Task Administration via Peer-to-Peer Communication

Yasin Sahin*, Sven Oberwalder, Konstantin Lindorfer, Fabian Hitzenberger
Department of Computer Science
Höhere Technische Bundes-Lehr- und Versuchsanstalt Wiener Neustadt
(Federal Technical Secondary College)
Wiener Neustadt, Austria
*Corresponding author email: sahin.yasin@student.htlwrn.ac.at

***Abstract*—The time a system requires to complete an operation is a substantial aspect of various automation tasks, such as the Botball competition. The efficiency of each machine, and thus the speed at which the system operates, can be increased by optimizing task scheduling and task allocation between them. Many algorithms for dynamic task allocation have been proposed, but few studies focus on their practical implementation. This work proposes a concept for a novel library that provides task coordination among all types of machines and robots. Through experiments conducted within the Botball environment, the authors demonstrate the increase in efficiency achieved with this library. The impact of this work extends beyond the Botball competition, enabling roboticists and engineers worldwide to increase the efficiency and extensibility of their systems to new levels.**

***Index Terms*—task allocation, network communication, multi-robot system, ubiquitous computing**

I. Introduction

In most scenarios one robot alone could be sufficient to complete a task. However, having multiple robots cooperating has several advantages over single-robot systems. This especially applies to scenarios where the task contains more than one objective, as is the case in the Botball competition. For example, the task can be split up among the robots to complete it more quickly. Additionally, the performance and the reliability of the robots can be increased as well [1]. However, the main problem of working with a multi-robot system (MRS) is that the robots must be coordinated in advance and work in a coherent manner. In an uncoordinated system, the robots may hinder each other in various ways. While this problem can be solved statically, i.e. by assigning tasks to each robot in advance, dynamic task allocation has the following unique benefits:

First, it is possible to pause a task if necessary to avoid a collision or other type of interference with another task. Moreover, through effective management, insufficient completion of a task can be detected and a fallback plan can be executed. Another benefit is the ability to perceive errors in execution and instruct other participants of the system to help out the machine in need.

To achieve the above, this study aims to develop a library that handles the communication and task management between participants of a multi-robot system using so-called Task Administration Data Exchange (TADE) requests. In this context, a "task" is defined as an activity conducted by a machine. This activity is defined in advance by the user and, if completed successfully, increases the value of the operation performed by the TADE network as a whole by a certain value, here referred to as the "weight" of the task.

However, this library does not engage with the implementation of algorithms to detect possible errors in execution or other interference. In this study the authors evaluate the effectiveness of TADElib in facilitating task coordination and increasing efficiency when using multiple robots, especially with regard to the Botball competition.

II. Study of Literature

The first prerequisite for a successful implementation of the TADElib is dynamic task allocation. As stated by Lerman et al., dynamic task allocation should be applied to increase the overall performance of a multi-robot system (MRS) [2]. This typically also minimizes the overall time required to complete a task.

As of today, many algorithms and tools for dynamic task allocation have been proposed and developed. Moreover, there are numerous works and studies which deal with the concept of MRS and their optimization. For example, Tolmidis and Petrou describe an auction-based algorithm for optimizing multi-objective task allocation problems, in which the robots evaluate the bids assigned to each task [3].

Another approach, the so-called Multi-Objective Particle Swarm Optimization (MOPSO) based algorithm, considers multiple parameters such as task execution cost and transfer time to achieve efficient task allocation and scheduling [4]. Furthermore, [5] introduces the Robot Operating System (ROS) as a useful tool that allows robots to communicate with each other. The study shows that ROS can be used to efficiently create a collaborative environment for robots.

Considering the works above, several revolutionary achievements have already been made in the field of MRS. This prototype of TADElib proposes a manifestation of a simplified version of the MOPSO-based approach, using only one parameter, the importance of a task, to evaluate incoming TADEs in conjunction with a novel packet protocol. However, no

Usage	Tool	Version / Specification
Programming Language	C++	C++ 17
Build System	CMake	ver. 3.0
Concurrency	C++ Threading Library	std::thread
Communication	Sockets	sys/sockets.h
Hardware	Botball Kit	2023
Operating System	Wombat OS	ver. 25.6

TABLE I: An overview of tools used for the development of TADElib

research has been done on the handling of errors that may occur during the execution of critical tasks.

