A Multi-Agent System Framework for Miniaturized Satellite

Plataforma de desarrollo software basado en agentes para sistemas satelitales miniaturizados[1]

Carmen Chan-Zheng[2], Johan Carvajal-Godínez[3]

Fecha de recepción: 7 de abril de 2018
Fecha de aprobación: 8 de julio de 2018

Chan-Zheng, C; Carvajal-Godínez, J. A Multi-Agent System Framework for Miniaturized Satellite. *Tecnología en Marcha*. Vol. 32-1. Enero-Marzo 2019. Pág 54-67.

DOI: https://doi.org/10.8845/tm.v32.i1.4118

1 Work based on Thesis work of [19].
2 Universidad de Costa Rica. Puntarenas, Costa Rica. Correo electrónico: carmen.chan@ucr.ac.cr
3 Instituto Tecnológico de Costa Rica. Cartago, Costa Rica. Correo electrónico: johcarvajal@tec.ac.cr. ORCID: https://orcid.org/0000-0003-1489-7894

Keywords

MAS engineering; framework; embedded systems; real-time; agent; multi-agent

Abstract

Developing newer satellite missions faces increased onboard software complexity. Next generations of small satellites need to enable the infrastructure for implementation of concurrent and deterministic onboard algorithms for mission coordination and control. Multi-agent-based architectures are a new developing approach adopted in the software engineering field due to its flexibility, scalability, and adaptability to dynamic operating environments. This paper describes the design and implementation of a deterministic multi-agent system framework to develop applications for highly constrained embedded computers used in small satellite missions. As a result of the implementation of this framework the user coding effort for describing complex onboard software applications is reduced up to 50% with minimum impact on CPU load and program memory allocation. This paper also shows a set of benchmarks that demonstrate not only the feasibility of MAS-based software for small satellite missions but its value to achieve aggressive development schedules.

Palabras clave

Sistema multiagente; sistema embebidos; agente; tiempo real; multi-agente.

Resumen

El desarrollo de nuevas misiones satelitales se enfrenta a un incremento en la complejidad del software a bordo. Las próximas generaciones de satélites pequeños deben habilitar la infraestructura para la implementación de algoritmos a bordo concurrentes y deterministas para la coordinación y el control de la misión. Las arquitecturas basadas en multi agentes son un nuevo enfoque de desarrollo adoptado en el campo de la ingeniería de software debido a su flexibilidad, escalabilidad y adaptabilidad a entornos operativos dinámicos. Este artículo describe el diseño y la implementación de un *framework* basado en sistemas multi-agentes para desarrollar aplicaciones para computadoras embebidos con recursos altamente limitados que son utilizadas en pequeñas misiones satelitales. Como resultado de la implementación de este *framework*, el esfuerzo de codificación del usuario para escribir aplicaciones complejas de software a bordo se reduce hasta en un 50% con un impacto mínimo en la carga de la CPU y la asignación de la memoria del programa. Los resultados obtenidos en esta investigación demuestran no solo la viabilidad del software basado en MAS para misiones de satélites pequeños, sino también su valor para lograr planeamientos agresivos.

Introduction

Embedded computers are the core of any electronic system, and satellites are not the exception. In fact, satellite designers are adopting the use of distributed spacecraft architectures to improve their performance and making their subsystem's integration easier and faster. The shift from a centralized computing architecture towards a distributed architecture offers new advantages. For example, it improves fault-tolerant capabilities by enabling resource sharing among subsystems [1]. Also, it is easier to reconfigure and to upgrade the onboard software on the fly, which brings flexibility in the mission operations [2] [3]. However, these new advantages come at a cost: the onboard software complexity of the space missions is increased. According to [4], the primary cause of the growth in satellite software complexity comes from the mission requirements related to coordination and control-related tasks during the mission operation phase.

Both coordination and control activities demand a deterministic (real-time) behavior to the computers onboard of the satellites. This onboard software is characterized by being autonomous. Thus, it can make decisions without human intervention with specific time constraints. In particular, spacecraft maneuvers and fault detection, identification and recovery tasks are the most critical features needed to ensure the safety of the spacecraft [5] [6]As the onboard coordination and control activities require a high degree of autonomy, the use of Multi-Agent Systems (MAS) is proposed as a software architecture style to develop satellite software with intelligent capabilities [7]. The critical feature of this architectural style lies in its capacity for addressing problems by distributing them to different agents [8]. According to [9], an *agent* is an autonomous computational entity that is assigned to a specific role within the system, communicates with other agents and perceives their environment. A group of agents (MAS) work in a proactive sense to achieve a specific mission goal.

