Computational Adjustable Autonomy
for NASA Personal Satellite Assistants

Henry Hexmoor
University of Arkansas
Computer Science & Computer Engineering
Department
Engineering Hall, Room 313
Fayetteville, AR 72701

hexmoor@uark.edu

Justin Tyrel Vaughn
University of Arkansas
Computer Science & Computer Engineering
Department
Engineering Hall, Room 313
Fayetteville, AR 72701

jtvaugh@uark.edu

ABSTRACT

We will describe a simulator and simulated teamwork among a number of Personal Satellite Assistants (PSA) onboard the simulated space station patrolling for problem detection and isolation. PSAs reason about autonomies of potential helpers while helpers reason about their autonomies for deciding to help or to break away from prior commitments to help. We describe algorithms for computing PSA autonomies when there are concurrent and conflicting situations. We also offer empirical results about qualities of help a recruiting PSA receives when there are multiple, concurrent problems.

1. INTRODUCTION

The Personal Satellite Assistant (PSA) is a small flying robot that is being developed at NASA Ames for deployment on the international space station. These robots are an outgrowth of a need to free astronauts from routine tasks of inventory control, safety checks, and fault detection and isolation. PSAs are loaded with a variety of sensors including sensors for gas and pressure sensing as well as vision and speech. We describe implementation of a simulator that shows movement of several simulated PSAs in the corridors of the space station. We present algorithms that allow for PSA to reason about their autonomy and level of collaboration. We discuss tasks for the PSA including fire and locating a source of gas leak onto a station module or from the station module (called on gassing and off gassing, respectively).

Figure 1 shows a snapshot of three PSAs that are converging on the problem with one PSA already in the module with the problem. In the next section we will describe the steps involved in reasoning about the problem. The main issue being addressed is automatic adjustment of autonomy for the PSA.

Autonomy is defined and used in multi-agency [6, 7, 10, 12], and other disciplines including sociology [9], and philosophy [13, 14]. Autonomy is important in multi-agent interaction since it relates abilities in a self or a group to the individual's freedoms and choices. Agent-centered understanding of autonomy is required for coherent inter-agent interaction.

The notion of autonomy has been used in a variety of senses and has been studied in different contexts. The concept of autonomy is closely related to the concepts of power, control and dependence [5, 7]. An agent is autonomous with respect to another agent, if it is beyond the influences of control and power of that agent. In other words, autonomy presupposes some independence or at least restricted dependence. Further exploration of the relationship between power, control, and autonomy is beyond the scope of this paper. Autonomy is defined in [6] as the agent's degree to which its decisions depend on external sources including other agents. This can be called a cognitive autonomy. We have explored this further in [7]. The work described in this paper promotes the relativistic view of autonomy we introduced in [3, 4]. It is possible to differentiate autonomy into dynamic and deterministic types. Dynamic autonomy might capture the agent's initiative and ability to self-start, whereas deterministic autonomy might capture the agent's ability to refrain from actions it can perform.

Adjustable autonomy is a related notion that captures the idea of a human operator intervening and guiding actions of a machine [8]. Another example of the work on adjustable autonomy is [2]. A quantitative measure of agent autonomy is proposed in [1]. They define the degree of autonomy as an agent's relative voting weight in decision-making. This approach has several advantages. For example, it allows for explicit representation and adjustment of the agent autonomy. To our knowledge, it has been the first attempt to describe an agent's autonomy from a decision-theoretic point of view. Our own work elsewhere introduced another measure of relative autonomy [4]. In this

paper we will discuss a computational approach to determining autonomy for a PSA. This extends the quantitative approach we presented earlier in [11].

The remainder of this paper first presents a simplified strategy for handling a single problem. We then consider concurrent problems and discuss reasoning about autonomy computation. Then, we outline an alternate strategy and present experiments that illustrate quality of help. We complete the paper with concluding remarks.