III. Project Setup

This section introduces and explains the tools used to develop this TADElib prototype. The version number as well as other relevant information for each tool are shown in Tab. I.

A. *Threading*

A TADE participant should be able to continue executing a task while still processing and sending TADE requests in the background. This requires TADElib to be able to execute code in parallel threads. Utilizing threads also eases the implementation: Each task is assigned to its own thread, which can easily be started, paused, resumed, and terminated. The exact utilization of threads in our TADElib implementation is further described in IV-E.

The C++ standard header "thread" is used to provide the required functionality for this implementation[1].

B. *CMake*

The researchers employ CMake as the tool of choice for compiling and building TADElib. CMake neatly handles the compiler-agnostic process of building the project, creating a native build environment that seamlessly integrates the components into the executable [6].

C. *Sockets*

As all members of the system are connected over a network, sockets are used to send TADE packets over the given connection. The C++ header "sys/socket.h" is included in TADElib for this purpose[2].

D. *Wombat*

Since the robot controller currently used in Botball is the KIPR Wombat, it was used for the basic robot construction, as shown in Fig. 1. The network card inside the Wombat enables it to host or connect to a network so that communication between robots can take place.

[1]Documentation for the threading library used is available at en.cppreference.com/w/cpp/header/thread

[2]Documentation for the socket library used is available at pubs.opengroup.org/onlinepubs/009695399/basedefs/sys/socket.h.html

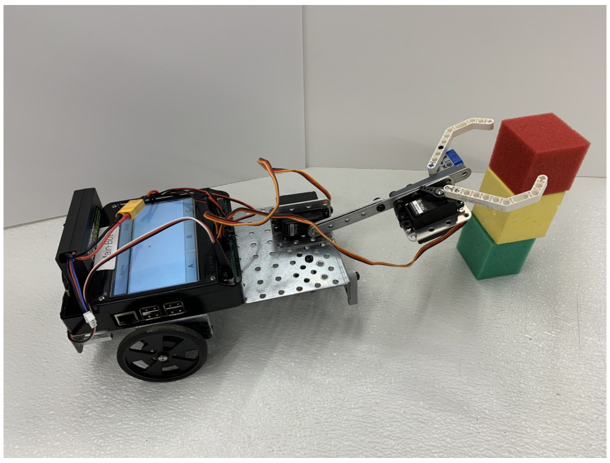
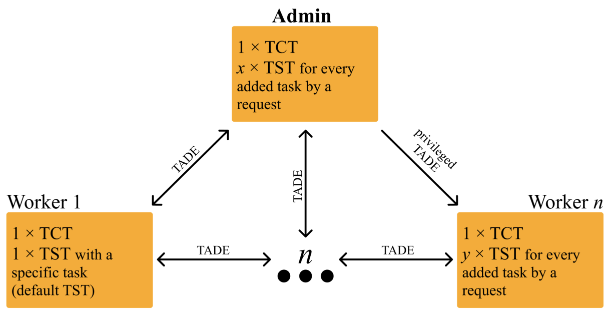


Fig. 1: A simple robot made from Botball parts and a KIPR Wombat


Admin
1 × TCT
x × TST for every added task by a request
TADE
TADE
privileged TADE
Worker 1
1 × TCT
1 × TST with a specific task (default TST)
TADE
n
TADE
Worker n
1 × TCT
y × TST for every added task by a request
Fig. 2: A visual description of the correlation between participants. Note that the terms TST and TCT will be explained in chapter IV-E.

IV. Task Administration Data Exchange

A. *Concept*

As shown in Fig. 2, all participants in the system have their own jobs and the arrows indicate the communication between them, allowing for a TADE between all devices. Within the system, there is an administrator that can be designated by the user and has the ability to send privileged TADE requests. The administrator is also responsible for calculating the total output and deciding whether a change in the plan should be made in case the output falls below the expected minimum. In order for this to occur, all recorded output from worker devices must be transmitted to the administrator device, and a new plan can be established via privileged requests. An unprivileged TADE from the administrator is treated like a regular TADE from any other device and as such may be declined by the receiving participant.

