Scalable Computing: Practice and Experience
Volume 8, Number 1, pp. 41–48. http://www.scpe.org
ISSN 1895-1767

COMPUTATIONALLY ADJUSTABLE AUTONOMY

HENRY HEXMOOR* AND BRIAN MCLAUGHLAN*

Abstract. Reasoning about autonomy is an integral component of collaboration among computational units of distributed systems. This paper introduces an agent-level algorithm that allows an agent to continuously update its autonomy with respect to recurring asynchronous problems with the aim of system-wide collaboration efficiency. The algorithm is demonstrated in a relevant scenario involving NASA space station-based Personal Satellite Assistants, which can handle dynamic situation management that frustrates global collaboration protocols.

Key words. Agents, Autonomy, portable satellite assistant.

1. Introduction. Computer-controlled systems feature prominently in large-scale projects currently under development by the military, commercial, and scientific agencies. Examples of these projects include the US military's Network-Centric Warfare doctrine, IBM's Autonomic Computing initiative, and NASA's space station project. As these systems have increased in complexity, self-governing components have come to feature prominently in their design and control. This change in paradigm from direct human control to indirect human oversight has forced designers to address issues involving the autonomy of these sub-systems.

Autonomy is defined and used in multi-agent system research [6, 7, 11, 12, 13] and other disciplines including sociology [10] and philosophy [14, 15]. It is important in multiagent interactions since it relates the abilities of an agent to its freedoms and choices. The understanding and quantification of an agent's autonomy is required for coherent interagent interaction.

The concept of autonomy is closely related to the concepts of power, control, and dependence [5, 7]. The notion of autonomy has been used in a variety of senses and has been studied in different contexts. It generally presupposes some independence or restricted dependence. However, it can describe many different but related concepts. An agent can be autonomous with respect to another agent if it is beyond the influences of control and power of that agent. It can also be used to describe quality of choice and can even encompass self-imposed "sense of duty" concepts.

While autonomy can be intuitively understood, it unfortunately is a complex topic whose exact definition and implementation is rather elusive. However, by identifying "types" or "subclasses" of autonomy, specific aspects of the concept can be defined and quantified. The multiagent system designer can then utilize these models to focus on the particular attributes of autonomy that would be most beneficial for the particular implementation.

Autonomy is defined in [6] as the agent's degree to which its decisions depend on external sources including other agents. This form of autonomy can be called Cognitive Autonomy. This concept has been explored further in [7]. This paper utilizes this definition of autonomy and promotes the relativistic view introduced in [3, 4].

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 [1] with quantitative measure proposed in [2]. In this, the degree of autonomy is defined as an agent's relative voting weight in decision-making. This approach has several advantages including the allowance for explicit representation and adjustment of agent autonomy.

The remainder of this paper presents our work regarding computation and determination of adjustable autonomy levels for collaborative, problem-solving agents in a multi-agent system. Section Two describes our approach, including the generalized algorithm. Section Three portrays an implementation of this algorithm for NASA's PSA program. Experiments performed on this system are chronicled in Section Four. Section Five presents the conclusions drawn from this work.

2. Approach. This paper addresses adjustable autonomy in a distributed system where agents discover, announce, and complete asynchronously occurring tasks. The tasks are generic and require multiple participant collaboration to solve. The collaboration process is facilitated through a four-stage bidding process:

1. Announcement
2. Priority

*Southern Illinois University Carbondale, Illinois, 62901 {hexmoor, brianm}@cs.siu.edu

3. Permission
4. Acceptance

In addition to providing a mechanism for collaboration on tasks, the algorithm must be able to scale well and handle dynamic and complex situations. That is, it must be able to handle multiple, conflicting tasks. It must be able to handle changes to the problem topology such the introduction or removal of key agents or tasks. Ideally, the algorithm will handle variations without excessive setback in its ongoing computations.

Announcement

Upon discovery of a new task, the discovering agent—known here as the originating agent—broadcasts the discovery to the group. Each agent maintains a list of announced tasks. The task data structure is shown in Figure 2.1.

