

A Game-based Approach to Comparing Different Coordination Mechanisms

Kenneth M. Ford, Niranjani Suri
Institute for Human and Machine Cognition
40 St Alcaniz Street,
Pensacola, 32502 FL
Email: {kford, nsuri}@ihmc.us

Karel Košnar, Pavel Jisl, Petr Benda, Michal Pěchouček, Libor Přeučil
Gerstner Laboratory
Czech Technical University in Prague
Technická 2, Prague 6, 166 27, Czech Republic
Email: {kosnar, jisl}@labe.felk.cvut.cz

Abstract—Achieving coordinated behavior among multiple physical or logical entities is a challenging problem. Research in coordination has traditionally focused on either centralized or distributed approaches, but both approaches have inherent problems. The Process Integrated Mechanism (PIM) is a novel approach to coordination that relies on strong mobility to rapidly move a single process (the Coordinating Process or CP) among all the nodes that are part of the PIM. The continuous migration of a single process that performs the coordination combines the best of centralized and distributed approaches - simplicity of programming and reliability without a single point of failure. This paper proposes a method for the comparison of a traditional distributed approach based on Multi Agent Systems (MAS) and the new PIM approach. A simplified version of the Capture-the-Flag game is used for evaluation. One side is controlled by the MAS whereas the other side is controlled by the PIM. The paper analyzes the complexity and performance of each solution and provides a means to compare and contrast the two approaches.

I. INTRODUCTION

Many scenarios ranging from search and rescue to combat operations can benefit significantly from teams of humans, robots, and computers that collaborate and coordinate together to solve a problem. Designing and building systems of humans and computers that coordinate well is a challenging problem, both in terms of the complexity of developing and verifying the algorithms as well as the runtime efficiency. Current research has focused on both centralized and distributed (negotiation-based) approaches, but both these approaches have specific problems and limitations.

The centralized approach requires a coordinator to operate at a single point in the system and this coordinator needs to have complete information about all the entities. The negative implications are that the entire state of the system has to be transferred and collected at a single point and that the failure of the coordinator will cause the system to fail.

Distributed approaches such as multi-agent systems or systems based on emergent behavior (inspired by swarms or ants [1], [3]), do not have this single point of failure. Compared to the centralized approach, the multi-agent approach significantly increases the complexity of the programming model and the runtime. The complexity is caused by the requirement that each entity construct and maintain a model of the state and the expected behavior of the other entities. Moreover, these approaches introduce uncertainty in the behavior of the

system, which is undesirable. While the emergent approaches may utilize fairly simple programming models, their behavior is even more unpredictable.

A Process Integrated Mechanism (PIM) is a novel programming model and a runtime architecture that addresses the distributed coordination problem. The PIM approach retains the advantage of a single coordinating authority while avoiding the structural difficulties that have traditionally led to its rejection in complex settings. The components in the PIM architecture are conceived as parts of a single mechanism, even when they are physically separated and operate asynchronously.

The idea is to retain the perspective of the single controlling authority but abandon the notion that this process must have a fixed location in the system. Instead, the computational state of the coordinating process is rapidly moved among the component parts of the PIM. From this perspective, the operation of the PIM is the inverse of multiprocessing or timesharing. Instead of rapidly switching a single CPU between multiple processes, the PIM rapidly switches a single coordinating process between multiple CPUs in multiple physical entities. Just as a multitasking system effectively hides the process switching, the PIM runtime system effectively makes process cycling transparent.

This paper compares the PIM approach to the multi-agent approach for distributed coordination, analyzing the strengths and weaknesses of each approach. The multi-agent approach provides a good barometer for judging the PIM approach. A simplified version of the Capture-the-Flag game, played by a team of robots on each side is used for evaluation. One side is controlled by the MAS whereas the other side is controlled by the PIM.

