A Mission-Oriented Service Discovery Mechanism for Highly Dynamic Autonomous Swarms of Unmanned Systems

Vincent Autefage, Serge Chaumette, Damien Magoni

A Mission-Oriented Service Discovery Mechanism for Highly Dynamic Autonomous Swarms of Unmanned Systems

Vincent Autefage
Univ. Bordeaux, LaBRI
Talence, France
autefage@labri.fr

Serge Chaumette
Univ. Bordeaux, LaBRI
Talence, France
serge.chaumette@labri.fr

Damien Magoni
Univ. Bordeaux, LaBRI
Talence, France
magoni@labri.fr

***Abstract*—Over the past few years, many research projects have begun to focus on swarms of mobile unmanned systems (e.g., drones, ground robots) globally referred as UMS. These systems, because of the many sensors and actuators they can embed, are suitable for autonomous missions in 3D (Dull, Dirty and Dangerous) environments for instance. However, embedding a large number of capabilities in all of members of a swarm is expensive in terms of cost, weight and energy consumption. Thus, it is usually more efficient to embed only a single or a few capabilities within each UMS. It then becomes necessary to provide a discovery mechanism built into the swarm in order to allow its members to share their capabilities and to collaborate for achieving a global mission. These shared capabilities are called services. In this paper, we propose a new service discovery system called AMiRALE for *Asynchronous Missions Relay for Autonomous and Lively Entities* dedicated to highly volatile, autonomous and mobile swarms of UMS. Our solution is independent of both nodes' mobility and connectivity patterns. Moreover, it supports heterogeneous swarms and degraded conditions of operation (i.e., message loss, UMS loss and disconnected network). It is also totally decentralized and enables both discovery and service usage. We provide a description of the theoretical model of our AMiRALE system as well as several simulation results obtained from a park cleaning scenario.**

***Keywords*-Autonomous Swarm, UMS, Service Discovery, Collaboration.**

I. Introduction

Mobile swarms of UMS (*Unmanned Systems*) offer a lot of opportunities in both civil and military domains. Still, they raise a number of issues that need to be addressed in order to make it possible to use them in real world applications.

In the civil context, swarms can be used for applications like surveillance or search and rescue operations. We give two illustrative examples here. Among recent research is a project developed at the University of Pennsylvania [1] that targets the exploration of a damaged building with a number of tiny ground and aerial vehicles. The goal is to collect information about its state of degradation and to construct a 3D map of the inside (i.e., SLAM - *Simultaneous Localization and Mapping*). Another example is the CARUS project [2] which has been developed at the University of Bordeaux. The goal is to enable a swarm of UAVs (*Unmanned Aerial Vehicles*) to autonomously achieve the surveillance of a region by dynamically sharing small areas of this region between a number of UAVs. It can for instance be used to monitor the occurrence of forest fires. CARUS is focused on distributed and autonomous decision-making (including retasking) within the swarm contrary to most other research projects that are more or even totally centralized.

Of course, in the military domain swarming raises many opportunities to increase troops support and situation management. The army is pushing forward the research effort both in terms of algorithms and scenarios. In the US, the Tactical Technology Office (TTO) of the DARPA has issued a solicitation entitled Collaborative Operations in Denied Environment [3] (CODE), whose goal is to support research projects around aerial swarming. Among the scenarios put forward in the solicitation is tactical reconnaissance. In Europe many projects are also funded by governmental agencies and large companies.

Among the many issues raised by autonomous swarms is *service discovery*. Because of the cost of the hardware (e.g., sensors, actuators) and the limited weight and space of the payload that unmanned systems can embed, it is worth dispatching a subset of the capabilities required to achieve a mission among the entities of a swarm. It then becomes necessary to provide a system that enables an entity to locate a remote capability (and to use it by relocating its carrier if necessary). This is called *service discovery*. Of course, such a mechanism also has to provide access to the capability once it has been located which sometimes means moving the UMS that embeds this capability (e.g., a camera) to the appropriate location.

Mobile swarms of UMS are compliant with the MANET (*Mobile Ad-hoc Network*) model [4]; this kind of swarm defines a dynamic topology in which unmanned systems are the mobile communicating nodes. Even though service discovery has been widely studied in the context of MANETs [5], most of existing solutions are mainly focusing on the location problem; i.e., they do not provide any access mechanism, focusing only on the discovery aspect. In our context, capability relocation furthermore impacts the network and the discovery process itself. Indeed, both

operations (i.e., discovery and use) are achieved over a network configuration which keeps on changing all the time. This is due to the dynamic nature of such networks where the entities that embed the capabilities move around if required. Unfortunately, existing discovery mechanisms for ad-hoc networks are dependent on the mobility model of the nodes and usually require specific connectivity patterns between them. They thus cannot be used in our context.

Another key issue is that to ensure autonomy and to support degraded mode of operation, the approach has to be totally distributed with no node playing a central role. All nodes should thus run the same algorithm.

To the best of our knowledge, no discovery mechanism has been designed that would offer these specific characteristics.

In this paper, we propose an original and dedicated service discovery mechanism which we call AMiRALE (*Asynchronous Missions Relay for Autonomous and Lively Entities*) to solve these issues. The remainder of this paper is organized as follows. We first provide a state of the art focusing on the main types of existing service discovery systems in Section II. In Section III, we detail the theoretical model describing our service discovery mechanism for highly mobile and volatile swarms. In Section IV, we present several simulation results showing the performances of our service discovery model compared to two other models. We finally conclude and discuss future work in Section V.

II. State of the Art

Basically, *service discovery* is the way for an entity x (i.e., client) to locate a capability (i.e., service) which is available on another node (i.e., provider) of the network. Service discovery requires the availability of information on the services that can be used including their location in the network. Three major architectures are usually considered [5]:

- *centralized directory-based* where a unique node hosts the location of all the services of the network;
- *distributed directory-based* where the service directory is split over several nodes;
- *directory-less* where each node hosting a service is in charge of announcing it to the rest of the network nodes.