2. STEPS OF PROBLEM IDENTIFICATION

In the first step, the PSA who detects the problem formulates a broadcast alert to send to other PSAs. This is initiated when a PSA locates an abnormality in its environment. The abnormality could be a variation in the ambient temperature, or an atmospheric imbalance (pressure variation, excess oxygen, nitrogen, carbon dioxide, etc.). Once it is discovered, the PSA proceeds to broadcast the alert to all other PSAs since pinpointing a problem is much more accurate with 2 or 3 agents reading from different angles than by a single PSA. This process is similar to the method used in radio signals localization; we refer to it as triangulation. The values passed in the alert message are *ID*, *ProbType*, *Loc*, and *Time*.

ID is used as an identifier so each contacted PSA knows whom they are dealing with. This is necessary, as they must respond with their calculated priorities later in this process. Also, it should be noted that upon initialization, each PSA is assigned a unique *ID* number, so that no duplicate *ID*s exist.

ProbType is a static numeric value assigned to a specific abnormality. This value is arbitrary, as it is used as an identifier later for calculating priority. For example, fire (excess heat) is read as problem type 1, on gassing (abnormally high readings of some gas) is type 2, and off gassing (too little pressure, or gas) is recognized as type 3. These numbers could easily be referenced as letters or even symbols; it makes no difference so long as the lookup tables in the heuristic for priority are changed accordingly.

Loc is the location of the detected problem. This is not an exact location, but rather a room number. This is used for comparison of distances needed through the entire station. When the exact location of the problem is not yet known, the variation within the room from the room center inside the destination module is of minor consequence in our computations. The loss of accuracy is minimized, since this is consistent throughout the priority process.

The *Time* variable is the relative amount of time, during which the problem has been known. For instance, if at time 13 an abnormality is discovered, and instantly a call for help is issued, but for some reason no one replies, or no replies are favorable, we might wait until time 18 and call again. In this case, at time 13 we report a time 1 as this represents the first time unit at which the problem was known and at time 18 we return 6 as it

has been known for 5 additional time units. This is used by the priority heuristic to compute increasing anxiety in the PSAs concurrent with the time period during which the problem remains unresolved.

In the second step, every PSA except the message originator completes evaluation of its capabilities to help. At this stage, it is assumed that the agent has a means of evaluating its resources (R), which for the PSA is based on its battery charge level. The calculated Cost (C) will be in terms of an estimated energy usage needed for the task at hand. C is subsequently compared to R for that PSA.

C is initially computed by first calculating the distance needed to be traveled. We assume all PSAs are constructed identically, move at identical rates of speed, and consume energy at the same rate. Of greater importance than the distance to the target is the distance from the target to the nearest recharge station. It does the system little good for a PSA to assist in locating a problem only to run out of energy and shut down. These distances are added together to produce a total distance to use in computing C.

The total distance to be moved is multiplied by the energy consumption rate. We then add to this an estimation of the amount of energy, which must be used during the task. The total is C.

$$C = (\text{Distance to target} + \text{Distance from target to recharge}) \times \text{Energy Consumption Rate} + \text{Energy Required for Task} \quad (1)$$

If C > R then an unfavorable priority is returned, indicating unavailability. Otherwise, C <= R, and the PSA can successfully help locate the problem and still recharge itself. When C <= R, Priority (P) is calculated by first determining what type of room the problem is located in. This is done since some locations are inherently more important than others. For instance, laboratories are relatively less important than habitation modules. Preference numbers (we will call *RoomFactor*) are assigned to each of the 4 available room types, 1 for storage module, 2 for crew quarters, 3 for labs, and 4 for the control station. Inside each room, values for each problem type are different. These numbers were chosen based upon a relative importance of the room and the amount of damage believed to be possible based upon a particular problem in that room. This value, which we denote as Q, is used in calculating the job weight. Q is determined through equation 2 below, and is used in the final priority calculation for P.

$$Q = \ln(\text{Time} + \text{RoomFactor}) \quad (2)$$

The natural log is used for this equation because it causes Q to change along a predictable curve as either Time or RoomFactor increase.

