Multi-agent Swarm Control in Virtual Worlds

by

Andreas Rydell

LIU-IDA/LITH-EX-G--13/053--SE

2014-02-20
Multi-Agent Swarm Control in Virtual Worlds

Andreas Michael Rydell

February 20, 2014

EXAMINER: ALEXANDER KLEINER
Abstract

Flocking birds in a virtual world through adaption of Craig Reynolds, Boids algorithm. In this paper I will demonstrate how I adapted the Boids algorithm to an already existing code to create a flocking behaviour and discuss options to the solutions. I have been given a virtual world with existing birds that is moving around based on random elements that in a way mimics flying.
Contents

1 Introduction 1
   1.1 Problem Description ................................. 1
   1.2 Method and Sources ................................. 2
   1.3 Limitations ........................................... 2
   1.4 Structure of the thesis .............................. 2

2 Related Work 3

3 Boids Flocking 3
   3.1 Theory ................................................ 3
      3.1.1 Cohesion ......................................... 4
      3.1.2 Separation ....................................... 4
      3.1.3 Alignment ........................................ 5
   3.2 Pseudo code .......................................... 5
      3.2.1 Move boids ...................................... 5
      3.2.2 Rule 1 ............................................ 6
      3.2.3 Rule 2 ............................................ 7
      3.2.4 Rule 3 ............................................ 8

4 Adaption 8
   4.0.5 Gametick ........................................... 9
   4.0.6 Rangecheck ......................................... 9
   4.0.7 Flock ................................................ 10
   4.0.8 Update .............................................. 12
   4.0.9 Decisions .......................................... 12
   4.0.10 Steer ............................................... 13
   4.0.11 Separation ........................................ 13
4.0.12 Alignment

4.0.13 Cohesion

5 Description of the Environment

6 Experimental Results

6.1 Bugs and problems

7 Discussion

7.1 Leader/Pseudo-leader

7.2 Information from closest neighbor

7.3 Emotions

7.4 Known flightpaths

8 Conclusions

9 Acknowledgement

10 References
1 Introduction

Out in the world, birds often tend to fly together as a group. This behaviour is called flocking and helps to fend of other birds that would otherwise want to harm the individual bird. Whether the birds are tiny birds like sparrows or if they are bigger birds like magpies, flocking serves a purpose. This thesis will show how agents in a virtual world can flock together and fly as a group. We will achieve this with a set of simple rules explained later in the thesis.

The coding has been done for a company named Code Club AB which has a game called Wurm Online. That means that the virtual world has been provided, including a standard bird that is already in the game. The birds provided are just billboards with a random flight pattern which changes every game tick. A billboard is a 2D model, which in this case is a bird with an animation. They are not very detailed and are ment to fly on very high altitudes. The virtual world is a huge modifiable world where you can cut down trees, build houses and dig tunnels etc.

1.1 Problem Description

The company wishes to have a realistic virtual world with birds that have a natural behaviour and group together. Today the birds have no realistic flight behaviour and fly around without knowledge of each other. Each bird has vectors for a three dimensional velocity that updates itself once per game loop. The problem is to adjust n velocities (v1...vn) to n birds(b1...bn) and make it efficient in calculation speed and making it look natural to the viewer. Since the game is a multiplayer game and the world is huge, a lot of calulations must be made each game loop. Therefor we need to keep the birds velocity calculations to a minumum.

How I shall achieve a realistic flight behaviour and make the birds group together will be explained in the section below.
1.2 Method and Sources

There are two big categories of flocking. There is the V-form and the crowd form. The V-form is a form of leader flocking where birds fly in a pattern, following a leader. This type of flocking can be achieved by using pseudo-leaders [9]. The crowd form is where the whole group flies in a mix and adjusting their flight paths to each other, moving as a whole group and not following anyone. Craig Reynolds gave us an algorithm which mimics this behaviour [2,6]. In this thesis I am focusing on the crowd form, since the agents in the virtual world are similar to crows or magpies.