According to previous studies [6] [7], the *directory-less* architecture is the most robust when a high level of mobility is considered but it is more adapted to small scale rather than large scale networks. Moreover, it is admitted that a decentralized architecture is more fault-tolerant, has a natural exploitation of parallelism, is reliable and scalable [8]. This kind of service discovery mechanisms can operate in two ways [9]:

- *pull mode* which is a *reactive-like* strategy;
- *push mode* which is a *proactive-like* strategy.

In *push mode*, providers constantly broadcast the list of services they offer which means that a client will be able to know (with a good probability) where a service is located before trying to use it. In *pull mode*, the client does not know the holder of a service in advance. The location is computed at the request time. In a highly dynamic network, the *push mode* is the best in terms of delay because accessing the service does not require to calculate the location at the request time [6] [10] which can be costly and inefficient if the network keeps on changing. However, the presence of a large number of services can decrease the performance of the approach because of the many announcement messages that can lead to a high packet collision rate [11].

Despite the existence of many service discovery mechanisms, several problems are still open [12]:

- *Service usage* (i.e., the way to access a service once it has been discovered) is not addressed in most of existing solutions.
- *Service selection* which is required when a same service is hosted by several different nodes is also often ignored.
- The *performance* of MANET applications is closely dependent on the mobility of the nodes of the network [6] [13]. This is due to the fact that mobility has a strong impact on connectivity patterns [14]. A service discovery mechanism should thus be able to adapt its behavior to this parameter.

We also want to address two other problems in this paper:

- *Autonomy*: so as to ensure degraded mode of operation, each node of the swarm should be totally autonomous and take its own decisions. In other words, there can be no supervisor; i.e., no entity should play a central role.
- *Resilience*: the support of the volatility in the network. Indeed, nodes (providers or clients) may join or leave the network at any time without jeopardizing the current tasks of the swarm.

As explained before, mobility has a strong impact on the efficiency of the discovery process and the size of the network is important too. In order to provide an efficient and scalable solution, it is thus crucial to make it context-adaptive [7]:

- a *small scale* and a *highly mobile* swarm should use a proactive directory-less mechanism (flooding based approach) ;
- a *medium scale* swarm with *limited mobility* should use a directory-less mechanism on top of a routing overlay (cross-layer approach) ;
- a *large scale* swarm with *little mobility* should use a directory-based mechanism on top of a routing overlay.

To the best of our knowledge, other mixes of scale and mobility have not been studied because they do not correspond to effective configurations. In this paper, we

focus on a part of this global system, AMiRALE, which is our solution for reasonable scale (i.e., few dozens of nodes) and highly mobile swarms of autonomous UMS. It should nevertheless be noticed that AMiRALE remains operational, even if not the best solution, for swarms with low mobility. In addition to service discovery, it also addresses service usage and service selection as introduced in the previous section. It is totally decentralized and supports dynamic changes of the topology (i.e., mobility and resilience).

III. Theoretical Model

A. Overall Description

Most existing service discovery mechanisms focus on the location of the services rather than on the actions induced by them. We can draw a parallel between *IP networks* and *content-based networks* [15]. A content-based network is more focused on the data itself than on its holder. AMiRALE is based on a similar idea. In the autonomous swarms that we consider, UMS are programmed to collaboratively achieve missions. Usually, missions are partly known at swarm ignition time, while they can still evolve because of external events. Events are caught by sensors and actions are achieved by using UMS capabilities (i.e., sensors and actuators).

For a specific event e, we call $Sens_e$ (sensor) a node which can capture this event. This node generates a new mission called $m_e^{n:k}$ where n is the identifier of the creator node and k is a mission sequence number relative to this node. This mission is described by a message (a *view* called $v_e^{n:k}$ which is a reduced form of the mission $m_e^{n:k}$) that travels the network through intermediate nodes called $Forw_e$ (forwarders), until it reaches a final node which can solve the mission $m_e^{n:k}$. Such a final node is called $Solv_e$ (solver). One of the fields of the mission description $m_e^{n:k}$ is an internal state which evolves while the view travels through the network. This state can take five different values which are strictly ordered. This order is used later in this paper to describe the evolution of the system:

1) *start*: the mission has been created but no node is currently trying to solve it;
2) *will*: the mission has been caught by a solver, and it is preparing to solve it (e.g., moving to the location where the mission takes place);
3) *do*: the mission is currently being solved;
4) *abort*: the mission has been dropped because it has apparently already been solved (e.g., a target object that the solver is supposed to remove has disappeared);
5) *end*: the mission is solved.

The *start* state is only set at mission creation time by the node that generates it. The other states can only be set by solvers. A forwarder is a *read-only node*; i.e., it cannot modify the state field. As explained before, states are strictly ordered: $start < will < do < abort < end$. A finite state machine of the evolution of the state field is shown in Figure 1.

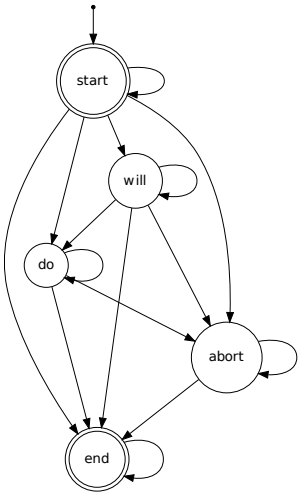

Figure 1. Finite state machine of the state mission field.

We assume here that each UMS has a unique identifier and its own clock. Each mission description contains a state, a creator identifier and creation date, the date of modification and the identifier of the last node that updated it. At regular time intervals, the nodes of the swarm broadcast to their neighborhood the list of mission views they are aware of in order to update each other. Note that nodes communicate only asynchronously and one way.