P is computed by using distance as a scalar and comparing the new job weight to the old job weight.

$$P = Q_{new} \times (1 - \text{Distance to New Target} / \text{MAXDISTANCE}) - Q_{old} \times (1 - \text{Distance Remaining to Old Target} / \text{MAXDISTANCE}) \quad (3)$$

MAXDISTANCE is the maximum distance a PSA can move through the entire station. The distance plays an important role in the calculation of P. This is due to the observation that the PSA with the smallest distance to move will be the most likely to arrive earlier. Thus, the time to completion of the task is lower with this PSA.

In the third step, all P values are collected by the alert originator and are compared to one another. This can be done in a number of ways, and the actual method used is not important. The two PSAs with the highest P values are selected as helpers. Permission (Y) is granted to the two PSAs selected, while the rest are given Denial (N). The granting of N is just as important as the granting of Y as it brings closure to the issue and allows the other PSAs to return to their previous tasks.

Now we describe the 4th step. The PSAs who received Y proceed to move to the new problem destination in this step. While this takes place, the originator PSA begins the process of triangulating the problem by cutting the search space down to less than ¼ of the room in question. At this time, the PSA, which located the problem, is unable to depart the job site unless the problem disappears (is dealt with by an outside party). If the originator were to leave the job site, the two helper PSAs might become confused, and wait indefinitely after their arrival.

The fifth step begins once the helper PSAs arrive at the job site. Upon arrival, a helper PSA sends a message to the originator PSA telling it that help has arrived. At this time, the originator picks a face on the prism defining the search space, and the helper picks another. When the second helper arrives, it picks the remaining third face. In this scheme no two PSAs are checking the same axis. Each PSA checks 9 points on the face it has chosen, thus a measure of completion can be taken at any point in time during triangulation.

Upon completion of this task, the PSA either awaits a new assignment or it proceeds to the recharge area. At this point in time, the PSAs' returned P will be the highest possible for a given problem type and a given location based upon the PSAs' current location because the second half of equation 2 is reduced to 0. This can be seen when the PSAs' situation is analyzed in relation to the equation. If the PSA is idle, then Q_{old} is set to 0, as there is no task, which to compare the new value. It is obvious that the P returned will depend on the distance to travel and the Q value of the problem.

Figure 1 shows a module of the space station. The round objects which can be seen (one at the top of the image, two in the lower portion) are PSAs. The diamond is used to represent the location of the problem. The PSA at the top of the image is the one, which located the problem; the other two are "helpers" responding to an alert. The Figure shows a module and entry corridors from other modules.

3. PSA PERSPECTIVE ON PROBLEM

The previous section dealt solely with the simplest form of this process. Figures 2 and 3 illustrate problem detection with conflicts and concurrent problems. Figure 1 is the state diagram for a PSA that locates a problem, while Figure 2 shows the state diagram for a PSA responding to the alert message.
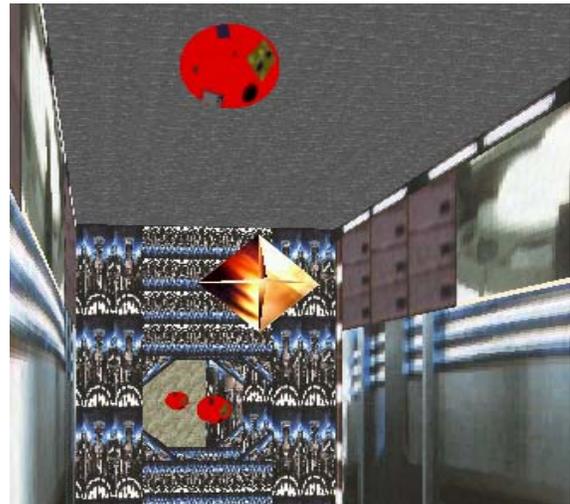

