Resource Determination and Allocation in Sensor Networks: A Hybrid Approach

MURAT ŞENSOY[1], THAO LE[1], WAMBERTO W. VASCONCELOS[1], TIMOTHY J. NORMAN[1], ALUN D. PREECE[2]

[1]*Department of Computing Science, University of Aberdeen, AB24 3UE, United Kingdom*
[2]*Computer Science, Cardiff University, Cardiff CF24 3AA, United Kingdom*
Email: {m.sensoy, thao.le, w.w.vasconcelos, t.j.norman}@abdn.ac.uk, A.D.Preece@cs.cf.ac.uk

Many organizations depend on critical sensory information to achieve their tasks. As the number of those tasks increase, efficient determination and allocation of required resources in sensor networks become crucial. In this paper, we propose means to describe tasks semantically with their requirements and constraints so that software agents can reason about those tasks and determine what type of sensor resources they may need. Based on the semantic description and reasoning mechanisms, we propose a distributed agent-based approach to efficiently allocate sensor resources to tasks. Our evaluation of the proposed approach shows that not only it enables fully automated determination and allocation of resources for tasks, but also the resulting allocation is efficient and close to optimum.

Keywords: Sensor networks, Semantic Web, Resource allocation

Received 01 Month 2009; revised 01 Month 2009

1. INTRODUCTION

Information acquired from networks of sensors plays an important role in many critical tasks including the detection of environmental phenomena and surveillance of a region. Sensor networks are typically required to support multiple tasks, each having varying or competing information needs and requirements. Consequently, only a subset of available sensor resources may be useful for a specific task, but, at the same time, some tasks may be in competition for specific resources. The information requirements of, and constraints on tasks determine what types of sensors should or could be used. Only after analyzing task requirements and constraints may the most useful sensor types be determined, and instances of sensor resources allocated. These sensor resources are organized in (often ad-hoc) networks, providing the infrastructure to support their coordination and the transfer of sensor data to enable the delivery of the information needs of tasks.

Sensor networks can be categorized as stationary, mobile, and hybrid. In stationary sensor networks, sensors are static and distributed over an area to sense a specific activity. In this setting, resources within the network are mainly the sensor instances with different properties (e.g., type, range, battery life and so on). In mobile sensor networks, sensors are attached to autonomous platforms that provide mobility in different environments (e.g., air, underwater, and so on). Unmanned Aerial Vehicles (UAVs), Autonomous Underwater Vehicles (AUVs) and robots are some examples of platforms with different operational capabilities. Mobility enables sensors to move and sense activities in a larger area. Similarly, platforms in a network may coordinate in order to explore a region collaboratively with fewer sensors. Moreover, a set of sensors with different capabilities, in some cases, can be attached to the same platform to sense different types of activity simultaneously in the same area. In this setting, resources within the network are not only sensor instances, but also platform instances that have a capacity to mount and support specific sensor types. We note that platforms not only mount sensors, but they may also affect the capabilities of sensors attached to them; some sensors may provide certain capabilities only if they are attached to a specific type of platform. Some sensor networks are hybrid — composed of both stationary and mobile sensors. These networks inherit properties of stationary and mobile sensor networks.

A task may be atomic or composite. A composite task is composed by subtasks, which are either composite or atomic, while an atomic task cannot be divided further into subtasks. In this way, we can define a composite task recursively and create a hierarchy of its subtasks that should be carried out in a specific order. In this hierarchy, different subtasks of a task may have highly different requirements and goals. For example, a composite task for monitoring global warming is shown in Figure 1. This task is composed of three subtasks, where a subtask requires to monitor icebergs using a stationary sensor network, another task

requires a mobile sensor network to monitor the atmosphere, and a third task requires a hybrid network of mobile and stationary sensors to detect changes in the ocean. This illustrates that tasks may need a variety of resources to meet their requirements.

Determining resource types required by a task is non-trivial; it requires comprehensive domain knowledge and reasoning capabilities. Existing systems assume that human experts determine the types of resources required by a task at design time, then the resources of those types are allocated using various methods [7, 14, 20, 3, 10]. However, in highly dynamic and uncertain environments, requirements of tasks may change. As a result, previously determined resource types should be revised. This may result in serious delays in a task, because the human experts should be "in the loop" all the time.

After determining the necessary resource types for a task, specific instances need to be identified and allocated. If a task cannot secure all of its necessary resources, it may fail. Although the resources are critical for tasks, they may be on high demand, scarce in quantity and hard to deploy. This creates a motivation for tasks from different or the same organizations to cooperate by sharing their resources to achieve their goals. However, the cooperation between tasks is not trivial, because each task is autonomous and has its own success metrics, constraints and policies.

In this paper, we propose a framework for the resource determination and allocation problem for tasks in wireless sensor networks. First, we propose an ontology-based approach to describe tasks and determine resource types required by atomic tasks making up a composite task. Our approach relies on a knowledge rich semantic representation to describe tasks and uses a novel combination of ontological reasoning and logic programming to determine resource types required by atomic tasks. Then, we propose a multi-agent framework to allocate instances of these resources to the tasks. In the proposed approach, each task is represented by an agent, which is responsible for the task and initiation of its subtasks if any. Similarly, sensors in an area are represented by autonomous agents. Based on the interactions between the agents, required resources for tasks are allocated incrementally. Our evaluation shows that the proposed approach has a good performance in terms of time complexity and successfully allocation of resources.

The rest of the paper is organized as follows. In Section 2, we overview our agent-based architecture with an example. Section 3 describes how tasks are represented semantically and the relationship between resource types and tasks. Section 4 presents the combination of ontological reasoning and logic programming for flexible matchmaking of resource types to atomic tasks. Section 5 proposes a method for allocating sensors of specific types to the atomic tasks. Section 6 evaluates our approach using simulations. Section 7 discusses our proposal with references to existing related work. Lastly, Section 8 concludes the paper with an overview of our contributions and plans for future work.

2. ARCHITECTURE OVERVIEW

We propose a multi-agent system where each task is represented by a task agent. The task agent is responsible for the task. If a task is composed of subtasks, then that task's agent delegates those subtasks to other task agents. If a task agent represents an atomic task, then the agent is only responsible for the determination and allocation of resources required to execute the task. In summary, agents of composite task are responsible for the delegation of subtasks to other agents while the agents of atomic tasks are responsible for the determination and allocation of resources.

Resource determination and allocation for each atomic task is managed by the agent of that task. Hence, for a composite task, overall resource determination and allocation is achieved in a decentralized manner by the agents representing atomic tasks within the composite task. The agents of atomic tasks first determine the necessary resource types and then interact with the sensors (resources) on the area of their interest to allocate necessary resources. Sensors may have different types, locations, ranges, batter life and so on. This means each sensor has a different utility for each task. In our approach, each sensor is represented by a sensor agent, knowledgeable about the location, range, type, battery life and utility of its sensor. Therefore, in order to allocate sensors for a specific atomic task, the agent of the task should interact with the sensor agents considering its requirements and constraints (e.g., location, range, utility, and so on).

Using the composite task example in Figure 1, Figure 2 illustrates how tasks are delegated to agents and how these agents interact in our framework. We can summarize the scenario in Figure 2 as follows:

1. The semantic description of Global Warming Monitoring (GWM) task is fed into the system.
2. The GWM task is delegated to a task agent (GWM agent), which initiates and coordinates subtasks of the GWM task using the task description.
3. The GWM agent delegates the atomic task *Monitoring Icebergs (MI)* to a task agent (MI agent). The MI agent reasons about the required resource types using the description of the MI task.
4. After determining the required types of sensors, the MI agent allocates sensors by interacting with the sensor agents in the area.
5. During the execution of MI task, a melting iceberg is detected and the GWM agent is informed by the MI agent.
6. As defined in the description of GWM task, the GWM agent initiates *Detect Changes in Ocean (DCO)* task by delegating it to a new task agent (DCO agent). The DCO task is composed of three parallel atomic tasks. Therefore, the DCO agent delegates those tasks to three task agents, which are responsible of the atomic tasks: *Monitoring Thermal Changes*, *Monitoring Ocean Animals* and *Monitoring Seismic Activities* respectively. These agents autonomously reason about

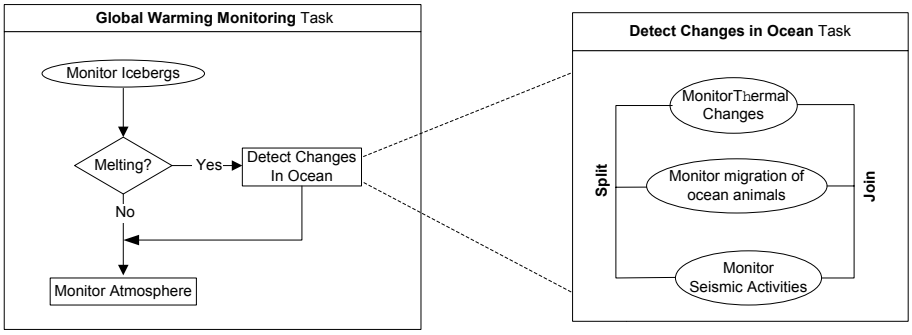

FIGURE 1. A composite task example for monitoring effects of global warming. In the figure, composite tasks are represented using rectangles while atomic tasks are represented using ellipses.