In general, most of the MAS-based applications do not use a standard platform or framework for its implementation [10]. However, there are several solutions available for MAS-based applications development. Some of them are based on the specifications established by The Foundation for Intelligent Physical Agents (FIPA), for instance, JADE [11], SPADE [12], and Mobile-C [13]. The main advantage of being a FIPA-compliant platform/framework is the ability of applications interoperability among different platform/frameworks.

There are also other successful MAS-based non-FIPA compliant frameworks designs such BESA-ME [14], EmSBot [15] and ObjectAgent [3] that enable the implementation of embedded MAS-based software.

Despite that JADE is the most used MAS-based application development platform [16] and the JADE agent platform runtime's memory footprint is around 100kb (making it suitable for embedded devices) [17], it does not fulfil the real-time requirements of the spacecraft system. Similarly, SPADE does not offer real-time support either. Moreover, Mobile-C framework runs on OSs that are not suitable for highly-constrained embedded platforms.

Lastly, the above-mentioned non-FIPA compliant frameworks (BESA-ME, EmSBot and ObjectAgent), although successfully implemented, they do not discuss the implementation of the minimum technology required to build a MAS-based platform specified by FIPA.

Furthermore, as shown, there are several approaches of MAS-based platform/framework/application implementations in embedded systems, but there are no reports on the literature about how the mapping methodology of an agent to a RTOS environment might affect the CPU's load, power consumption and latency.

Therefore, the purpose of this paper is to present a MAS framework that bridges the gap between real-time features and implementation based on the FIPA specifications. This framework is built based on a design strategy that considers the effect of the mapping on the CPU's load, power performance and latency of the system. Multi-Agent Framework for Embedded Systems (MAES), is a FIPA-based framework with real-time capabilities designed to be suitable for highly constrained embedded devices used in computers for pocketqube satellites.

The outline of the paper is as follows: Section 2 describes the software architectures aspects considered for the design of MAES, then Section 3 focuses on the implementation strategy, specifically in the key components to make the platform FIPA-compatible. Section 4 shows a characterization of MAES using a set of applications with different levels of complexity to show the value of adopting MAS-based software design in small satellites missions. Finally, Section 5 draws the main conclusions and discusses the future work intended.

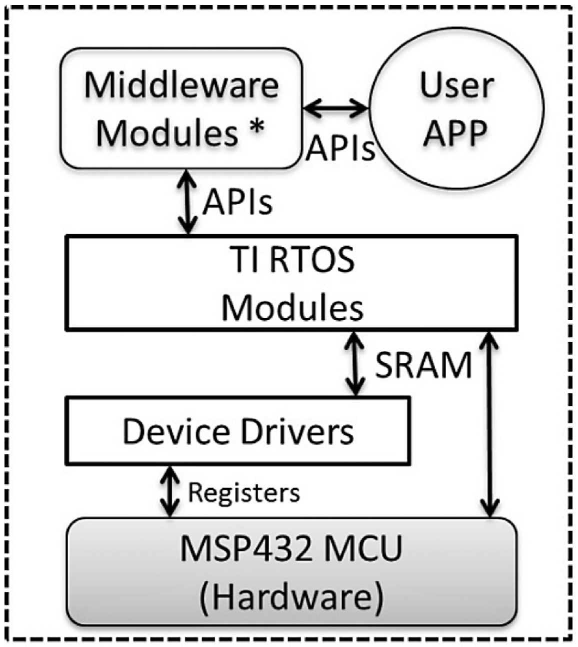

Figure 1. Onboard Software Architecture for PocketQube Computer.

MAES Architecture

The development of a multi-agent systems framework for satellite systems responds to the need of having distributed and concurrent execution environment for autonomous onboard software execution. This section starts with a description of the satellite execution infrastructure, to continue with the MAES framework design. The end goal is to show the feasibility and value of adopting multi-agent-based software development approach for satellite software development.

Satellite Software Architecture

This paper considers a highly miniaturized satellite using the pocketqube form factor. As discussed in [16], this highly integrated spacecraft, with an approximated volume of 5cm x 5cm x 15 cm, consists of a distributed electronic system with three primary embedded computers for electrical power management, for attitude determination and control and for command and data handling tasks. All these computers are based on the MSP432P401R microcontroller that features an ARM Cortex-M4 processor capable of running multi-threading applications with the TI-RTOS operating system. In principle, this tiny satellite is capable of running distributed and concurrent software for its operation.