B. *Structure*

TADElib's underlying network protocol consists of five different packet formats, which are summarized in Tab. II. The type of a TADE packet is determined by the first hexadecimal digit of its ID. Note that the parameter "importance" will be explained in chapter IV-C.

Type	Parameter Name	Value
0	tade_id	0xxx xxxx
	importance	integer
	mode	P or T
	task_id*	xxxx xxxx
1	tade_id	1xxx xxxx
	importance	integer
	task_id	xxxx xxxx
	mode	S, SC, P or T
2	tade_id	2xxx xxxx
	output*	integer
	info*	string
3	tade_id	3xxx xxxx
	target_tade	xxxx xxxx
4	tade_id	4xxx xxxx
	target_tade	xxxx xxxx
	response	ACK or NAK

TABLE II: An overview of the packet structures of the different TADE types. Parameters marked with * are optional.

1) Warning TADE (Type 0): This type of packet is sent when a participant needs another participant to pause (P) or even terminate (T) a specific task. If no tasks are specified, all tasks currently running on the other device are targeted.

2) Requesting TADE (Type 1): A Requesting TADE is sent to ask another machine to perform a certain task. It can be decided whether the task should be executed simultaneously (S) with the currently running tasks, the task should be scheduled (SC) and executed after the current tasks are finished, the current tasks should be paused (P) or even terminated (T).

3) Reporting TADE (Type 2): The Reporting TADE is used to communicate the output of a completed task and other data to other devices. This packet is typically sent to the administrator.

4) Reassuring TADE (Type 3): A Reassuring TADE is used to inform other system participants that a task requested by a type 1 TADE no longer needs to be executed and may be terminated. In addition, tasks paused by a type 0 TADE may also be resumed by sending a Reassuring TADE.

5) Responding TADE (Type 4): Finally, a type 4 TADE is sent in response to another TADE to inform the sender whether it has been accepted (ACK) or not (NAK). It also has the additional function of ensuring that a previous TADE has been successfully received.

If the original TADE is privileged, the response attribute of the Responding TADE packet must be ACK.

C. Evaluation

To determine whether a TADE packet should be accepted or declined, all requests go through an evaluation process. Only privileged requests do not need to be evaluated, since they must always be accepted.

All tasks are assigned an importance attribute, which is determined by the user and should reflect the weight of the task for optimal results. The recipient compares the importance of the request with the importance of its currently running tasks. A request will be declined if current tasks are more important, and vice versa.

Furthermore, all tasks contain the boolean attributes is_pausable and is_terminable. If a type 0 or type 1 TADE request has its mode attribute set to "pause" (P) and the task is not pausable, the request will be denied regardless of the importance attribute. This also applies when the request mode is set to "terminate" (T), but the task may not be terminated.

D. Connection Configuration

For a successful TADE configuration, a connection must be established, and all system participants need to be in the same network, which is hosted by the administrator. The device that is configured as the administrator has a file named TADEsys.xml stored locally. All other participants send broadcasts so that the administrator can identify them. After a quick handshake between the administrator and the worker, information about the latter, such as its IP address, is recorded in the TADEsys.xml file by the administrator.

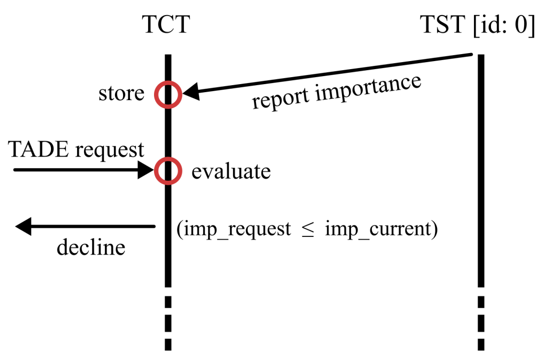
Similarly, worker devices have a members.xml file, which is a copy of the administrator's TADEsys.xml file, but with the administrator included and marked respectively. The members.xml file is updated each time a new participant introduces itself to the administrator. This type of update is communicated to all members of the system using type 2 TADEs.