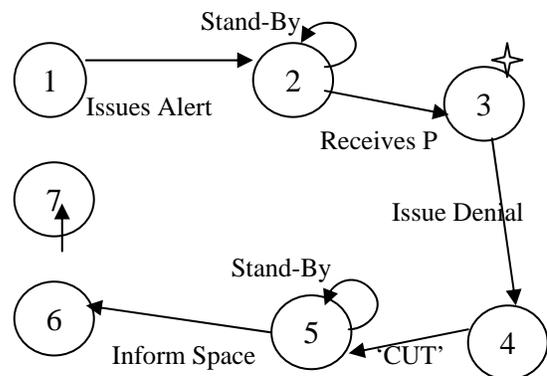**Figure 1: Three PSAs are converging to the module with problem**

Figure 2: State Diagram of PSA that Locates a Problem

As soon as a problem is detected (state 1 in Figure 2), the PSA that locates it sends out an alert signal. It is assumed at this stage that the alert is received and understood by all other agents (state 2 in Figure 2). Upon completion of the alert, originating PSA enters a standby mode awaiting the receipt of the P's returned by potential helpers. When all Ps have been received, originating PSA determines the best helpers based upon the values received (state 3 in Figure 2) and proceeds to issue permission or denial. Permission is issued to the two PSAs with the highest P,s and denials are sent to all others. Immediately after granting permission to its helpers, the originating PSA begins the process of *cutting* the search space (state 4 in Figure 2). This is done by taking a series of readings from various points in the room, and this reduces the area likely to hold the

source of the problem by as much as 75%. After the originating PSA has completed *cutting* the search space, it once again enters a standby mode, awaiting the arrival of the two helper PSAs (state 5 in Figure 2). When the arrival messages have been received, the originating PSA informs the helpers of the search space and begins triangulating by picking a face on the search space (state 6 in Figure 2). After completion of triangulation, the PSA enters a finished stage, and is allowed to resume normal activities (state 7 in Figure 2).
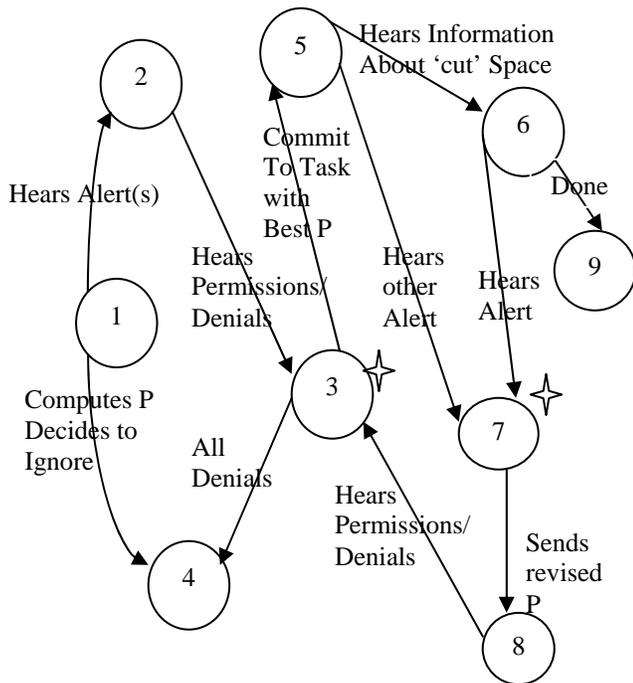
Figure 3: State Diagram of PSAs that Respond to Alert

For the PSAs receiving the alert message, the process begins immediately upon hearing an alert (state 1 in Figure 3). Upon receipt of the alert message, the PSA computes a P for the problem (state 2 in Figure 3). If multiple alerts are received, the PSA computes an individual P for each problem and retains these P's for later comparison. If some condition, such as low energy level, would prevent the PSA from completing the task it is being asked to evaluate, the PSA determines that it must ignore the alert message, and resumes its previous task (state 4 in Figure 3). Otherwise, the PSA considers itself committed (state 3 in Figure 3).