This work assumes that satellite's onboard software (OBSW) can be modeled as a Multi-Agent System (MAS), and these agents are mapped into one of the pocketqube computers for their execution, so each of the onboard computers provides an execution platform for agents to live (logical containers). Either the agents or their private behavior functions can rely their execution on TI-RTOS threads called tasks. Also, this work takes as inputs the FIPA standards for implementing the MAS components. The software architecture model for this pocketqube mission is presented in figure 1, where the interface between different software layers is highlighted. MAES framework is implemented as a middleware module in charge of enabling the operation of the MAS based capabilities for a single pocketqube computer.

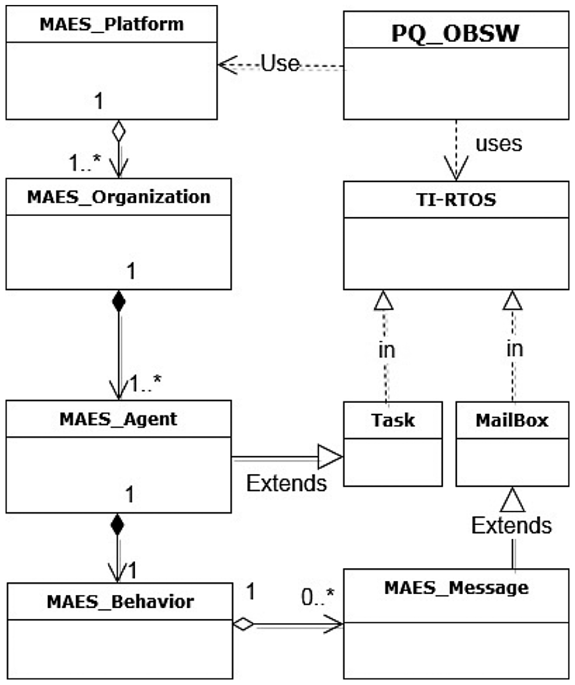

Figure 2. MAES Class Diagram.

MAES Framework Design

A set of software components were engineered using object-oriented abstractions to provide with the required FIPA-based features in MAES framework. The core of the MAES design is the agent class that is an extension of the task module in the TI-RTOS. This class allows the composition of agents with one behavior supplied by the behaviors class. These behaviors can be either generic or composed, depending on the software requirements, which provides flexibility. Agents can interact with each other by sending and receiving messages. For that purpose, agents use the mailbox module from the TI-RTOS.

Agents also can be grouped into organizations, depending on the software requirements. For that purpose, the MAES organization class provides the abstractions required for agents to organize and execution within the MAES execution platform, that aggregates all the organizations living in the execution environment. Figure 2 shows the relationship of these classes within the MAES framework. Section 3 focuses on the implementation aspects of each class and their integration.

MAES Framework Implementation

The implementation of the MAES framework was carried out using the integrated development environment (IDE) Code Composer Studio (Version: 7.1.0.00016) with the Texas Instruments Compiler (Version: TI v16.9.1 LTS) and the XDCtools (Version: 3.32.0.06_core).

The hardware platform provided for this research was the SimpleLink^{TM} MSP432P401R LaunchPad^{TM}

Development Kit. The microcontroller used by the development board is the MSP432P401R microcontroller (ARM Cortex-M4) running at 48MHz. Additionally, a Sensors Booster Pack Plug-in Module was connected to the Launchpad. This module contains a light sensor, an infrared sensor, an Inertial Measurement Unit (IMU) (featuring an accelerometer and gyroscope), a magnetometer and an environmental sensor.

Table 1. Class implementation for MAES components.

MAES Component	MAES Class
Agent	Agent Class Generic Behavior Class Agent Organization Class
Agent Platform	Agent Platform Class
Agent Management Service	Agent Platform Class
Message Transport Service	Agent Platform Class Agent Message Class

The framework was written using C++, which is an Object-Oriented Programming (OOP) Language. Table 1 shows that each of the mandatory FIPA components is implemented accordingly through different classes provided in MAES framework.

The following subsection describes in detail the implementation of each MAES class.

Agent Class

An instance of the Agent class contains the variables that describe the Agent's AID, mailbox handle, local name, priority, and organization characteristic. The instance also has a pointer to a memory stack defined by the developer. This stack is used for storing the agent's context and variables.