An agent will update the information about a task as it receives relevant information. For simplification, this paper assumes that all agents have some method of hearing announcements and other bidding related information, whether through direct or indirect means. If this simplification is not the case, the algorithm will yield as best a solution as is possible with the information available.

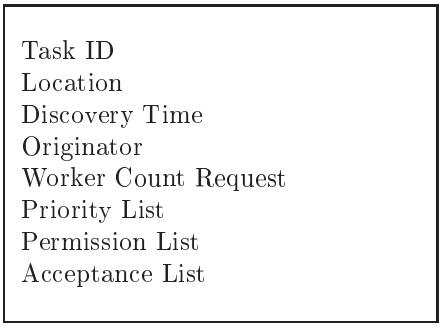

FIG. 2.1. *Task Data*

Priority

Upon receiving and archiving the task announcement, an agent will reason about its objective suitability to address the task. The agent may include several attributes, e.g., necessary skills, energy usage, and the time that the task has been active. It incorporates these factors in assigning some meaningful priority to the task. It is important to note that, at this stage, the agent will not account for alternative tasks. That is, it will not rank a task higher or lower according to its personal preferences. Reasoning along subjective considerations will occur later. Upon determining its priority for the task, the agent will announce the score to the other agents. In the most basic version of this system, only the originator needs to maintain all the priorities. However, as will be described later, some enhancements are possible in which agents can adjust their acceptance based on the priority scores made by other agents.

Permission

The originating agent collects these priority scores and generates a permission list. In its simplest form, the permission list is an ordered list of the priority scores. However, the algorithm utilized by the originating agent can be much more complex, taking into account abstract concepts such as trust and affinity the originating agent has towards particular agents or even known synergies among bidding agents. Ultimately, this permission list contains the bidding agents in the order of most to least desirable for joining the task. Although the originating agent only needs a specific number of agents to perform the task, it will create an ordered list containing all bidding agents in the event that some of the most desirable agents will be unable or unwilling to participate. The originating agent publishes this list to the group.

Acceptance

Unlike many contemporary systems such as online auctions, a bid does not constitute a contract in this system. Each agent is allowed to tentatively accept or reject the permission granted by the originating agent. Additionally, a tentative acceptance is not enforceable. If an agent finds a task for which it is more suitable, it is free to abandon its current task. As will be shown later, it is assumed that the agent has taken into account any disruption its action would make on its current task if it were to act. Thus, the acceptance becomes an announcement of which task the agent is currently considering to perform.

The bidding agent makes her acceptance determination by accounting for several factors including its desire or suitability for this relative to other tasks, the level of permission granted by the originating agent for this and other tasks, and the priority of alternative agents should the agent decline to perform the task.

The bidding agent takes into account competing tasks at this stage rather than in the priority stage so that it can provide benevolence for the system. For example consider an agent X that has placed bids on two tasks, Task 1 that has been announced by agent A and Task 2 that has been announced by agent B. Agent X determines its priority for Task 1 to be quite low, but sees its priority for Task 2 to be high. Both agents A and B have published permission lists in which agent X is among the top choices. If agent X were to take a greedy stance, it would accept the task for which it gave the highest priority, in this case Task 2. However, if it further inspects the permission lists, it may discover that the agents that would be forced to perform Task 1 in agent X's absence are not particularly well-suited for the task and would struggle, while the alternative agents for Task 2 are only slightly less-suitable than agent X and could still perform adequately. To provide for optimal system performance, agent X could choose to accept Task 1 even though it would personally prefer Task 2.

There are three caveats to accepting tasks. First, an agent may only give its acceptance to one task. If it has already accepted a previous task, it must announce its withdrawal from that previous task.

Second, an agent cannot accept a task that has been locked. A task is locked if n higher-ranked agents have accepted the task, where n is the requested number of agents for the task[1].