II. PIM ARCHITECTURE

A Process Integrated Mechanism (PIM) integrates multiple physical computing nodes into a single virtual mechanism that can be controlled by a single process. This integration is achieved by rapidly moving a single process (called the Coordinating Process or CP) between each node that is part of the PIM. The CP is written as a single program that can access any of the component nodes just by referring to them, like objects in an object-oriented programming model. The CP

may invoke methods on a node, thereby performing actions, or transfer data to and from the node.

As shown in Figure 1, the role of the CP is to perform coordination. Each node may still have local processing that handles tasks strictly related to the local node. In a robotic environment, these tasks include sensor data processing, motor control, collision avoidance, and waypoint navigation. Tasks that require more than one node to communicate and coordinate are handled by the CP. In the PIM model, nodes may not communicate with each other directly. Unlike the Multi-Agent approach, each node does not maintain any awareness or model of the other nodes. The CP has knowledge of all the nodes and directs any behavior on a node that depends on the state of a different node.

Implementation-wise, the code that realizes the CP is installed on all components and each component maintains a current run-time state of the CP. At any given moment, only one copy of the CP is actually running on any of the components. While running on a node, the coordinating process has full access to any local data and can directly control any locally performed activity.

After a configured timeslice, the runtime state of the CP is captured and moved from the current node to the next node, where the CP immediately continues its execution. This movement of the CP state between components is rapid compared to the necessary global reaction time of the overall system, providing the illusion that the same process is running everywhere.

An important aspect of the PIM model is that the movement of the CP is transparent to the program itself and is completely managed by the runtime system. The CP is programmed as though it is simultaneously running, and has access to, every node in the PIM. The fact that at any given moment in time, the CP is running only on one of the nodes is invisible to the CP. In this sense, the PIM is the inverse of timesharing. A process in a multi-tasking environment is unaware of the fact that it is being suspended and switched out in order to run other processes. When the process switching happens fast enough, the interruption does not influence the behavior of the process. The process executes under the assumption that it is the only process running on the host. Likewise, the CP is unaware of being moved from one node to another. The assumption is that if the movement is fast enough, the system will be unaware that the CP is only on one of the nodes at any given moment in time.

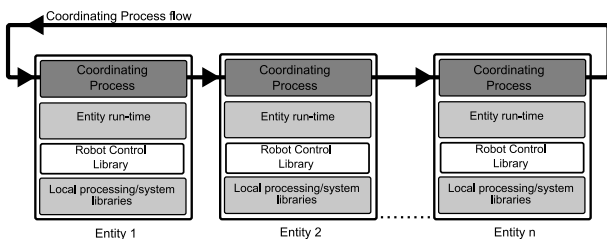


Fig. 1. Flow of Information in Process Integrated Mechanism architecture

III. A-GLOBE MULTI-AGENT ARCHITECTURE

We have used A-GLOBE¹ multi-agent technology [4] for comparing a MAS-approach with the PIM. A-GLOBE has been used successfully for free-flight collision avoidance among autonomous aerial vehicles and also for modeling collaborative underwater mine-sweeping search. While in the former application A-GLOBE was used as a simulation environment, the later A-GLOBE deployment was also migrated onto the robotic soccer platform [2].

A-GLOBE organizes distributed multi-agent applications into several containers that integrate or represent the autonomous hardware platforms (robots in our scenario). Each robot is controlled by a single A-GLOBE agent that is performing independent decision making and interacts in a peer-to-peer manner with the other agents controlling the other platforms. These agents are physically deployed on the platforms. Each container can host multiple agents, each running in a separate thread. The system is shown in Figure 2. The A-GLOBE Container Manager takes care of starting, executing, and finishing containers. The Agent Containers manage the life-cycle of agents and also provide services for sending and receiving messages. The Message Transport component ensures an efficient exchange of messages between two agents.