When a $Sens_e$ node catches an event *e*, it is possible that another mission initiated because of *e* already exists. However, the decision to generate a new mission is application dependent, therefore a user function called f_{ignore}^e will decide if the event *e* has to be ignored or not. When a node receives a newer view of one of the missions it is aware of, it updates this mission in its local memory and broadcasts the new mission view. When a solver gets a mission in the *start* state that it is able to solve, it turns the state field of the mission to *will* and prepares itself to solve it (e.g., by moving to a specific location if required). When it begins solving the mission, it turns its state field to *do*. If the solver detects that the mission has apparently already been solved (e.g., a target object that the solver is supposed to remove has disappeared), it turns the state field to *abort*. Finally, when it has succeeded in solving the mission, it turns the state field to *end*. A solver can be in *will* or *do* mode for only one mission at a time. Furthermore, a solver $Solv_e$ can stay in the *will* (resp. *do*) state only for a limited time Ψ_{will}^e (resp. Ψ_{do}^e). If one of these thresholds is exceeded, the solver leaves the mission if and only if it is informed by another solver that this last one is now taking care of the same mission. Additionally, if a solver is informed that the mission it is dealing with (*will* or *do* state) has evolved to a greater state, it leaves the mission and updates the relative description. For a specific mission relative to an event *e*,

if a node is not a $Sens_e$ nor a $Solv_e$, it is considered as a $Forw_e$. By default, a forwarder is able to deal with a mission of type *e* even it has not been aware of this kind of event before. Indeed, views contain specific information (i.e., thresholds) of the relative type. In other words, it is possible to add several new mission types during the swarm operation.

B. Time Synchronization and Clock Issues

As explained before, each node has its own internal clock and the time can thus drift differently from one node to the other. Consequently, it is necessary to take this problem into account in our discovery system. Three main approaches exist [16]:

- *relative ordering* where events are ordered without time reference;
- *relative timing* where nodes take into account the time drift relative to the others;
- *global timing* where all the clocks are synchronized (by using GPS information for instance).

Consequently, we have developed three versions of AMiRALE, one for each synchronization technique. In this paper, we only detail the global timing one. The complete description of each version of the theoretical model is available in a technical report [17].

C. AMiRALE Global Timing Model

We decided on formalizing this model on top of the ADAGRS [2] [18] (*Asynchronous Dynamicity Aware Graph Relabeling System*) which is only based on one way broadcasts and local computations. The limitations imposed by this model are the keys to ensure degraded mode of operation and autonomy by fostering a totally decentralized and distributed approach.

As explained before, a node can have three different roles:

- $Sens_e$ which means that the node can sense an event *e* and creates the relative mission $m_e^{n:k}$;
- $Solv_e$ which means that the node can solve a mission $m_e^{n:k}$;
- $Forw_e$ which means that the node can forward a mission $m_e^{n:k}$ (default role if the node is not a sensor nor a solver for the type *e*).

We define an additional generic role called Any_e that stands for any of the above roles and what we use to simplify the description of the formal model.

Rules are used to describe the interactions of a node with a mission of type *e*. There are 5 different rule types which are presented in Figure 2. For each of these rules, the 2 circles represent the role of the node before and after applying the rule. The set under the circles represents the current value of the mission $m_e^{n:k}$. The rule is applied only if the precondition c is satisfied.

We now describe the different rules:

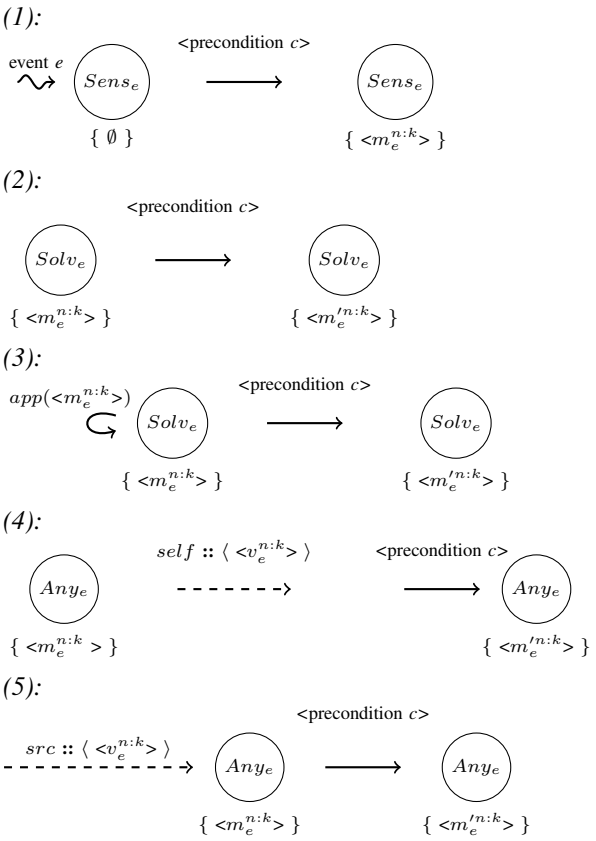

Figure 2. Rules types of the AMiRALE model.

1) This rule is the only one that causes a new mission to be created. A $Sens_e$ node named *n* creates the mission $m_e^{n:k}$ if precondition c is true. This precondition will be used in the instantiation to check if a similar mission initiated because of the same event *e* is already being solved or under resolution.
2) This rule enables a solver node to autonomously modify the state of a mission. After applying this rule, the mission $m_e^{n:k}$ is updated to its new version $m_e'^{n:k}$.
3) This rule enables a solver node to react to an applicative event $app(m_e^{n:k})$ (e.g., action success, action aborted) and to modify the related mission description.
4) This rule enables a node to broadcast a mission view. This broadcast operation is described as $self :: v_e^{n:k}$ where $self$ is the identifier of the current node and $v_e^{n:k}$ the view of the current mission $m_e^{n:k}$.
5) This rule enables a node to modify a mission after being informed of a new version thereof. The received message is described as $src :: v_e^{n:k}$ where $v_e^{n:k}$ is the view of the mission $m_e^{n:k}$ received from the sender called src.

Each mission $m_e^{n:k}$ is a 7-tuple $\{e, k, n, t, s, n', t'\}$ where e is the type of the mission (i.e., event type), k is a mission sequence number relative to its initiator node and n is the

identifier of the node that has created the mission (i.e., the initiator node). The 3-tuple $\{e, k, n\}$ is the identifier of the mission. t is the date of its creation, s is its current state (i.e., *start*, *will*, etc.), n' is the identifier of the last node which updated the current state, and t' is the date of the last mission update. Data are not represented in our formal model. As explained before, each node broadcasts the view of the missions it is aware of at regular time intervals. In the global timing approach, a view is equal to the relative mission; i.e., $m_e^{n:k} = v_e^{n:k}$. A view can also embed extra information (e.g., position of the sender, battery life time).

