A Distributed Architecture For Heterogeneous Multi Sensor-Task Allocation

Diego Pizzocaro, Alun Preece
School of Computer Science and Informatics
Cardiff University, UK
Email: D.Pizzocaro@cs.cardiff.ac.uk

Fangfei Chen, Tom La Porta
Dept. of Computer Science and Engineering
Penn State University, US

Amotz Bar-Noy
Dept. of Computer Science
Graduate Center
City University of New York, US

***Abstract*—Heterogeneous sensor networks are increasingly deployed to support users in the field requiring many different kinds of sensing tasks. There may be multiple alternative kinds of sensors suitable for a given task. Sensing tasks might compete for the exclusive usage of available sensors. Such an environment is highly dynamic with users moving and generating tasks at different rates. Users typically lack the time and expertise to manually decide which are the best sensors for their tasks. We need therefore to design a distributed system to automatically allocate sensors to tasks. We formalize this problem as Multi-Sensor Task Allocation (MSTA) and show that the heterogeneity of sensors and tasks requires knowledge-based sensor-task matching. We extend a pre-existent well known coalition formation protocol to propose a novel layered distributed system which by using qualitative and quantitative measures provides allocation flexibility. We demonstrate that it is feasible to perform the knowledge-based sensor-task matching on a user's device by presenting a proof-of-concept mobile app which allows a user in the field to interact with the system. We run simulations to demonstrate that our architecture is scalable and that the allocation quality improves by allowing preemption of sensing resources from ongoing tasks and a reallocation mechanism.**

I. Introduction

Heterogeneous sensors are increasingly used to support emergency responders in the field usually requiring many different sensing tasks, like detecting people who may need help and monitoring a collapsing building[1]. Upon deployment in the field heterogeneous sensing devices will form an ad hoc network using wireless links or cables to communicate. This heterogenous sensor network is required to support multiple sensing tasks of different types simultaneously. Sensing tasks might share the usage of a sensing resource, but more often they compete to exclusively control it - for example in the case of directional sensors. A mobile user moving on the field needing many different sensing tasks would not have the time to manually decide what is the best set of sensors to use for each task. In addition, a user might not have the expertise to decide what type of sensors could best match each task's sensing requirements. Most importantly, if they were to manually choose the sensors, they would probably consider only a subset of the sensor and task parameters. We need therefore to design a distributed system to automatically allocate sensors to the tasks they best serve, considering the task information requirements and the sensor capabilities. We refer to this problem as *Multi-Sensor Task Allocation* (MSTA).

Such a problem can appear not only in emergency response scenarios but it can be easily identified in many humanitarian relief operations conducted by international coalitions. In our problem settings tasks have different priorities and are generated over time with different rates. Each task requires one set out of any possible set of sensors to be accomplished, we call these sets of sensors "*sensor bundles*". To satisfy each task's requirements we want to allocate exactly one *sensor bundle*. These bundles need to be dynamically generated by the system considering the relevant sensors and computing their joint utilities for each of the tasks.

Our main contribution is an extension of a pre-existent well known protocol [11] to implement a distributed system solving our MSTA problem instance. The novelty of our architecture consists in a layered approach which by integrating a knowledge base with the allocation protocol provides flexibility in the choice of sensors to use in order to satisfy the users' task requirements. Therefore we solve the allocation problem taking into account both qualitative and quantitative measures.

The remainder of the paper is organized as follows. In Section II we analyze which features we need to add to the allocation protocol chosen as a skeleton for our architecture. In Section III we provide an overview of our layered architecture and we formally analyze our MSTA problem. In Section IV we describe the *Knowledge-based bundle generator* component of our system which supports the *allocation protocol* described in Section V. In Section VI we show the performance of our distributed system implemented in a discrete time event simulator. We also include the benchmarks of a prototype mobile app developed for iOS devices. Finally, Section VII concludes the paper outlining future work.

II. Background

There are many possible instances of the MSTA problem, depending on a number of key parameters. We use the taxonomy of MSTA problems introduced in [7] (as an extension of [3]) to identify the problem instance which is relevant

[1] "Global Hawk collects reconnaissance data during Haiti relief efforts", http://www.af.mil/news/story.asp?id=123185754 — "Small, Unmanned Aircraft Search for Survivors in Katrina Wreckage", http://www.nsf.gov/news/news_summ.jsp?cntn_id=104453

to our scenario. Given that military and emergency response operations use heterogeneous sensing devices to coordinate missions, we focus on heterogeneous sensor networks - identified by the label *HE* using the MSTA taxonomy; also given that the dynamic environment does not provide enough information to plan for future allocation we aim at Instantaneous Allocation (*IA*). In the most general case each task usually requires a group of sensors therefore we consider Multi-Sensor tasks (*MS*) which often involves non-additive joint utility functions to evaluate the performances of sensor bundles for a particular task (i.e. functions in which the utility coming from each sensor does not sum-up linearly). Finally we consider a particular subset of sensors which can serve exclusively one task at a time, like directional sensors - i.e. Single Task (*ST*) sensors . For these reasons, our problem instance is identified by *ST-MS-IA-HE* following the terminology of the MSTA taxonomy in [7]. A version of the *ST-MS-IA-HE* is referred to as *disjoint coalition formation* problem in the multi-agent community as stated in [3] and it has been formally studied in [10] and [11]. A well known efficient allocation protocol has been proposed in [11]. This protocol was designed for generic multi-agent systems, thus we need to adapt it to our problem settings. In particular we extended the protocol for the *disjoint coalition formation* problem adding four main features which we found were necessary.