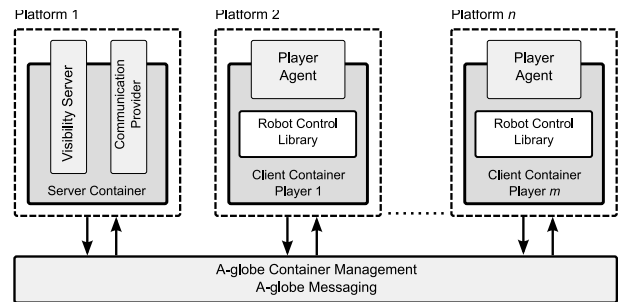


Fig. 2. System architecture in A-globe

A-GLOBE is a lightweight multi-agent platform that supports mobility and integration with powerful environment simulation components. The environment simulation components support realistic testing of the multi-agent algorithms and facilitate straightforward migration to real distributed environments (such as robotic hardware). The simulation components are the only domain specific centralized processes. There is no centralized coordinator. The agents act independently and use negotiation strategies to achieve coordinated behavior.

There are several specific modes of multi-agent operation that can be used for the comparison with the PIM-based approach:

- 1) Agents performing complex computations on top of complete, identical sets of data that are maintained by each agent. Each agent performs computations aimed at optimizing behavior of the entire community. Possible conflicts within the agents (given the high dynamics

¹<http://agents.felk.cvut.cz/aglobe/>

of the application domain) are solved by negotiations among the agents.

- 2) Agents performing only lightweight, local decision making on top of complete, identical sets of data. Coordination, plan merging, and resource sharing is achieved by negotiations among the agents.
- 3) Agents performing only lightweight, local decision making on top of partial data that are available to each individual agent (limited sensing capabilities based on geographical position, restricted communication range among the robots, etc.). This approach requires more negotiation for coordination than the previous two approaches.

The multi-agent approach, by its nature, is particularly suitable in highly distributed domains with partial knowledge and communication inaccessibility – mode 3. However, PIM is particularly suitable for dynamic real time environments with full connectivity, data accessibility and possibility centralized the data needed for coordination. That is why, the first experiment was directed towards comparing programming complexity and behavioral efficiency of PIM and \mathcal{A} -GLOBE in the mode 1. This mode also allows testing the mechanics of the two approaches using the identical decision making strategies, which would no longer be possible if modes 2 and 3 would be deployed. Future enhancements to the PIM will allow the PIM to operate over unreliable networks - which will allow comparisons using the other modes as well.

IV. CAPTURE-THE-FLAG GAME

To compare these two completely different approaches, this paper uses a simplified version of the *Capture the Flag* game. The goal of the game is for each team to protect its own flag while attempting to knock down the opponent's flag, which is achieved by a robot running into the flag. The game ends when one of the teams knock down the opponent's flag or the time expires. A time limit is imposed to prevent stalemate.

The flag is surrounded by a circular safety area into which defending robots are not allowed to enter. The safety area is used to prevent the situation where robots stand right alongside their own flag and thereby fully block the path to the flag.

Another simplification is that each team provides the position of its members to a position server, which then distributes the position information to each node using a virtual sensor. The virtual sensor is configured with a sensing range, which is used to determine the visibility of other robots to any given robot. This simplification reduces the impact of sensor inaccuracy on the comparison of the system.

The size of the field was chosen according the robot's size and number of team members. Robots are not allowed to touch each other and are therefore designed to stop 10 cm before any obstacle.

V. GAME PLAYING STRATEGIES

The game playing strategies are designed to best exploit the models and capabilities provided by the PIM and MAS systems. In the PIM-based approach, the algorithm for the

coordination is developed as though it is centralized, since the PIM abstraction hides the distributed nature of the system.

In the MAS approach, each robot has an independent instance of an agent that is running as a separate process. Therefore, each robot effectively has its own coordinator and actions to be performed are negotiated with other robots and their coordinators.

In order to make the comparison meaningful, it is useful to have as many invariants as possible between the two teams. Therefore, both teams are as identical as possible. All of the robots are the same at the hardware level and provide the same set of low-level behaviors (such as waypoint navigation). This section also develops a formal description of the common strategies used by both approaches. The common strategy will be the highest level of the decision process and will be realized by both the PIM and MAS approaches. The differences in the specific implementations will serve to highlight the advantages and disadvantages of each approach.