Generic Behavior Class

The MAES framework provides classes that can be used by the developer to implement customized behavior. An instance of Generic Behavior class allows the user to implement one behavior for an agent. The instance contains methods that can be overwritten by the developer: setup(), action() and done(). Additionally, the developer can also overwrite methods related to Failure Detection, Identification and Recovery tasks (FDIR). The FDIR methods intend to detect faults and identify the origin of the fault in the shortest time possible. Therefore, reducing the diagnostic time and increasing the system availability [17]. With the FDIR methods, the agent autonomy and reliability are increased as it can detect, identify and recover from its failure instead of having a centralized managing authority to perform those actions. The execution flow of these methods is shown in figure 3.

To execute the methods shown in figure 3, the method executed() has to be called from a wrapper function. If required, several behaviors can be implemented as separate subroutines in the wrapper function, and its execution order is determined by the developer by using patterns such as Finite State Machines.

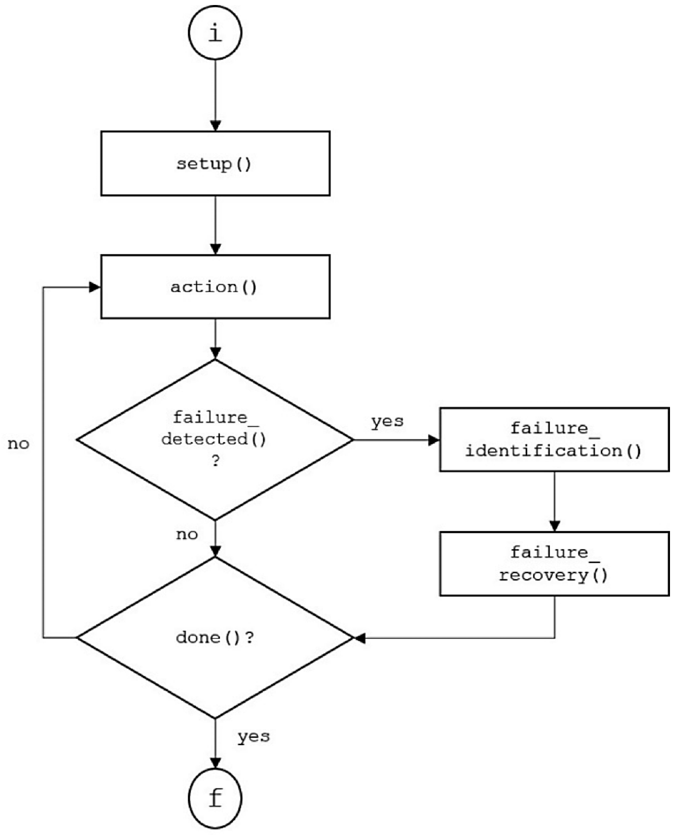

Figure 3. Generic behavior instance execution flow diagram.

Agent Organization Class

The Agent Organization class allows the developer to create organizations. Each organization groups the agents into one of the following topology types: Hierarchy and Team. The Agent Organization object contains methods such adding an agent to the organization, banning an agent, removing an agent from the organization, among several others.

Agent Platform Class

An instance of the Agent Platform Class contains methods to initialize the constructed agents, to initialize the platform and to perform the services. Additionally, this class creates the AMS agent.

The AMS agent performs actions such register/de-register an agent, suspend/resume an agent, kill an agent and restart an agent. The behavior of the AMS agent is shown in figure 4.

Agent Message Class

Each agent's mailbox is created in the Agent Platform class. However, the message object to be exchanged during the agent's communication is not constructed yet. Therefore, MAES framework provides a method for the message object creation and management through the Agent Message class. An instance of the Agent Message class contains the message object to be exchanged between two agents and the methods to manipulate the object. The message object contains the sender's AID, the target's AID, the message type and the message's content. Also, there are methods to manipulate the message object such send, receive, add receivers, among others.

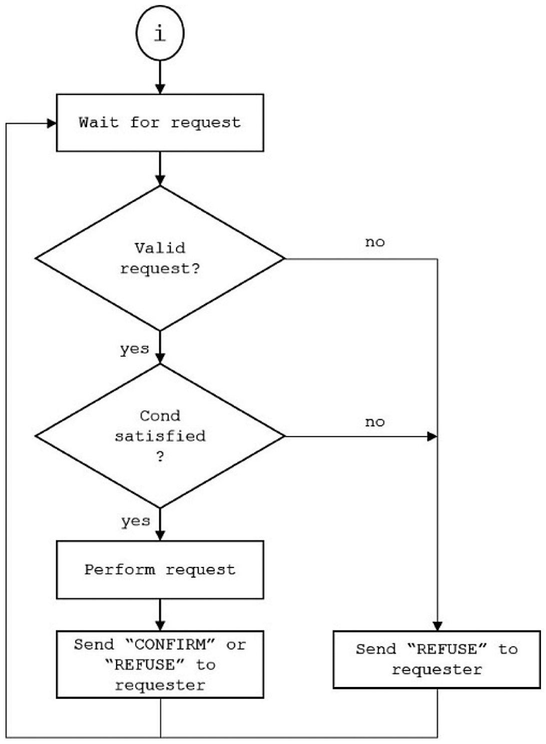

Figure 4. AMS Agent Behavior.