First, the original protocol assumed the presence of a central "matchmaker" agent to which the other agents would have asked if their capabilities were a fit for the task's requirements — we instead take advantage of the user's mobile devices which can host such a "matchmaker" which we call *Knowledge-based bundle generator*. We also completely move the calculation of the joint utility values of sensor bundles to the user devices, whereas in [11] it was carried out distributedly on the sensors with a preliminary protocol stage aimed at the calculation of the "coalitional value".

The second feature which we introduce is the ability for a task to request a different bundle of sensors in case the resources requested the first time were not available - we call this *rebid mechanism*. There are in fact multiple ways of satisfying a task, as stated in [12] which considers alternative coalitions for satisfying a rescue task if the first attempt to get those resources fails. In general this extension is inspired by the concept of *substitute goods* widely used in economics [1].

The third extension consists in making more explicit the *preemption mechanism* which is mentioned in [11] when the authors describe its implementation in the RETSINA dynamic agent system. We allow preemption of sensors which are already serving a task, whereas the original protocol allows only preemption of sensors which have already committed to a task but have not already started to serve it. Our approach is similar to the multi-robot system architecture proposed in [2] which assigns tasks to robots with first-price auctions, but allow (in some circumstances) later reassignment.

Finally we need to deal with tasks characterized by an expected duration and an *expiration time* or deadline before which we need to decide if we can serve them. In fact providing support to the task too late might be useless for a user. We therefore extend the protocol by including deadlines, although keeping our focus on maximizing the overall user/task "happiness", without trying to reduce the tasks' waiting time. We achieve this by using the *rebid mechanism* mentioned above, while there is still time to satisfy the task.

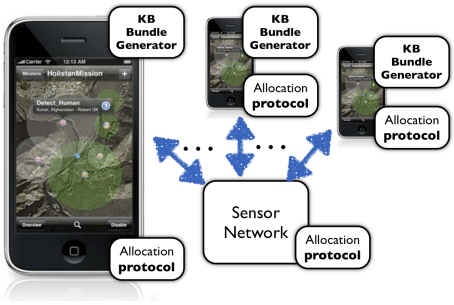

Fig. 1. Fully distributed system architecture.

III. Architecture overview & model

In this section we give an overview of our distributed architecture highlighting the steps which are executed to find an efficient solution to the MSTA problem instance, for which we also provide a formal IP model. The architecture is comprised of four main components as shown in Figure 1: a mobile user in the field, a Knowledge-based bundle generator (KB bundle generator), an allocation protocol, and the sensor network. The mobile device represents the point of entry of tasks into the system: i.e. the mobile user submits sensing tasks through a mobile app interface, by specifying a local area-of-interest (represented by coloured circular areas in the interface in Fig. 1) and its information requirements. As an illustration-of-concept, we prototyped the interface to the system on the Apple iPod Touch[2] motivated mainly by its wide adoption and the quality of the development tools provided. The *Knowledge-based bundle generator* on the user device recommends bundles of sensors that are known to be "fit-for-purpose" for that particular type of task specified by the user. The user device then communicates directly to the sensors (e.g. using WiFi or Bluetooth) the best sensor bundle which would be needed to satisfy its newly created task. Then, the sensors autonomously negotiate through the *allocation protocol* which one is the best task to serve as part of a bundle, thus finding a solution to the MSTA problem instance. Note that a single sensor bundle fully satisfies the sensing requirements of the task, therefore we have a one-to-one relationship between sensor bundles and tasks. Finally, the sensor network is configured accordingly and begins serving tasks by delivering sensor data to users.

MSTA problem model

This MSTA problem can be modelled as a tripartite graph whose vertices consist of a set of sensors $S = \{S_1, \ldots, S_n\}$, a set of bundles $B = \{B_1, \ldots, B_l\}$ and a set of tasks $T = \{T_1, \ldots, T_m\}$ as represented in Fig. 2.

[2]http://www.apple.com/ipodtouch/

Each task j is associated with p_j, representing the priority and/or importance of the task. For each task, we are given a set of sensor bundles, each of which would at least minimally satisfy the task's utility demand d_j. Each possible assignment of a bundle k to a task j is associated with a joint utility value e_{kj}, which is an estimate of how good a bundle of sensors could be to satisfy the sensing requirements of a task. We make no assumptions on the utility function: it could be for instance subadditive, superadditive or linear. Note that we assume that each of the bundle utilities would at least minimally satisfy the task's utility demand, i.e. $e_{kj} \geq d_j$. Instead if bundle k has a joint utility smaller than the task's demand or it cannot serve task j at all, we set $e_{kj} = -1$.

