A System Architecture for Exploiting Mission Information Requirement and Resource Allocation

Fangfei Chen[a], Thomas La Porta[a], Diego Pizzocaro[b], Alun Preece[b] and Mani B. Srivastava[c]

[a]Dept. of Computer Science and Engineering, The Penn State University, US
[b]School of Computer Science and Informatics, Cardiff University, UK
[c]Electrical Engineering Department, University of California, US

ABSTRACT

In a military scenario, commanders need to determine what kinds of information will help them execute missions. The amount of information available to support each mission is constrained by the availability of information assets. For example, there may be limits on the numbers of sensors that can be deployed to cover a certain area, and limits on the bandwidth available to collect data from those sensors for processing. Therefore, options for satisfying information requirements should take into consideration constraints on the underlying information assets, which in certain cases could simultaneously support multiple missions. In this paper, we propose a system architecture for modeling missions and allocating information assets among them. We model a mission as a graph of tasks with temporal and probabilistic relations. Each task requires some information provided by the information assets. Our system suggests which information assets should be allocated among missions. Missions are compatible with each other if their needs do not exceed the limits of the information assets; otherwise, feedback is sent to the commander indicating information requirements need to be adjusted. The decision loop will eventually converge and the utilization of the resources is maximized.

1. INTRODUCTION

In a military scenario, commanders need to determine what kinds of information will help them execute missions. They issue high-level queries describing the missions such as "Monitor high value targets in North road" and expect to know which information is required for the missions and whether the information is available to them. However, the amount of information available to them is subject to the underlying information assets. For example, a limited number of sensors can only be deployed to cover a certain area and all their data may not be collected immediately due to the network constraints. Therefore, options for information sets should take into consideration of the underlying information assets.

In this paper, we propose a system architecture for 1) exploiting missions' information needs and 2) allocating resources among them. We define a mission to be a collection of ISR* tasks with temporal and causal relations. The intent of the commander decides the nature of the mission and the relation among its tasks. The commander's queries enter the system via a knowledge base which matches the mission descriptions with predefined task transition graphs. A task transition graph defines the spatial and causal relation among tasks of a mission. An ISR analyst either confirms or adjust this graph and passes on the graph to the rest of the system which analyzes the requirement of all tasks and allocates resources.

Resources are allocated among different missions but resources are associated with tasks. Resource requirement for each task is derived by analyzing its information needs. At a particular time, the resource requirement of a mission is decided by its current active tasks. Missions are compatible with each other if their needs do not exceed the limit of the information assets; otherwise, feedback is sent back to the commander suggesting an adjustment to the information choice. The decision loop will eventually converge and the utilization of the resources is maximized. Note that the focus of this paper is exclusively on ISR tasks, and therefore non-ISR tasks such as "send troops to intercept a suspect" or "fire a weapon" will not be considered by the system described below.

For a particular scenario, the task transition graph, the resource matching graph (which expresses task resource requirement) and the capabilities of resources are sent as input to a resource allocation problem solver.

*ISR stands for Intelligence, Surveillance and Reconnaissance

The abstract problem is a stochastic multi-dimensional knapsack problem. We formulate this problem and provide reference on potential solutions to this problem. The solver returns admission decision to the commanders, in the form of whether there is sufficient resource for such a mission, or the subset of missions that can be executed simultaneously consistent with the current resource inventory. The commanders then decide if this is satisfactory and either re-issue queries into the system or proceed to execute the mission with allocated resources.

The rest of this paper is organized as follows. Section 2 presents our system architecture. Section 3 discuss in details the resource allocation problem. Section 4 provides an example scenario and a walkthrough of the system. Finally, Section 5 concludes the paper.

2. SYSTEM ARCHITECTURE

In this section, we define several basic concepts and present our system. We describe how commanders express mission queries that enter the system and trigger a series of operations in the system. We illustrate how the system infers resource requirements for each mission, and decides on the effective allocation of ISR assets providing feedback to commanders.

2.1 Resource, Mission and Task

Before we present the system, first we define three important concepts: mission, task and resource. A mission needs multiple resources to execute. Resources are essentially every possible asset able to produce an ISR output. These include sensor platforms (e.g. UAVs), sensor motes, vehicle patrols, local informants, etc. However, only resources with capacity constraints are taken into our resource allocation problem; resources with infinite quantity or those that can be shared by any number of missions are not included.