MAES Framework Workflow

Figure 5 shows the workflow to be followed to use the MAES framework properly. This consist of three phases namely project setup, agent construction, and platform construction. In the setup phase, all the dependencies are configured and verified in the integrated development environment. For the agent construction phase both the stack and behavior parameters are established, then during the platform constriction phase, the boot agent service is instanced the application is launched by the TI-RTOS scheduler.

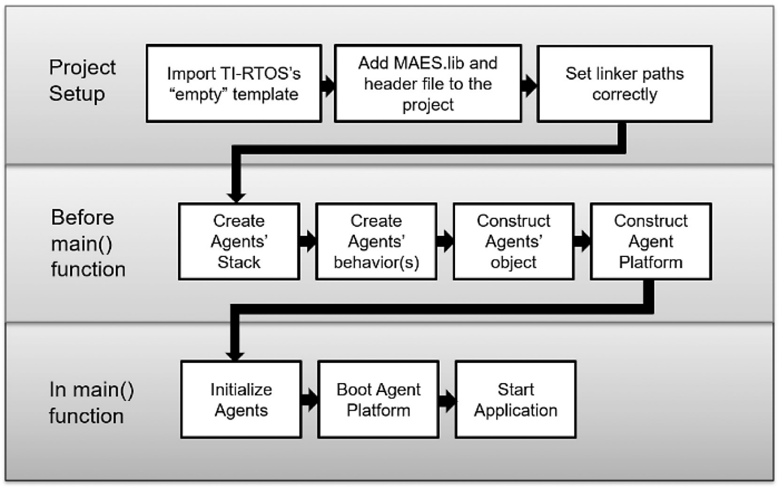

Figure 5. MAES framework Workflow Sequence.

MAES Benchmark

This section describes the experiments conducted to study the real-time characteristics and the impact on the memory allocation, CPU's load and power consumption of the MAES framework on any application. For this, four applications with a different level of complexity were used.

The following list describes the used applications, and it is ordered from the simplest to the most complex level:

1. Blink LED application: Blinks two LEDs.
2. Telemetry Logger application: Logs temperature, voltage and current value and outputs the value in UART interface.
3. Command and Data Handling System application: Receives commands from the user through UART interface and performs the according action.
4. Attitude Estimation algorithm application: Implementation of a quaternion-based Extended-Kalman Filter (EKF) based on the work of Sabatini (2006) [18].

The Attitude Estimation Algorithm was implemented in MAES and JADE framework to study the real-time characteristic of MAES framework; the JADE library was chosen for comparison purposes since it is the most common known-tool for MAS-based development.

Then, to study the impact of MAES framework on the memory, CPU's load and power consumption it was developed two versions for each of the applications: without MAES and with MAES. Each of the benchmark tests is explained in further detail in the following sections. To achieve consistency with the results, all the experiments were run on the same Launchpad board and were implemented in the same environment with the same compiling tools.

Real-time characteristics

This experiment compares the execution time of the EKF algorithm in MAES and in JADE. The execution time of the algorithm is measured each time that a new data arrives. The data is logged during 5 minutes at 10Hz sampling rate. The mean and variance of both framework's execution time are shown in table 2.

Table 2. Mean duration and variance of the execution time

	MAES Framework	JADE Framework
Min (ms)	2.564792	0.054312
Max (ms)	2.593562	64.982879
Mean (ms)	2.574432	0.182392
Variance ($[ms]^2$)	0.000023	2.328159

JADE's execution time varies on each call as the processor might be busy executing other system's processes. On the other hand, MAES' execution time is consistent the real-time operating system ensures predictability in its behavior. Figure 6 shows the probability density function for both frameworks.