Our main interest lies in a dynamic scenario in which tasks arrive overtime and have different durations, therefore we adopt a discrete model of time in which we have timeslots. Tasks might arrive at the start of any timeslot and may last for any discrete duration. We assume that the *profit* for a task that lasts for multiple timeslots is the sum of the profits earned over all timeslots during the task's lifetime; we define the *profit* as the utility e_{kj} of a sensor bundle allocated to the task scaled by the priority p_j. In such a dynamic scenario where tasks might arrive over time while other tasks have already been allocated bundle of sensors, we consider the possibility of *preempting* sensors already serving other tasks in the case these might be more useful for satisfying the newly arrived (maybe more critical) task. Of course we want to avoid preempting sensors too often from ongoing tasks otherwise the allocation strategy could interrupt the support of a task too frequently, quite literally making sensors undecided as to which task they should contribute. We express this by including in the objective function a cost c subtracted from the profits for each individual preemption event, i.e., a sensor assigned to an ongoing task j at time $t-1$ which at time t is reassigned to a different task j'. Thus the goal in this problem is to maximize the profits of the tasks and minimize the cost of the sensor-task allocation over all the timesteps.

$$
\begin{aligned}
&\textit{Maximize: } \textstyle\sum_t (\sum_{kj} p_j e_{kj} y_{kjt} - c \sum_{ijj'} z_{ijj't}) \\
&\textit{Such that: } \textstyle\sum_k y_{kjt} \leq 1, \text{ for all } j,t \\
&\qquad \textstyle\sum_{kj} I_{ik} y_{kjt} \leq 1, \text{ for all } i,t \\
&\qquad y_{kjt} \leq C_{jt}, \text{ for all } k,j,t \\
&\qquad z_{ijj't} \geq y_{k'j't} + y_{k,j,t-1} - 1, \\
&\qquad\qquad \text{for all } i, t > 1, j \neq j', k, k' \\
&\qquad\qquad \text{such that } I_{ik} = I_{ik'} = C_{j,t-1} = C_{jt} = 1 \\
&\qquad y_{kjt}, z_{ijj't} \in \{0,1\} \text{ for all } j, j', k, t
\end{aligned}
$$

We now explain the IP. The decision variables y_{kjt} indicate whether bundle k is assigned to task j at time t. The first set of constraints prevents more than one bundle from being assigned to any one task at each timestep. The second set of constraints prevents any sensor from being used more than once, in multiple chosen bundles. Matrix I specifies the membership relationship between sensors and bundles. The third set of constraints prevents a bundle to be assigned to an inactive task; for this we define a matrix C as $C_{jt} = 1$ if task j is active at time t and 0 otherwise. Additionally, we use a decision variable $z_{ijj't}$ which is 1 if sensor i was assigned (within some bundle k) to one ongoing task j at time $t-1$ but is then reassigned (within some bundle k') to another task j' at time t. The fourth set of constraints forces $z_{ijj't}$ to be 1 if both $y_{k',j',t}$ and $y_{k,j,t-1}$ are equal to 1. These constraints apply only if sensor i appears in bundles k and k' (i.e. $I_{ik} = I_{ik'} = 1$), and task j is active at both times $t-1$ and t (i.e. $C_{j,t-1} = C_{jt} = 1$). Note that no reassignment cost is charged when a sensor switches from a task that is ending. For small instances, optimal solutions can be obtained by solving this IP, although as a generalization of Semi-Matching with Demands (SMD) [5] the problem is NP-hard, even to approximate. Larger problem instances require therefore to be solved with heuristics, which in our case are implemented by the entire distributed system architecture.

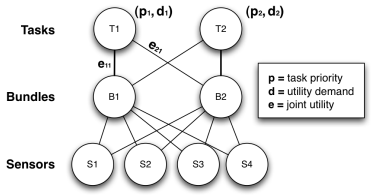

Fig. 2. MSTA problem instance (ST-MS-IA-HE) as a tripartite graph.

IV. Knowledge-based bundle generator

Task allocation in heterogeneous sensor networks requires knowledge of which sensor types are applicable to which kinds of task. We separate two issues: whether a type (or combination of types) of sensor *can* potentially satisfy a task, and *how well* might particular sensor instances perform on a given task. We encode this knowledge in a knowledge base (KB). The KB stores, for each kind of task, each type (or combination of types) of sensor that can theoretically achieve that task, and a *joint utility model* that allows us to compute the utility of particular sensor instances for that task. We refer to the types of applicable sensors as *bundle types* (because they determine the types of sensors that form our allocated bundles). These types may be defined using a *sensor ontology*, such as the one described in [4].

Figure 3 shows the reasoning process enabled by the KB in more detail. For each newly created task T_j at time step t the KB recommends a Joint Utility Model (JUM) to compute the sensor bundle utility and a Bundle Type (BT) to select the sensor types compatible with the task. The identification of Bundle Types was originally implemented as described in [4], by dynamically matching the sensing capabilities provided by each single sensor and the capabilities required by each task type. Sensor and task features were defined using ontologies expressed with the Web Ontology Language (OWL); the matching process was implemented using the Pellet open source OWL reasoner[3]. The output of this step is a set of Bundle Types, where each of the entry is composed by a set of sensor types which altogether would satisfy the information requirements of the sensing task. The BTs recommended at

[3]http://pellet.owldl.com/

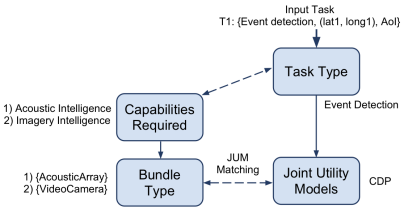

Fig. 3. Reasoning procedure.

Task Type	Recommendation
I	(BT_I + JUM_I)
I	(BT_2 + JUM_2)
2	(BT_3 + JUM_I)
2	(BT_2 + JUM_I)
2	(BT_2 + JUM_2)
...	...
N	

Fig. 4. Lightweight KB bundle generator.

this stage are just sets of sensor types, and they do not contain any kind of information regarding the number of instances to choose for each sensor type in the BT.