One of the most common algorithms for flocking is the Boid algorithm [2], created by Craig Reynolds. A lot of articles cite Reynolds work as a source and adapt his algorithms to their need. I too have used this algorithm as a base and adapted it to my needs.

I have studied different approaches for the flocking. Flocking with leader or a pseudo-leader [9] which would work and could function very well in the game. A leader that would lead the flock could be used for the V-form flight pattern, which could be seen by migratory birds. If the groups are large, the pseudo-leader approach would be more preferable even in the crowd form. Details why a pseudo-leader is more preferable in large groups will be explained later on, in the discussion.

The method used when coding was first to understand the code, then to figure out how to adapt the already existing code to fit with the flocking.

1.3 Limitations

Due to the fact that I was adapting code for a company that already had a finished product, I had to narrow down some options I had thought of implementing. Such things as landing in trees, splitting the flock and other things that flocking bird behaviour could imply. This is because it would have taken too much time to implement these actions.

1.4 Structure of the thesis

In this thesis I will show how basic flocking works, based on Craig Reynolds algorithm, Boids[2]. After the basic algorithm I will show how I adapted the algorithm for the virtual world provided. Lastly there will be a summery of the results, errors and bugs I have encountered and then a conclusion.
2 Related Work

One significant inspiration and almost invaluable algorithm is the Boids algorithm. Craig Reynolds developed this algorithm 1986 for his artificial life program called Boids\cite{2} and wrote an article about it\cite{6}. I will later go into detail how the Boid algorithm works.

A lot of other related work that uses flocking birds are based on the Boids algorithm. Therefore, in this thesis, it is considered a keystone, which without, I would not be able to program the birds.

One of those works which have used Craig Reynolds algorithm is Killer Game Programming \cite{3} which in chapter 22, explains flocking as well as giving code examples for flocking rules and other behaviours.

3 Boids Flocking

In this section I will go though the Boids algorithm that I have used and show how it works. A Boids is an agent which can be thought of as a bird. They often look like triangles and I will refer to to the triangles in this thesis as Boids.

3.1 Theory

In the following subsections there will be descriptions of how basic flocking works. There are three rules that Craig Reynold stated in his work and is used in the Boids algorithm \cite{2,6}. The three rules for navigation are:

- A rule to match the speed of the other agents.
- A rule to steer away from other agents that are too close.
- A rule to find the center of the flock.
3.1.1 Cohesion

In the first rule the agents try to steer towards the center of the flock. We can see this in figure 1 where an agent calculates the center of the flock and steers towards it.

![Figure 1: Cohesion](image)

3.1.2 Separation

In the second rule the agents try to avoid collisions with other agents. It tries to separate a bit from the flock. In figure 2 we see that an agent one is getting close to another agent, it checks it tries to avoid collision and steers away from the agent.

![Figure 2: Separation](image)
3.1.3 Alignment

The last rule states that the agent should have the approximate heading and velocity of the flock. In figure 3 we can see an agent adjusting its heading to match an approximate of the group.

![Figure 3: Alignment][2]

3.2 Pseudo code

To get a deeper understanding of how the Boids algorithm works, I will in this section show pseudo code and explain how it works. The pseudocode seen below can be found at [1].

3.2.1 Move boids

In the procedure move_all_boids_to_new_positions() we see that each boid follow three rules. A loop will run these rules for all boids. The results of the rules are set to three vectors, v1, v2 and v3. These vectors plus the selected boids' velocity add up to a new velocity and that velocity is added to its current position.
Algorithm 1 Move all Boids to new positions

procedure MOVE_ALL_BOIDS_TO_NEW_POSITIONS
    Vector v1,v2,v3
    Boid b
    for each Boid b do
        v1 = rule1(b)
        v2 = rule2(b)
        v3 = rule3(b)
        b.velocity = b.velocity + v1 + v2 + v3
        b.position = b.position + b.velocity
    end for
end procedure

3.2.2 Rule 1

This rule states that each boid should fly towards the center of the flock of boids. The procedure will take a boid as input and will be called bj as short for "boid number J". A vector for the "percieved center" of boidJ created, called pcJ. The full list of boids will be looped and the current boid is named b. As long as the boid bj, that is not the same boid as boid b, we will add bs' position to pcJ.