The common strategies can be divided into two types. Individual strategies are based on roles of the player (attacker or defender). Team strategies defines basic behavior of the team as a whole.

A. Team Strategies

As mentioned above, the team strategies dictate the behavior of the team as a whole and is a key issue for overall team performance. The team strategy in this case consists of the role assignment in the Capture the Flag game. It is necessary to analyze the situation of the game and choose an appropriate number of the attackers and defenders. It is also necessary to assign specific roles to each of the robots, considering which role, attacker or defender, is better suited for each robot. Assignment of roles to players and optimal distribution of roles in the team are optimization tasks. Every team tries to maximize utility of the players in the team. As the robots are identical at the hardware and the basic behavior level, each robot can play any role. Assignment of the optimal role can be done by taking into account factors such as the actual position of the robot.

While the paper compares different coordination approaches, we propose unified criterion of the optimization tasks and computation of utility for each role and player. The objective function is a function of the:

a) *Utility of attacker*: Time to the flag is the key property for the attacker. It depends on speed of robot and length of trajectory. Robot needs to find optimal (shortest, collision free) trajectory.

b) *Utility of defender*: Utility of the defender depends on how quickly it can block a dangerous attacker. The function is defined to compute utility of the defender based on position of defender and corresponded attackers. $F: D \times A \rightarrow R$. The goal is to assign defenders to attackers in such a way as to maximize the sum of this utility over all defenders.

c) *Dangerousness of the opponent attacker*: The dangerousness of the opponent attacker is computed based on the distance of the attacker to the flag.

Formally, the distribution of the roles over the team members can be seen as a projection $\mathcal{S} : P \rightarrow R$ from the set of players P to the set of roles $R = \{attacker, defender\}$. Optimization is finding such \mathcal{S} that maximizes the objective function F . Additional constrains can be given to \mathcal{S} .

$$\mathcal{S}^* = \operatorname{argmax}_{s \in \mathcal{S}} \sum_{p \in P} F(p, s(p))$$

We propose function F which reflects all of the above mentioned criteria. For the attacker, F equals to the reciprocal of the length of the shortest trajectory to the opponent's flag.

$$F(p, attacker) = \frac{1}{T_p}$$

where T_p is player's trajectory length.

For the defender, the computation of function F 's value is more complex because the value of F depends not only on player p and its role $s(p)$, but also on the defended opponent player. The function F can be defined as a tree-parameter function for the defender. Also, to keep the function uniform, some implicit value of "defended player" for an attacker can be defined.

The set of roles can be redefined from $R = \{attacker, defender\}$ to the $R = \{attacker, defender_i : i \in \{1, \dots, n\}\}$, where n is the number of the opponent and $defender_i$ means that the robot defends opponent player i . In this case, the solution for the optimization problem also finds the optimal assignment of the defenders to the opponent's players.

Let pi_a be the point of intrusion of attacker a , defined as the intersection of the circle around the flag and the line joining the actual position of the attacking robot and the position of the opponent's flag. Then, the advantage of the defender is the difference between defender's and attacker's distances to the point of intrusion. The third criterion for the dangerousness of the opponent's attacker also needs to be defined. Dangerousness $D(a)$ is similar to the function of a team's own attackers.

$$D(a) = \frac{1}{\|a, of\|}$$

Let $\|a, b\|$ stand for Euclidean distance between points a and b . Then

$$F(p, defender_a) = (\|p, pi_a\| - \|a, pi_a\|)D(a)$$

The actual solution that realizes these optimizations tasks differ in PIM and MAS approaches.