When a PSA considers itself to be suitable to help with a task, it is not guaranteed that the PSA will be selected for the task. The PSA must await receipt of all permissions and/or denials on ongoing tasks, and then determine its next course of action. If a task denial is received, the PSA then ignores the alerts it has received and resumes its previous task (state 4 in Figure 3). Otherwise, it performs an internal evaluation on the importance of the permissions received to that individual PSA. This evaluation is performed based upon the P values it calculated for those tasks. When multiple permissions are received, the PSA sorts through them to find the one it has the highest P value. Once the permitted task with the highest P value is selected, the

PSA begins moving toward the problem location (state 5 in Figure 3). If during transit the PSA hears another alert message, it must once again calculate a P for the new problem and its existing problem. The P value for the previously committed task is going to be scaled by a Measure of Completion (MOC) (state 7 in Figure 3). MOC is the percentage of the current task that has been completed, with 10% being considered complete upon arrival at the target location. The equation for P will be:

$$P = [Q_{new} \times (1 - CurrentDistance / MaxDist)] \times (1 - MOC) \quad (4)$$

The PSA sends P after it computes it, and awaits Permission or Denial (state 8 in Figure 3). If permission is received, the PSA will proceed to the new target location, ignoring its previous assignment (state 5 in Figure 3). If denial is received, then the PSA resumes its previous task (state 4 in Figure 3). Upon arrival at the problem site, the PSA must await the completion of the *cutspace* operation by the alert originator. As soon as the PSA receives the search space information, it begins triangulation by picking a face on the search space and taking readings (state 6 in Figure 3). At this time the PSA can receive other alerts. If an alert is received, the PSA will evaluate P by using the MOC method previously described (state 7 in Figure 3). MOC increases by 10% for each reading the PSA takes on its selected face of the prism forming the search space. Since the PSA takes 9 readings on the face, if all readings are complete but the final location is still being calculated, the PSA will not leave the site. Therefore, MOC measure are between 0.1 and 1.0, and P calculated by the MOC equation will range between 100% and 0% of the P value which would be determined normally, with lower values calculated the closer the task is to completion. Upon completion of triangulation, the PSA enters the finished stage and is allowed to return to patrolling or to recharge if needed (state 9 in Figure 3).

The originating PSA reasons about autonomy in state 3 in Figure 2. P values of potential helpers refer to their abilities and when the originating PSA issues permission/denial, it is giving each agent a deontological attitude. In State 3 of Figure 3, the potential helper reasons about autonomy by considering only the tasks for which it has been given permission. From these it selects the task for which it has the highest P value. In the current implementation, we treated all permission equally. But in general, agents should assess the permission according to degree or type. Perhaps, inter-agent relationships would dictate how to evaluate permission. We leave this for future work. In state 7 of Figure 3, PSAs are in the middle of helping. When they receive additional alerts, they must recompute their P values. This involves reasoning about their autonomies. A helping PSA must decide whether to continue helping with a current task or to abandon it and help with another task. Here we consider the rate of task completion. A more complex reasoning process will be needed to account for other factors. An objective factor is the consequence of helping or not helping with any task. A subjective factor is the relationships it has with other PSAs and how its decision about help will affect its relationship in the future.

The strategy described so far will eventually complete each task. We will call this strategy 1. We have a new, suggested process, which we will call strategy 2. In strategy 2, all PSAs that reply to an alert will receive an ordering rather than simply awarding

permission and/or denial to a PSA. That is to say that the most desirable helper receives a 1, the second 2, and so on such that all other PSAs are ordered 1 to *n* with 1 being the most favorable and *n* being the least favorable from the perspective of a particular alert originator. The helper PSAs can then compare their calculated Ps and the associated ordering returned to them, and then select the task for which it is best suited. This is very similar to an inverted contract net. In this way, the Quality of Help (QOH) should be balanced more evenly across all tasks. The more important tasks will still get those PSAs closest to them since the P values calculated for it should be higher than those calculated for a less important job, especially if that job lies farther away. This increases the degree of autonomy for the PSAs, as it allows them to decide when and where they go.

Since the algorithm used in calculating P for the PSAs is so heavily reliant upon distance, there will be very few cases where the most serious problem doesn't get its top two selections. One case would be that in which a PSA was in the same room with a problem but was not the originator (i.e., finder) of it. In this case, it is likely that the problem the PSA is in the room with will take precedence for it because the distance it must move to that problem is zero. These cases, however, are probabilistically rare in practice.