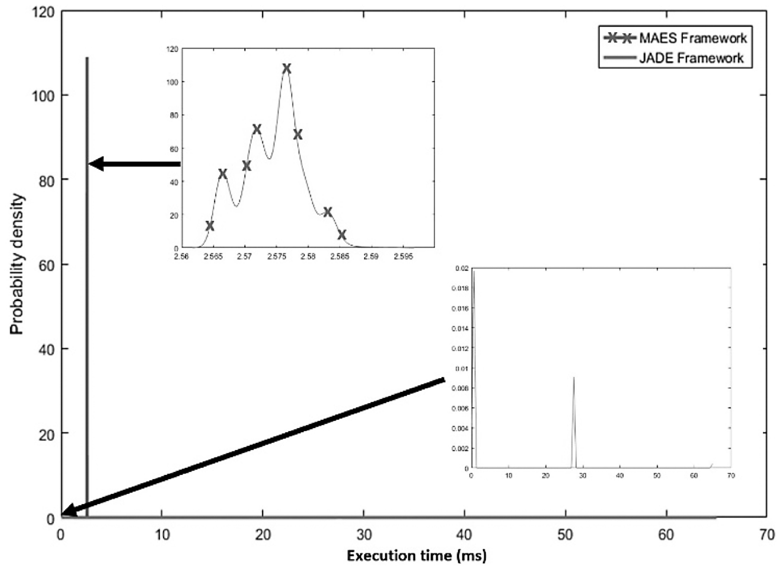

Figure 6. Probability density function comparison.

As seen, JADE's execution time is widely spread while it is concentrated in MAES. Despite that the algorithm executes faster in JADE in average, JADE cannot guarantee that on each call the algorithm execution time will be consistent. On the other hand, the execution time in MAES is consistent as it lies on top of a real-time operating system that ensures predictable execution pattern behavior.

Memory Performance

The Launchpad includes 256KB Flash memory and 64KB SRAM memory. Figure 7 shows the memory usage for the application on both implementations.

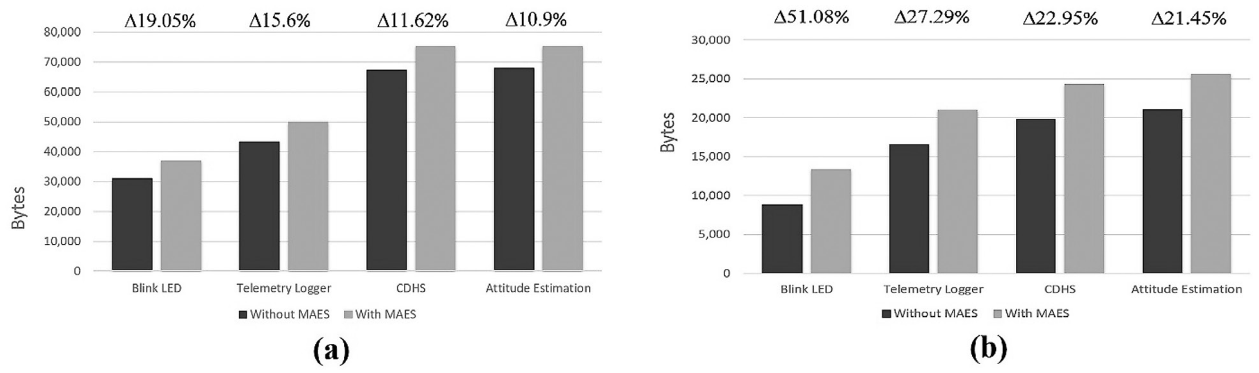

Figure 7. (a) Flash Memory allocation (b) SRAM memory allocation

For all the applications, there is an increment in memory utilization (program memory and SRAM) when using a MAES implementation. The increment in the SRAM memory is due to the dynamic object allocation of MAES framework. An Agent Platform object requires 4,400 bytes, and an additional Agent object requires 36 bytes in the dynamic memory. Then, the increment in the Flash memory is due to the additional code size of the MAES framework as shown in table 3.

Table 3. Code size characterization for each class.

Class	Size (bytes)
Agent Platform	2,364
Agent Message	1,280
Agent Organization	1,738
Agent	236
Behavior	136

It is noteworthy that the difference between the two implementations is reduced as the application's complexity increases. Therefore, the MAES framework is more suitable for the more complex application than the simple software implementations.

Even though that the memory usage is increased using MAES framework, the user coding effort is reduced as shown in table 4.

Table 4. main() function code size per application

Application	Without MAES (bytes)	With MAES (bytes)	%Decrease
Blink LED	252	120	52.39
Telemetry Logger	444	236	34.48
CDHS	344	156	46.85
Attitude Determination	232	152	54.65

The without-MAES implementation requires additional coding effort for creating tasks and signaling mechanisms. On the other hand, the with-MAES implementation already contains all the agent routines and communication method standardized for the developer use.

