0) \\ \Delta\mathbf{v}_2 = \mathbf{v}_f - (\Phi_{\mathbf{vr}_0}\mathbf{r}_0 + \Phi_{\mathbf{vv}_0}\mathbf{v}_0 + \Phi_{\mathbf{vv}_0}\Delta\mathbf{v}_1) \end{cases} \tag{3}$$

5.4. Case Study of Reconfiguration

A case study of reconfiguration is presented to integrate the basic behaviours. Assume there are three satellites in a formation flying, and the five parameters of satellite 1, 2 and 3 are $(0,0,0,0,0)$, $(0,0,\text{-}10000,0,0,0)$ and $(5000,0.0011569,\pi/2,0,\pi/2)$, respectively. Satellite 1 is in a 300Km circular earth orbit. The initial configuration is shown in Fig.7. The expected configuration is a “A-Train” formation flying, where satellite 2 is 10 km behind satellite 3 that is also 10 km behind satellite 1. The transfer time is set to 0.745h.

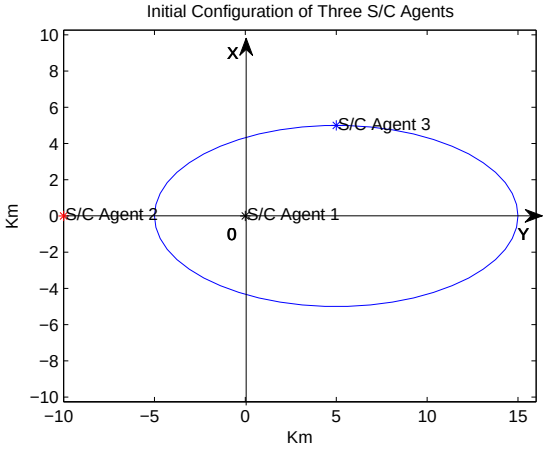

Fig.7. Initial Configuration

For this case study, the auction algorithm, initially used to solve the network flow problem, is applied to determine the optimum assignment of aforementioned three satellites in the along-track direction [18]. The along-track direction is discretized every 1 km. Each satellite agent maintains an information state, which consists of current assignment of three satellites and related delta-v requirements. The satellite agent communicates its information state with others, and based on the auction algorithm the information state is converged to the optimum reconfiguration, as shown in Fig. **8**, where each diamond represents the final position of each satellite in the along-track direction.

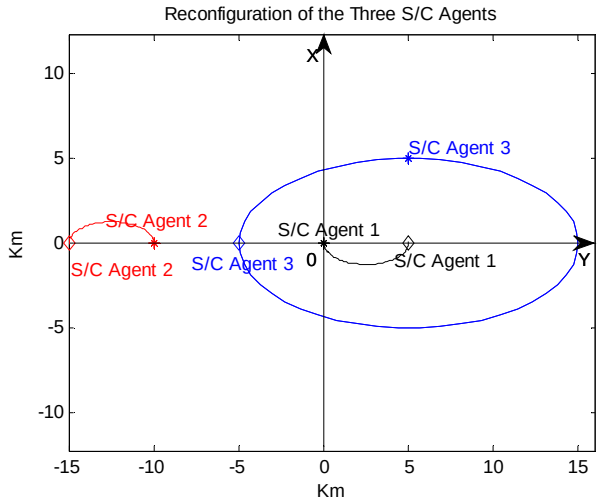

Fig. 8. Reconfiguration of the Satellite Agents

6. Conclusions

An onboard autonomous architecture with cooperative mission planning capability is grounded on Multi-Agent System technology for Distributed Space Systems in the context of the DelFFi CubeSat formation flying mission. This architecture is based on the smartphone technologies, including ARM processor, Android operating system and open-source MAS development environment. Preliminary experimental results of implementing the onboard autonomy architecture are presented, which show the efficiency and potentials of the proposed architecture for low-cost missions utilizing multiple satellites.

7. References

[1] Gill, E., et al., "DelFFi: Formation flying within the QB50 constellation of nano-satellites," The 63rd International Astronautical Congress, Naples, Italy, Oct. 2012.

[2] Muylaert, J., et al., "QB50, an international network of 50 double cubesats for multi-point, in-situ measurements in the lower thermosphere and re-entry research," The ESA Atmospheric Science Conference, Barcelona, Spain, Sep. 2009.

[3] Marshall, W., et al., "Phonesat: A smartphone-based spacecraft bus," The 62nd International Astronautical Congress, Cape Town, South Africa, Oct. 2011.

[4] Kenyon, S., et al., “Strand-1: Use of a $500 smartphone as the central avionics of a nanosatellite,” The 62nd International Astronautical Congress, Cape Town, South Africa, Oct. 2011.

[5] Hamann, R., et al., “Nanosatellites for microtechnology prequalification: The Delfi program of delft university of technology,” The 6th Symposium on Small Satellites for Earth Observation, Berlin, Germany, 2007.

[6] Doyle, R., et al., “Autonomy and Software Technology on NASA’s Deep Space One,” IEEE Intelligent Systems, Vol. 14, No. 3, 1999.

[7] Chien, S., et al., “Onboard autonomy on the earth observing one mission,” AIAA 1st Intelligent Systems Technical Conference, Chicago, Il, USA, Sep. 2004.

[8] Wikipedia, “Deep space 1”. Available: http://en.wikipedia.org/wiki/Deep Space 1

[9] Schetter, T., et al., “Multiple Agent-Based Autonomy for Satellite Constellations,” Artificial Intelligence, Vol.145, No. 1-2, 2003, Pages 147-180.

[10] Campbell, M. and Schetter, T., “Comparison of Multiple Agent-Based Organizations for Satellite Constellations,” Journal of Spacecraft and Rockets, Vol.39, No.2, 2002.

[11] Mueller, J.B. and Surka, D.M., “Agent-Based Control of Multiple Satellite Formation Flying,” The 6th International Symposium on Artificial Intelligence and Robotics & Automation in Space: i-SAIRAS 2001, St-Hubert, Quebec, Canada, June 18-22, 2001.

[12] Curits, S.A., et al., “ANTS (Autonomous Nano-Technology Swarm): An artificial intelligence approach to Asteroid Belt resource exploration.” The 51st International Astronautical Congress, Oct. 2000.

[13] Hinchey, M.G., et al., “Autonomous and autonomic swarms,” The 2005 International Conference on Software Engineering Research and Practice (SERP’05), Las Vegas, Nevada, USA, June 2005.

[14] Eickhoff, J., “Onboard Computers, Onboard Software and Satellite Operations: An Introduction,” Springer Series in Aerospace Technology, 2011.

[15] LEON3FT-RTAX Data Sheet and User’s Manual, Aeroflex Gaisler, 2012.

[16] Meier, R., Professional Android Application Development. Wiley, 2008.

[17] Bellifemine, F., et al., “Developing Multi-Agent Systems with JADE,” Wiley Series in Agent Technology, 2007.

[18] deWeck, O.L., et al., “Optimal reconfiguration of satellite constellations with the auction algorithm,” Acta Astronautica, Vol. 62, pp. 112-130, 2008.