We provide in Figures 3 to 8 the complete system of rules that composes the global timing AMiRALE model. We note *self* the node which applies the rule and *now* the current date according to the clock of *self*. We remind that clocks are globally synchronized in this version of AMiRALE. *free()* indicates that the node applying the rule is not in a *will* or *do* state for any mission for the moment. As indicated here-before, the model uses several user functions which are application dependent. We call those user functions *filters*. The complete list of filters is detailed as follows:

- $f^e_{ignore}(m_e^{n:k})$ is used to decide if a new mission should be created or not when a new event is captured by a sensor node (e.g., a similar mission was perhaps already initiated because of the same event *e*).
- $f^e_{select}(self : m_e^{n:k}, src : v_e^{n:k})$ enables the local node $self$ to decide if it has to leave the mission $m_e^{n:k}$ because src, which has sent the view $v_e^{n:k}$, is also taking care of the mission and is more advanced in the process.
- $f^e_{blind}(m_e^{n:k})$ is used to decide if a view should be broadcasted or not. This filter can help to reduce potential network traffic and collisions.
- $f^e_{pass}(m_e^{n:k})$ is used by a $Solv_e$ to decide if the mission $m_e^{n:k}$ should not be selected. This function enables to implement a mission selection scheduler and to prevent several nodes to select certain missions (e.g., battery is too weak to solve this mission).
- $f^e_{check}(v_e^{n:k})$ is used by nodes to ignore several views. This function enables to implement safety verifications or security policy (e.g., several nodes are not allowed to share or to modify certain mission types).

Rules of Figures 5 and 7 enable the swarm to be resilient since it ensures that a mission can be taken back by another solver if the first one has encountered several failures (e.g., engine failure, low battery, solving problem). Service usage is provided by solver local rules (Figure 5 and 6). Service selection is implemented through both solver local decisions rules and solver message reception rules (Figure 5 and 7). The autonomy is given by the use of only local computations and asynchronous communications (Figure 4). Finally, filters enable to customize the model behavior to a target scenario and therefore improve the collaboration performances.

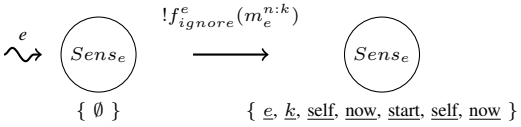

Figure 3. Sensor rule of AMiRALE.

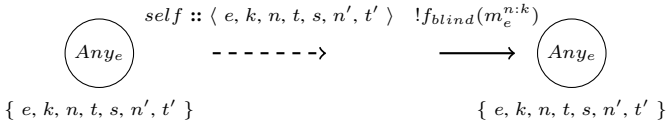

Figure 4. Message (view) sending rule of AMiRALE.

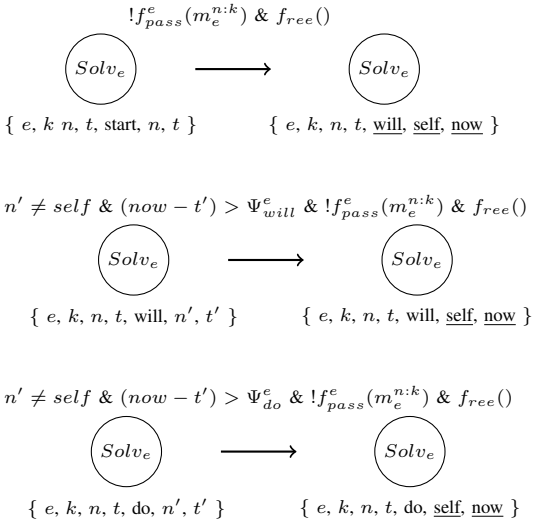

Figure 5. Solver local-decision management of AMiRALE.

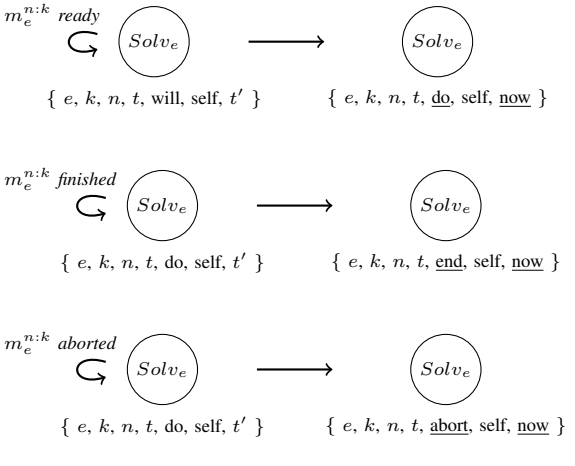

Figure 6. Solver local-event management of AMiRALE.

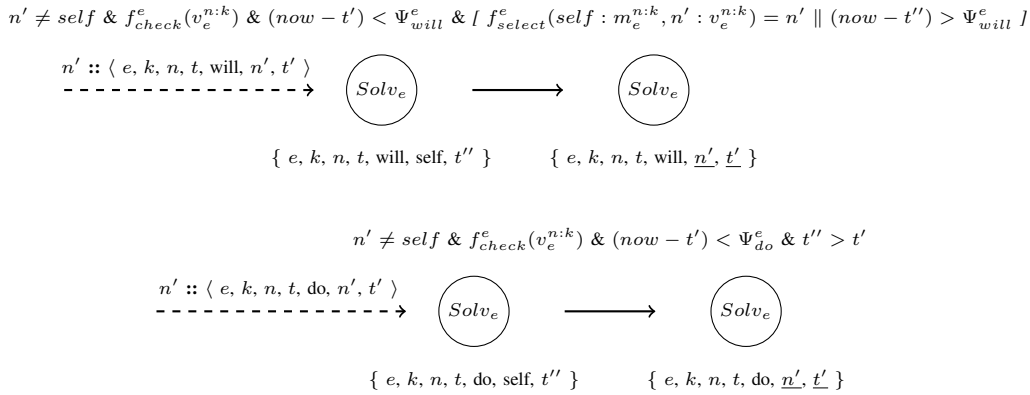

Figure 7. Solver message (view) reception of AMiRALE.