Third, an agent cannot accept a task where it is not ranked in the first n non-rejecting agents in the permission list where n is the number of agents required to perform the task. That is, if a task needs three agents, and agent X is ranked fourth, it cannot accept the task unless one of the first three decline it. Conversely, any agent may decline a task regardless of its ranking in the permission list. These scenarios are shown in Figure 2.2.

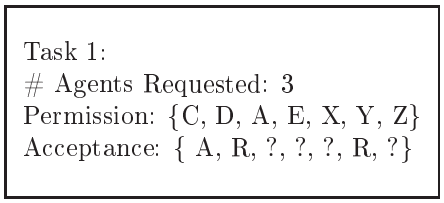

FIG. 2.2. *Agent X cannot accept the task until either agent A or E rejects it.*

Algorithm

An algorithm has been developed to facilitate this bidding scheme. This algorithm is implemented at the agent level and runs continuously. The pseudo code for this algorithm is shown in Algorithm 1.

Some notes regarding this algorithm. In the final If statement, the agent does nothing if its chosen task could be filled by more qualified agents. This forces the agent to wait to see if the desired task will become available. As an alternative, the agent could change this to a rejection and recalculate a "second best choice". Then, if the desired task becomes available due to top-ranked agents rejections, it can change its acceptance back to the original task. This alternative keeps all agents busy, but it may cause additional start-up costs from changing tasks

It is the task originator's responsibility to ensure that the task does not get lost in the shuffle. To this end, the originating agent will periodically broadcast the current state of the task.

Rather than rigidly define the four phases of the bidding process, the algorithm allows each agent to proceed independently. This precludes the need for coordination of phase changes that may be difficult in some environments. However, this could cause the originating agent to publish a permission list before all agents have given their priority scores. With the publication of this list, the agents are free to begin the acceptance process before potentially ideal agents announce their priority. To prevent unnecessary shuffling as new agents bump out less ideal workers, the agents should take potential shuffling into account when bidding. Alternatively, if the agents can communicate with all other agents in the system, then the originating agent can delay publishing the permission list until all agents have announced their priorities.

To illustrate the algorithm, consider the following scenario. To simplify the illustration, the scenario will be shown from the perspective of the tasks.

[1]In the PSA application, "n" is three. I.e., three robots are required to triangulate source of the problem.

Algorithm 1 Bidding Scheme Pseudocode

```
while 1 do
  Sense surroundings
  Task List update
  Append new discovered tasks
  Append new heard tasks
  Update existing tasks
  for Each task t in Task List do
    Calculate and announce t_priority
    if t_originator == self then
      Calculate t_permission List
      Announce task t
    end if
  end for
  Calculate best non-locked task
  for Each task t in Task List do
    if t ≠ best then
      Announce rejection
    else
      if Self rank < n^th non-rejecting then
        Announce acceptance
      else
        Do nothing
      end if
    end if
  end for
end while
```

Agents A and B have discovered and announced Tasks 1 and 2, respectively. Agents A, B, C, and D are within responding distance to these tasks. Figure 2.3 shows the state of the tasks after the agents have begun to respond with their priority to the tasks and the originating agents have published permission lists. For simplicity, permission is granted based solely on announced priority.

Task 1	Task 2
# Agents Needed: 2	# Agents Needed: 2
Priority: {A,7},{B,3}, {C,8},{D,9}	Priority: {A,1},{B,8}, {C,2},{D,7}
Permission: D, C, A, B	Permission: B, D, C, A
Acceptance: ?, ?, ?, ?	Acceptance: ?, ?, ?, ?

FIG. 2.3. *Permission List Publication*

With the publishing of the permission lists, agents are now free to begin accepting or rejecting the tasks as shown in Figure 2.4. Agents C and D are the most ideal candidates for Task 1. C will accept this task as B accepts Task 2. They will quickly reject the alternate tasks.

However, D has been accepted for both tasks. Greedily, it could accept Task 1, but its rejection of Task 2 would force Task 2 to be performed by A, a very unsuitable agent. It must decide on a course of action—greedy or benevolent.