A mission is a collection of ISR tasks with temporal and causal relations. Each task requires a subset of all the available resources. A mission can be executed only when its resource requirements can be satisfied during the whole course of the mission. However, at one particular moment, not all the tasks are active. That is, although we allocate a set of resources to a mission, the mission may use them at different times. One way to describe the relation among tasks of a mission is to use the task transition graph. Similar to a state-machine graph, each object (cycle) in the graph stands for a task, and arrows between the tasks indicate the temporal or causal relation between tasks. See Figure 1 for some examples.

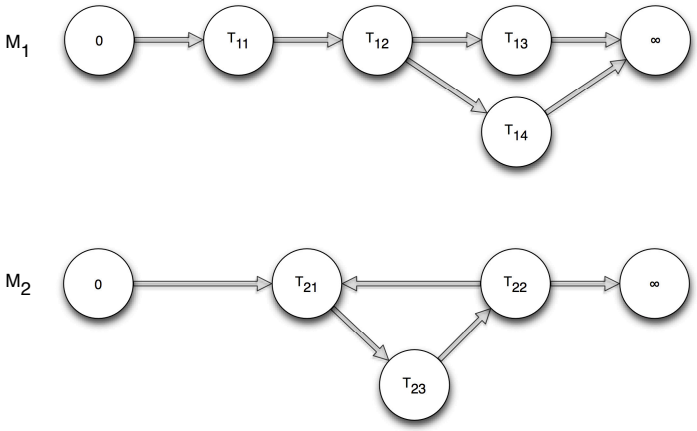

Figure 1: Task transition graph

In Figure 1, there are two missions M_1 and M_2. The tasks marked as "0" and "∞" indicate the start and end of the mission, respectively. All tasks in M_1 are *transient* in that each task can be executed at most once and once done never executed again; either T_{13} or T_{14} will be executed after T_{12}. Tasks are *recurring* (not transient) in M_2 and all will be executed.

However, the resource requirement of each task is not shown in the task transition graph. In fact, each cycle in Figure 1 is also a delegate of a subset of the resource. For the purpose of displaying the resource competitions among missions, we introduce another type of graph: resource matching graph. Using the same example, we draw this graph in Figure 2. An edge between a task and a resource indicates that this task requires this resource.

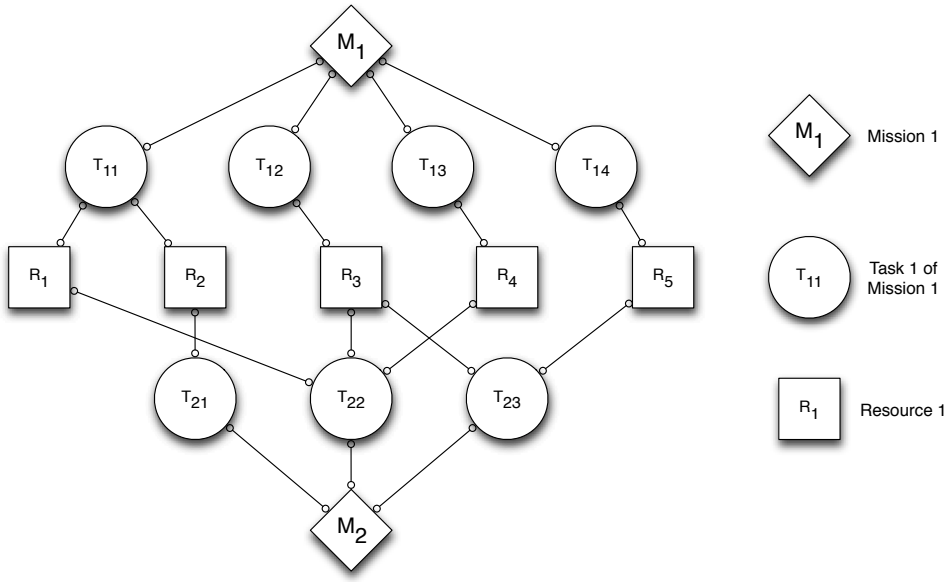

Figure 2: Resource matching graph

2.2 System Components

We propose the following system architecture. In the big picture, the information flows in the system and forms a feed-back loop starting from the commander, traversing every component of the system (see Figure 3). Based on the admission/allocation result obtained by a Resource Allocation problem solver, either the commander needs to readjust his command/query, or the mission obtains sufficient resources and get executed.