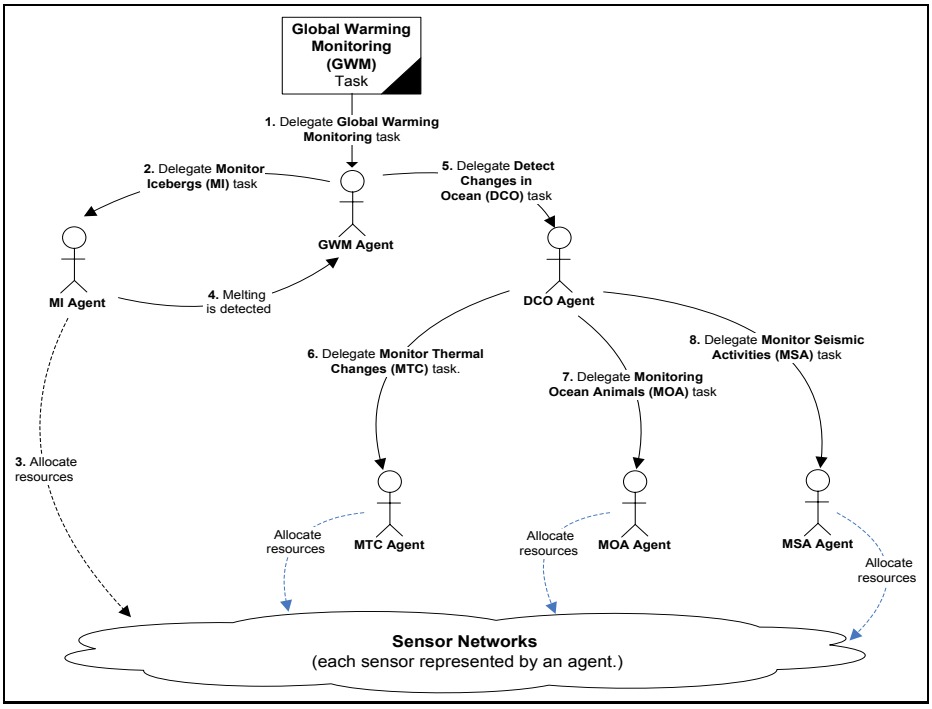

FIGURE 2. A partial scenario for **Global Warming Monitoring** task in Figure 1.

the atomic tasks they represent and determine the most useful sensor types. Lastly, they interact with the sensor agents to allocate required sensors of the determined types.

The scenario in Figure 2 reveals two important challenges. The first one is the representation of tasks so that agents can reason about them. The reasoning at a task level involves the interpretation of a task's flow (e.g., which subtasks should be activated) and determination of required types of resources. Semantic Web technologies provide knowledge representation languages and tools to support task-level reasoning. The second challenge is the allocation of specific resources for an atomic task, once the types of necessary resources are determined by its agent through reasoning. In this paper, we integrate Semantic Web technologies with multi-agent systems in a novel way to handle these two challenges. The next two sections give details about our approach for the representation of tasks and determination of required resource types for a specific task.

3. KNOWLEDGE REPRESENTATION

In this section, we propose means to semantically represent tasks using OWL-DL [24] and describe what kind of resources a task may require.

3.1. Semantic Representation of Tasks

We represent a task as a workflow composed of a set of subtasks with temporal and logical relationships between them (e.g., sequence, if-then-else). In order to provide a formal grounding to our model of semantic task workflows, we build upon the OWL-S process ontology [13] combined with an OWL-based domain ontology that captures properties of tasks within a specific domain. Therefore, task agents can easily understand, interpret and reason about the task representations.

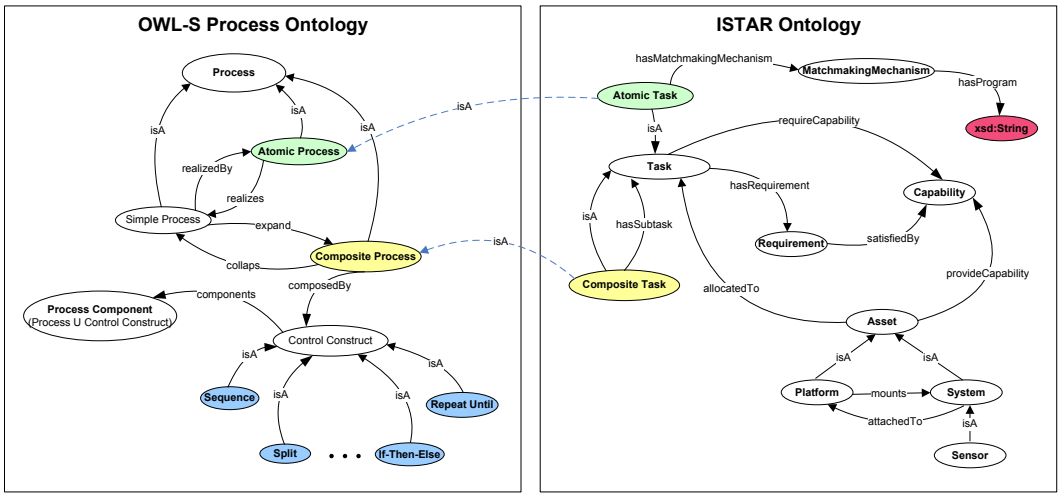

FIGURE 3. Combination of OWL-S's process ontology and ISTAR ontology.

The OWL-S process ontology is proposed to describe Web Service processes as semantic workflows [13]. OWL-S defines three categories of process: simple, atomic and composite. Composite processes are described using atomic or other composite processes using temporal and logical relationships (e.g., sequence, split, if-then-else and so on). In our context, atomic tasks are considered as atomic processes. Similarly, composite tasks are considered composite processes that are composed of other processes. Thus, we can combine the OWL-S process ontology with domain specific ontologies in an intuitive manner to describe tasks recursively with formal semantics.

Our approach is flexible enough to work with different domain ontologies, that is, we can describe tasks for different domains. However, to ground our presentation, we address the Intelligence, Surveillance, Target Acquisition and Reconnaissance (ISTAR) domain[1] in the rest of the paper. In order to describe tasks in this domain, an ISTAR ontology is combined with the OWL-S process ontology as shown in Figure 3. The ISTAR ontology describes the relationships between resources and tasks requiring those resources, while the OWL-S process ontology describes the relationships between tasks. More specifically, the ISTAR ontology states that each task may require capabilities to achieve its objectives and resources may provide various capabilities. The *Platform* and *System* concepts are both resources, but systems can be attached to platforms. Sensors are regarded in the ontology as a specialization of systems. We note that the ISTAR ontology shown in Figure 3 contains only core concepts and relationships. However, it is easily extended by adding other OWL ontologies to further elaborate on different concepts.

Each task may have requirements, which are used to select the most appropriate resources for them. The requirements of a composite task are inherited by its subtasks. Requirements can be associated with the task in at least three ways. First, the task can be defined abstractly in an ontology together with its default requirements. Second, new requirements can be explicitly placed onto the task during design time. Third, constraints defined within the context of the task may add new requirements to the task, or modify its existing requirements. Figure 4 shows how an atomic task *Road Surveillance* may be defined. This task has one operational requirement[2], namely *Constant Surveillance*, and one intelligence requirement, namely *Imagery Intelligence* (IMINT) capability. If we assume that a road surveillance task is an instance of this task, then it inherits these two requirements.

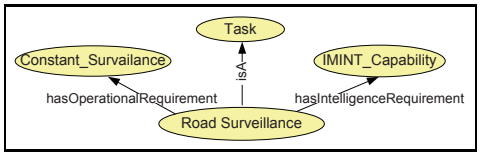

FIGURE 4. Abstract task example.

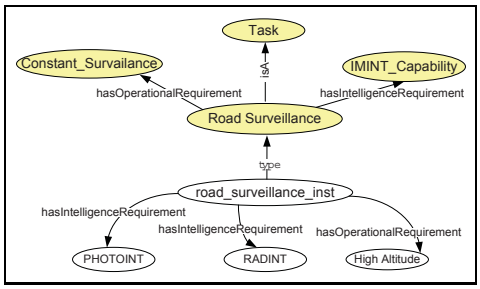

FIGURE 5. Task instance example.

Let us suppose that the road surveillance task to be executed in a mountainous area during the winter. The constraints imposed on the road surveillance task affect its requirements as follows. First, a *high altitude* requirement is added, because this task will be executed in a mountainous

[1] http://en.wikipedia.org/wiki/ISTAR

[2] The object properties *hasOperationalRequirement* and *hasIntelligenceRequirement* are defined as sub-properties of *hasRequirement* in the ISTAR ontology.

area. Second, because the road surveillance will be carried out during the winter (when snow, rain and fog are highly probable and imagery vision can be badly affected), *Radar Intelligence* (RADINT) is added to the requirements of the task. Figure 5 shows the resulting road surveillance task instance along with its full set of requirements.

As explained above, constraints may affect the requirements of a task. For this purpose, in the domain ontology, we have to use semantic rules [1] to represent the relationships between constraints (e.g., terrain and weather conditions) and requirements (e.g., high altitude and radar intelligence). These rules are important because they capture the crucial domain knowledge critical to the overall tasks.

3.2. Resources Required by Tasks

Allocating resources to a task corresponds to allocating resources to its subtasks. This leads to the argument that allocating resources for a task amounts to allocating resources to all of the atomic tasks within the task. Therefore, henceforth we mainly describe our approach using atomic tasks.

Each atomic task may require different types of resources, because each atomic task may have different requirements. Moreover, some atomic tasks may have requirements that cannot be met by a single resource type. In those cases, different resource types should be allocated together to meet the requirements of an atomic task. We use the term *Deployable Configuration* in order to refer to the set of resource types that an atomic task needs. Deployable configurations are defined formally in a domain-independent way as follows. First, we make use of three finite and non-empty sets:

- Resource *types* $T = \{t_1, \ldots, t_n\}$
- Resource *capabilities* $C = \{c_1, \ldots, c_m\}$
- Task *requirements* $R = \{r_1, \ldots, r_p\}$

Sample sets for ISTAR domain are

- $T = \{GlobalHawk, EOCamera, Reaper, DayLightTV\}$
- $C = \{LargeAreaCoverage, NightVision, HighResImage\}$
- $R = \{HighAltitude, IMINT\}$

A set of types $T' \subseteq T$ is formally related via the function κ to a set of capabilities $C' \subseteq C$, that is, $\kappa : 2^T \mapsto 2^C$. This formalization aims at capturing dependencies between resource types while providing certain capabilities – when resource types are put together, they provide *combined* capabilities, as in, for instance, $\{GlobalHawk, EOCamera\}$ provides $\{LargeAreaCoverage, HighResImage\}$ but $\{GlobalHawk\}$ only offers an empty set of capabilities.