1) *PIM Implementation*: Since the PIM requires only one coordinator, it is better suited to perform a central computation of the optimization. For this purpose, binary linear programming works well for solving the optimization task in the PIM. Since the variables can take only the value 0 or 1, we define

matrix M of the values of the function $F = (player, role)$.

$$M = \begin{pmatrix} F(p_1, d_1) & F(p_1, d_2) & \dots & F(p_1, d_n) & F(p_1, a) \\ F(p_2, d_1) & F(p_2, d_2) & \dots & F(p_2, d_n) & F(p_2, a) \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ F(p_n, d_1) & F(p_n, d_2) & \dots & F(p_n, d_n) & F(p_n, a) \end{pmatrix}$$

We maximize

$$\sum_{i=1}^n \sum_{j=1}^{n+1} M(i, j)x_{(j+(i-1)*n)},$$

where x_i is the binary variable. To assign exactly one role to each player we define constrains

$$\sum_{j=1}^{n+1} x_{(j+(i-1)*n)} = 1$$

for each player $i \in \{1, \dots, n\}$.

Additionally we should ensure that every attacker is defended by no more than one defender. Therefore, we add the constrains

$$\sum_{i=1}^n x_{(j+(i-1)*n)} \leq 1$$

for each opponent's attacker $j \in \{1, \dots, n\}$.

2) *MAS Implementation*: As stated in Section III, the same set of strategies have been used for comparison between PIM and A-GLOBE. Thus, the potential for the use of negotiation based coordination in this experiment was somewhat limited. Negotiation was used mainly for detecting and solving conflicts within the role assignment computed by each individual agents. These conflicts arise as a result of the high dynamics of the game. Due to the fact that physical movement intervenes with the nontrivial reasoning of an agent, the computation always works with slightly outdated data. The experiment described in Section VII will illustrate how efficiently each of these two architectures deal with this type of impreciseness.

The agent computation is implemented in three phases:

1) Each agent computes its local utility of playing the role of attacker and defender (against each attacker). The agent then selects the role that maximizes its utility

$$\operatorname{arg} \max_{x \in \{a, d_1, d_2, \dots, d_n\}} F(p_1, x).$$

2) Each agent sends its selected role and locally computed utilities to other agents and receives utilities from other agents.

- If no two agents selected the same role, there is no conflict and the actual solution is adopted.
- If there are two or more agents selecting the same role (e.g., two robots defending the same attacker), the agent(s) with lower utility value updates its M matrix with the utility value of the agent with the highest utility value and proceeds to step 1.

If two agents selected the same role and they computed the same utility value for that particular role, the agent with lower id number performs re-planning as described in step 2.

B. Individual strategies

Individual strategies are the means by which a robot plays a particular role - either an attacker or a defender. The goal for an attacker is to knock down the opponent's flag and conversely the goal for a defender is to protect the local flag.

These two strategies are implemented in the common underlying core and the behavior for a particular robot is changed by the team strategy (that is, each robot is assigned the role of being an attacker or a defender). Much of the implementation is the same across the MAS and PIM implementation and only differ where necessary - in the planning algorithm.

1) *Attacker*: An attacking robot tries to find the optimal trajectory to knock down the flag. This trajectory must be collision free at the moment of planning. Of course, given the highly dynamic environment, the robot uses collision avoidance algorithms during trajectory execution. Shortest trajectory is planned by different algorithms. Dynamic programming is more suitable for the PIM as it provides trajectories for all attackers after one computation (given that all the robots are controlled by a single process). A* path planning algorithm is more suitable for the MAS approach, given that each attacker behaves independently and computes the trajectory independently.

2) *Defender*: The defending strategy is based on blocking entities. If an opponent's attacker assigned to a defender, the defender blocks the attacker by moving along a blocking trajectory. This trajectory is computed based on the attacker's shortest path to the flag.

VI. PERFORMANCE MEASUREMENT

Various performance measures were taken to compare the different approaches of the PIM and MAS systems. The expectation is that the PIM with a single coordinating process and the MAS system with negotiation will exhibit different behavior. The performance measurements try to cover as many different performance criteria as possible for evaluation.