When the commander has an intent to initiate a mission, he sends a query such as "Monitor high value targets in North road" to a Knowledge Base (KB). The query may be expressed through an app interface, such as the one in Figure 4 which provides a convenient mission-entry form. The interface formulates commands using a controlled vocabulary and with limited semantic combinations. It also allows the commander to set different parameters of the mission such as its location on a map or its priority level. For more details about such interface and extensions to the system presented in this paper we refer the reader to another paper published in these proceedings.[1]

The query is then matched with a set of predefined *task transition graphs* in the knowledge base. As described above, these graphs include information regarding how a mission should evolve as a composition of ISR tasks. The most appropriate task transition graph from the set associated to the mission is automatically selected by utilizing mission parameters such as its type, location and relations among tasks. The ISR analyst then confirms the system choice or makes proper adjustments in the task transition directions and probabilities (e.g. consulting with the commander). Finally a task transition graph is associated with the current mission. Note that if the ISR analyst makes modifications to a task transition graph for a particular mission, then the edited graph could be stored in the KB as the new best choice for any other newly generated mission having the same settings as the one just submitted.

The set of ISR tasks composing the transition graph is then parsed by a reasoner-based component called

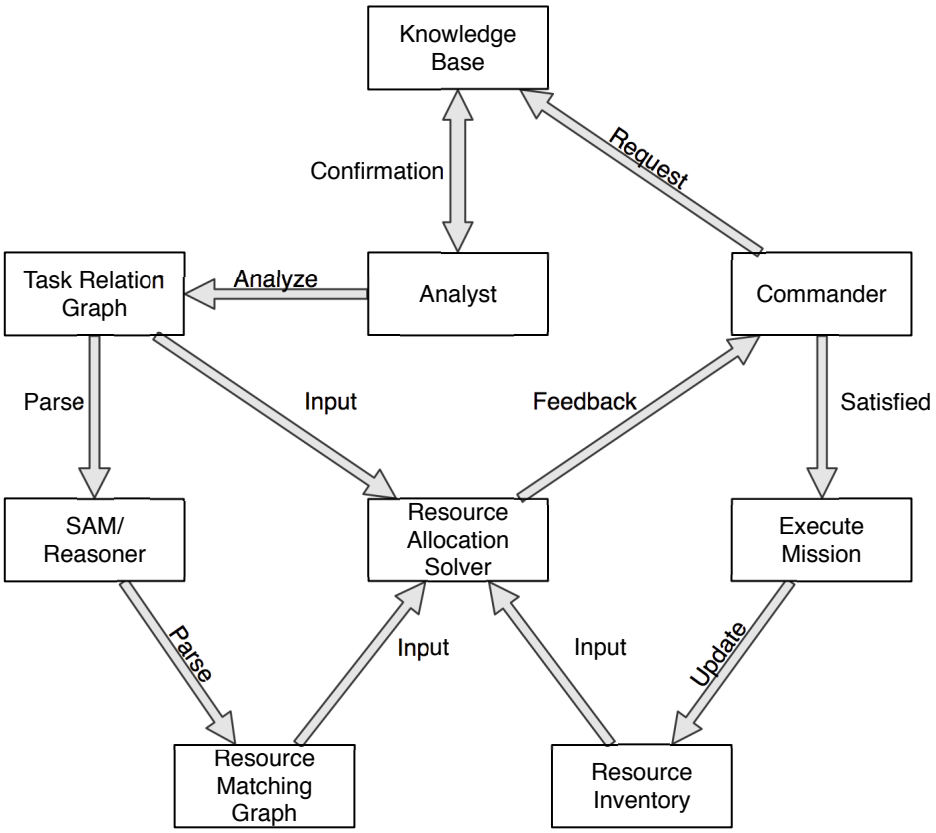

Figure 3: System Architecture

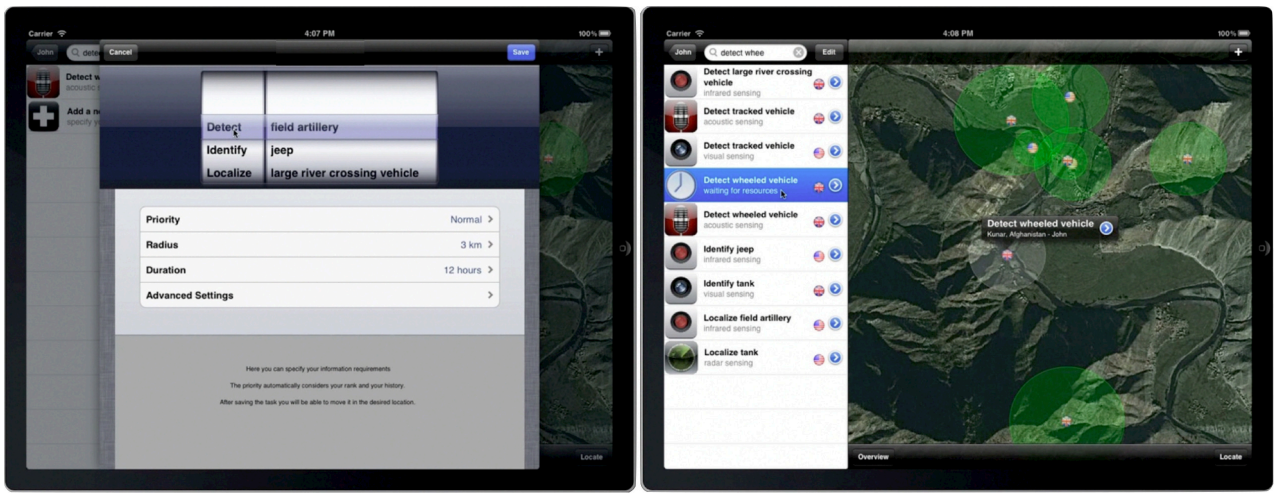

Figure 4: App interface for submitting missions to the system.

SAM/Reasoner[†], which recommends an ISR asset bundle semantically matching each tasks' requirements and satisfying tasks' utility demands (e.g. in terms of minimum quality of service requested). In particular, the reasoning process is split into two parts: first a reasoner decides what type of ISR asset bundles could potentially match the task requirements, we call the output of this step *Bundle Types* (BT). Second, we search in the resource inventory for feasible asset bundle instances matching the BT recommended at the previous step and we evaluate if each particular instance matches the utility demand of the task. Finally, the most useful ISR asset bundle is selected as the set of resources required by the task.