Second, a set of requirements $R' \subseteq R$ is formally related via the function σ to a *set of sets* of capabilities $\{C'_0, \ldots, C'_q\}, C'_i \subseteq C, 0 \leq i \leq q$, that is, $\sigma : 2^R \mapsto 2^{2^C}$. This formalization aims at capturing another important aspect – a set of requirements can be met by various different capabilities put together. An example of this is how requirement $\{IMINT\}$ can be met differently by $\{NightVision\}$ or $\{HighResImage\}$.

Lastly, a *deployable configuration* is a set of resource types providing the necessary (and sufficient) capabilities for a set of requirements. More formally, we have:

DEFINITION 1. Given a set of requirements $R' \subseteq R$ with associated capability sets $\sigma(R') = \{C'_0, \ldots, C'_q\}$, a deployable configuration $DC \subseteq T$ is a set of types such that, for at least one $C'_i, 0 \leq i \leq q$, $C'_i \sqsubseteq \kappa(DC)$ which means that each capability $c' \in C'_i$ is semantically subsumed by a capability $c \in \kappa(DC)$. Moreover, for any proper subset of a deployable configuration $DC_j \subset DC, DC_j \neq DC$, there is no $C'_i, 0 \leq i \leq q$, such that $C'_i \sqsubseteq \kappa(DC_j)$. ∎

The definition above forges the *necessary* (first part of the definition) and *sufficient* conditions (second part of the definition) for deployable configurations. These must be minimal: only those essential types should be in the configuration and nothing else.

As illustrated in Example 1 below, there may be different ways of executing an atomic task using different types of resources. In the definition of deployable configurations, we keep the functions κ and σ as abstract as possible, as they should be defined differently for distinct domains. This means that the mechanism used to determine deployable configurations of an atomic task may change for different tasks. In some settings, each resource type may be independent and provide specific capabilities. In other settings, however, resource types may depend on one another to provide capabilities. Example 2 and 3 below demonstrate how the determination of deployable platforms may vary in distinct settings.

EXAMPLE 1. Assume that an atomic task has two deployable configurations: [*GlobalHawk*, *EOCamera*] and [*Reaper*, *DaylightTV*], where *GlobalHawk* and *Reaper* are autonomous Unmanned Aerial Vehicles (UAVs) while *EOCamera* and *DaylightTV* are sensor types. This means that there are two different ways of executing this task. The first way is to use only the resource types *GlobalHawk* and *EOCamera*. Alternatively, another way is to use only the resource types *Reaper* and *DaylightTV*. The task may select one of these configurations before allocating resources. The selection may depend on the utility of these configurations for the task.

EXAMPLE 2. In stationary wireless sensor networks, the capabilities of a sensor type may not depend on other sensor types. For instance, a thermal sensor does not depend on other types of sensors to sense thermal activity within its range. In this setting, capabilities provided by sensor types are additive. That is, a surveillance task requiring thermal and acoustic information simply needs two different types of sensors: thermal sensors and acoustic sensors.

EXAMPLE 3. In mobile sensor networks, sensors are attached to platforms such as Unmanned Aerial Vehicles (UAVs), Autonomous Underwater Vehicles (AUVs), autonomous robots and so on. Hence, they can move within

the region of interest and provide the required sensing information. Platforms have a pre-defined number of slots onto which particular kinds of sensors can be attached. There are domain-specific constraints that determine what platforms can be used with which sensors to provide a specific capability. For instance, a surveillance task may require imagery intelligence. This requirement can be met by sensors that provide the imagery intelligence, however the task may also need platforms to carry and support those sensors. Let us assume we have only two types of platforms that provides constance surveillance capability: *GlobalHawk* and *Reaper*. Additionally, we have only three types of sensors with imagery intelligence capability: *EOCamera*, *IRCamera*, and *DaylightTV*. *GlobalHawk* can only carry and support sensor types *EOCamera* and *IRCamera*, while *Reaper* can only mount *DaylightTV*. As a result, we can compose only three different deployable configurations for the task: [*GlobalHawk, EOCamera*], [*GlobalHawk, IRCamera*], and [*Reaper, DaylightTV*].

Determination of deployable configurations for a specific task typically requires extensive domain knowledge and expertise as the previous examples illustrate. In the following section, we propose to combine ontological reasoning and logic programming to provide a flexible solution for the computation of deployable configurations.

4. DETERMINING REQUIRED RESOURCE TYPES

Determination of deployable configurations for a specific task can be modeled as a semantic matchmaking problem where sets of resource types are matched to tasks. During matchmaking, we may use ontological reasoning to determine the best combinations of resource types that provide capabilities required by the tasks.

Example 2 and Example 3 show that different tasks may require different mechanisms for computing deployable configurations. Hence, we must be able to use different matchmaking mechanisms in order to compute deployable configurations for different tasks. That is, assuming that a generic matchmaking mechanism is able to handle all tasks is not realistic and can be very restrictive in many settings. Instead, in this section, we propose to enable different tasks to use different matchmaking mechanisms. For this purpose, we propose Ontological Logic Programming (OLP), which is a novel combination of ontological reasoning and logic programming. Different matchmaking mechanisms are described using OLP and associated with an ontology as an instance of *MatchmakingMechanism* concept. In order to associate different tasks with different matchmaking mechanisms, an object type property *hasMatchmakingMechanism* is created, whose domain is the *Task* concept while its range is the *MatchmakingMechanism* concept. In order to find the most adequate resource types for a specific task, our matchmaking algorithm simply gets the matchmaking mechanism related to the task and execute the associated OLP program.

Figure 6 shows the OLP stack used for matchmaking. At the base of the stack, we have domain ontologies

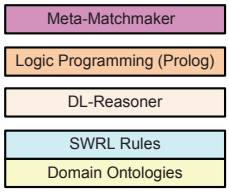

FIGURE 6. OLP Stack.

in OWL [24]. Some rules are associated with these ontologies using SWRL [1]. Above the ontologies and the rules, we have a Description Logic (DL) [2] reasoner such as Pellet [23]. This reasoner is used to infer facts and relationships from the ontologies and rules. Above the reasoner, we have a Logic Programming (LP) interpreter. Our choice of LP language is Prolog and in our implementation, we use a pure Java-based Prolog interpreter [15]. At the top of the OLP stack, we have our meta-matchmaker. The meta-matchmaker is responsible for retrieving the right matchmaking mechanism for a task from the ontology, and then it uses an appropriate Prolog meta-interpreter to interpret the related OLP program. The meta-interpreter uses the DL reasoner during an execution and returns the matchmaking results to the meta-matchmaker.

Figure 7 shows a simplified version of the Prolog meta-interpreter used to interpret OLP programs through the *eval* predicate, while Figure 8 shows a simple matchmaking mechanism written in OLP. The code in Figure 8 is a Prolog program, where concept and properties from the underlying ontologies are referenced directly. In order to differentiate between ontological and other predicates, we use name-space prefixes separated from the predicate name by a colon. Predicate names from the ontology are written in single quotes, so as to allow for any strings to name predicates; this is necessary due to Prolog's convention of representing variables with strings starting with capital letters. For example, we can directly use ontological predicate *istar:requireCapability* in a OLP program without defining its semantics in Prolog, where *istar* is a prefix that refers to *http://www.csd.abdn.ac.uk/ita/istar*. Note that the Prolog knowledge-base does not have any information about ontological predicates, since these predicates are not defined in Prolog, but described separately in an ontology. Therefore, in order to interpret OLP program, the Prolog meta-interpreter needs the ontological reasoning provided by the DL reasoner. The meta-interpreter accesses the DL reasoner through the *loadFromOntology* predicate as shown in Figure 7. This predicate is a reference to a Java method, which queries the reasoner and loads the necessary facts into the Prolog knowledge-base.

During the interpretation of an OLP program, if a predicate in *prefix:name* format is encountered, the DL reasoner below the interpreter in the OLP stack is queried to get direct or inferred facts about the predicate in the underlying ontologies. For example, if the meta-interpreter encounters the ontological predicate *istar:requireCapability* during its interpretation of an OLP program, it queries the

```
eval(not(G))  :- not(eval(G)).
eval((G1,G2)) :- eval(G1),eval(G2).
eval((G1;G2)) :- eval(G1);eval(G2).
eval((O:G))   :- ontology(O,G).
eval(G)       :- not(complex(G)), (clause(G,B), eval(B);
                 not(clause(G,_)),call(G)).
complex(G)    :- G=not(_) ; G=(_;_) ; G=(_,_) ; G=(_:_).
ontology(O,G) :- loadFromOntology(O,G),call(O:G).
```

FIGURE 7. Simplified Prolog meta-interpretter for OLP.

```
getConfigurations(Task,Sensors):-
  extendSolution(Task,[], Sensors).
extendSolution(T,Prev,Next):-
  requireSensor(T,Prev,X),
  A=[X|Prev],
  extendSolution(T,A,Next).
extendSolution(T,S,S):-
  not(requireCapability(T,S,_)).
requireSensor(T,S,X):-
  requireCapability(T,S,C),
  istar:'Sensor'(X),
  istar:'provideCapability'(X,C).
requireCapability(T,S,C):-
  istar:'requireCapability'(T,C),
  not(provideCapability(S,C)).
provideCapability([Y|Tail],C):-
  istar:'provideCapability'(Y,C),!;
  provideCapability(Tail,C).
```

FIGURE 8. A matchmaking mechanism example, where a deployable configuration is composed of sensor types whose capabilities are additive. The function *getConfigurations* computes deployable configurations for a specific task.

DL reasoner. The *requireCapability* predicate is defined in ISTAR ontology using SWRL rules, so the reasoner interprets these rules to derive facts. Then the facts returned by the reasoner are loaded into the Prolog knowledge-base and interpretation of the OLP program is resumed. Therefore, we can directly use the concepts and properties from ontologies while writing logic programs and the facts are imported from the ontology through a reasoner when necessary. Caching mechanisms are used to improve performance. OLP enables us to combine the advantages of logic programming (e.g., complex data types/structures, negation by failure and so on) and ontological reasoning.