When we have looped over all the boids, we divide pcJ with the number of boids minus one to get an average. We return 1 % of the way bj should move by dividing the vector towards the center by 100. This is modifiable if the boid should move faster or slower.

Algorithm 2 Rule 1

procedure RULE1(boid bj)
    Vector pcJ
    for each Boid b do
        if b != bj then
            pcJ = pcJ + b.position
        end if
    end for
    pcJ = pcJ / N-1
    return (pcJ - bj.position) / 100
end procedure
3.2.3 Rule 2

In the second rule each boid try to avoid collision with each other. Here we create a vector $c$ and initialize it to zero to get a vector "away" from the boids that are too close. We go through each boid $b$ and make sure it is not the same boid as the input boid $b_J$. If the absolute value of the two birds positions are less than a given value (in this case 100 units), the difference in the positions are subtracted from $c$. After the loop is done we will have a vector pointing away from all boids that are too close to $b_J$.

```
Algorithm 3 Rule 2

procedure rule2(boid bJ)
    Vector $c = 0$
    for each Boid $b$ do
        if $b != b_J$ then
            if abs($b$.position - $b_J$.position) < 100 then
                $c = c - (b$.position - $b_J$.position)
            end if
        end if
    end for
    return $c$
end procedure
```
3.2.4 Rule 3

The final rule will try to match the velocity of the other boids. A vector for the input boid $bJ$’s ”perceived velocity” will be created and named $pcJ$. As we go through all other boids in the loop we check that the loops’ boid $b$ is not the same as $bJ$. As long as they do not match, we update $pvJ$ with $b$’s velocity. After the loop we divide $pvJ$ with the number of boids minus one to get an average velocity of the group. We return a small portion (in this case, an eight) of the speed based on the average minus the current boids’ velocity.

Algorithm 4 Rule 3

```plaintext
procedure rule3(boid bJ)
    Vector pvJ
    for each Boid b do
        if b != bJ then
            pvJ = pvJ + b.velocity
        end if
    end for
    pvJ = pvJ / N-1
    return (pvJ - bJ.velocity) / 8
end procedure
```

4 Adaption

The following section of the thesis is my adaption of the pseudo code with added decision making for migratory birds. It uses the same basic functions as the Boids algorithm but is adapted for the already existing birds in the game. They still have the three behaviour rules, but to find each other, a range check to calculate the distance between the different birds is needed. A couple of thing to point out is that the variable $z$ used in all procedures are the height variable and the variable `thisBird` is the bird currently running the `gametick()`.
4.0.5 Gametick

All birds have their own game tick which updates their position. The update will give a small change to the agents’ velocity in the axis and new positions are updated according to the new velocity. All birds should not be able to flock and to achieve this, it is decided upon creation if the bird can flock or not. In the case that the bird is not able to flock it uses the basic code provided by the company, which makes the bird have a random flight pattern. Otherwise they use the flocking procedures I have written. All particles in Wurm Online have a gametick procedure (specific for each particle) and once per update in the game each particle will run their own gametick(). Since gametick() is the first chance to chose if each bird should use the flocking procedures or if they should fly around randomly, we can encapsulate this choise in an if, else if expression. If they are able to flock, they will go forth and search for nearby birds.

In the case where the bird is able to flock, we first search an area around us for other birds and get an array with who is close enough. Then we apply the rules as explained earlier in the thesis and lastly we update their new positions with the flocking rules.