For example, based on our latest work[2] for an ISR task such as *Detect Wheeled Vehicle* the result of the first reasoning stage is to use $BT_1 = \{VideoCamera, AcousticArray\}$, i.e. we can use both video cameras and acoustic sensors for accomplishing this task. Later on we look for the recommended BT in the inventory of ISR assets and evaluate if those match the task demands. The output is therefore a set of ISR resources which semantically match the task requirements. This allows us to be flexible in terms of allocation as the same task could be satisfied using different BTs, and therefore the chances of satisfying that task with different ISR asset bundles are increased.[2] Note that literature is available fully describing the first reasoning stage,[3] while for the second stage we refer to our latest work.[2]

Based on semantic reasoning/matching, SAM will therefore parse each task requirements and output their resource requirements. The outcome of this procedure is the resource matching graph in Figure 3. Note that, task descriptions are sent to the SAM/Reasoner for obtaining the resource requirement per task. Then the system gather task requirements for each mission as the input to the next component (i.e. the solver), given that missions are the unit for the admission decision.

A resource inventory maintains the current resource capacity information. A solver takes as input the capacity, resource matching graph and task transition graph and answers the question whether this mission can be satisfied by the current resource inventory. In cases where more than one missions are competing for the resources, missions will have associated profits/priorities (e.g. set using the app interface in Figure 4). Then the solver answers the question which subset of the missions can be admitted using the resources in the inventory and maximize the total profits/priorities. We will see in next section that this problem is in fact a stochastic multi-dimensional knapsack problem.

Admission decision is sent back to the commander for approval: if the commander is satisfied, the admitted missions get executed; otherwise they may adjust the mission queries and send them to the system for a re-evaluation. The decision loop will eventually converge and the utilization of the resources is maximized.

3. RESOURCE ALLOCATION PROBLEM

In this section, we define and discuss the abstract resource allocation problem. The input to the problem include the task transition graph, resource matching graph and the resource capacity. The output of the problem is the admission decision— which subset of missions can be executed with sufficient resources.