To determine the number of instances of each sensor type assigned to a task we use a Joint Utility Model. Such model consists of a suggested maximum, minimum or exact number of sensor instances for each of the sensor types forming a BT. Moreover it includes a utility function which is used to compute the estimated value e_{kj} for a group of sensor instances implementing the recommended BT. The KB indicates which JUMs are appropriate for each task type. Each JUM is only compatible with certain sensors, so the final step in the reasoning procedure is to match applicable JUMs with BTs.

To illustrate the above, consider two *task types*: "event detection" and "target localization". As result of the first reasoning stage the reasoner suggests to use $BT_1 = \{Videocamera, AcousticArray\}$ for "event detection", i.e. we can use both videocameras and acoustic sensors; instead $BT_2 = \{AcousticArray\}$ for "target localization". Later on it associates one or more JUMs to each recommended BT by searching the knowledge base. In this case (and also in Section VI) we assume that in the knowledge base there are just two utility models compatible with either of the two BTs. Based on the models proposed in [9] we associate to the *event detection* task a variant of the Cumulative Detection Probability model (which we call *CDP*) and to the *target localization* a 2D-localization function (referred to as *2D-Loc*) based on the distance and angle from the target. Therefore for the *event detection* task the reasoner will recommend ($\{Videocamera, AcousticArray\}$, *CDP*); instead for *target localization* it will output ($\{AcousticArray\}$, *2D-Loc*). Note that this potentially allows us to be very flexible in terms of allocation as the same task could be satisfied using different combinations of BTs and JUMs, and therefore would increase the chances of satisfying that task.

Following we list the JUMs mentioned above which will be also used in Section VI. Note that these functions have been chosen as examples and are extensively discussed in [9]:

CDP Given a set of candidates of size l for a task, this model chooses the k sensors maximizing $(1 - \prod_{S_i \to T_j}(1 - e_{ij}))$, where e_{ij} is the probability that a single sensor S_i detects an event for T_j. Here the joint utility is monotonically increasing as sensors are assigned but nonlinear. (Note that in Section VI for our simulations we will choose $k = 10$.)

2D-Loc Given a set of candidates of size l for a single task, this model chooses the best *pair of sensors* which maximize $1/U_{i,i'}$, where U represents the uncertainty of the target localization provided by the pair $(S_i, S_{i'})$. The uncertainty is given by $U_{i,i'} = \frac{\sqrt{d_i^2 + d_{i'}^2}}{|\sin(\theta_i - \theta_{i'})|}$; where d_i and θ_i are the distance and bearing, respectively, from the task location to the i-th sensor. This function is maximized when the angle separating the sensors is $90°$ and the distances are minimal. Also in this case the joint utility is non-linear.

Note that we cannot compare directly utilities computed with the first and with the second JUM because the objective function values have different meanings: CDP computes the cumulative detection probability, and 2D-Loc the opposite of the uncertainty of the target localization. Therefore we need to normalize those values (e.g. between 0 and 1) to obtain the percentage of satisfaction of the task with a particular utility value generated with one of the two models. This normalized value is exactly the e_{kj} parameter mentioned in Section III.

Lightweight KB Bundle Generator

The original implementation of the reasoning process is computationally expensive [4] due to the exponential-time complexity of the classification algorithm used by the Pellet reasoner. However, because the task types and sensor types are relatively stable (it is rare for new kinds of sensor or task to become available during an operation) it is feasible to precompute the results of the reasoner and store these in a look up table; such an implementation is more suitable for deployment on a mobile device, to avoid wasting battery life on expensive computation. The only assumption that this approach makes is that the device will have a sufficiently large storage capacity, which is reasonable for modern mobile devices. The structure of the look up table is represented in Figure 4. Note that each row of the table is a tuple composed by a Task Type (represented as an ID), a Bundle Type and a JUM.

V. Allocation Protocol

In this section we describe how we have extended the *disjoint coalition formation protocol* in [11] to solve the MSTA problem instance. The protocol runs on the two main entities composing our distributed system: sensors and user devices. When the user creates a task (e.g. using the iPhone interface illustrated in Figure 1), the user device computes *feasible sensor bundles* and their *joint utilities* using the Knowledge-based bundle generator; we call these pairs *bids*. These are then sent to the sensors which choose greedily the task to serve based on the average utility per sensor until there are no more bids.

The protocol consists of two main stages: in the preliminary stage - which we call *initial negotiation* - the user devices compute and distribute the bids to the sensors by first discovering which sensors are good candidates for each task; in the main stage — called *bundle formation* — the sensors decide upon which bundle to join in order to serve a particular sensing task. The *initial negotiation* works as follows:

1) At time t the users create tasks on their devices, characterized by a task type, a priority, an expected duration, a

deadline to be satisfied and a geographical area of interest — representing the entire geographical area on which the sensing task will take place. E.g. "event detection" at $(lat_1, long_1)$ within a circular area of radius 100 meter.

2) The user devices query the sensors in the geographical area of interest, asking for their location, sensor type and current status (unallocated or already allocated to another task).
3) The user devices then generate bids of the type $bid_\alpha = \langle U_\alpha : (T_j, B_k, v_{\alpha,t}) \rangle$, where U_α is a user, T_j is a task generated by U_α, and B_k is the bundle of sensors with highest value $v_{\alpha,t}$ computed as:

$$v_{\alpha,t}(T_j, B_k) = \begin{cases} p_j \cdot e_{kj} - c_{kj,t} & \text{if } e_{kj} \geq d_j \ , \\ & I_{ik} = 1 \ \forall S_i \in B_k \\ 0 & \text{otherwise} \end{cases}$$

Where e_{kj} is calculated by the *Knowledge-based bundle generator* using the recommended Joint Utility Model and considering only the sensor instances of the types specified in the Bundle Type which is implemented by B_k. Also note that we subtract from the profit the cost $c_{kj,t}$ proportional to the number of sensors that are already tasked and that we would need to preempt from other tasks at time t, as discussed in Section III.