The OLP program in Figure 8 is a simple matchmaking mechanism, where the *getConfigurations* predicate computes deployable configurations for a specific task. This is an incremental algorithm, which starts with an empty set and iteratively attempts to add new resource types to this set if the resource type provides a required capability, which is not provided by the resource types in the current set. The algorithm assumes that resource types are independent and capabilities are additive. Hence, this algorithm can be used to determine deployable configurations for the case in Example 2. In an ontology, OLP programs are associated with instances of the *MatchmakingMechanism* concept using a data type property and tasks are associated with different matchmaking mechanisms using *hasMatchmakingMechanism* object type property.

Although our example in Figure 8 is simple, it is straightforward to create sophisticated matchmaking mechanisms for the cases where resources are co-dependent and their capabilities are not additive. For example, the OLP

```
getConfigurations(T,[P|S]):-
  deployablePlatform(T,P),
  extendSolution(T,P,[],S).
deployablePlatform(T,P):-
  istar:'Platform'(P),
  not((istar:'requireOperationalCapability'(T,C),
       not(istar:'provideCapability'(P,C)))).
extendSolution(T,P,Prev,Next):-
  requireSensor(T,P,Prev,X),
  istar:'mounts'(P,X),
  A=[X|Prev],
  extendSolution(T,P,A,Next).
extendSolution(T,P,S,S):-
  not(requireCapability(T,P,S,_)).
requireSensor(T,P,S,X):-
  requireCapability(T,P,S,C),
  istar:'provideCapability'(X,C).
requireCapability(T,P,S,C):-
  istar:'requireCapability'(T,C),
  not(provideCapability(S,C)),
  not(provideCapability([P],C)).
provideCapability([Y|Tail],C):-
  istar:'provideCapability'(Y,C),!;
  provideCapability(Tail,C).
```

FIGURE 9. A matchmaking mechanism example, where a deployable configuration is composed of platform and sensor types whose capabilities are not additive; instead sensors and platforms are interdependent.

program in Figure 9 computes deployable configurations for settings such as those in Example 3. In such settings, each sensor must be carried by an available platform that provides all of the operational requirements of the task (e.g., constant surveillance). If a sensor cannot be carried by an available platform, there is no point in considering deployable configurations with that sensor type. Using this knowledge, a tailored and efficient matchmaker can be employed. This matchmaker first identifies the deployable platforms that meet the requirements of the task. Once many possibilities are narrowed down by determining deployable platforms, the sensor types that provide the intelligence capabilities required by the task are determined so that those sensors can be mounted on the deployable platforms.

5. ALLOCATION OF RESOURCES

In the previous sections, we have described how tasks can be semantically represented and how deployable configurations for atomic tasks can be computed efficiently. Given a set of deployable configurations for an atomic task, various resource allocation approaches can be used to allocate the most adequate resources to the task. In this section, however, we present a novel approach to this important problem for sensor network management that offers significant efficiency improvements over existing solutions, while generating high quality solutions.

In our framework, a task agent is responsible for each task. The execution order of subtasks within a composite task is explicitly described using the OWL-S process ontology. Using this description, the agent of the composite task delegates subtasks of the task to other agents, when necessary, as illustrated in Section 2. Agents responsible of atomic tasks determine the deployable configurations and allocate resources for their tasks.

An agent A^t for an atomic task t is knowledgeable about the constraints and requirements of the task t. As described in Section 4, A^t computes the set of deployable configurations for t. Each deployable configuration may have a different value for the task, hence computed deployable configurations are ordered by the agent depending on a predefined criteria of the task (e.g., budget of the task, cost and utility of resources, and so on). The resulting ordered list of deployable configurations is denoted as $L^t_{dc} = \langle dc^t_0, \ldots, dc^t_n \rangle$, where dc^t_0 is the most desirable deployable configuration of t while dc^t_n is the least desirable.

Agents for the atomic tasks are responsible for the allocation of resources to the tasks they represent. For this purpose, A^t tries to allocate instances of resource types in the most desirable deployable configuration dc^t_0. If it fails to allocate instances of those resource types, the agent moves onto the next desirable deployable configuration. Task t fails if A^t cannot allocate the required resource instances for any deployable configuration of t. This may also lead to the failure of other tasks that have t as a subtask.

For example, if the task agent has L^t_{dc}= $\langle [GlobalHawk, EOCamera], [Reaper, DaylightTV] \rangle$, it first tries to allocate *GlobalHawk* and *EOCamera* instances. If they cannot be allocated for the task, the agent tries to allocate instances of *Reaper* and *DaylightTV*. If the agent fails to allocate instances, the task cannot be carried out.

In the following section, we propose an approach for task agents to allocate instances of resource types in their deployable configurations.

5.1. Resource Allocation Model

In this section, we propose an approach for continuous resource allocation problem in sensor networks, where new tasks with different sensing requirements can appear at any time [12]. Although those resources can be both sensor and platform instances in our model, we consider only sensors, for the sake of simplicity. We model each sensor as having its own location and sensing range, and being able to provide different utilities to different tasks. The utility is computed by a predefined function for each task and depends on various factors such as sensor type, range, battery life, location and so on.

Each sensor instance on the environment is wrapped by an autonomous computational entity, that is, a software agent, which communicates with others by means of message-passing. A task consists of a number of atomic tasks, which has a specified location, operational range, and sensing demand. An atomic task can only be satisfied if its demand is met (within a threshold) and all of its required sensor types are present in its allocation. If an atomic task is not satisfied, the other tasks requiring that task as a subtask may not be successful.

More formally, a sensor s_i is defined as the tuple $\langle \gamma_i, l_i, r_i, u_i \rangle$ where $\gamma_i \in T$ specifies s_i's type (T is the set of all resource types), l_i and r_i are the location and sensing range of s_i, u_i is the maximum utility s_i can provide in a single time unit. Each atomic task t_j within a composite task is defined as the tuple $\langle L^j_{dc}, l_j, r_j, d_j, b_j \rangle$ where L^j_{dc} is the ordered list of t_j's deployable configurations; l_j and r_j specifies t_j's location and operational range; d_j is the sensing demand that t_j requires; lastly b_j is the overall budget for the task.

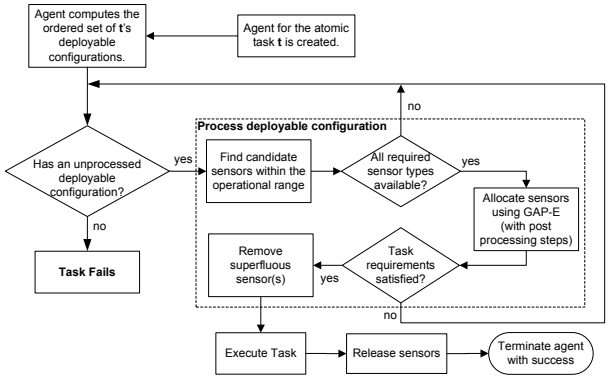

FIGURE 10. The proposed approach as a flowchart.

The task agent A^j_T has an interest in sensors of a specific type if any of the deployable configurations of its task (t_j) contains this sensor type. Let t_j be interested in sensor s_i, then u_{ij} denotes the utility that s_i can provide to t_j, which is defined as a percentage of u_i calculated by the ratio between the overlap of the ranges of s_i and t_j and the range of s_i. If the operational areas of s_i and t_j do not intersect or t_j do not have any interest of s_i, the value of u_{ij} will be 0.

Here we assume that the sensors cooperate with each other and they know both their locations as well as the location of other sensors. Moreover, we also allow a sensor to provide its service to multiple tasks (for example, an audio sensor can provide the same information to all the detecting tasks that require its service). The sensor agents communicate with each other based on the message exchange protocol detailed in [10]. A task can arrive at any time and there may be more than one task active at any given time. Our approach enables sensors to be shared and reassigned between tasks. By having this feature, the tasks which are executed later can secure previously assigned sensors which, otherwise, will be unavailable.

A^j_T interacts with the sensor agents and allocate the required sensors to the task t_j as follows (see Figure 10):

1. For its most desirable deployable configuration dc^j_0, the agent A^j_T identifies the available sensors within t_j's operational range and adds those sensors to its sensor list. This is done by querying each sensor agent with a message containing the task information (sensor types in dc^j_0, location and range of the task) and waiting for an answer from that sensor[3]. Each sensor analyzes the task and if it decides to bid for the task, its bid represents the amount of utility the sensor can provide to the task; the higher the utility the higher the sensor agent's bid

[3]We do not consider failure cases in which, for example, sensor agents do not respond to such requests. There are, however, well known mechanisms for handling such situations such as setting a deadline for the receipt of responses.

for the task. Even though a sensor has already been allocated to one or more tasks, it may also provide utility to the task if the sensor can provide service to the task without decreasing its service quality for the other tasks. This enables more tasks to be successfully allocated without reducing the practicability of the approach or putting more constraints on the sensors. After all the sensors within the operational range have been queried, if the available sensors do not cover the sensor requirements of t_j (i.e. a required sensor type cannot be found), then allocation attempt for dc_0^j fails. In this case, the task agent tries other deployable configurations of t_j. If the task agent fails for all of the deployable configurations, then the required sensors for t_j cannot be allocated and the task fails.

2. If sensors for all required sensor types are available, the task agent uses the bids from the sensors to determine which sensor instances should be allocated given its budget. For this purpose, the task agent converts the sensor allocation problem into the well-known knapsack problem[4]. The solution of knapsack problem is known to be NP-complete. Therefore, finding an exact solution is very costly. In order to come up with a polynomial time approximate solution, we propose a novel multi-round algorithm (GAP-E), which is based a polynomial time approximation to solve the knapsack problem. The details of the GAP-E algorithm are presented in Section 5.2.
3. After all the rounds are completed and all task requirements are satisfied in terms of allocation, there is a final post processing step to release all the superfluous sensors (the one that can be released without violating t_j's requirements – both in terms of utility and sensor type). If there is more than one sensor which can be released, select the one with the smallest utility. This is to ensure that t_j is never allocated more sensors than needed.
4. When the GAP-E finishes, if the final allocation does not satisfy t_j's requirements for the current deployable configuration, the process is repeated for the next deployable configuration. If the allocation fails for all deployable configurations in L_{dc}^j, t_j fails. However, if t_j is allocated necessary resources, its execution has been started. After the task has been processed, the next step is releasing the allocated sensors so that these sensors can be reassigned to other tasks that are active.