Agent A cannot announce its acceptance of Task 1 despite its likely preference toward it. Rather, it will wait to see what Agent D announces so that it will not have to begin its inept performance of Task 2 and then possibly switch mid-execution to Task 1.

Next, consider how the algorithm will react to a dynamic situation. For this we introduce another agent, agent E. This agent hears the updates given by the two task originators and determines its priority for the tasks. Additionally, agent C detects a new task, Task 3. This situation is shown in Figure 2.5.

Task 1	Task 2
# Agents Needed: 2	# Agents Needed: 2
Priority: {A,7},{B,3}, {C,8},{D,9}	Priority: {A,1},{B,8}, {C,2},{D,7}
Permission: D, C, A, B	Permission: B, D, C, A
Acceptance: ?, A, ?, R	Acceptance: A, ?, R, ?

FIG. 2.4. *Partially Accepted*

Task 1	Task 2	Task 3
# Agents Needed: 2	# Agents Needed: 2	# Agents Needed: 2
Priority: {A,7},{B,3},	Priority: {A,1},{B,8},	Priority: {A,7},{B,3},
{C,8},{D,9}, {E,0}	{C,2},{D,7}, {E,5}	{C,9},{D,8}, {E,5}
Permission: D, C, A, B, E	Permission: B, D, E, C, A	Permission: C, D, A, E, B
Acceptance: ?, R, ?, R, R	Acceptance: A, ?, ?, R, ?	Acceptance: C, ?, ?, ?, ?

FIG. 2.5. *New Task and Agent*

In this situation, C chooses its own task and rejects its previous acceptance of Task 1. Additionally, E immediately sends rejection to Task 1 due to its absolute inability to perform the task as demonstrated from its priority announcement of 0.

This leaves several issues to be resolved. First, it allows A to accept its ideal Task 1 as it is now in the first 2 non-rejecting agents and does not need to wait for D's rejection.

Agent D is now desired by all three tasks. It still has some determination to make before choosing. For instance, D's choice could depend upon whether agent A chooses Task 1 or Task 3. It also depends upon whether important tasks will be left without adequate workers.

The exact method utilized for determining its choice depends on how much complexity the system designer imbues in the agents' decision-making process. Ideal efficiency is a difficult problem that is most likely beyond the practical scope of real-world agents regardless of the algorithm. However, the agent could play the prisoner's dilemma game to second-guess what other agents may choose. Perhaps the simplest and most computationally efficient method when faced with such incomplete information would be for Agent D to take the greedy choice and let the other agents adjust to maximize the remaining system performance.

Additionally, this example illustrates a problem with all task allocation algorithms—maximizing utility when not enough workers are present. If such a scenario is likely in the system, the designer could include a task priority that would modify the agents' behavior such that they would be more likely to accept critical tasks and leave less vital tasks understaffed.

Despite the problems, this example demonstrates how the algorithm can adapt to changes made mid-calculation. Rather than toss out the bidding process and start over or exclude new agents and tasks from the proceedings, the agents make some quick adjustments and continue.

3. An application: The Personal Satellite Assistant (PSA). A PSA is a small (basketball-sized) flying robot that is under development at NASA Ames (at the Moffet field AFB[2]) 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 equipment for gas and pressure sensing. In the remainder of this section we describe an implementation of our algorithm that allows PSAs to perform several appropriate tasks such as fire and gas leak (i. e., on- and off-gassing) detection while reasoning about their autonomy and level of collaboration.

As per the algorithm, the PSA that detects the problem formulates a broadcast alert to send to the 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 such as high or low pressure, or excess oxygen, carbon dioxide, or nitrogen. The PSA sends the alert containing the type of problem and type of room in which the problem is located to persuade other agents to help it pinpoint the source of the problem more accurately. This process is similar to the method used in radio signal triangulation.

[2]We thank Yuri Gawdiak for a tour and discussions in 2002.

To determine its suitability for this task, the PSA must account for its energy resources. Each PSA has a limited battery power that will be consumed during transit as well as during the task execution. It is assumed that the PSA has a means of evaluating its resources R, which in this case is its battery charge. It will then calculate its cost C to perform the task.