4) Finally the user devices send to all the sensors included in the bundle B_k the bid bid_α.

In the protocol we do not specify any particular discovery mechanism. In the simulations instead we use a very simple device discovery policy based on the communication range of sensors and users. We do not consider the overhead generated by the discovery process, i.e. step (2) of the initial negotiation. This will depend highly on the discovery mechanism which will be implemented on the nodes, e.g. [6], therefore we leave this choice to the reader.

The main stage of the protocol is represented by the *bundle formation* part which mainly resides on the sensors:

1) Each sensor node keeps a list of bids in which it is involved sorted by decreasing average value $u_{ij,t} = v_{\alpha,t}/|B_k|$, where $|B_k|$ is the cardinality of bundle k.
2) If the sensor is currently not serving any task, it chooses the bid to which it can contribute the most, i.e. with the highest $u_{ij,t}$. It then sends an ACCEPT message to the sensors that are present in this bid (i.e. to all $S_i \in B_k$).
3) The sensor clears this bid (i.e. it officially commit to that task) only when it receives an ACCEPT message from all the sensors involved in the bid. This ensures that a bid is cleared if and only if all the sensors in the bid agree to clear it.
4) When a sensor node clears a bid, it sends a CLEARED to all its neighbors — the set of sensors with which it shares some bids and the users who generated those — notifying them that it has cleared (i.e. it was allocated to task T_j) and all bids including that sensor should be dropped from consideration.
5) The sensors that receive a CLEARED message from another sensor, delete from their bid list all the bids in which the sender sensor is included. The sensors stop the execution when they clear a bid, when their list of bids is empty or when an ACCEPT *timeout* for receiving all the ACCEPTs from the sensors in B_k is expired. If a sensor remains unallocated, it sleeps until it receives the next bid starting again from step 1.
6) The users that receive a CLEARED message from a sensor assigned to another user's task, delete the sensor from the set of neighbor sensors and recompute a new bid with the remaining sensors through the *KB bundle generator*. The users stop execution when they obtain a CLEARED message from a sensor assigned to one of their tasks or when a *convergence timeout* for satisfying the created task is expired since the beginning of the negotiation.

Note that if a task is not satisfied, the user device generates a new bid asking for an alternative sensor bundle to the KB bundle generator until a *convergence timeout* to satisfy the task expires (Step 6). This extends the original protocol in [11] with a *rebid mechanism* including an *expiration time* for the task, as discussed in Section II. We also introduced an ACCEPT *timeout* which expires if sensors have waited too long for an ACCEPT, allowing the user to rebid before the task's *convergence timeout* expires. In fact a sensor might be stuck in the "waiting for accepts" state if at least one sensor involved in the bid had previously committed to another task (sending a CLEARED before the arrival of the new task). This previously allocated sensor might not be freed by its current task before the expiration of the newly generated task's convergence timeout. Therefore the other sensors which accepted the new task would be stuck in a waiting for accepts state. The ACCEPT *timeout* allows the sensors waiting for an ACCEPT to drop the bid in case of a long wait and start from Step 1; allowing the user device to generate a new bid for the task thus improving the chances to satisfy the task thanks to a smart use of the *rebid mechanism*.

Preemption mechanism

The bid valuation function $v_{\alpha,t}$ already considers the cost of reassigning sensors from an ongoing task to a newly arrived one. Preempting sensors is important to increase the likelihood of satisfying higher priority tasks appearing overtime as discussed in Sec. II. Following we describe the *preemption* steps that we add after Step (2) of the *bundle formation* stage:

2.a When a sensor receives a $bid_{\alpha'}$ and is *currently allocated*, it will only be preempted from the current task j — and therefore it will send an ACCEPT — if the average contribution to the new task j' is strictly greater than the previous one, i.e. $u_{ij',t} > u_{ij,t}$.
2.b When a sensor is *preempted*, it sends a PREEMPTED message to all the sensors participating in the current bundle and to the user device owning the task. The sensors receiving this message will stop serving the current task, and the user device will drop it.
2.c If instead a sensor is not preempted (i.e. $u_{ij',t} \leq u_{ij,t}$.) it will keep serving the task for all its duration, after

which all the sensing resources assigned to the task will be released.

The procedure for each sensor implementing the protocol has a computational complexity of order $O(n^k \cdot |T|)$ to arrive at a decision (as proved in [11]), where n is the number of sensors in the network, k is the maximum size allowed for a sensor bundle and $|T|$ is the number of current tasks.

VI. Implementation and Performance

In this section we describe the implementation and performance of our fully distributed system. We include benchmarks of the *KB bundle generator* which we implemented on an Apple iPod Touch. We also show the performance of the proposed *allocation protocol* implemented in the REPAST Symphony Java simulation environment.