When all *initiated subtasks* in a task have been processed successfully, the task is assumed to complete successfully for the related task agent; otherwise the task fails.

5.2. The GAP-E Algorithm

In this section, we detail the proposed GAP-E algorithm that is used by task agent A_T^j to allocate sensors to a particular task t_j for the current deployable configuration dc. In order to realize the idea of stricter governing of selected sensors round-by-round, we introduce new data structures, P_j and C_j. P_j is the priority array that indicates the importance relationship amongst sensor types in the current deployable configuration of t_j. A sensor type is more important to allocate for a task if the task has not allocated sensors of that type. Hence, sensors of the most importance type should be allocated first. Once, sensors of a specific type is allocated, the importance of that type decreased, so P_j is a dynamic data structure and reflects what types of sensors should be allocated next. C_j is the cost array that specifies the cost that will need to be met if t_j requires the service of a certain sensor. In addition, a budget b_j acts as a constraint that governs the number of sensors that can be allocated to t_j.

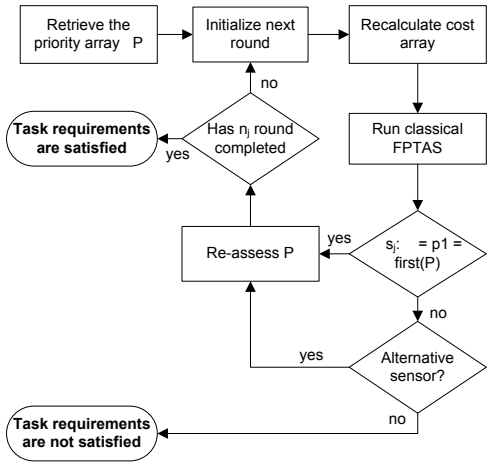

FIGURE 11. GAP-E algorithm as a flow chart.

Figure 11 summarizes the steps that GAP-E takes to find an allocation to t_j for current deployable configuration, the details of which as follows:

1. Let the current deployable configuration have n sensor types, then the priority array is represented as $P_j = \{p_x | x = 1...n\}$, where p_x is the sensor type that has the x^{th} importance with respect to t_j. For example, if t_j has the current deployable configuration $dc = \{a_1, a_2, a_3\}$ and $P_j = \{a_3, a_2, a_1\}$ meaning that a_3 is the most important sensor type and a_1 is the least important one in dc. Nonetheless, all these types must be presented in the final allocation; otherwise the task agent fails to allocate resources for the current deployable configuration.
2. The utility array of the candidate sensors to t_j is updated by the task agent. Based on the value of P_j, GAP-E introduces the concept of the cost array $C_j = \{c_{ij}\}$ for all s_i that has $u_{ij} > 0$. (c_{ij} is the cost for t_j to use s_i's service and computed using the bid received from s_i) and b_j as the overall budget of task t_j. Here, C_j has a similar objective as P_j; it is used to specify the relation between the sensor types and a particular task. The budget, b_j, is used to control the number of sensors allocated to t_j.

[4]Given a set of items, each with a weight and a value, the knapsack problem is to determine the number of each item to include in a collection so that the total weight is less than a given limit and the total value is as large as possible

3. Next, there will be n rounds, with each round $r = 1 \ldots n$ composed of the following steps:
   (a) Reconstruct the matrix C_j so that if sensor type k is more important than type l then all the sensors of type k will have a lower cost than those of type l. That is, if $P_j = \{\ldots, k, \ldots, l, \ldots\}$ then $c_{k'j} < c_{l'j}\ \forall s_{k'}, s_{l'} : \gamma_{k'} = k, \gamma_{l'} = l$. Therefore, the knapsack algorithm has a tendency to allocate sensors of the most important type.
   (b) Run the *Fully Polynomial Time Approximation Scheme (FPTAS)* algorithm [25] with the input of C_j, $U_j = \{u_{ij}\}$ and budget b for the current round. This budget is computed using Equation 1, where A' is the allocation from the last round and U_j is the utilities of the sensors to the task t_j. The obtained solution is then merged with A' to form the temporary allocation A.

$$b = b_j * r/n_j - cost(A') \quad (1)$$

   (c) If A does not contain any sensor of type p_1 (the most important sensor type), we need to replace some sensors in the current allocation A with the sensors of type p_1. Providing there exists some available sensors of type p_1 for t_j, the ones with a minimum cost is selected as the target sensors. If, however, there are some sensors of type p_1 within t_j's range but they are all allocated to other tasks, the task agent A_T^j communicates with the agents representing those tasks to reassign some of those sensors to t_j. In more details, if A_T^j can identify a task t_x that is currently holding sensor s_x of type p_1 and A_T^x can find a replacement sensor s_l without violating t_x's allocation requirements, s_l is assigned to t_x, s_x is released by A_T^x and s_x is selected as the target sensor by A_T^j. If no target sensor can be found then t_j fails to allocate resources for dc.
   If adding this target sensor to A causes the allocation to go over-budget then we need to remove other sensors from A to accommodate this new sensor. This is done by repeatedly removing a sensor, which is (1) a sensor of type k that has more than two members within A or (2) a sensor of type l that has never been the first member of P_j in this or previous rounds. If more than one removable sensors can be identified, the one which provides the lowest utility is removed first. If the budget constraint is still violated but no removable sensor can be identified then t_j fails to allocate resources for dc.
   (d) Reassess P_j so that the most important sensor type p_1 is now the sensor type that does not appear in A. The order of the some sensor types in P_j is shifted right as shown in the next section. If all the required sensor types are presented in A, P_j will revert back to the initial constructed version at the end of n rounds.
4. After n rounds has been completed, the GAP-E algorithm terminates.

In the following section, we present an example to illustrate the algorithm in more detail.

5.3. GAP-E Example

The following example demonstrates how the proposed approach works. Let us assume that $t_1 = \{L_{dc}^1, l_1, r_1, d_1, b_1\}$ where $L_{dc}^1 = \langle[a_1, a_2, a_3, a_4]\rangle$ (the task has only one deployable configuration that contains sensor types a_1, a_2, a_3 and a_4), $d_1 = 1.5$ (the task has a demand of 1.5), and the initial budget for the task is set to $b_1 = 2$. The following candidate sensors exist within t_1's operational range: $S' = \{s_{11}, s_{12}, s_{21}, s_{22}, s_{31}, s_{41}, s_{42}\}$ with s_{ij} denoting sensor type a_i and index j. Here we have $U = [u_{11}, u_{12}, u_{21}, u_{22}, u_{31}, u_{41}, u_{42}] = [0.5, 0.25, 0.4, 0.7, 0.5, 0.15, 0.75]$ where u_{ij} is the utility that s_{ij} can provide for t_1.

Initial priority array is $P_1 = [a_1, a_2, a_3, a_4]$ (i.e., sensor type a_1 is most important ($p_1 = a_1$), followed by type a_2 and so on.). The initial cost array is generated from the bids received from the candidate sensors as $C = [[c_{11}, c_{12}], [c_{21}, c_{22}], [c_{31}], [c_{41}, c_{42}]] = [[0.1, 0.15], [0.03, 0.17], [0.2], [0.05, 0.16]]$. There will be 4 rounds as follows:

- Round 1:
  - C_1 is recalculated[5] based on C as $C_1 = [[c_{11}, c_{12}], [c_{21}, c_{22}], [c_{31}], [c_{41}, c_{42}]] = [[0.1, 0.15], [0.28, 0.42], [0.7], [0.8, 0.91]]$.
  - The current budget is computed as $b = 0.5$ using Equation 1 (the total budget is initially split equally between the four sensor types required for this task). For those parameters, FPTAS returns $A = \{s_{11}, s_{12}\}$.
  - Now we have all sensors of type a_1, the priorities among sensor types are now revised such that $P_1 = [a_2, a_1, a_3, a_4]$. The allocation is within budget, $cost(A) = 0.25$ and $U(A) = 0.75$. So, we do not consider sensors to be removed from the allocation.
- Round 2:
  - C_1 is recalculated as $C_1 = [[c_{21}, c_{22}], [c_{31}], [c_{41}, c_{42}]] = [[0.03, 0.17], [0.7], [0.8, 0.91]]$.
  - $b = 1.0 - 0.25 = 0.75$, FPTAS returns $A = \{s_{21}, s_{22}\}$, combined with A' (the allocation from the previous round), we have $A = \{s_{11}, s_{12}, s_{21}, s_{22}\}$.
  - Now we have sensors of type a_1 and a_2, thus $P_1 = [a_3, a_1, a_2, a_4]$. The removal of sensors from the allocation need not be considered. $cost(A) = 0.45$, $U(A) = 1.85$.

[5] Since we have 4 sensor types and $P_1 = [1, 2, 3, 4]$, c_{11} and c_{12} remains the same, c_{21} and c_{22} are increased by $\frac{1}{4}$, c_{31} is increased by $\frac{1}{2}$, c_{41} and c_{42} are increased by $\frac{3}{4}$.