3.1 Problem Definition

Given is a set R resources and a set M of m mission. Each resource R_i has a capacity c_i. Each mission M_j requires D_{ij} of resource R_i and is associated with a profit p_j if all its demands are satisfied. Let $p = [p_1, p_2..., p_m]$ denote the profit vector of missions. Note that in the system described here, the profit could simply represent the priority of a mission. We use a binary decision variable x_j to indicate whether M_j's demand is satisfied. That is, $x_j = 1$ if M_j is admitted; otherwise, $x_j = 0$. The problem is defined as the following linear program:

$$\max \quad \sum_j p_j x_j \tag{1}$$

$$\text{subject to} \quad \sum_j D_{ij} x_j \leq c_i \qquad \forall i \tag{2}$$

[†]SAM stands for Sensor Mission Assignment

The objective is to maximize the profit of admitted tasks with respect to capacity constraints. This problem is a multi-dimensional knapsack problem. In our problem, however, D_{ij} is a random vector based on a certain distribution or having some known statistics. If we solve this linear problem, it guarantees that the resource is sufficient even in the worst case. But this allocation is very low in efficiency, for the worst may only happen with extremely small chance. For example, if a mission's demand in a certain resource follows exponential distribution, the demand quantity may range from zero to infinite. As a result, this mission will never get chance to be executed even the chance it may need a great amount of such resource is very small.

In order to allocate more efficiently, we allow the total demands of admitted mission to overflow with a certain probability. Let ρ denote the overflow probability, which indicates the maximum frequency that admitted missions may violate the capacity constraints. We replace Equation 2 with the following:

$$Pr(\sum_j D_{ij}x_j > c_i) \leq \rho \qquad \forall i \tag{3}$$

These constraints upper bound the overflow chance. This problem is in the form of a *chance-constrained program*,[4] and may be solved by *scenario approximation*[5,6] or *sample average approximation* (SAA).[7,8] It is a hard problem in general; for some distributions such as Bernoulli,[9] it is even #P-hard to compute the probability $Pr(\sum_j D_{ij}x > c_i)$. Detailed solution for this problem is out of the scope of this paper. Please refer to the work by Chen et al.[10] for heuristics to solve this problem. Their basic ideas is to transform this chance constrained problem into a regular multi-dimensional knapsack using the concept of *effective size*. Effective bandwidth/size was introduced by Kleinberg et al[9] and has been used in solving problems such as stochastic load balancing and stochastic knapsack.[11,12]

4. AN EXAMPLE SCENARIO

In this section, we present an example scenario and a walkthrough of our system. The scenario (Figure 5) consists mainly of two crossing roads: Horizontal Rd and Vertical Rd. Potential targets may move along these roads. Static ISR assets deployed on the field consists of three directional cameras: Cam1 is on Horizontal Rd, Cam2 is on Vertical Rd and Cam3 is right at the crossroads. In addition, mobile ISR assets already deployed on the field are: a police patrol on the Horizontal Rd – note a patrol is considered as an ISR asset given that it produces reports on observed events; and an Unmanned Aerial Vehicle (UAV) flying on top of the Vertical Rd. Note that the patrol and UAV could be moved to different locations from the ones where those assets are deployed, although the system will try to make use of the resources already deployed first rather than trying to move them to different locations. Therefore the decision of moving or not the mobile ISR assets is part of the allocation mechanism.

Two missions of the same type, "Monitor any High Value Target on the X road", enter the system via two queries issued by the commander through the app interface in Figure 4. Here X is either Horizontal or Vertical road and as we will cover later this will translate into different ISR task requirements for each mission. Note that through the app interface the commander will be able to specify different details for each mission, such as location or mission priority.

The queries are forwarded to the Knowledge Base and matched with those predefined task transition graphs associated with that particular mission type (i.e. "Monitor any High Value Target on the X road"). The best task transition graph is sent to the analyst for confirmation. The analyst may use information such as where each task is located (i.e. Horizontal and Vertical Road) in order to tune the best transition graph. Since the two missions are of the same type and have intersecting areas of interest, it is reasonable to assume that the KB will generate the same task transition graphs for them as in Figure 6.

Besides two dummy tasks indicating the start and the end of the missions, the mission consists of three tasks. The first task is to detect any object on the road that is potentially a high value target. Once detected, the mission transits into the next task which identifies the detected object. If the detected target is of high value, the mission shifts into the third task, in which assets need to be allocated in order to track the target; otherwise, the mission reverse into the detecting task.