CPU's load

The RTOS Analyzer's Load Analysis tool from Code Composer Studio is used to measure the CPU utilization for each application in the benchmark. For that, it captures the average CPU utilization for each task/agent. According to this tool, there is an increment for all the tested applications. As an example, table 5 shows the increment for the Attitude Determination Application.

Table 5. CPU Utilization per function for the Attitude Determination

Source	Without MAES	With MAES
Kalman function	2.43%	2.44%
Sensor function	0.26%	0.28%
UART function	1.60%	2.16%

The increment of the CPU load in the MAES implementation is due to the communication method used in MAES. MAES not only implements the mailbox module but also, additional instructions are enclosed in the method send() from the Agent Message class. The additional instructions check the recipient validity. Table 6 shows the average duration for each communication method.

Table 6. Average time duration for different communication

	Average duration (ms)	Average duration (cycles)
Mailbox Post/Pend	17.412	836
MAES Post/Pend	27.375	1,314

Even though that the MAES post/pend pair is mailbox-based, this pair contains additional instructions to verify the recipient validity. Therefore, this increases the average number of cycles for the MAES communication method.

Power consumption

A set of experiments were conducted to verify the MAES's implementation impact on the power consumption. For that, the power profile from each application implemented with MAES is compared against its non-agent implementation. The results are shown in table 7.

Table 7. Mean power consumption for each application.

Application	Without MAES (mW)	With MAES (mW)	Difference
Blink Led	152.60	152.85	0.160%
Attitude Determination	149.21	149.57	0.245%
Telemetry Logger	134.17	134.18	0.004%
CDHD	158.95	160.20	0.785%

There is an increase in the power consumption for an application using the MAES implementation. The additional power consumption is the MAES framework is due to the extra CPU utilization required for the MAES' communication method as reported in Section 4.3. However, the impact of the MAES framework on the power performance was deemed negligible as the difference is lower than 1%.

Conclusions and Future Work

Conclusions

This paper has shown the feasibility of a Multi-Agent Framework for Embedded Systems (MAES) that is a FIPA-based framework with real-time capabilities designed to be suitable for highly constrained embedded devices used in highly miniaturized satellites.

The framework was developed on top of a Real-Time operating system (TI-RTOS) to guarantee determinism on agent's execution. The MAES framework's real-time characteristic was demonstrated in the benchmark analysis with an Attitude Determination application based on the Kalman filter. The experiments demonstrated that algorithm execution time in MAES is consistent with a variance in the order 10^5 s^2. Based on that experiment, MAES ensures predictable behavior in its execution.

Results have also shown that the user coding effort is reduced as the tasks and communication routines are standardized and encapsulated into MAES' class methods. However, it comes at a cost as the MAES-based applications show a minimum increase in memory, CPU's load, and power consumption. Furthermore, it was also shown that MAES is more suitable for more complex applications.

Even though that there is an increase in the memory allocation, it is demonstrated that the framework is lightweight as this only requires additionally 5,826 bytes in the Flash memory. Furthermore, an Agent Platform object requires 4,400 bytes, and an additional Agent object requires 36 bytes in the SRAM memory.

In conclusion, MAES is a real-time, lightweight and scalable framework compatible with highly resource-constrained embedded computers.

Future Work

Despite the MAES framework was developed based on the FIPA specifications, the framework is not fully FIPA-compliant. Specifically, the MAES framework messages are not compliant with the FIPA Agent Communication Language (ACL). Thus, it presents an implementation opportunity to expand MAES' functionality. The framework can be expanded to perform agents' inter-platform communication when FIPA ACL is integrated.

References

[1] J. Carvajal-Godínez, J. Guo and G. Eberhard, "Agent-based algorithm for fault detection and recovery of gyroscope's drift in small satellite missions," *Acta Astronáutica,* vol. 139, pp. 181-188, 2017.

[2] R. Radhakrishnan, W. W. Edmonson, F. Afghah, R. Martínez, F. Pinto and S. C. Burleigh, "Survey of Inter-Satellite Communication for Small Satellite Systems: Physical Layer to Network Layer View," *IEEE Communications Surveys Tutorials,* vol. 18, no. 4, pp. 2442-2473, 2016.

[3] D. Surka, M. Brito and C. Harvey, "The real-time ObjectAgent software architecture for distributed satellite systems," in *Aerospace Conference, 2001, IEEE Proceedings.*, Big Sky, MT, USA, USA, 2001.

[4] D. Dvorak, "NASA Study on Flight Software Complexity," in *AIAA Infotech@Aerospace Conference*, Seattle, Washington, 2009.