- Round 3:
  - C_1 is recalculated as $C_1 = [[c_{31}], [c_{41}, c_{42}]] = [[0.2], [0.8, 0.91]]$.
  - $b = 1.5 - 0.45 = 1.05$, FPTAS returns $A = \{s_{42}\}$, combined with A' we have $A = \{s_{11}, s_{12}, s_{21}, s_{22}, s_{42}\}$.
  - Now we have sensors of type a_1, a_2 and a_4 but since $p_1 = 3$ and there is no sensor of type a_3 in A, we need to add a sensor of type a_3 to A. Since there is only one sensor of such type, s_{31} needs to be added. However, adding s_{31} causes the budget to be exceeded and we, therefore, need to remove sensors from the current allocation. Any of the sensors in A can be removed and thus, s_{12} is removed first since it has the lowest utility (u_{12}). By doing so, the cost of the solution falls below the budget and we have a valid allocation $A = \{s_{11}, s_{21}, s_{22}, s_{31}, s_{42}\}$ with $cost(A) = 1.41$ and $U(A) = 2.85$. Additionally, we have all the required sensor types, $P_1 = [a_1, a_2, a_3, a_4]$.
- Round 4:
  - C_1 is recalculated as $C_1 = [[c_{12}], [c_{41}]] = [[0.15], [0.8]]$.
  - $b = 2.0 - 1.41 = 0.59$, FPTAS returns $A = \{s_{12}\}$, combined with A' we have $A = \{s_{11}, s_{12}, s_{21}, s_{22}, s_{31}, s_{42}\}$.
  - As we have all the required sensor types, $P_1 = [a_1, a_2, a_3, a_4]$ $cost(A) = 1.56, U(A) = 3.1$.

Thus, GAP-E returns a valid allocation of $A = \{s_{11}, s_{12}, s_{21}, s_{22}, s_{31}, s_{42}\}$. However, since $U(A) = 3.1$ is greater than $d_1 = 1.5$, we will need to release superfluous sensors. Of the sensors in A, we can release any sensor in the set $\{s_{11}|s_{12}, s_{21}|s_{22}\}$. Thus, s_{12} will be released first because u_{211} is smallest of all (0.25). This brings the $U(A)$ down to 1.6. After removing s_{12}, $U(A) = 2.85$ and still greater than d_1. This time we can only release either s_{21} or s_{22} otherwise the sensor type requirements of t_1 will be violated. Thus, s_{21} will be released since $u_{21} < u_{22}$. As a result, $A = \{s_{11}, s_{22}, s_{31}, s_{42}\}$ is the final allocation to t_1.

Having defined our model, next section will details the results of our experiments.

6. EVALUATION

We have evaluated the proposed approaches in two steps. First, we present a case study to highlight the importance of automated determination of deployable configurations for tasks. Second, we empirically evaluate our sensor allocation approach in terms of time complexity and successful allocation of resources.

6.1. Determining Resource Types

Determination of deployable configurations for a task requires extensive domain knowledge and expertise. Currently, those deployable configurations are determined by human experts during the design of tasks. Considering that a task may contain 100s or 1000s of atomic tasks, the process of manually determining deployable configurations is both not feasible and highly error-prone. Moreover, during the execution of tasks, the environment (e.g., climate, terrain and so on) may change. This may lead to addition of new requirements to the task and invalidate some of the presumed constraints. As a result, design time decisions about the required sensor types (deployable configurations) may get invalidated. Unlike the manual determination of deployable configurations, the proposed approach in Section 4 enables computation of new deployable configurations promptly in response to the changes in the definition of a task (e.g., constraints and requirements of the atomic tasks in the task). Automating computation of deployable configurations enables autonomous execution of tasks on sensor networks with minimal human intervention. The following case-study highlights the importance of automatically determining deployable configurations for the atomic tasks.

We consider a hypothetical country in which there is an ongoing civil war between ethnical groups. An earthquake has recently hit and destroyed the Eastern-most part of the country where thousands of civilians lost their homes and some of them are dead or seriously injured. This region is surrounded by mountains, where caves in the terrain are used by armed groups as shelters. Humanitarian volunteer organizations such as the Red Cross are willing to help the civilians. However, possible attacks by the armed groups in the region prevent these organizations from operating in the region. In this setting, our task is to enable these organizations to provide relief to those affected by the earthquake in the region without facing any security issue.

This task is composed of many atomic tasks. Let us assume that one of the critical atomic tasks in this setting is a road surveillance task that aims to detect and prevent hostile activities on the road of the humanitarian organizations to the camp. The surveillance task requires high quality imagery sensors to monitor activities. Those sensors should be attached to UAVs that have a long endurance and high altitude capabilities. The task is based on the matchmaking mechanism shown in Figure 9, because sensors and platforms are not independent. The relationships between sensors and platforms are defined in ISTAR ontology and used by the matchmaker mechanism in Figure 9 to select the best combinations of resource types for the task. Depending on the ontological domain knowledge, the matchmaker found two deployable configurations: *[Global Hawk, IRCamera]* and *[Global Hawk, EOCamera]*. The second deployable configuration is selected for the task and five Global Hawks with Electro Optical Cameras are allocated, configured and deployed on the region.

In the middle of the surveillance task's execution, a part of the road between mountains has been covered by a fog. Two UAVs that are responsible of monitoring this part become immediately useless, because the attached optical sensors cannot be used to sense activities under fog. In order to save the whole task, new UAVs with the necessary equipment should be allocated immediately. Hence, the task agent computes new deployable configurations using the

new constraints. The task agent finds only one deployable configuration, *[Global Hawk, SAR]*, where *SAR* is a radar sensor that can be used to detect activities under fog. The task agent tries to allocate Global Hawks with SAR sensors to resume the task on the foggy area.

Without an approach like the one we propose, the human experts should always be "in the loop" during the design and execution of tasks to determine necessary deployable configurations. Depending heavily on human experts for determining deployable configurations may result in delays, which may lead to severe failures. Similar to our approach, Gomez *et al.* propose an approach for automatically determining deployable configurations for tasks in mobile sensor networks [8]. Their work depends on a minimal set covering algorithm. In order to determine deployable configurations for an atomic task, this algorithm enumerates all possible sets of resource types so that each set has at most n members. Then, a set is considered as a deployable configuration of the task if it is composed of a platform type P and sensor types S so that P can mount all of the sensor types in S. This approach is based on exhaustive search with limit defined by n.

Given a specific task, output of the OLP program in Figure 9 and that of the approach proposed by Gomez *et al.* should be the same because they are using the same definition of deployable configurations. However, the proposed algorithm in Figure 9 is based on the idea that the search space can be significantly reduced using domain knowledge (i.e, dependencies between sensors and platforms; not every types of sensors can be used with a specific type platform). Using this principle, at each iteration, it rules out many possibilities and significantly reduces the time required to compute deployable configurations.

To confirm our assertions, we have implemented the exhaustive search algorithm in Java and empirically compared it and our OLP-based algorithm in Figure 9 in terms of time consumption. Our results are demonstrated in Figure 12, where the x-axis is the maximum number of items in deployable configurations and y-axis is the average time consumed by each approach to find all of the deployable configurations of an atomic task. When the maximum size of deployable configurations is lower than four, exhaustive search algorithm is faster then our approach. This performance difference is originated from the overhead of using OLP stack to interpret the logic program in Figure 9. However, when the maximum size of deployable configurations is greater than three, the proposed approach outperforms the exhaustive search algorithm significantly; time consumption of the exhaustive search increases exponentially while that of the proposed approach looks mostly linear.

6.2. Resource Allocation

This section evaluates our approach in a range of different environments and assesses its performance in terms of the number of successfully completed composite tasks and the amount of processing time required. There are a number of internal variables which control the behavior of our model as well as external variables which define the environment in which our model is being used. The system developed in Java allows us to manipulate these variables, conduct experiments and analyze the results.

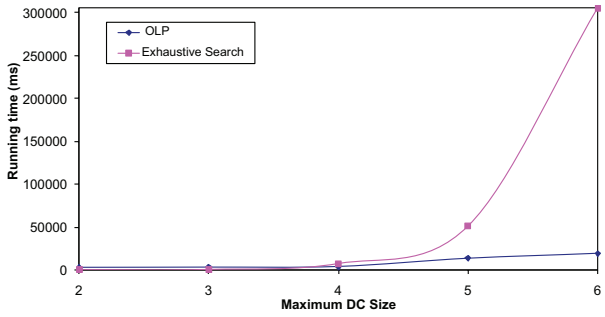

FIGURE 12. Comparison of OLP program in Figure 9 and exhaustive search algorithm to find deployable configurations.

6.2.1. Simulation Setup

Our approach is evaluated using randomly generated problems. A set of data is generated for each run. We compare the performance of the proposed model (named *Multiple-Sensor-Mode* or *MSM* now on) with the following alternatives:

1. **Exclusive sensor mode (ESM):** This is our implementation for the approach presented in [16] (see Section 7). Each sensor can only be assigned to one task at a time. For each task, we test all combinations of sensors within the available candidates with the restriction that each sensor type only has one member[6]. The combination providing the highest utility value will be checked against the demand of the task, and, if acceptable, it will be selected as the final allocation for that task.
2. **Shared sensor mode (SSM):** This approach operates in a similar way to ESM. The difference being that it allows sensors to be shared between two tasks as our approach does. In this approach, the combination of sensors providing the highest utility is selected as the final allocation if it satisfies the demand constraint of the task in question. In both ESM and SSM, an allocation may not satisfy the demand of a task, because only one sensor of each required type is allocated. However, with such a simplification, those approaches work much faster than they would. On the other hand, their performance in terms of achieved tasks is expected to be much lower, because the demand constraints of the tasks may not be satisfied. Those approaches is included in our benchmark comparisons mainly to evaluate the *time consumption* of the proposed

[6]The assignment problem is NP-complete and it is time and memory consuming to go through all possible combinations of sensors. Thus one single sensor per type is chosen to reduce the running time of this case.

approach, instead of evaluating its performance in terms of successful resource allocation.

3. **Shared sensor mode without demand checking (SSM-NC)**: This approach operates in a similar fashion to SSM. However, there is no evaluation against the task's demand while testing success of the allocation. Instead, the combination providing the highest utility will be selected as the final allocation for that task without considering the demand of the task. Therefore, if SSM-NC can allocate at least one sensor for each required sensor type of a task, the task is assumed to have a successful allocation without considering its demand constraints. This means that in terms of success in sensor allocation, SSM-NC defines an upper bound for the case that we consider demand constraints. This will give us an idea of what the optimal success rate of the tasks might be[7]. In our benchmark comparison, we use SSM-NC to evaluate how well our approach in terms of *successful allocation* of sensors to tasks.