A. Mobile device

As mentioned in previous sections, we implemented a prototype app on an Apple iPod Touch 2nd Generation device. In particular we tested if it was feasible to implement the *Lightweight KB bundle generator* on a modern mobile device, given that the lookup table size could be very large and therefore computationally expensive to use. We implement the table showed in Figure 4 as a combination of 4 tables using the standard ER concept of *foreign keys*. We implement this ER schema using SQLite which is the integrated db engine in iOS (the OS installed in the Apple iPod Touch), and as a consequence the measured performance will be related to the chosen db engine. We run experiments first using a *Synthetic KB* filling the tables with synthetically generated data. Then we use a *Prototype KB* containing real knowledge from the literature regarding sensor/task types and the sensing capabilities required and provided by both. We then measure the size of the full KB stored on the iPod Touch flash memory (summing up the sizes of the 4 tables), and the time to retrieve all the (BT, JUM) entries associated with a single task type.

To generate a *Synthetic KB* we randomly generate 1000 task types, each consisting of an integer ID and a string of random characters of length 50 representing the task type description (e.g. "event detection"). We then generate a constant number of BTs descriptions with a length of 100 characters and we do the same with the JUM table generating random utility functions represented by strings of source code. We then fill the relationship table associating every task type with all the possible combinations of (BT, JUM). Keeping constant the number of randomly generated task types — 1000 task types — we increase the number of (BT, JUM) per task type and we measure the memory usage and the time to retrieve all the (BT, JUM) entries related to a particular task type. As shown in Figure 5 and 6 we run the experiments generating first 10, then 50 and finally 100 (BT, JUM) per task type. This leads to an increase in the size of the relationship table respectively containing 10000, 50000 and 100000 entries.

We also carry out benchmarks on a KB filled with realistic data from the literature; we refer to this as *Prototype KB*. To generate these realistic task types and sensor types we use openly available information about sensor devices/platforms and military sensing tasks mainly using the Missions and Means Framework as described in [4]. We feed this realistic data to a reasoning process like the one described in Section IV implemented on a laptop computer, and we then generate all the possible outputs given all the possible inputs. We store these results on the iPod Touch using the previously mentioned ER schema. We provide 4424 task types as an input to the user. Below we give 2 examples of realistic task type descriptions: *Detect_NonMilitaryVessel*, *Identify_RefuelingEquipment*. On average the reasoning process generates 5 different (BT, JUM) per task. Note that all the benchmarks conducted in this section are repeated for 20 times and then averaged to get significant values also with randomly generated entries/queries.

We summarize the experiments in Figure 5 and Figure 6, which show 3 main results. First, the storage space occupied by the KB grows linearly with the number of entries in the relationship table. Second, the query time increases logarithmically with the size of the relationship table. This is mainly due to the db engine used. More precisely, SQLite uses an R*-Tree which is a refinement of the R-Tree[4]. The search time complexity in an R*-Tree is basically the same as a B-Tree: $O(log(n))$ as it performs a binary search. Finally, the last and most important result is that with a realistic knowledge base the memory requirements and lookup performance on the mobile device are acceptable. In fact, our realistic knowledge base with 21378 entries occupies 12 megabytes of the flash memory of the iPod Touch – with its size growing linearly with the number of entries in the table. Most importantly, the time required to retrieve an entry from this lookup table is around 20 milliseconds, which increases logarithmically with the number of entries. The Apple iPod Touch which we used as an example of mobile device has the following technical specifications[5]: iOS 4.1, CPU ARM11 620 MHz (underclocked to 533 MHz), 128 MB DRAM, and 8GB of storage on flash memory.

B. Distributed protocol

We implement the distributed protocol in Java using *REPAST Simphony* — an open source agent-based discrete time simulation environment[6] — as a platform to simulate a sensor network composed of static wireless sensors of different types and mobile users on the field. We tested the distributed architecture on randomly generated problem instances in which tasks arrive over time without warning, and depart after spending a certain amount of time being active. In our simulation setup we deploy 250 sensor nodes with a uniform random distribution on a 2D grid of 500m×500m, generating randomly *2 sensor types*: acoustic and video. Each sensor node has a Sensing Range (SR) and a Communication Range (CR),

[4]http://www.sqlite.org/rtree.html – checked on 24th of January 2011

[5]http://daringfireball.net/linked/2008/11/25/new-touch-cpu – checked on 24th of January 2011. Apple does not officially publish full tech specs of its devices, these where obtained using a tool based on Unix `sysctl()` call.

[6]http://repast.sourceforge.net/ - checked 24th January 2011.

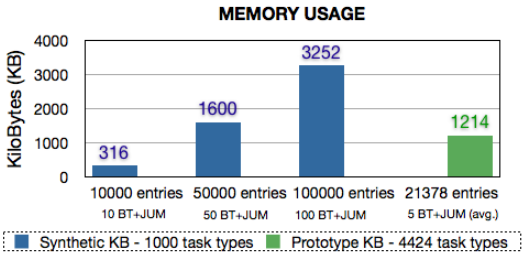

Fig. 5. Memory usage of Lightweight KB on a mobile device.

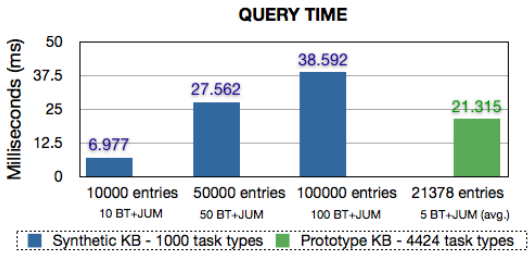

Fig. 6. Query time for Lightweight KB on a mobile device.

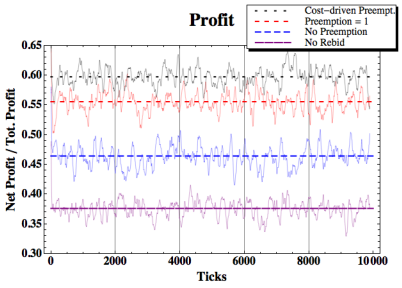

Fig. 7. Total Profit (5/3 tasks per ts).