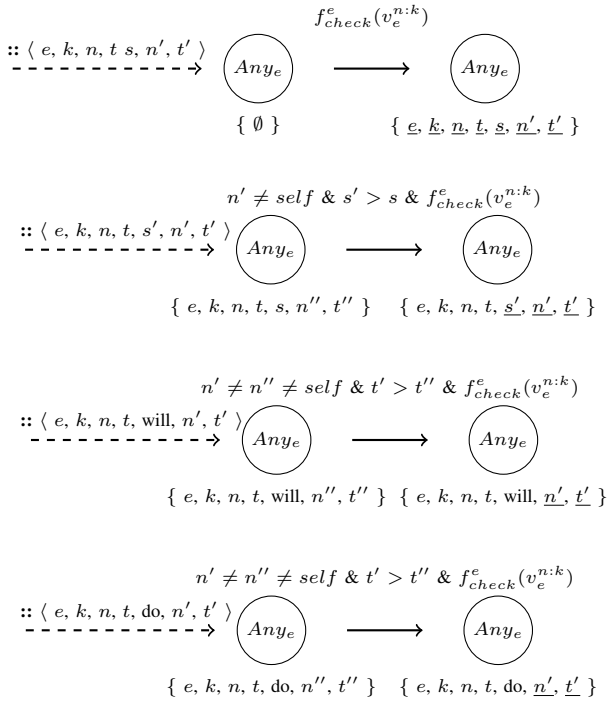

Figure 8. General message (view) reception of AMiRALE.

D. Flooding Control

The service discovery mechanism that we propose is based on flooding. Consequently, if the density of nodes is very high, network congestion can appear [19]. Several solutions exist to solve this issue. AMiRALE uses the aggregation of messages [6]. All views are sent in one unique broadcast packet which significantly reduces the network congestion. It is furthermore possible to use three other efficient techniques [20]:

- *counter-based approach* which prevents a node from broadcasting a view that has been received or sent more than *n* times in a given time interval;
- *probabilistic-based approach* which causes a view to be broadcasted depending on a pseudo-random draw;
- *jitter-based approach* in which a small delay is added before sending a message. This technique enables to de-synchronize the broadcasts issued by nodes and to remove part of the collisions.

These techniques can easily be added to our service discovery system in both the formal model (by means of the f^e_{blind} user function) and the real implementation.

IV. Evaluation

A. Scenario and Collaborative Models

In order to assess the benefits of AMiRALE, we chose to evaluate it in a park cleaning scenario called ParCS [21]. Indeed, this scenario offers a solution to achieve a selective collection of waste in a park by using an autonomous swarm composed of UGVs (*Unmanned Ground Vehicles*). These vehicles are self-organized so as to manage the collection of waste and to clean the park. Each UGV is specialized, which means that it can only clean one kind of garbage (e.g., paper, glass, compost, plastic). This specialization makes the heterogeneity aspect of the swarm. However, each UGV is able to sense any kind of garbage. In this scenario, each mobile robot is moving by following a *random way-point mobility model* [22]. All waste pieces are thrown uniformly in the park.

In order to evaluate the performances of our proposed solution, we compare it with two other existing schemes:

- the *worst-case* solution: a full random behavior with no communication between robots what we call *Mute model*;
- the *best-case* solution: a centralized and omniscient *Black Board* system with a shared-memory what we call *Black Board model*.

1) Mute Model: in this solution, all mobile robots are moving randomly and independently from the others. Each UGV cleans a garbage of its own type when it senses it.

There is no communication here, which means that a garbage will be cleaned only if a mobile robot moves close enough to it during its travel. This solution is the worst possible in terms of collaboration and efficiency since it is more a collective approach than a collaborative scheme [23] [24].

2) Black Board Model: the *Black Board* [25] is a well known model usually used for solving expert problems [26]. The main idea of the Black Board is that a group of specialized entities are working together on a common problem. This problem can be subdivided in several pieces (i.e., tasks) that can be achieved by the different specialists in function of their specific capacities. When an entity is able to solve a task, it locks it on the black board. When the task is over, it is removed from the Black Board. This concept has been widely used in computer science for many years and possesses a lot of benefits [27]. This approach is based on *opportunistic cooperation* as AMiRALE is. The *shared-memory* Black Board implementation is the best in terms of collaboration efficiency but it is completely unrealistic [28]. Indeed, in this implementation all the nodes share a single memory which means that any event, action or knowledge from a node is simultaneously available for the others. In this approach, no communication is needed. This model is theoretically the best since the information dissemination is provided atomically. For this solution, each entity is also moving randomly and cleaning compliant garbage. However, when a mobile robot senses a garbage of an incompatible type, its location is written on the Black Board. Each robot consults periodically the content of the Black Board and locks the closest compatible garbage that it finds when it is possible. When a garbage is locked, the responsible robot goes directly to the garbage position in order to remove it. When the garbage is cleaned, the corresponding information is removed from the Black Board. Te preemption is not allowed, which means that a locked garbage cannot be cleaned by another robot. In the case of robot failure (i.e., robot is done), the possible locked garbage is automatically unlocked by the Black Board itself. In other words, not any *dead lock* can appear in this scheme [29].

B. AMiRALE Configuration

AMiRALE is used here to enable the collaboration between mobile robots. The mission type *e* represents a type of garbage (e.g., paper, glass, compost, plastic). Thus, a robot which can clean an *e* garbage is a $Solv_e$. For all *e*, all robots are $Sens_e$; i.e., each robot can sense garbage of any types. User defined functions and variables are configured as follows:

- f^e_{ignore} returns true when a mission already exists for the sensed garbage in the sensor missions database.
- f^e_{pass} is used to select the closest available and compatible garbage for a $Solv_e$.
- f^e_{blind} is used to reduce the mission diffusion when this last one is in the *end* state and not any other robots has emitted a contradictory information for more than 1000 seconds.
- f^e_{select} enables to select locally the closest solver between the two implied in the information exchange (i.e., each view contains the position of the sender).
- f^e_{check} is not used here since the security aspect is out of the scope of the paper.
- Ψ^e_{will} and Ψ^e_{do} are fixed to 1000 seconds.
- Each robot broadcasts its views every 5 seconds.