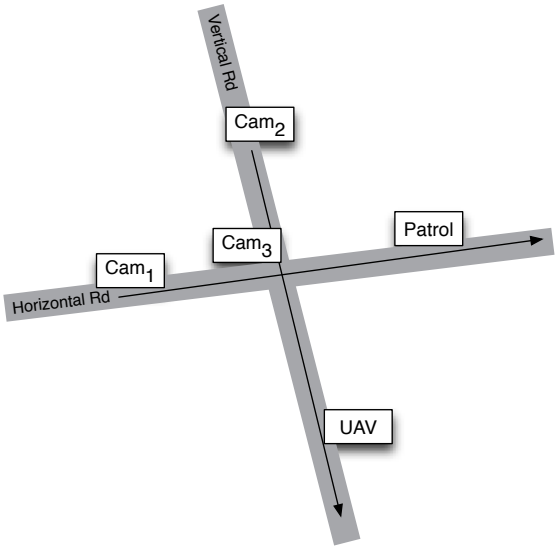

Figure 5: Map showing scenario and assets

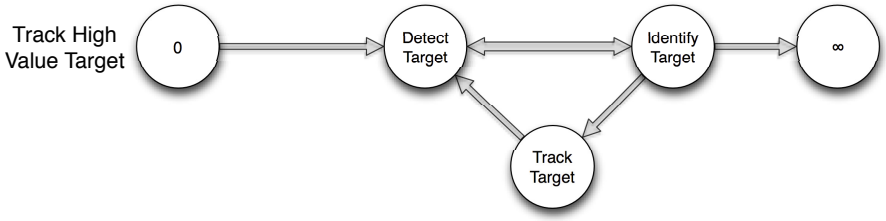

Figure 6: task transition graph for both two missions in the scenario

The set of tasks together with other information inherited from the mission description such as on which road each mission takes place is sent to the SAM/Reasoner. SAM matches each ISR task instance with the best set of resources. As described in Section 2.2, first each task is matched to an ISR asset Bundle Type (BT) and then we search in resource inventory for ISR asset instances matching the type of resources contained in the BT. In this specific case we have that the "tracking" task can be satisfied by the $BT_1 = \{Patrol, UAV\}$, that is either a police patrol or a UAV could potentially satisfy the task. Instead for both "identify" and "detect" tasks the ISR asset types which could potentially satisfy them are $BT_2 = BT_3 = \{VideoCamera\}$, i.e. any number of video cameras on the field allocated to the task might potentially satisfy the tasks. SAM then uses utility functions to decide which exact ISR asset instances matching the BT recommended are expected to fully satisfy each of the tasks' demands (e.g. in terms of quality of information required). These utility functions could be based on asset-task distances, and might include also the cost of moving a mobile ISR asset from one location to the other. Alternatively these functions could be more complex such as non-linear functions similar to the ones described in our previous works.[2]

The result of the SAM/Reasoner can then be represented in the form of a resource matching graph. For this particular scenario (see Figure 7), the detect tasks of the two missions match with two separate cameras. The identify tasks of the two missions match to the same "Camera 3" at the crossing, which provides not only video capturing function but also the capacity of identifying vehicles by their license plates. However, "Camera 3" may not be shared because it is a directional camera and it will necessarily point in opposite directions for each of the two missions. The tracking tasks of the two missions can use different types of resources as recommended by the SAM/Reasoner: in our case given the current deployment of the resources and the task locations, the vehicle patrol will be the best choice for the mission on the Horizontal Rd and instead the UAV for the one on the Vertical Rd.

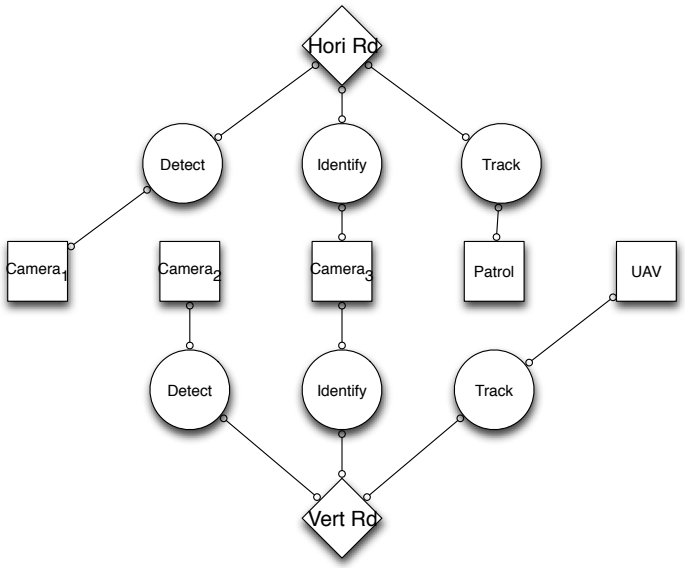

Figure 7: Resource matching graph for the example

Resource requirement of each task together with the task transition graph of a mission is used to calculate the distribution of the resource requirements of the mission. This distribution is sent to the resource allocation solver. Note that the current availability and quantity in the field of each resource is obtained from a resource inventory.