We will use QOH as a measure of quality of our algorithms. QOH quantifies the degree of satisfaction the finder has with the helpers received for a task. In the next section we will discuss the experiments with the quality of help an originator receives.

4. EXPERIMENTS

We conducted an experiment with 10 sets of calculations to determine which PSAs would go to what problem. The locations of the problems and the PSAs were arranged such that the system was relatively balanced. The Q value for each of the 10 problems was generated randomly using a four-sided die for the ones digit and a ten-sided die for the tenths. The PSAs were ordered by their returned priority value for each problem. For quantifying the results, numeric values were associated with the ordering. Receiving one's first choice was worth *n* points, second choice *n-1*, third *n-2*, etc., such that receiving ones last choice on a problem was worth 1 point.

The number of helpers available in the system was sufficient in each test to meet the demands. The exact same scenarios were run under strategy 1 and strategy 2. The results are shown in Figure 4. QOH is the vertical axis and the Number of Simultaneous Problems is the horizontal axis.

The method of selecting helpers used for the strategy 1 set was the first method described above, in which a PSA that locates a problem sends permission to its top two picks. If either of those picks is unavailable, it requests its third pick, and then its fourth, and so on. In this manner, the PSA, which locates the problem, is selecting its own helpers. This method could be considered drafting. If a PSA is granted unchallenged permission in round 1, for example, then that PSA is unavailable for the future rounds.

The second method was that first proposed in the previous section, and was deemed strategy 2. This involves each PSA that locates a problem ordering all available helper PSAs from 1 to *n* where *n* is the maximum number of helper PSAs in the system. The helper PSAs then proceed to request permission from the originator to which they responded with the highest priority value. The locator PSAs take the two highest requests and grant those two permissions. It is necessary to still grant permissions and denials so that all problems receive help. This is very similar to a contract net in which bids are offered and either accepted or rejected. In this case, the two highest "bids" are granted permission while the rest are given denial. This method involves the helper PSAs attempting to select where they go while those PSAs requesting help simply act as mediators.
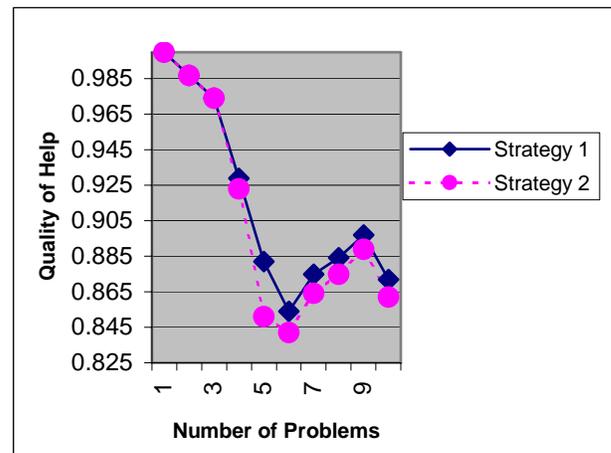

Figure 4. Quality of Help vs. Number of Simultaneous Problems for Two Proposed Strategies

The results of the experiment are shown in Figure 4. This shows that while the two strategies produce very similar results, strategy 1 actually works better in this system than strategy 2. The Quality of Help value can be seen to drop off slower in set 1, bottom out at a higher value in set 1, and achieve a secondary peak with a higher value in set 1. Additionally, the first method of selection is quicker and computationally less intensive.

However, from the perspective of autonomy, the first method restricts the agents to a greater degree. PSAs have less freedom in mobility and choice of tasks. Priority only plays a role in the very first stage of the process; after that it is up to the locator PSAs. The second method allows the helper PSAs to undertake whichever task is both best fitting to them and compliant to the greater needs of the system. While the results show that this method produces slightly inferior performance, the differences are not overly significant. In most cases, differences were calculated at 3% or less.