C. Simulation Parameters

We have run our simulations in a dynamic graph simulator called JBotSim [30] [31]. This software is actually a Java library which enables the design of low and high level scenarios and behaviors of communicating heterogeneous mobile nodes. Our simulation parameters are described in Table I.

Table I
EXPERIMENTATION PARAMETERS

Parameter	Value
Park size	1000 x 1000 m
Robots size	1 x 1 m
Robots speed	5 m / seconds
Robots sensing range	30 m
Robots communication range	30 m
Number of runs per simulation	200

We evaluate here the complete cleaning time in function of several parameters:

- number of garbage: number of garbage per type of garbage (Figure 9);
- number of robots: number of robots per type of garbage (Figure 10);
- number of types: number of existing garbage types in the park (Figure 11);
- frequency of failures: frequency at which a random robot is deleted from the map and replaced by a new similar and memory-empty one (Figure 13).

D. Simulation Results

Figure 9 exhibits the total cleaning time in seconds for the different solutions (Mute, Black Board and AMiRALE) as a function of the number of garbage per type. The number of types is fixed to 6 as the number of robots per type (i.e., 36 robots in total). As expected, the cleaning time increases as the number of garbage is more and more important. The important values of the standard deviation on the Mute plot is due to the pure random aspect of the solution. Indeed, in this case, each robot moves randomly and independently from the others in the map, cleaning garbage during their travel. The performance of this solution is strongly linked to the correlation between the initial robots' position, the distribution of garbage on the map and the behavior of the random way-point mobility model. This is the the reason

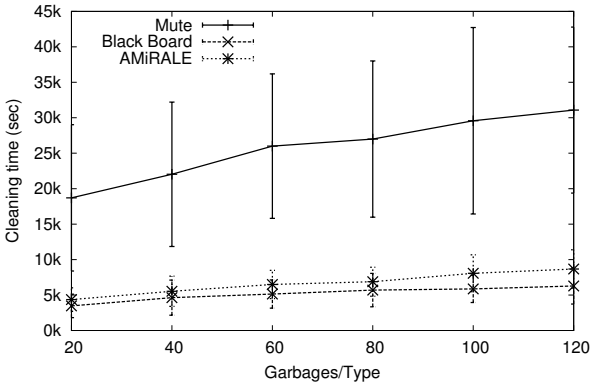

Figure 9. Cleaning time for 6 types and 6 robots per type as a function of the number of garbage (36 robots in total).

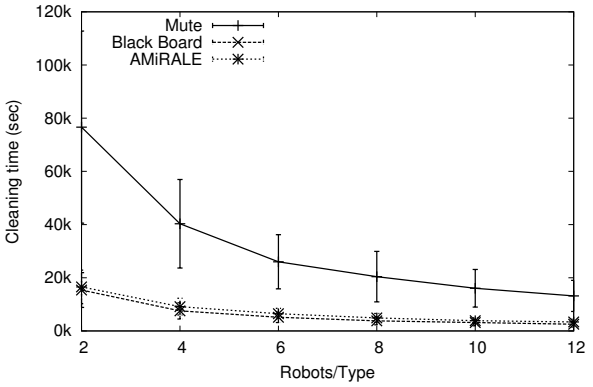

Figure 10. Cleaning time for 6 types and 60 garbage per type as a function of the number of robots (360 garbage in total).

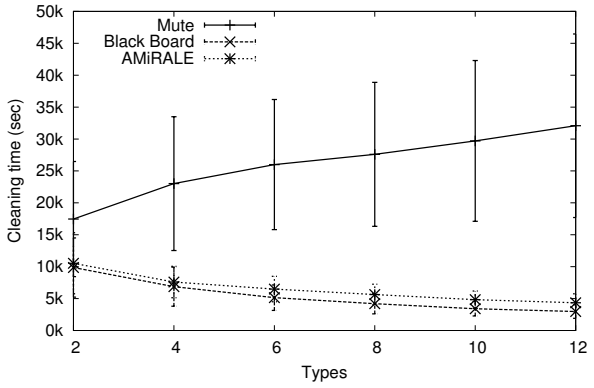

Figure 11. Cleaning time for 60 garbage per type and 6 robots per type as a function of the number of types.

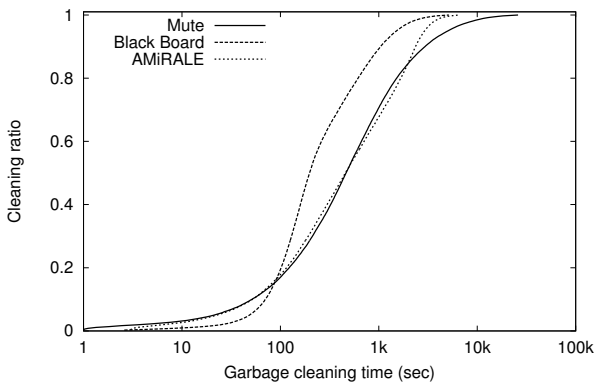

Figure 12. Cleaning time CDF for 6 types, 6 robots per type and 60 garbage per type (36 robots and 360 garbage in total).

why results are fluctuating. We can also notice that the Black Board and the AMiRALE plots are very close. Of course, the Black Board model is better since it does not require any communication between robots and thus does not suffer from any distance effect between them. Also, it is not facing synchronization issues between versions of the global mission state since it is using a single shared-memory between robots. AMiRALE provides good results compared to the Black Board since it only uses local computations and asynchronous messages.