E. TADE Processing

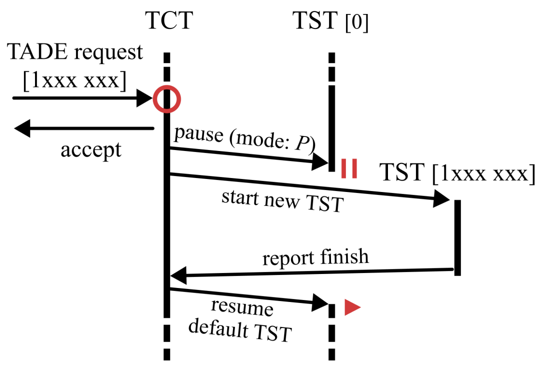
There are two types of threads running on every system participant:

1) The TADE Control Thread (TCT) performs all actions necessary to manage TADEs, such as receiving or sending TADE requests, processing them, updating the TADEsys.xml or members.xml file, storing or retrieving data received in a type 2 TADE, etc. Only one TCT is allowed per participant.
2) Task Specific Threads (TST) are threads that contain a specific task. At the start of its program, a device already has one TST, which is referred to as the default TST. It has the ability to create more TSTs and assign tasks to them. To make a participant idle, such as a robot that is only waiting and processing requests, the default TST can be set to a task that is actively waiting. All TSTs have an ID equal to their TADE ID, unless the thread is the default TST. By definition, the ID of the default TST is always 0.

As shown in Fig. 3a, a TST informs the TCT of the weight of its task at the beginning of execution. This allows the TCT to easily compare it with the importance attribute of TADE requests and decide whether to accept or decline a request. A detailed visualization of a type 1 TADE request, whose mode attribute is set to P, being accepted is shown in Fig. 3b.



(a) A TADE request being declined



(b) A TADE request being accepted

Fig. 3: A flowchart demonstrating the cooperation between TCT and TST

F. Cryptography

Cryptography is the utilization of mathematical algorithms to encrypt and decrypt data, as well as to generate and verify digital signatures. The goal of cryptography is to fulfill the key security requirements, which are [7]:

- Authentication: The procedure of proving someone's identity
- Confidentiality: Ensuring that only the intended recipient has access to the data.
- Integrity: Ensuring that the recipient receives the message in its original form and that the message has not been tampered with
- Non-repudiation: A method to verify that the message was in fact sent by the purported sender

Cryptography is essential to ensure the integrity and confidentiality of TADE packages. For the experiments conducted in this study, cryptography was omitted in order to speed up the development and testing process. However, when deploying a system like TADElib in a real-world environment, the use of cryptography is mandatory [8].

One option for encrypting the communication between the robots is symmetric encryption. It works by converting plain text to ciphertext using a secret key shared among the robots in the current session. Symmetric encryption is a reliable and efficient way to protect TADElib from potential attacks, because it is reasonably fast and requires less computational power than many other methods [9].

G. Security Issues

While symmetrical encryption in a fully developed version of TADElib may cover some security issues, there are still vulnerabilities that must be addressed. One of the most significant weaknesses is the connection network between the participants. It is especially vulnerable against any type of DoS or DDoS attack [10]. These attacks paralyze the network, disrupting the communication. In that case, all communication would be blocked, and sent TADE packets would not be processed properly anymore.

Fortunately, there are several ways to prevent these attacks: For example, by using globally coordinated filters, such as ingress filtering [11] or route-based distributed packet filtering [12]. If an attack could not be prevented, it should be detected immediately, followed by an appropriate intrusion response that identifies the source of the attack and blocks it [13].

Another major security problem is the secret key sharing process. The secret key used for symmetric encryption must be securely exchanged before communication can begin [14].

V. Experiments

A. Using TADElib to Increase Efficiency

The basic robot construction shown in Fig. 1 is used to perform the experiments. In the first scenario, also shown in Fig. 1, the red cubes must be stacked on top of each other, while the yellow cube, currently in the middle of the stack, is required for another task. The robot, hereafter referred to as robot A, is instructed to move the red cube aside, remove the yellow cube from the stack, and then put the red cube back in its place to form the desired stack.