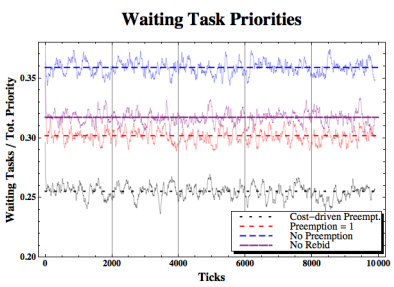

Fig. 8. Waiting Tasks Priorities (5/3 tasks per ts).

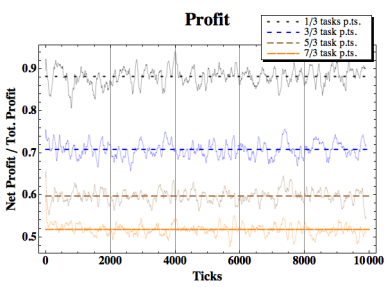

Fig. 9. Total Profit varying task arrival rate.

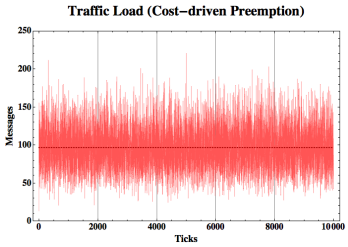

Fig. 10. Network Traffic for Cost-driven Preemption (5/3 tasks per ts).

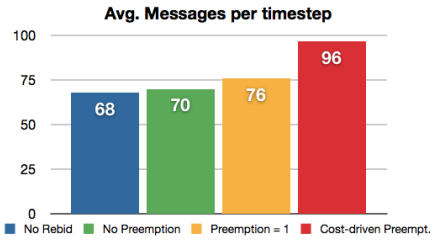

Fig. 11. Avg Messages (5/3 tasks per ts).

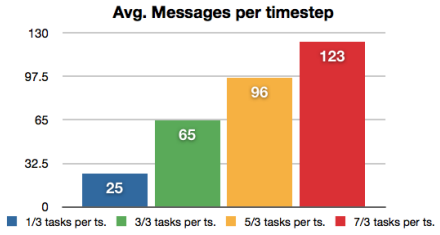

Fig. 12. Avg Messages varying task arrival rate.

which in our simulation are SR = 30m and CR = 60m. We randomly deploy 50 user nodes on the field, each moving with a random-waypoint model and having a CR = 60m. Every 3 timesteps, 5 randomly selected users create a new sensing task in their surroundings with a uniform distribution (i.e. arrival rate = $5/3$ tasks per timestep), with a maximum distance of the task from the user node equal to CR. Each task is owned by a single user and is characterized by a location (x,y) on the 2D grid, a task type and a duration (randomly chosen between 10 and 20 timesteps). Each task's priority and utility demand are generated with a uniform random distribution between 0 and 1. We discard a-priori tasks for which there are not enough sensors to be satisfied. Finally each task's expiration time is equal to half of its duration. In the simulation we use the two *task types* described in Section IV: "event detection" and "target localization". The type is chosen on task's creation with a uniform random distribution among these two types. We implement *CDP* and *2D-Loc* with a brute force approach enumerating and evaluating all the possible subsets of the candidates, which in the worst case is 2^l (where l is the number of sensor candidates). A brute force approach to be implemented on a user device is computationally efficient by assuming that in an emergency response scenario the number l of sensors surrounding the user is on the order of tens.

We run simulations for the full version of our protocol which includes both the rebid and preemption mechanisms (referred to as *Cost-driven Preemption* approach in the graphs), and we compare this with 3 other versions of our protocol to analyze the traffic overhead and the benefit of using the preemption and rebid mechanisms. We consider a version without rebid and preemption (*No Rebid*), a version which uses our rebid mechanism but no preemption (*No Preemption*) and a variant in which we use both rebid and preemption but we directly limit a sensor to be consecutively preempted only once (*Preemption = 1*). We repeat each simulation 10 times for 10000 timesteps and we average all the measurements. To compare the quality of the allocation achieved by each version of the protocol we consider the profit calculated at each timestep using the objective function in Section III. In Figure 7 our experiments show that the *Cost-driven preemption* protocol offers higher profit at each timestep, achieving on average 60% of the total profit. Note that the total profit is obtained by

summing at each timestep the profit of all the tasks being served by a sensor bundle (*satisfied tasks*), all the tasks which were not allocated a sensor bundle (*unsatisfied tasks*) and all the tasks which were previously preempted (*preempted tasks*). We keep the unsatisfied tasks and preempted tasks in the field for their remaining duration, so that the total profit represents the total potential task allocation quality. The *Preemption = 1* version has performance very similar to the *Cost-driven preemption*, while the *No Preemption* and *No Rebid* reach less than 50% of the total profit on average. In Figure 8 we show the fraction of waiting task priorities over the total task priorities — calculated as the sum of the priorities of the satisfied, unsatisfied, preempted and waiting tasks — for each timestep. This graph shows that the *Cost-driven preemption* version keeps only 26% on average of the most important tasks waiting for a decision. The worst waiting task rate is offered by the *No Preemption* (36%), while *No Rebid* (32%) performs a bit better because it sets the tasks to unsatisfied or satisfied just after the first bid.