Figure 10 exhibits the total cleaning time in seconds for the different solutions as a function of the number of robots per type. The number of types is fixed to 6 and the number of garbage per type is fixed to 60 (i.e., 360 garbage in total). Here, increasing the number of robots reduces the total cleaning time. Again, the Mute solution exhibits poor results since the Black Board and AMiRALE provide quite equivalent results. We can also notice that the steepness of both the Black Board and the AMiRALE plots tends towards zero as the number of robots per type increases.

Figure 11 exhibits the total cleaning time in seconds for the different solutions as a function of the number of types. The number of robots per type is fixed to 6 and the number of garbage per type is fixed to 60. Therefore, the global numbers of robots and garbage are increasing with the number of types. The Mute solution is less and less efficient as the number of types increases. This means that the increase of the number of garbage has a stronger effect than the increase of the number of robots. On the contrary, Black Board and AMiRALE plots decrease with the growing of the number of types. Indeed, each robot is able to locate and communicate to the swarm the existence of a garbage through sensing even if this last one has a different type. Again, the performance of AMiRALE and the Black Board are close.

Figure 12 is a cumulative distribution function of garbage cleaning time. The number of types is fixed to 6 as the number of robots per type and the number of garbage per type is fixed to 60 (i.e., 36 robots and 360 garbage in total). We can notice here that the benefit of the AMiRALE approach is more visible for the cleaning of the last 20% of garbage.

Figure 13 exhibits the total cleaning time in seconds for the different solutions as a function of the failures frequency.

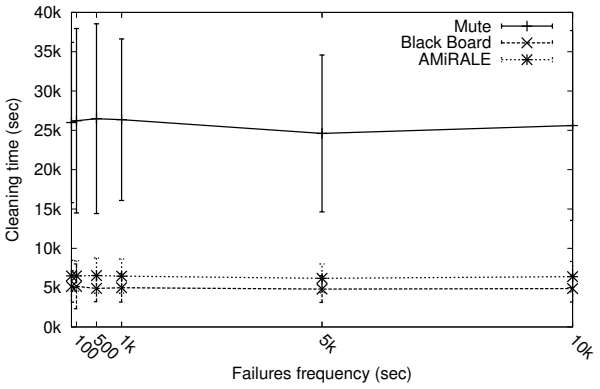

Figure 13. Cleaning time for 6 types, 6 robots per type and 60 garbage per type as a function of the failure frequency (36 robots and 360 garbage in total).

The number of types is fixed to 6 as the number of robots per type and the number of garbage per type is fixed to 60 (i.e., 36 robots and 360 garbage in total). At each reset, a random robot is removed from the map and replace by a fresh new similar and empty-memory one. As expected, the impact of a reset on the Black Board is negligible since the memory of the removed robot is not lost due to the presence of the shared-memory. The Mute solution is not facing any important effects of the reset either. The small variations observed are due to the random behavior of the robots. We can notice that AMiRALE is also not affected by the reset. Indeed, even if the memory of the removed robot is lost, the replication mechanism inside the AMiRALE model allows the fresh new replacing robot to recover a coherent global mission state of the scenario by communicating with the other nodes. We can thus claim here that AMiRALE is resilient to node failures even for a high frequency (e.g., every 100 seconds).

V. Conclusion

We have presented AMiRALE, a novel service discovery mechanism for highly mobile, autonomous, volatile and collaborative swarms of UMS. It solves issues related to services location, selection and usage. It supports a degraded mode of operation thanks to the approach that relies only on one-way broadcast communications and local computations. It is totally distributed and decentralized.

We have detailed the theoretical model and have performed several simulations on a park cleaning scenario in order to compare the benefits of AMiRALE to a collective solution (i.e, Mute model) without communication between robots and an optimal theoretic solution based on the Black Board model which relies on a global shared-memory for each robot. We have shown that AMiRALE exhibits results close to the Black Board model and that it is resilient to a notable frequency of node failures.

We have already evaluate the two other versions of AMiRALE (i.e., relative ordering and relative timing) with the presented one (i.e., global timing) on the same park cleaning scenario [32]. One of our future goals is to add other vehicle types such as *Unmanned Aerial Vehicles* to our park cleaning scenario in order to improve the sensing process. We will run a prototype of AMiRALE on virtual machines interconnected by a network emulator such as *NEmu* [33] [34] which will enable us to carry out realistic performance measurements. We also plan to experiment our scenario in a real park with real vehicles [21]. We have already started a collaboration with the Mugen Company[1] for this purpose. Finally, we plan to demonstrate the benefits of AMiRALE for other applications and devices related to connected objects, smart cities and the Internet of Things.

Acknowledgment

This work is partly funded by the *Direction Générale de l'Armement*[2] and the *Région Aquitaine*[3].

References

[1] N. Michael, S. Shen, K. Mohta, Y. Mulgaonkar, V. Kumar, K. Nagatani, Y. Okada, S. Kiribayashi, K. Otake, K. Yoshida, K. Ohno, E. Takeuchi, and S. Tadokoro, "Collaborative mapping of an earthquake-damaged building via ground and aerial robots," *Journal of Field Robotics*, vol. 29, no. 5, pp. 832–841, September 2012.

[2] S. Chaumette, R. Laplace, C. Mazel, R. Mirault, A. Dunand, Y. Lecoutre, and J.-N. Perbet, "Carus, an operational retasking application for a swarm of autonomous uavs: First return on experience," in *Proceedings of the 30th IEEE MILCOM*, November 2011, pp. 2003–2010.

[3] DARPA, *Collaborative Operations in Denied Environment (CODE)*, 2014, solicitation Number: DARPA-BAA-14-33.

[4] J. Mackar and S. Corson, *Mobile Ad hoc Networking (MANET): Routing Protocol Performance Issues and Evaluation Considerations*, IETF, 1999, rfc 2501.

[5] C. Ververidis and G. Polyzos, "Service discovery for mobile ad hoc networks: a survey of issues and techniques," *IEEE Communications Surveys Tutorials*, vol. 10, no. 3, pp. 30–45, March 2008.

[6] J. Hoebeke, I. Moerman, B. Dhoedt, and P. Demeester, "Analysis of decentralized resource and service discovery mechanisms in wireless multi-hop networks," in *Wired/Wireless Internet Communications*, ser. Lecture Notes in Computer Science. Springer, 2005, vol. 3510, pp. 181–191.

