Allocation scheme

Ronnie Johansson

23rd April 2004

Abstract

Abstract

1 Introduction

2 Available and feasible services

The *service management* maintains the status of services and decides which are *available* (a service is available if its underlying resources are available). A service is *feasible* with respect to a task if both the task and service agent find it sensible to connect that task to the service (see Figure 2). For instance, a service might be considered infeasible if it will take too much time for the service to complete or if the underlying resources are occupied with other tasks and cannot afford to perform more tasks simultaneously. Adaptive systems should also be able to remove resources from previously assigned tasks to support other more urgent ones.

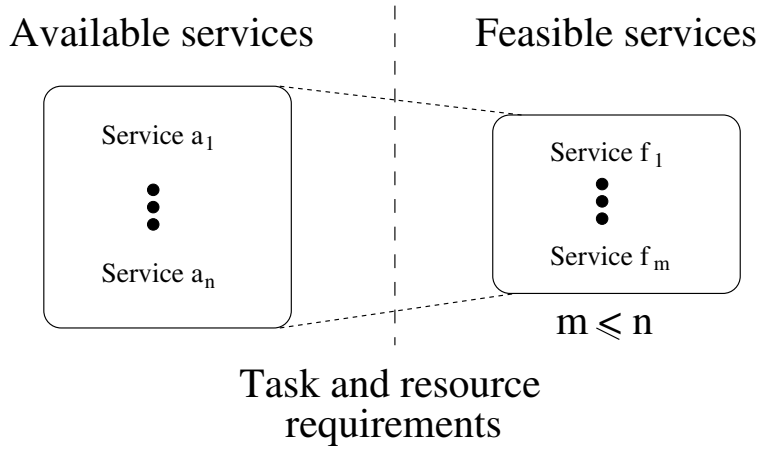

Figure 1: The set of services which are *feasible* for a given tasks is a subset of the *available* services.

2.1 Payoff

In decision theory, the (expected) outcome of a decision is sometimes expressed as the *utility* (i.e., the system gain) of the decision. In other applications, it is more appropriate to speak of *cost*. In the former case, the decision-maker tries to make the decision that maximizes the utility, and in the latter case the cost should be minimized. When multiple objectives are considered, it might also be appropriate to combine both utility and cost into an integrated value (e.g., by subtracting the cost from the utility). Here we call such a value *payoff*.

In the case of information acquisition, it is natural to consider both benefit and harm of a decision (where our decision is a candidate allocation between a task and a service). Utility is here associated with the desires of the task, e.g., measurement accuracy, how soon the measurement can be made, etc.). Costs are associated with the resources that realize the service.

Utility and cost may also be considered separately. An agent representing a task might refuse an allocation with a service that provides a utility that is too low (using the rationale that a utility value that is sufficiently low has no qualitative value to the agent). Conversely, a service can refuse an allocation if it is too costly to satisfy the corresponding task. Such interests should typically be handled by the allocation scheme, but in an agent-based implementation this responsibility assignment is intuitive.

3 Allocation

For every task in an ordered priority queue (this could be a continuous process), try to find the most *beneficial* (according to expected payoff) allocation (if any feasible services exist at all). If a feasible service does not exist, put the task back into the queue. Some tasks may get a priority below some threshold. Those should be ignored and deleted.

There might occur live-locks for low priority tasks (this, however, might be in order, since high priority tasks should be served first. One could, if appropriate, make pending tasks increase their priorities the longer they wait).

4 A centralized allocation problem

In this section, we study a centralized multi-sensor multi-target problem. The centralized allocation scheme re-considers its current sensor allocation, say, every t seconds. The tasks are the known targets which, according to our objectives, we want to track as long as they are within our reach. The allocation scheme, furthermore, tries to optimize the combined measurement accuracy of all targets. The subsequent estimates (and estimate uncertainty) is the result of the fusion of measurements from all sensors that track the same target.

Here, each service is directly connected to a sensor station which, during a time-interval (before the re-consideration by the allocation scheme), may acquire measurements from all targets within its range. There is one and only one service for each sensor. Each service carries information about where its sensor is located in the environment (in some common coordinate system), and its capabilities in terms of position estimation (it typically has an estimated standard deviation).

We propose this centralized allocation scheme, Algorithm 1, for illustration. Lines 3 to 6 iterate over all generated tasks. The tasks are assumed to have been ordered according to priority. Line 3 checks whether there are still tasks to be treated. If there are not, then naturally, the function can return the result temporarily stored in A^*. If the there are no more services, then the allocation scheme can do nothing else and should return.

Line 4 (described in detail in Algorithm 2) finds the best allocation (if any) between the task currently considered, t, and the feasible services. Line 6 removes all services affected by the allocation a^* from S, i.e., the set of available services.

Algorithm 1: Centralized allocation
Input: A queue of tasks T ordered according to priority, the set of available services S
Output: A set of allocations A^*
CENTRALIZED_ALLOCATION(T, S)
(1) $A^* = \emptyset$
(2) **while** $\neg$EMPTY(T) $\wedge$ $\neg$EMPTY(S)
(3) $t \leftarrow$ DEQUEUE(T)
(4) $a^* \leftarrow$ BEST_ALLOC(t,S)
(5) $A^* = A^* \cup a^*$
(6) $S \leftarrow S \setminus$ AFFECTED_SERVICES(a^*)

In Algorithm 2 lines 4 through 7 iterates over all available services, rates their expected payoff (see Section 2.1) solving task t, and selects the one that gets the highest score. If a particular service cannot at all fulfill the requirements of a its payoff value will be $-\infty$.

Algorithm 2: Best allocation
Input: A task t, the set of available services S
Output: A best allocation a^*
BEST_ALLOC(t, S)
(1) $best\_payoff = -\infty$
(2) $a^* = \emptyset$
(3) **foreach** $s \in S$
(4) $cur\_payoff \leftarrow$ CALC_EXP_PAYOFF(t,s)
(5) **if** $cur\_payoff > best\_payoff$
(6) $a^* \leftarrow a$
(7) $best\_payoff \leftarrow cur\_payoff$

Since the algorithm builds the solution (i.e., the set of allocations A^*) starting with the most important tasks first, we may add a line somewhere in the code that checks whether the function should abort its work immediately or continue. By adding such a line, we give Algorithm 1 an *anytime property*. This means that the longer the algorithm is allowed to run the better the outcome will be.

Comments and future work

- try some type of commonality (i.e., commonalities between tasks/services)
- study the centralized and decentralized case