Figure 10 shows the network traffic generated by the *Cost-driven preemption* protocol described in Section V. Our version exchanges the highest number of messages compared with the other 3 versions as highlighted in Figure 11. Nonetheless the maximum number of messages exchanged every timestep is around 220 messages with a much lower average of 96 messages per timestep, and with an acceptable variance around ± 50 messages. Figure 11 confirms that the other 3 variants of the protocol provide similar network traffic levels and that the least overhead is provided by the *No Rebid* variant. The number of messages exchanged on average at each time step is highly dependent on the maximum size k allowed for each bundle, which is stored in the KB bundle generator. We limited this number to 10 sensors for *CDP*, while instead *2D-Loc* is already limited to exactly 2 sensors. Finally, in Figure 9 we vary the task arrival rate showing the total profit achieved overtime by the *Cost-driven preemption* version. We run simulations with 1/3, 3/3, 5/3 and 7/3 tasks per timestep and the results show that the performance of our allocation protocol decreases sub-linearly by increasing linearly the task arrival rate. Figure 12 shows how the traffic load generated by the protocol increases linearly arriving at a maximum of 120 messages per timestep on average which is much less than the total number of nodes (i.e. 300). This shows that the allocation protocol is scalable and resilient to high task arrival rates.

VII. Conclusions and Future research

We formalized the Multi-Sensor Task Allocation problem in an emergency response scenario. We proposed a distributed architecture containing as the main novelty the use of both qualitative and quantitative metrics to allocate sensors to tasks. We extended a well-known pre-existent *allocation protocol* to deal with our highly dynamic problem settings. We also developed a prototype mobile app as an interface to our sensor allocation system, showing it is feasible to implement the *Knowledge-based bundle generator* on the device and that its performance is acceptable. We implemented the *allocation protocol* in a discrete time simulation, proving it is scalable and that preemption and rebid mechanisms provide a higher total profit. In future work we plan to compare this architecture with a fully centralized and a "hybrid" architecture; also addressing different MSTA problem instances. Further, we plan to use scheduling to handle deadlines similar to [8]. However in their work they consider a static scenario for which they propose a centralized solution. We are currently working on how to integrate data delivery/dissemination mechanisms in our architecture, to "close the loop" between requested and delivered information.

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. We acknowledge also Matthew P. Johnson & Yosef Alayev (CUNY), Matthew J. Williams & Konrad Borowiecki (Cardiff University).

References

[1] F. M. Bass, D. J. Tigert, and E. A. Pessemier. Complementary and substitute patterns of purchasing and use. *Journal of Advertising Research*, 9(2):19–27, 1969.

[2] L. Chaimowicz, M. F. Campos, and V. Kumar. Dynamic role assignment for cooperative robots. In *Robotics and Automation, 2002. Proceedings. ICRA'02. IEEE Conference on*, volume 1, pages 293–298, 2002.

[3] B. P. Gerkey and M. J. Mataric. A formal analysis and taxonomy of task allocation in Multi-Robot systems. *The International Journal of Robotics Research*, 23(9):939, 2004.

[4] M. Gomez, A. D. Preece, M. P. Johnson, G. de Mel, W. Vasconcelos, C. Gibson, A. Bar-Noy, K. Borowiecki, T. F. L. Porta, D. Pizzocaro, H. Rowaihy, G. Pearson, and T. Pham. An ontology-centric approach to sensor-mission assignment. In *EKAW'08*, volume 5268 of *Lecture Notes in Computer Science*, pages 347–363. Springer, 2008.

[5] M. P. Johnson, H. Rowaihy, D. Pizzocaro, A. Bar-Noy, S. Chalmers, T. F. La Porta, and A. Preece. Sensor-mission assignment in constrained environments. *IEEE Transactions on Parallel and Distributed Systems*, 21:1692–1705, 2010.

[6] M. Kohvakka, J. Suhonen, M. Kuorilehto, V. Kaseva, M. Hännikäinen, and T. D. Hämäläinen. Energy-efficient neighbor discovery protocol for mobile wireless sensor networks. *Ad Hoc Netw.*, 7:24–41, January 2009.

[7] D. Pizzocaro and A. Preece. Towards a taxonomy of task allocation in sensor networks. In *Proceedings of the 28th IEEE international conference on Computer Communications Workshops (INFOCOM)*, pages 413–414, Rio de Janeiro, Brazil, 2009. IEEE Press.

[8] S. D. Ramchurn, M. Polukarov, A. Farinelli, C. Truong, and N. R. Jennings. Coalition formation with spatial and temporal constraints. In *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems: volume 3-Volume 3*, pages 1181–1188, 2010.

[9] H. Rowaihy, M. Johnson, D. Pizzocaro, A. Bar-Noy, L. Kaplan, T. L. Porta, and A. Preece. Detection and localization sensor assignment with exact and fuzzy locations. In *DCOSS'09, Marina Del Rey, California, USA*, pages 28–43, June 2009.

[10] T. W. Sandhlom and V. R. T. Lesser. Coalitions among computationally bounded agents. *Artificial Intelligence*, 94(1-2):99–137, July 1997.

[11] O. Shehory and S. Kraus. Methods for task allocation via agent coalition formation. *Artificial Intelligence*, 101(1-2):165–200, May 1998.

[12] R. Wang, T. Mullen, V. Avasarala, and J. Yen. A Market-Based adaptation for resolving competing needs for scarce resources. In *Intelligent Agent Technology, 2006. IAT '06. IEEE/WIC/ACM International Conference on*, pages 350–356, 2006.