This section reports on the following measurements to compare performance:

- *Code complexity* is one of the most important measurements and describes the complexity of code needed to realize the algorithm for implementing the game. It is measured in lines of code and number of classes.
- *Communication traffic* is another important measurement. The goal is to measure the amount of data and number of messages transferred between entities.
- *Communication overhead* measurement counts the percentage of processing time used for communication (preparation, transfer, processing, and utilization of data).
- *Reaction time* is one of the properties of the Capture the Flag game. This property describes the time needed for changing the roles of entities in the game from being an attacker to a defender (and vice versa). It can be described as time needed from making the decision to change to completing the assignment of all the new roles.
- *# of role changes in team* describes the team activity in the team planning. It measures how often a player

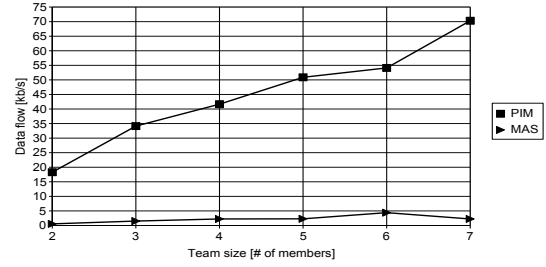


Fig. 3. Data flow comparison

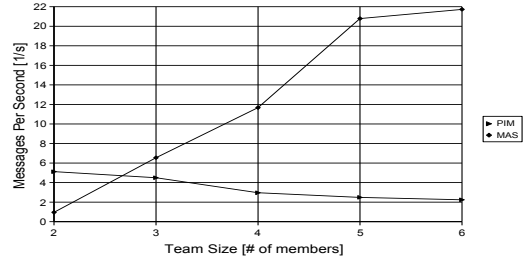


Fig. 4. Messages sent by one entity/node per 1 second

changes its role from a defender to an attacker or vice versa (mean value over team).

VII. EXPERIMENTS

The following experiments were conducted to measure performance. The game was played with different sized teams, varying from 2 to 7 members on each side. For PIM approach, the residency time was set to 100 ms.

The comparison of data flow needed to ensure coordinated behavior is shown in Figure 3. The data flow is measured in terms of the size of all transmitted messages by all the robots in a team in 1 second. Amount of data transferred by the PIM is larger, because it includes the complete run-time state of the coordination process.

Number of messages transmitted by one robot per second are shown in Figure 4. The decreasing trend in the PIM approach is caused by having a fixed residency time but an increase in the number of nodes. The increasing trend in the MAS approach is caused by more difficult negotiation over a larger group of team members.

Communication overhead (Fig. 5) is the percentage of processing time, spent for preparation of communication. The

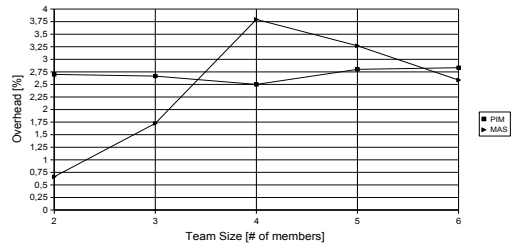


Fig. 5. Communication overhead

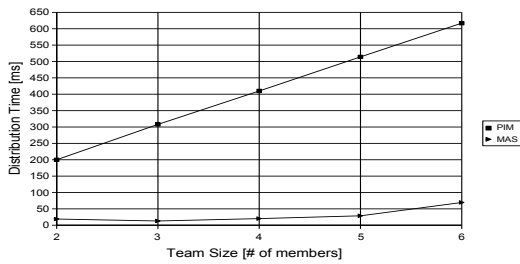


Fig. 6. Time needed to spread information over whole team

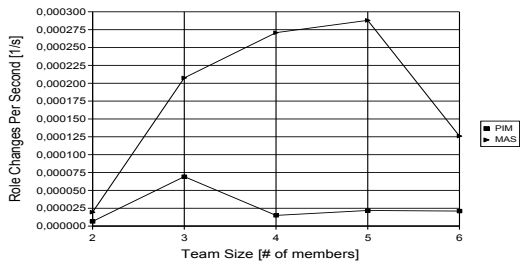


Fig. 7. Role change rate all changes per 1 second

PIM approach needs to save the complete runtime state, serialize it and also de-serialize it on the next node. The time needed for this is more or less fixed and hence the overhead strongly depends on the chosen residency time. The MAS approach prepares, sends, and parses XML messages used for negotiation. Differences are insignificant.