In the second scenario, robot A picks up the red cube and moves it away, still holding it in its claw. In order for the second robot, here referred to as robot B, to pick up the yellow cube, robot A sends a TADE to inform robot B that it has positioned itself correctly and has cleared the path to avoid any collisions. Robot B then grabs the yellow cube with its claw, moves it away and can now continue with its previous task. After safely removing the cube, robot B sends a reassuring TADE to robot A, which then moves the red cube to its intended position.

Even though this does not have to be solved using TADEs and could be implemented by simply letting the robots wait for each other for a specified amount of time, by using this system, all of the collisions can be avoided. The process as a whole should also be more efficient than the static approach, which would require additional waiting times.

B. *Using TADElib for Error Avoidance*

For this second experiment, the robot construction as well as the environment from the previous experiment are reused. The only difference is that robot B is additionally equipped with a camera. In this scenario, robot A picks up the red cube at the top of the stack and then deliberately drops it to the ground to simulate failure of its task. It then sends a TADE to robot B, which upon receiving it tries to find the dropped cube by color, grabs it, and puts it back on the stack.

16 out of 20 runs were successful, indicating a significant increase in the repairability of initially failed tasks. In this example, the risk of failure for the specified task was reduced by approximately 75%. These results demonstrate that TADElib is effective at repairing errors that occur during execution.

VI. Real World Applications

Communication between machines such as robots, cars, and other devices has the potential to revolutionize the way we live and work in the future. By enabling them to exchange information and collaborate with each other, more efficient and intelligent systems can be created that can perform tasks and make decisions on our behalf.

One of the key benefits of device communication is the ability to coordinate and optimize the performance of multiple devices at once. For example, a fleet of robots working together on a manufacturing line could communicate with each other to divide tasks, handle errors that occur, share resources, and coordinate their movements. This would allow the robots to work more efficiently and effectively, improving the overall productivity of the manufacturing process.

Similarly, a network of connected cars could communicate with each other and with the transportation infrastructure to optimize routes, reduce the likelihood of congestion, and improve safety. By sharing real-time traffic data and other information, cars could make more informed route planning decisions, reducing potential accidents and improving fuel efficiency.

VII. Discussion and Conclusion

As the authors of this paper have demonstrated through these two experiments, the current prototype has successfully achieved its goal of increasing the overall performance of an MRS. In the first experiment, the mean execution time of the operation with TADEs is approximately twice as fast as the mean execution time without TADEs, which can be seen in Fig. 4. Execution errors were also reduced to 25% because robot A held onto the red cube, minimizing the chance of dropping it. While this improvement is significant, it is important to note that the TADElib is a new technology and the code provided was not fully optimized for performance. Therefore, further optimization of the code used for the specific tasks has the potential to yield even greater performance improvements for the MRS.

In the second experiment, it was observed that only 75% of the runs were successful. While this may suggest that TADElib is not a reliable technology, it is important to note that these failures were not necessarily caused by TADElib, but may have been caused by the code provided for the tasks. Therefore, it is important to recognize that the reliability of TADElib is highly dependent on the quality of the code provided to it.

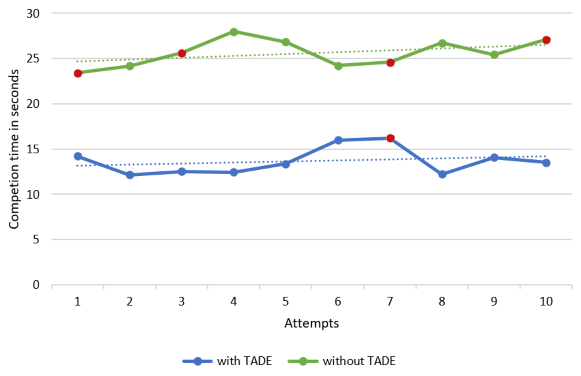


Fig. 4: This graph shows the time it took the robots to complete the task for each attempt in the first experiment. The red dots indicate a failed execution where the robot dropped a cube.