C is initially computed by calculating the distance to travel to the task and the subsequent distance to a power 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. The total distance to be moved is multiplied by the energy consumption rate. An estimation of the amount of energy required to perform the task is added to get the total cost C.

$$\begin{aligned} \mathrm{C} =& (\text{Distance to target} + \text{Distance from target to recharge}) \times \text{Energy Consumption Rate} \\ &+ \text{Energy required for task} \end{aligned}$$

If $\mathrm{C} > \mathrm{R}$ then an unfavorable priority is return indicating unavailability. Otherwise, when $\mathrm{C} \leq \mathrm{R}$, the PSA can successfully help locate the problem and still recharge itself. In this case, priority P is calculated by first considering what type of room in which the problem is located. This is done since some locations are inherently more important than others. For instance, laboratories are relatively less important than the control center. Additionally, the particular anomaly detected can influence the priority for a particular room. For example, off-gassing of oxygen in a equipment storage module would be less disastrous than the same problem in a habitation module. Conversely, high levels of magnetic interference may be dangerous for the equipment but could be of little consequence to humans inhabiting their quarters. The determined value, which we denote as Q, is used for calculating the job weight and is used in the final priority calculation for P.

$$\mathrm{Q} = \ln(\text{Time} + \text{RoomProblemFactor})$$

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

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

$$\mathrm{P} = \mathrm{Q}_{\mathrm{new}} \times \left(1 - \frac{\text{Distance to new target}}{\text{MAXDISTANCE}}\right) - \mathrm{Q}_{\mathrm{old}} \times \left(1 - \frac{\text{Distance remaining to old target}}{\text{MAXDISTANCE}}\right)$$

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 quickest. Thus, the time to complete the task is lower with this PSA.

As the PSAs proceed through the bidding process—priority declaration, permission, and acceptance—and the chosen PSAs begin to arrive at the problem location, they will take a prism on the face of the search space and begin scanning. This will allow PSAs that arrive quicker to begin the search process, while PSAs that arrive later can help refine the results. Thus, a measure of completion can be taken at any point in time during the triangulation.

4. Experiments. Experiments were performed utilizing the PSA scenario. The locations of problems and PSAs were arranged such that the system was relatively balanced. The Q value of each problem was randomly generated. The number of PSAs in the system was sufficient in each test to meet the demands.

The exact method of acceptance was performed under two strategies. In strategy 1, agents chose to accept the task in which they were highest ranked for permission. Note that this does not necessarily mean that the PSA greedily chooses the task for which it attributed the highest priority. Rather, it will choose the task of the originator that most values the PSA's assistance. For instance, if a PSA is listed as first in the permission list, it will accept that over a task where it is listed second. In strategy 2, PSAs perform as described in the algorithm—they choose to accept a task such that the sum of all priorities chosen is maximized. This strategy should spread the quality of help across the problems.

The results of the experiments are shown in Figure 4.1 and shows that the two strategies produce very similar results. However, the first strategy gives slightly better performance in this particular simulation and is computationally less intensive in general.

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

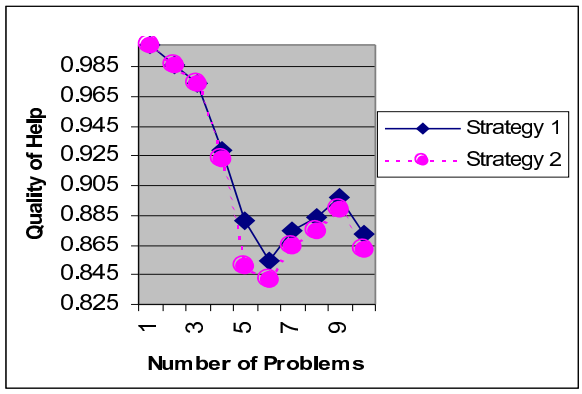

FIG. 4.1. *Quality of help for two strategies*

From the perspective of autonomy, the first strategy restricts the agents to a greater degree. The individual 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 originating PSA. The second method allows the bidding PSAs to undertake whichever task is both best fitting to them and compliant to the greater needs of the system.