A composite task is composed of a set of subtasks, which may be divided into other subtasks. The task can only be satisfied if all of its initiated subtasks can be allocated required resources successfully. In our simulations, the task arrival rates are controlled by the *rate_per_hour* parameter, which ranges from 2 to 8, and the life of the simulations is kept at 2 days. Each task can last for an arbitrary amount of time, ranging from 5 minutes to 4 hours.

The parameter *total_sensor_types* has a range $[4-8]$ and represents the number of different sensor types on the simulated area. The parameter *total_sensors_per_types* refers the maximum number of sensors distributed on the area for each sensor type. For each task, the number of subtasks are varied between 3 and 8. Each atomic task has only one deployable configuration that contains at most 4 sensor types, which is defined by the parameter *total_sensor_types*.

The simulated area has the size of $400m$ x $400m$. This is where the sensors and tasks are deployed in uniformly random locations. Each sensor range (r_i) is randomized between $20m$ and $40m$ and their maximum utility is calculated as $(r_i/40)^2$, which ensure the values between 0.25 and 1. The operational range of the tasks are set to be randomized between $40m$ and $80m$. We now turn to the specific results.

6.2.2. *Results*

In this section, we state three hypothesis about the performance of the proposed sensor allocation approach and provide evidences from the results of our experiments to support those hypothesis.

HYPOTHESIS 1. The proposed approach performs well in comparison to the estimated optimum.

[7]This cannot be considered to be the optimal result since it does not involve checking all the combinations of all the potential allocations for each individual task and then measuring their requirements. There are cases where optimal allocation involves non-local maxima allocation. Given the set of experiments need to be carried out, however, it is impractical to do a complete exhaustive search to find the optimal solution.

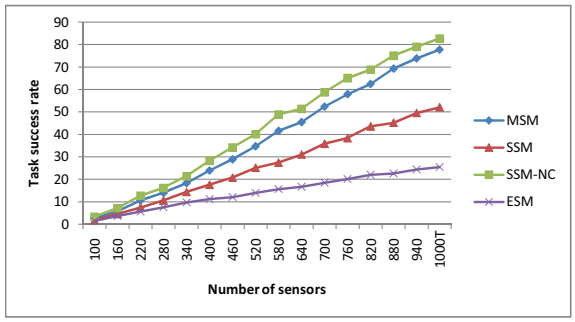

FIGURE 13. Task success rate with 4 sensor types and 4 composite tasks arriving per hour.

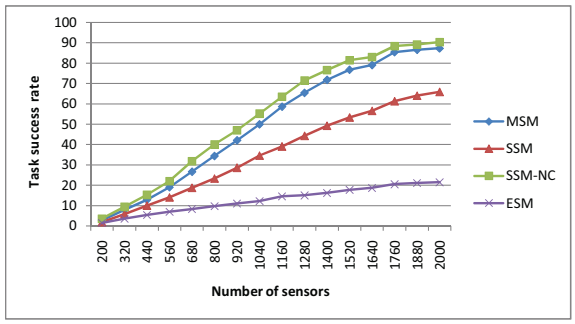

FIGURE 14. Task success rate with 8 sensor types and 8 composite tasks arriving per hour.

JUSTIFICATION. Figure 13 shows the task success rate of the four mechanisms with *total_sensor_types* = 4, *rate_per_hour* = 4 and *total_sensors_per_types* between 25 and 250. Figure 14 shows the task success rate with *total_sensor_types* = 8, *rate_per_hour* = 8 and *total_sensors_per_types* also between 25 and 250. As the overall objective is to increase the number of successful tasks, this control variable strongly indicate the performance of each mechanism.

As seen in both Figures 13 and 14, SSM provides better results than ESM. This is because in SSM, multiple tasks can share a single sensor and, thus, a task in SSM has a greater chance of being successfully allocated. However, in both SSM and ESM, only one sensor per sensor type can be presented in the allocation and in many cases this is not sufficient to satisfy the sensing demand of a task. This explains why MSM has a significantly better task success rate than SSM and ESM. With SSM-NC, since the demand requirement is not checked, the number of successful tasks is the largest. However, this does not reflect the optimum allocation since there are situations in which the sensor type requirements of a task can be met but the sensing demand cannot be achieved. Nonetheless, the assignment problem being identified in this paper is NP-hard and it is impractical to do an exhaustive search of all combinations to find the optimal solution. Thus, the SSM-NC is introduced here to give an idea of where the optimal solution might lie.

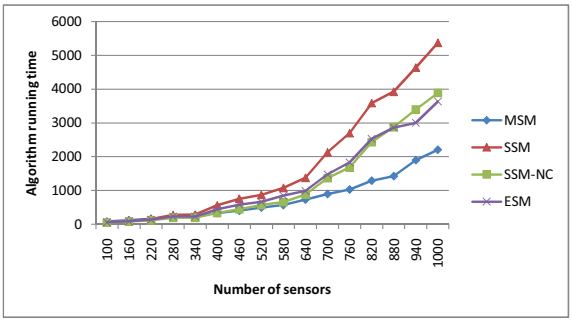

FIGURE 15. Running time (ms) with 4 sensor types and 4 composite tasks arriving per hour.

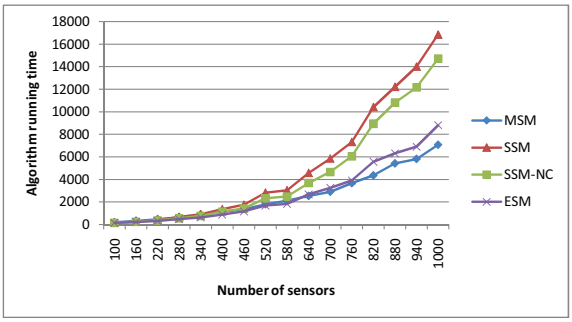

FIGURE 16. Running time (ms) with 8 sensor types and 8 composite tasks arriving per hour.

HYPOTHESIS 2. The computational complexity (running time) of the proposed approach is much less than that of the other mechanisms.

JUSTIFICATION. Similar to hypothesis 1, Figure 15 shows the running time of the four mechanisms with *total_sensor_types* = 4, *rate_per_hour* = 4 and *total_sensors_per_types* between 25 and 250. Figure 16 shows the running time with *total_sensor_types* = 8, *rate_per_hour* = 8 and *total_sensors_per_types* also between 25 and 250. Of course, the running time of each mechanism depends largely on the machine that the experiments are run on. However, putting them all together can give a picture of their overall complexity.

As shown in the figures, MSM's running time is always the smallest. Both SSM and SSM-NC has to exhaustively search through all the potential combinations of sensors to find the best allocation, their complexity will increase exponentially when the number of sensors increased. Thus, their running time are always highest and in many cases can be 2.5 times larger than that of our MSM. On the other hand, MSM employs the FPTAS algorithm (see Section 5.2) which has the polynomial complexity and therefore, its running time only increases steadily when the number of sensors increases.

HYPOTHESIS 3. The computational complexity of the proposed approach is increased in a steadily fashion with the number of tasks.

JUSTIFICATION. To evaluate this hypothesis, we measure

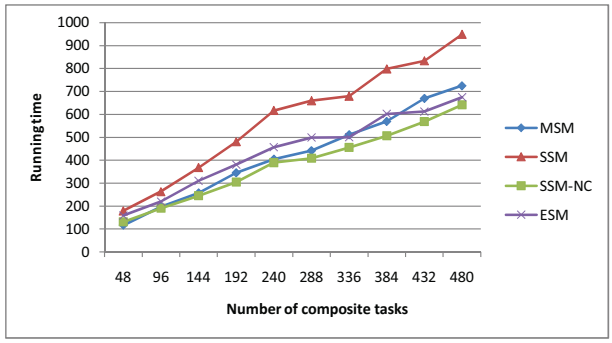

FIGURE 17. Running time (ms) with 4 sensor types and 25 sensors per type.

the running time of the mechanisms with *total_sensor_types* = 4, *total_sensors_per_types* = 25 and varies *rate_per_hour* between 1 and 10. The result is displayed in Figure 17. It is clearly that as the number of composite tasks increases from 48 to 480, the running time of MSM also increases steadily from 100 to just over 700ms. As our GAP-E algorithm has a polynomial complexity depending on the number of sensors and tasks, once we keep the former variable unchanged, therefore, MSM running time increases inline with the number of tasks.

7. DISCUSSION AND RELATED WORK

The first approach to matching intelligence, surveillance, and reconnaissance assets to tasks using an ontological framework was [8]. This work showed that the relationships between tasks and assets could be represented by means of *capabilities*: those required by tasks, and those provided by assets. This approach was well-founded in the sense it was a formalization of a well-known military planning framework: the Missions and Means Framework (MMF) [22]. The work reported in this paper extends this earlier work: the ontology shown on the left of Figure 3 is an updated version of the MMF ontology presented in [8].

Various ontologies have been proposed in the literature to describe sensors, platforms and so on. SensorML [17], OntoSensor [21] and Marine Metadata Interoperability project [4] are well-known examples. Ontologies for ISTAR tasks is a less well-developed area. The best-known taxonomy of tasks in a military context is the US Universal Joint Task List (UJTL) [11], which was formalized as a DAML ontology[8]. The earlier work in [8] reused elements of these sensor and task ontologies, but the representation of tasks was simplistic, being restricted to a simple task/subtask hierarchy (following the principles of the MMF). The representation of tasks introduced in this paper is much richer: tasks can be considered as workflows, whose components are described semantically using an ontology. In the literature, there are approaches that describe workflows semantically using an ontology [26]. However, these approaches do not consider multi-agent systems to

[8] http://orlando.drc.com/semanticweb/daml/ontology/condition/ujtl/condition-ont

allocate resources to carry out workflows as we do.

Sensor-task assignment problems in wireless sensor networks have been studied mainly using simplified models in which only a single sensor type is introduced and the exclusive use of resources is required. For example, in [7, 14] the authors propose distributed approaches to solve the assignment problem assuming that the same type of sensors are deployed in the battlefield and that there is no competition for the sensing resources between tasks. On the other hand, the decentralized approaches considered in [20, 3, 10] introduce competition between sensing tasks of same type. However, this approach prevents more than one sensor being assigned to any one task.