Distribution delay (Fig. 6) is time needed to change a role and make sure that the entire team knows and agrees with the change. Since there is only one decision-making entity in the PIM, this time depends purely on the number of robots. Distribution delay in MAS depends significantly on the team size and actual game state and can vary over a large range.

Change of roles (Fig. 7) represents the quality of the decision process. If a robot changes its role very often, performance of the robot is worse than if it keeps one role for long time.

Of course, the result of the games is also an interesting measure. Badly coordinated defense was main reason for a side losing the game. A win shows that the coordination was good enough to avoid collision in individual plans. During set of 50 runs, the PIM won 27 times while the MAS won 23 times. Note that this result is not dependent only on the underlying technology (PIM or MAS). Further optimization and improved algorithms are possible for both approaches.

VIII. CODE COMPLEXITY

The following table shows a comparison of code complexity for both approaches. The comparison is simplified by using line counts and ignoring comments and new lines.

The first row of the table compares classes needed to ensure coordinated behavior between members of the group. In this comparison, the PIM approach implements centralized coordination in contrast to all the negotiation protocols needed to implement in MAS approach.

Part	PIM		MAS	
	Lines	Classes	Lines	Classes
Coordination	671	5	4851	35
Role Assignment	538	3	1027	7
Path Planning	430	3	681	2

The second row on the table shows only parts needed to solve role assignment. In this case, it should be pointed out that the PIM approach uses an external library to solve the binary linear programming problem. The last row of table shows path planning implementation.

The comparison of these two approaches shows that the PIM approach uses far fewer lines of code (and classes) to achieve coordinated behavior in contrast to the MAS approach. This result was expected, given that the PIM allows an effectively centralized coordinator, which has all the knowledge about the members of the group and coordinates them at once. The MAS approach on the other hand needs to implement all the negotiation protocols, knowledge sharing, and synchronization mechanisms.

IX. CONCLUSION

The results validate the claim that the PIM approach simplifies the challenge of developing code to achieve coordinated behavior. Moreover, the behavior of the group is more predictable, when compared to the MAS approach. On the other hand, the PIM currently exhibits higher communication overhead and a small delay in information exchange. In the current implementation, the PIM is not able to handle unreliable communication links. This problem, as well as optimization of the communication overhead, will be the focus of future research.

X. ACKNOWLEDGEMENTS

This work was supported by the Office of Naval Research 'Coordinated Operations' Program, the Army Research Laboratories TEAM program under a subcontract from the University of Central Florida, and by research program number MSM 684077038 funded by the Ministry of Education of the Czech Republic.

REFERENCES

- [1] Rodney A. Brooks. Intelligence without reason. In John Myopoulos and Ray Reiter, editors, *Proceedings of the 12th International Joint Conference on Artificial Intelligence (IJCAI-91)*, pages 569–595. Sydney, Australia, 1991. Morgan Kaufmann publishers Inc.: San Mateo, CA, USA.
- [2] Milan Rollo, Petr Novák, and Pavel Jisl. Strategies for distributed underwater survey. In Michal Pěchouček, Paolo Petta, and László Zsolt Varga, editors, *Multi-Agent Systems and Applications IV*, volume 3690 of *Lecture Notes in Computer Science*, pages 657–660. Springer, 2005.
- [3] Erol Sahin. Swarm robotics: From sources of inspiration to domains of application. In Erol Sahin and William M. Spears, editors, *SAB*, volume 3342 of *Lecture Notes in Computer Science*, pages 10–20. Springer, 2004.
- [4] David Šišlák, Martin Rehák, Michal Pěchouček, Milan Rollo, and Dušan Pavlíček. A-globe: Agent development platform with inaccessibility and mobility support. In Rainer Unland, Matthias Klusch, and Monique Calisti, editors, *Software Agent-Based Applications, Platforms and Development Kits*, pages 21–46. Berlin, 2005. Birkhauser Verlag.