[7] A. Mian, R. Baldoni, and R. Beraldi, "A survey of service discovery protocols in multihop mobile ad hoc networks," *IEEE Pervasive Computing*, vol. 8, no. 1, pp. 66–74, January 2009.

[1] http://mugen-sas.com
[2] http://www.defense.gouv.fr/dga
[3] http://aquitaine.fr

[8] Y. Cao, A. Fukunaga, A. Kahng, and F. Meng, "Cooperative mobile robotics: antecedents and directions," in *Intelligent Robots and Systems 95. 'Human Robot Interaction and Cooperative Robots', Proceedings of IEEE/RSJ International Conference on*, vol. 1, August 1995, pp. 226–234.

[9] L. Cheng, "Service advertisement and discovery in mobile ad hoc networks," in *Proceedings of CSCW*, 2002.

[10] C. Mbarushimana and A. Shahrabi, "Comparative study of reactive and proactive routing protocols performance in mobile ad hoc networks," in *21st International Conference on Advanced Information Networking and Applications Workshops (AINAW)*, vol. 2, May 2007, pp. 679–684.

[11] U. Mohan, K. Almeroth, and E. Belding-Royer, "Scalable service discovery in mobile ad hoc networks," in *Networking Technologies, Services, and Protocols; Performance of Computer and Communication Networks; Mobile and Wireless Communications*, ser. Lecture Notes in Computer Science. Springer, 2004, vol. 3042, pp. 137–149.

[12] J. Su and W. Guo, "A survey of service discovery protocols for mobile ad hoc networks," in *International Conference on Communications, Circuits and Systems (ICCCAS)*, May 2008, pp. 398–404.

[13] T. Camp, J. Boleng, and V. Davies, "A survey of mobility models for ad hoc network research," *Wireless communications and mobile computing*, vol. 2, no. 5, pp. 483–502, 2002.

[14] G. Ravikiran and S. Singh, "Influence of mobility models on the performance of routing protocols in ad-hoc wireless networks," in *IEEE 59th Vehicular Technology Conference (VTC)*, vol. 4, May 2004, pp. 2185–2189.

[15] A. Carzaniga and A. L. Wolf, "Content-based networking: A new communication infrastructure," in *Developing an Infrastructure for Mobile and Wireless Systems*, ser. Lecture Notes in Computer Science. Springer, 2002, pp. 59–68.

[16] B. Kaur and A. Kaur, "A survey of time synchronization protocols for wireless sensor networks," in *International Journal of Computer Science and Mobile Computing*. IJCSMC, 2013, vol. 2, no. 9, pp. 100–106.

[17] V. Autefage, "Amirale formal model - a service discovery and collaboration system formalism based on dynamic graph relabeling," LaBRI - University of Bordeaux, Tech. Rep., February 2015, https://hal.archives-ouvertes.fr/hal-01114961/.

[18] R. Laplace, "Applications et services DTN pour flotte collaborative de drones," Ph.D. dissertation, University of Bordeaux, December 2012.

[19] S. Ni, Y. Tseng, Y. Chen, and J. Sheu, "The broadcast storm problem in a mobile ad hoc network," in *Proceedings of the 5th annual ACM/IEEE international conference on Mobile computing and networking*. ACM, 1999, pp. 151–162.

[20] B. Williams and T. Camp, "Comparison of broadcasting techniques for mobile ad hoc networks," in *Proceedings of the 3rd ACM International Symposium on Mobile Ad Hoc Networking &Amp; Computing*, ser. MobiHoc '02. ACM, 2002, pp. 194–205.

[21] V. Autefage, A. Casteler, S. Chaumette, N. Daguisé, A. Dutartre, and T. Mehamli, "Parcs-s2 : Park cleaning swarm supervision system – a position paper," in *9th AIRTEC International Congress*, October 2014.

[22] C. Bettstetter, H. Hartenstein, and X. Pérez-Costa, "Stochastic properties of the random waypoint mobility model," *Wireless Networks*, vol. 10, no. 5, pp. 555–567, September 2004.

[23] L. E. Parker, "Distributed intelligence: Overview of the field and its application in multi-robot systems," in *AAAI Fall Symposium on Regarding the Intelligence in Distributed Intelligent Systems*, 2007.

[24] ——, "Distributed intelligence: Overview of the field and its application in multi-robot systems," *Journal of Physical Agents*, vol. 2, no. 1, pp. 5–14, 2008.

[25] D. D. Corkill, "Blackboard systems," *AI expert*, vol. 6, no. 9, pp. 40–47, 1991.

[26] C. Ingram, R. Payne, S. Perry, J. Holt, F. Hansen, and L. Couto, "Modelling patterns for systems of systems architectures," in *IEEE 8th Annual Systems Conference (SysCon)*, March 2014, pp. 146–153.

[27] J. Hunt, "Blackboard architectures," *JayDee Technology Ltd*, vol. 27, 2002.

[28] D. D. Corkill, "Design alternatives for parallel and distributed blackboard systems," Tech. Rep., 1988.

[29] N. Kaveh and W. Emmerich, "Deadlock detection in distribution object systems," in *Proceedings of the 8th European Software Engineering Conference Held Jointly with 9th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, ser. ESEC/FSE-9. ACM, 2001, pp. 44–51.

[30] A. Casteigts, "The jbotsim library," *CoRR*, vol. abs/1001.1435, 2010.

[31] ——, *JBotSim*, http://jbotsim.sourceforge.net.

[32] V. Autefage, S. Chaumette, and D. Magoni, "Comparison of time synchronization techniques in a distributed collaborative swarm system," in *24th European Conference on Networks and Communications (EuCNC)*. IEEE, June 2015.

[33] V. Autefage and D. Magoni, "Network emulator: A network virtualization testbed for overlay experimentations," in *IEEE 17th International Workshop on Computer Aided Modeling and Design of Communication Links and Networks (CAMAD)*, September 2012, pp. 266–270.

[34] V. Autefage, *Network Emulator for mobile universes (NEmu)*, http://nemu.valab.net.