For this particular problem, the only resource that two missions compete for is "Camera 3". Only one mission can be admitted if we use Equation 1 with Equation 2. However, one mission does not necessarily occupy "Camera 3" all the time. As we mentioned, using Equation 1 with Equation 3 is more appropriate and leads to better utilization of resources. Therefore, for a tolerable overflow probability ρ, if each mission uses "Camera 3" for only a fraction of the time, these two missions may be both admitted with only a small chance of conflict. The tentative admission decision is sent from the solver to the commander for approval. If it is satisfactory, the missions get executed and the resource inventory is updated. Otherwise, the commander may adjust the mission query and send it into the system again.

5. CONCLUSION

In this paper, we designed a system architecture for exploiting mission resource requirement and assisting mission admission control. Missions are initially described as queries by the commander and are then parsed into task transition graphs and resource requirement of their tasks. This information is then sent to a solver as input to an abstract resource allocation problem. With some allowance of resource requirement overflow, the solver optimally allocates resources among multiple missions. The suggestion for admission control is sent back to the commander for approval. Negative results leads to modification of the commands or new queries into the system. The feedback loop converges when the resource utilization is maximized.

ACKNOWLEDGMENTS

Research was sponsored by US Army Research laboratory and the UK 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 authors and should not be interpreted as representing the official policies, either expressed or implied, of the US Army Research Laboratory, the U.S. Government, the UK Ministry of Defense, or the UK Government. The US and UK Governments are authorized to reproduce/distribute reprints for Government purposes notwithstanding any copyright notation hereon.

REFERENCES

1. A. Preece, D. Pizzocaro, D. Braines, and D. Mott, "Tasking and sharing sensing assets using controlled natural language," in *Proceedings of SPIE Defense, Security and Sensing 2012*, SPIE, 2012.
2. D. Pizzocaro, A. Preece, F. Chen, T. L. Porta, and A. Bar-Noy, "A distributed architecture for heterogeneous multi sensor-task allocation," in *Distributed Computing in Sensor Systems and Workshops, International Conference on*, **0**, pp. 1–8, IEEE Computer Society, (Los Alamitos, CA, USA), 2011.
3. 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*, *Lecture Notes in Computer Science* **5268**, pp. 347–363, Springer, 2008.
4. A. Charnes, W. Cooper, and G. Symonds, "Cost horizons and certainty equivalents: an approach to stochastic programming of heating oil," *Management Science* **4**(3), pp. 235–263, 1958.
5. G. Calafiore and M. Campi, "The scenario approach to robust control design," *Automatic Control, IEEE Transactions on* **51**(5), pp. 742–753, 2006.
6. A. Nemirovski and A. Shapiro, "Scenario approximations of chance constraints," *Probabilistic and randomized methods for design under uncertainty* , pp. 3–47, 2006.
7. J. Luedtke and S. Ahmed, "A sample approximation approach for optimization with probabilistic constraints," *SIAM Journal on Optimization* **19**(2), pp. 674–699, 2008.
8. B. Pagnoncelli, S. Ahmed, and A. Shapiro, "Sample average approximation method for chance constrained programming: theory and applications," *Journal of optimization theory and applications* **142**(2), pp. 399–416, 2009.
9. J. Kleinberg, Y. Rabani, and É. Tardos, "Allocating bandwidth for bursty connections," in *Proceedings of the twenty-ninth annual ACM symposium on Theory of computing*, pp. 664–673, ACM, 1997.
10. F. Chen, T. La Porta, and M. Srivastava, "Resource Allocation with Stochastic Demands," in *IEEE DCOSS*, 2012.
11. B. Dean, M. Goemans, and J. Vondrák, "Approximating the stochastic knapsack problem: The benefit of adaptivity," *Mathematics of Operations Research* **33**(4), pp. 945–964, 2008.
12. A. Goel and P. Indyk, "Stochastic load balancing and related problems," in *Foundations of Computer Science, 1999. 40th Annual Symposium on*, pp. 579–586, IEEE, 1999.