[5] C. Krupiarz, A. Mirantes, R. Doug, H. Adrian and W. Roger, "Flight Software," in *The International Handbook of Space Technology*, Berlin Heildelberg, Springer Berlin Heidelberg, 2014, pp. 471-491.

[6] K. Schilling, "Perspectives for miniaturized, distributed, networked cooperating systems for space exploration," *Robotics and Autonomous Systems,* vol. 90, pp. 118-124, 2017.

[7] C. P. Bridges and T. Vladimirova, "Agent computing applications in distributed satellite systems," in *International Symposium on Autonomous Decentralized Systems*, Athens, Greece, 2009.

[8] P. Lalanda, J. A. McCann and A. Diaconescu, "Sources of Inspiration for Autonomic Computing," in *Autonomic Computing: Principles, Design and Implementation*, London, Springer London, 2013, pp. 57-94.

[9] P. Lalanda, J. A. McCann and A. Diaconescu, "Autonomic Computing Architectures," in *Autonomic Computing: Principles, Design and Implementation*, London, Springer LOndon, 2013, pp. 95-128.

[10] J. P. Müller and K. Fischer, "Application Impact of Multi-agent Systems and Technologies: A Survey," in *Agent-Oriented Software Engineering: Reflections on Architectures, Methodologies, Languages, and Frameworks*, Berlin, Heidelberg, Springer Berling Heidelberg, 2014, pp. 27-53.

[11] F. Bellifemine, G. Caire, A. Poggi and G. Rimassa, “JADE: A White Paper,” *EXP in search of innovation,* vol. 3, no. 3, pp. 6-19, 2003.

[12] M. Escrivá Gregori, J. Palanca Cámara and G. Aranda Bada, “A Jabber-based Multi-agent System Platform,” in *Proceedings of the Fifth International Joint Conference on Autonomous Agents and Multiagent Systems*, Hakodate, Japan, 2006.

[13] B. Chen, H. H. Cheng and J. Palen, “Mobile-C: a mobile agent platform for mobile C/C++ agents,” *Software: Practice and Experience,* vol. 36, no. 15, pp. 1711-1733, 2006.

[14] D. M. Flórez, G. A. Rodríguez, J. M. Ortiz and E. González, “BESA-ME: Framework for Robotic MultiAgent System Design,” in *Proceedings of the 3rd International Workshop on Multi-Agent Robotic Systems*, Angers, France, 2007.

[15] L. Peng, F. Guan, L. Perneel, H. Fayyad-Kazan and M. Timmerman, “EmSBoT: A lightweight modular software framework for networked robotic systems,” in *2016 3rd International Conference on Advances in Computational Tools for Engineering Applications (ACTEA)*, Beirut, Lebanon, 2016.

[16] S. Speretta, T. Pérez-Soriano, J. Bouwmeester, J. Carvajal-Godínez, A. Menicucci, T. Watts, P. Sundaramoorthy, J. Guo and E. Gill, “Cubesats to pocketqubes: Opportunities and challenges,” in *Proceedings of the 67th International Astronautical Congress (IAC)*, Paris, France, 2016.

[17] J. de Oliveira, i. da Fonseca and H. Koiti, “Fault Detection and Isolation in Inertial Measurement Units Based on -CUSUM and Wavelet Packet,” *Mathematical Problems in Engineering,* vol. 2013, p. 10, 2013.

[18] A. M. Sabatini, “Quaternion-based extended Kalman filter for determining orientation by inertial and magnetic sensing,” *IEEE Transactions on Biomedical Engineering,* vol. 53, no. 7, pp. 1346-1356, 2006.

[19] C. Chan-Zheng, “MAES: A Multi-Agent Systems Framework for Embedded Systems,” TU Delft, Delft, Netherlands, 2017.

[20] A. Poggi and M. Tomaiuolo, “Integrating Peer-to-Peer and Multi-agent Technologies for the Realization of Content Sharing Applications,” in *Information Retrieval and Mining in Distributed Environments*, Berling Heidelberg, Springer Berling Heidelberg, 2011, pp. 93-107.

[21] E. Argente, J. Palanca, G. Aranda, V. Botti, V. Julian, A. García-Fornes and A. Espinosa, “Supporting Agent Organizations,” in *Multi-Agent Systems and Applications*, Berlin Heidelberg, Springer, 2007, pp. 236-245.

[22] F. L. Bellifemine, C. Giovanni and D. Greenwood, Developing Multi-Agent Systems with JADE (Wiley Series in Agent Technology), West Sussex, England: John Wiley & Sons, 2007.