Nevertheless, this prototype will eventually need to be reworked, because in its current form it does not seem feasible to implement the applications listed in section VI. First of all, unlike the current prototype, a finalized version of TADElib will have to employ cryptography, and the TADE network will have to be sufficiently secured against network threats, especially if TADElib is to be used for industrial purposes. Moreover, a large-scale system whose participants cannot connect to the same network cannot be configured as described in section IV-D. An example of such a system would be the use case of autonomous cars as described in section VI. In this case, a TADE request needs to be passed from one participant to another until the message reaches the intended recipient.

In summary, TADE offers many opportunities for the robotics community and Botball. Although this library only implements algorithms that have been previously developed and researched for the concept of dynamic task allocation, a completed version would support many roboticists as well as Botball teams in various ways. Therefore, the development of TADElib should be continued as a project for the Botball and robotics community.

Acknowledgments

The authors would like to thank Dr. Michael Stifter, Christine Zeh and all the other members of robo4you for their invaluable support throughout the creation of this work.

References

[1] A. Khamis, A. Hussein, and A. Elmogy, “Multi-robot task allocation: A review of the state-of-the-art,” *Cooperative Robots and Sensor Networks 2015*, pp. 31–51, 2015.

[2] K. Lerman, C. Jones, A. Galstyan, and M. J. Matarić, “Analysis of dynamic task allocation in Multi-Robot Systems,” *The International Journal of Robotics Research*, vol. 25, no. 3, pp. 225–241, 2006.

[3] A. T. Tolmidis and L. Petrou, ”Multi-objective optimization for Dynamic Task Allocation in a multi-robot system,” *Engineering Applications of Artificial Intelligence*, vol. 26, no. 5-6, pp. 1458–1468, 2013.

[4] F. Ramezani, J. Lu, and F. Hussain, “Task scheduling optimization in cloud computing applying multi-objective particle swarm optimization,” *Service-Oriented Computing*, pp. 237–251, 2013.

[5] M. Quigley, B. Gerkey, K. Conley, J. Faust, T. Foote, J. Leibs, E. Berger, R. Wheeler, and A. Ng, “ROS: An open-source robot operating system,” *Proc. ICRA Workshop on Open Source Software*, Kobe, Japan, 2009.

[6] “Overview,” *CMake*. [Online]. Available: https://cmake.org/overview/. [Accessed: 27-Jan-2023].

[7] A. J. Menezes, V. O. P. C., and S. A. Vanstone, in *Handbook of Applied Cryptography*, Boca Raton: CRC Press, 2001, p. 4.

[8] M. Cheminod, L. Durante, and A. Valenzano, “Review of security issues in Industrial Networks,” *IEEE Transactions on Industrial Informatics*, vol. 9, no. 1, pp. 277–293, 2013.

[9] M. B. Yassein, S. Aljawarneh, E. Qawasmeh, W. Mardini and Y. Khamayseh, ”Comprehensive study of symmetric key and asymmetric key encryption algorithms,” *2017 International Conference on Engineering and Technology (ICET)*, Antalya, Turkey, 2017, pp. 1-7, doi: 10.1109/ICEngTechnol.2017.8308215.

[10] S. A. Arunmozhi and Y. Venkataramani, “DDoS attack and defense scheme in wireless ad hoc networks,” *International Journal of Network Security Its Applications*, vol. 3, no. 3, pp. 182–187, 2011.

[11] P. Ferguson and D. Senie, “Network ingress filtering: Defeating denial of service attacks which employ IP source address spoofing,” 2000.

[12] K. Park and H. Lee, “On the effectiveness of route-based packet filtering for distributed DOS attack prevention in power-law internets,” *Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications*, 2001.

[13] C. Douligeris and A. Mitrokotsa, “DDoS attacks and defense mechanisms: Classification and state-of-the-art,” *Computer Networks*, vol. 44, no. 5, pp. 643–666, 2004.

[14] N. Ferguson, B. Schneier, and T. Kohno, *Cryptography engineering: Design principles and practical applications*. Indianapolis, IN: Wiley, 2010.