Our proposed E-GAP algorithm presented in Section 5.1 is an adaptation of the MRGAP algorithm proposed in [10]. The MRGAP algorithm aims to solve the static assignment problem, which is a common generalization of the problems presented in [6, 3, 19], incorporating both budgets and profit thresholds. The idea of that algorithm is to consider tasks as knapsacks that together form an instance of the Generalized Assignment Problem (GAP). The author of [6] give an approximation algorithm for GAP which takes a knapsack algorithm as a parameter. There, the standard knapsack algorithm FPTAS [25] offers an approximation guarantee of $2 + \epsilon$. MRGAP, however, still lacks the ability to consider multiple sensor types. Moreover, it does not take into account the trade-off between communication cost and utilities gained. Nonetheless, the battery life-time of individual sensor is typically limited by the power required to transmit their data. As a result, power conservation issues in wireless sensor networks are essential and attract the interest of many researchers. In [18], for example, the authors selected a sensor acting as a mediator to relay data for other sensors. It saves the battery life of other sensors to the detriment of its own battery. To choose such mediator, the authors proposed a payment scheme in which the power p to reliably transmit over a distance d (p is proportional to the square of d) is considered as the decision value.

A number of protocols to locally make decisions for a sensor in a wireless sensor network have begun to tackle the challenge of coordinating between the network's interconnected nodes given the absence of a central coordinator. Protocols in [18, 9] allow sensors to request other sensors to forward data, which provides sensors with a broader range of information, often leading to an improvement in the network's ability to meet its global objectives. However, it requires extra communication that imposes an energy cost on the network. Therefore, the authors of [5] observed that the proper level of coordination (the degree of hops a sensor broadcast message to) leads to a significant increase in the performance of the network.

Protocols have also been developed in [20, 16] where each task leader runs a local protocol to match sensors within two hops to the requirement of the task. These works are closely related to ours, considering the many-sensor type, many-task type assignment problem. They also use the results of matching sensor types to tasks to reduce the search space in order to find the allocation. However, they need to generate and check all the instances of feasible solutions of the matching sensor type problem, which is very inefficient in a dense sensor network. Moreover, unlike our work, they do not allow assets to be shared between tasks.

In our resource allocation model, we do not take into account the fact that some of the sensors can be mounted on mobile platforms. This is typical in many situations; for example, an image sensor can be mounted on an unmanned aircraft. Thus, such sensor will have a greater operational area compared to a static one. We plan to address this problem by incorporating new utility prediction model that will include the cost of relocating sensors within the battlefield.

8. CONCLUSIONS

In this paper, we integrate Semantic Web technologies and multi-agent systems in a novel way to determine and allocate resources to tasks in wireless sensor networks. First, we propose to use OWL-S process ontology with domain ontologies to describe tasks semantically. Second, we have combined ontological reasoning with logic programming to enable software agents to reason about the required sensor types for the tasks. Third, we have proposed a decentralized agent-based approach to solve the sensor-task assignment problem. In this approach, task agents interact with sensor agents to allocate the most useful sensors for the tasks. For this purpose, task agents convert allocation problem into a knapsack problem and utilizes the multi-round knapsack algorithm to provide concrete solution. Our approach has been evaluated in a number of different scenarios, and we have demonstrated empirically that good results can be achieved in a considerably less time compared to the traditional exhaustive search counterpart.

ACKNOWLEDGEMENT

This research was sponsored by the U.S. Army Research Laboratory and the U.K. Ministry of Defence and was accomplished under Agreement Number W911NF-06-3-0001. The views and conclusions contained in this document are those of the author(s) and should not be interpreted as representing the official policies, either expressed or implied, of the U.S. Army Research Laboratory, the U.S. Government, the U.K. Ministry of Defence or the U.K. Government. The U.S. and U.K. Governments are authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation hereon.

REFERENCES

[1] SWRL: A Semantic Web Rule Language Combining OWL and RuleML, 2004. http://www.w3.org/Submission/SWRL.

[2] F. Baader, D. Calvanese, D. L. McGuinness, D. Nardi, and P. F. Patel-Schneider, editors. *The Description Logic Handbook: Theory, Implementation and Applications*. Cambridge University Press, 2003.

[3] A. Bar-Noy, T. Brown, M. P. Johnson, T. F. L. Porta, O. Liu, and H. Rowaihy. Assigning sensors to missions

with demands. In M. Kutylowski, J. Cichon, and P. Kubiak, editors, *Lecture Notes in Computer Science*, volume 4837, pages 114–125. Springer, 2007.

[4] L. Bermudez, J. Graybeal, and R. Arko. A marine platforms ontology: Experiences and lessons. In *Proceedings of the ISWC 2006 Workshop on Semantic Sensor Networks*, 2006.

[5] E. Bulut, J. Zheng, Z. Wang, and B. K. Szymansk. Analysis of cost-quality tradeoff in cooperative ad hoc sensor networks. In *ACITA 2008*, 2008.

[6] R. Cohen, L. Katzir, and D. Raz. An efficient approximation for the generalized assignment problem. *Information Processing Letters*, 100(4):162–166, 2006.

[7] C. Frank. Algorithms for generic role assignment in wireless sensor networks. In *Proceedings of the 3rd International Conference on Embedded Networked Sensor Systems*, pages 230–242, 2005.

[8] M. Gomez, A. Preece, M. P. Johnson, G. Mel, W. Vasconcelos, C. Gibson, A. Bar-Noy, K. Borowiecki, T. Porta, D. Pizzocaro, H. Rowaihy, G. Pearson, and T. Pham. An ontology-centric approach to sensor-mission assignment. In *Proceedings of the 16th international conference on Knowledge Engineering (EKAW'08)*, pages 347–363, 2008.

[9] D. B. Johnson. Scalable and robust internetwork routing for mobile hosts. In *Proceedings of the 14th International Conference on Distributed Computing Systems*, pages 2–11, 1994.

[10] M. P. Johnson, H. Rowaihy, D. Pizzocaro, A. Bar-Noy, S. Chalmers, T. F. L. Porta, and A. Preece. Frugal sensor assignment. In S. E. Nikoletseas, B. S. Chlebus, D. B. Johnson, and B. Krishnamachari, editors, *Lecture Notes in Computer Science*, volume 5067, pages 219–236. Springer, 2008.

[11] Joint Staff. Universal joint task list (UJTL), 2002. http://www.dtic.mil/doctrine/jel/cjcsd/cjcsm/m350004c.pdf.

[12] T. P. Le, T. J. Norman, and W. W. Vasconcelos. Agent-based sensor-mission assignment for tasks sharing assets. In *Proceedings of the Third International Workshop on Agent Technology for Sensor Networks*, Budapest, Hungary, May 2009.

[13] D. Martin, M. Burstein, J. Hobbs, O. Lassila, D. McDermott, S. McIlraith, S. Narayanan, M. Paolucci, B. Parsia, T. Payne, E. Sirin, N. Srinivasan, and K. Sycara. OWL-S: Semantic markup for web services, November 2004. http://www.w3.org/Submission/OWL-S.

[14] H. Park and M. B. Srivastava. Energy-efficient task assignment framework for wireless sensor networks. Technical Report 26, 2003.

[15] G. Piancastelli, A. Benini, A. Omicini, and A. Ricci. The architecture and design of a malleable object-oriented Prolog engine. In R. L. Wainwright, H. M. Haddad, R. Menezes, and M. Viroli, editors, *23rd ACM Symposium on Applied Computing (SAC 2008)*, pages 191–197, 2008.

[16] A. Preece, D. Pizzocaro, K. Borowiecki, G. de Mel, M. Gomez, M. Vasconcelos, A. Bar-Noy, M. P. Johnson, T. L. La Porta, H. Rowaihy, G. Pearson, and T. Pham. Proceedings of the reasoning and resource allocation for sensor-mission assignment in a coalition context. In *MILCOM 2008*, 2008.

[17] A. Robin, S. Havens, S. Cox, J. Ricker, R. Lake, and H. Niedzwiadek. OpenGIS© sensor model language (SensorML) implementation specification. Technical report, Open Geospatial Consortium Inc, 2006.

[18] A. Rogers, E. David, and N. R. Jennings. Self-organized routing for wireless microsensor networks. *IEEE Transactions on Systems, Man and Cybernetics, Part A*, 35(3):349–359, 2005.

[19] H. Rowaihy, M. Johnson, T. Brown, A. Bar-Noy, and T. L. Porta. Assigning sensors to competing missions. In *Technical Report NASTR-0080-2007*, Network and Security Research Center, Department of Computer Science and Engineering, Pennsylvania State University, University Park, PA, USA, 2007.

[20] H. Rowaihy, M. Johnson, D. Pizzocaro, A. Bar-Noy, T. L. Porta, and A. Preece. Sensor-task assignment protocols. In *International Technology Alliance, Tech. Rep. (submitted for publication)*, 2008.

[21] D. J. Russomanno, C. R. Kothari, and O. A. Thomas. Building a sensor ontology: A practical approach leveraging iso and ogc models. In *Proceedings of the 2005 International Conference on Artificial Intelligence (ICAI)*, pages 637–643, 2005.

[22] J. H. Sheehan, P. H. Deitz, B. E. Bray, B. A. Harris, and A. B. H. Wong. The military missions and means framework. In *Proceedings of the Interservice/Industry Training and Simulation and Education Conference*, pages 655–663, 2003.

[23] E. Sirin, B. Parsia, B. C. Grau, A. Kalyanpur, and Y. Katz. Pellet: A practical OWL-DL reasoner. *Web Semant.*, 5(2):51–53, 2007.

[24] M. K. Smith, C. Welty, and D. L. McGuinness. OWL: Web ontology language guide, February 2004. http://www.w3.org/TR/owl-guide.

[25] V. V. Vazirani. *Approximation Algorithms*. Springer, 2001.

[26] Y. Wang, J. Cao, and M. Li. Goal-driven semantic description and query for grid workflow. In *SKG '07: Proceedings of the Third International Conference on Semantics, Knowledge and Grid*, pages 598–599, Washington, DC, USA, 2007. IEEE Computer Society.