5. Conclusion. As computer controlled systems increase in complexity, automated collaboration of subsystems becomes more relevant and critical to system efficiency. Utilizing the concept of adjustable autonomy—reasoning about commitments in particular—is a critical component to solving this problem. This work has shown how reasoning about autonomy can form the basis of moment-to-moment commitment making.

We have shown an algorithm that can be utilized for dynamic decision-making that is flexible enough to handle agents that join or leave before tasks are completed, as well as being able to handle tasks that appear during the execution of other tasks.

We have shown how this algorithm can be implemented in a relevant and current problem—that of task management of NASA's Personal Satellite Assistants on board the international space station. The domain of PSAs is a dynamic environment where multiple and possibly concurrent problems may develop, and is an area that will benefit from the teamwork made possible by this algorithm.

Future work in this area can take many directions. For instance, we could consider subjective attributes such as qualities of relationships and satisfaction of agents with the task assignment process. Additionally, we can look at the application of autonomy determination in reasoning about teams [3, 16] and its effect on this algorithm.

REFERENCES

[1] K. S. BARBER AND C. MARTIN, *Agent Autonomy: Specification, Measurement, and Dynamic Adjustment*, In Proceedings of the Autonomy Control Software Workshop, Agents '99, May 1–5, Seattle, WA., 1999, pp. 8–15.

[2] K. S. BARBER, A. GOEL, AND C. MARTIN, *Dynamic Adaptive Autonomy in Multi-Agent Systems*, In Journal of Experimental and Theoretical Artificial Intelligence, 12(2), Taylor and Francis, 2000, pp. 129–147.

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

[4] S. BRAINOV, AND H. HEXMOOR, *Quantifying Relative Autonomy in Multiagent Interaction*, In IJCAI-01 Workshop, Autonomy, Delegation, and Control, ACM, 2001.

[5] K. S. BRAINOV AND T. SANDHOLM, *Power, Dependence and Stability in Multiagent Plans.* In Proceedings of AAAI/IAAI 1999, AAAI, 1999, pp. 11–16.

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

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

[8] G. DORAIS, P. BONASSO, D. KORTENKAMP, B. PELL, AND D. SCHRECKENGHOST, *Adjustable Autonomy for Human-Centered Autonomous Systems on Mars*, Presented at Mars Society Conference, AIAA, 1998.

[9] G. DWORKIN, *The Theory and Practice of Autonomy*, Cambridge University Press, 1988.

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

[11] H. HEXMOOR, *Case Studies of Autonomy*, In proceedings of FLAIRS 2000, J. Etherege and B. Manaris, eds., Orlando, FL., AAAI, 2000b, pp. 246–249.

[12] H. Hexmoor, D. Kortenkamp, *Autonomy Control Software*, An introductory article of the special issue of Journal of Experimental and Theoretical Artificial Intelligence, Kluwer, 2000.
[13] S. Irani, S. K. Shukla, R. K. Gupta, *Online strategies for dynamic power management in systems with multiple power-saving states*. ACM Trans. Embedded Comput. Syst. 2(3), ACM, 2003, pp. 325–346.
[14] Mele, *Autonomous Agents: From Self-Control to Autonomy*, Oxford University Press, 1995.
[15] J. Schneewind, *The Invention of Autonomy: A History of Modern Moral Philosophy*, Cambridge Univ. Press, 1997.
[16] M. Tambe, D. Pynadath, C. Chauvat, A. Das, and G. Kaminka, *Adaptive agent architectures for heterogeneous team members*, In Proceedings of the International Conference on Multi-agent Systems (ICMAS 2000), Boston, MA, 2000.

Edited by: Marcin Paprzycki, Niranjan Suri
Received: October 1, 2006
Accepted: December 10, 2006