The reason for this decrease in performance lies in the nature of the decision making in the system as a result of the new process. By decentralizing the decision-making process, decisions are being made based upon less than the total amount of information in the system.

This study could be further extended to see how satisfied the helper PSA is with the task assignment. In this case, the task to which the PSA reports the highest priority receives a value of n, and the task it least wants to assist on receives a 1. The numbers for all PSAs involved aer averaged and divided by the maximum, n, to determine a satisfaction value.

5. CONCLUSION

The domain of Personal Satellite Assistants is a dynamic environment where multiple, and possibly concurrent, problems may develop. PSAs will benefit from teamwork in this environment. A challenging problem is how PSAs might reason about committing to help and revising their commitments in light of other problems. Reasoning about autonomy is an area we have explored as a basis for moment-to-moment commitment. In this paper we show how one PSA can reason about autonomy of other PSAs in recruiting them for problem resolution. The recruited PSA in turn reason about their own autonomies when other problems arise. Our solution has been based on objective information such as distances and problem severity. Our future work will consider subjective factors such as qualities of relationships and application of autonomy determination in reasoning about teams [3, 15].

6. ACKNOWLEDGEMENTS

7. REFERENCES

[1] Barber, S., Martin, C. 1999. Agent Autonomy: Specification, Measurement, and Dynamic Adjustment, In Proceedings of the Autonomy Control Software Workshop, Agents '99, pp. 8-15. May 1-5, Seattle, WA.

[2] Barber K. S., Goel A., and Martin C.E., 2000. Dynamic Adaptive Autonomy in Multi-Agent Systems, In *Journal of Experimental and Theoretical Artificial Intelligence*, 12(2): 129-147.

[3] Beavers G. and Hexmoor H., 2001. *Teams of Agents*, In *Proceedings of the IEEE Systems, Man, and Cybernetics Conference*.

[4] Brainov S and Hexmoor H., 2001. Quantifying Relative Autonomy in Multiagent Interaction, In IJCAI- 01 Workshop, Autonomy, Delegation, and Control.

[5] Brainov S., Sandholm T., 1999. *Power, Dependence and Stability in Multiagent Plans*. In *Proceedings of AAAI/IAAI* 1999: 11-16.

[6] Castelfranchi, C., 1995 *Guaranties for Autonomy in Cognitive Agent Architecture*, In N. Jennings and M. Wooldridge (eds.) *Agent Theories, Architectures, and Languages*, pp. 56-70, Spinger-Verlag.

[7] Castelfranchi, C. 2000. Founding Agent's Autonomy on Dependence Theory, In *proceedings of ECAI'01*, pp. 353-357, Berlin.

[8] Dorais G., Bonasso R.P., Kortenkamp D., Pell P., and Schreckenghost D.. 1998. Adjustable Autonomy for Human-Centered Autonomous Systems on Mars, Presented at *Mars Society Conference*.

[9] Dworkin G., 1988. *The Theory and Practice of Autonomy*, Cambridge.

[10] Hexmoor H., 2000a. A Cognitive Model of Situated Autonomy, In Proceedings of *PRICAI-2000 Workshop on Teams with Adjustable Autonomy*, Australia.

[11] Hexmoor H., 2000b. Case Studies of Autonomy, In *proceedings of FLAIRS 2000*, J. Etherege and B. Manaris (eds), p. 246- 249, Orlando, FL.

[12] Hexmoor H., Kortenkamp D., 2000. Autonomy Control Software, An introductory article of the special issue of *Journal of Experimental and Theoretical Artificial Intelligence*, Kluwer.

[13] Mele, A. 1995. *Autonomous Agents: From Self-Control to Autonomy*, Oxford University Press.

[14] Schneewind, J.B. 1997. *The Invention of Autonomy: A History of Modern Moral Philosophy*, Cambridge Univ. Press.

[15] M. Tambe, D. Pynadath, C. Chauvat, A. Das, and G. Kaminka, 2000. Adaptive agent architectures for heterogeneous team members, In Proceedings of the *International Conference on Multi-agent Systems (ICMAS 2000)*, Boston, MA.