Algorithm 5 Gametick

procedure GAME_TICK
    if !flockable then
        x += random
        y += random
        z += random
        Point3d velocity = new Point3d(x,y,z)
        currentPosition = currentPosition + velocity
        currentRotation += random
    else if flockable then
        Array nearbyBirds = rangeCheck(thisBird)
        flock(nearbyBirds)
        update()
    end if
end procedure

4.0.6 Rangecheck

To be able to find the other birds, I return an array with all the birds within an area (radius) of 100 meters. Depending on the virtual world and how the coordinates work in it, change the distance to a suitable number. If the current bird (bird b) is the same bird as the iteration bird in the for-loop, skip it. This is because we do not want to take ourselves into account when we are calculating speed and center of the flock etc.
Algorithm 6 Rangecheck

procedure RANGECHECK(Bird b)
    Array nearBirdFlock
    birdsMaxRadius = 100

    for all Bird bi do
        if bi != b && b.distance(bi) < birdsMaxRadius then
            nearBirdFlock.add(bi)
        end if
    end for

    return nearBirdFlock
end procedure

4.0.7 Flock

If there are birds to flock with within range, the different rule procedures are executed. Here we create a vector type, called Point3d that can take three values for x, y and z. These are named separation, alignment and cohesion. Each of the Point3d vectors are assigned values for each of the rules called separate(), align() and cohesion(). All of the rules are added to another Point3d called acceleration, which will be used later in update(). If there are no birds close enough to flock with the bird should fly around as normal with a random flight pattern.
Algorithm 7 Flock

procedure FLOCK(Array theFlock)
    if theFlock.size() > 0 then
        Point3d separation = new Point3d();
        Point3d alignment = new Point3d();
        Point3d cohesion = new Point3d();

        separation = separate(theFlock);
        alignment = align(theFlock);
        cohesion = cohesion(theFlock);

        acceleration += separation + alignment + cohesion
    else
        x += random
        y += random
        z += random

        Point3d velocity = new Point3d(x,y,z)
        currentPosition = currentPosition + velocity
        currentRotation += random
    end if
end procedure
4.0.8 Update

This is where the flocking bird updates its position. It uses the accumulated value from the different rules that were put into the variable \textit{acceleration} and adds it to the birds’ velocity. After that decisions can be made if it should change its mind and go somewhere else, in case that it is winter and it should head south. The current angle for the bird needs to be calculated and later used to update the birds’ animated rotation. This is needed to make sure the birds do not fly backwards. Lastly we set the \textit{acceleration} to zero until next \textit{flock()} updates it with new values.

\begin{algorithm}
\begin{procedure}[H]
    \caption{Update}
    \procedure{UPDATE}
    \State velocity.add(acceleration)
    \State decisions()
    \State $\theta = \text{calculateAngle()}$
    \State currentPosition = currentPosition + velocity
    \State currentRotation = calculateRotation($\theta$, currentRotation)
    \State acceleration.set(0,0,0)
\end{procedure}
\end{algorithm}

4.0.9 Decisions

If enabled, the birds can take decisions. Here is where those decisions are made. In this case, the birds fly south during the cold periods and north when it is getting warmer again. This is achieved by setting the birds velocity in the y-axis to either positive or negative. This will make the bird to fly north or south.

\begin{algorithm}
\begin{procedure}[H]
    \caption{Decisions}
    \procedure{DECISIONS}
    \If{Autumn} \State velocity.y = abs(velocity.y) \EndIf
    \ElseIf{Spring} \State velocity.y = abs(velocity.y) * -1 \EndIf
\end{procedure}
\end{algorithm}
4.0.10 Steer

Gives a correction to where the bird should steer, given a target location.

Algorithm 10 Steer

procedure steer(Point3d target)
    Point3d desired = target - location
    desired -= velocity
    return desired
end procedure

4.0.11 Separation

If the birds get too close to another bird, it should steer away from that bird. The separation can be modified though desiredSeparation. A loop will check the distance between the current bird (thisBird) and b. If they are too close we calculate a vector (difference) away from b to be used later to correct our flight path. We divide difference with the distance to modify how important it is to steer away from b. For example, if the current bird thisBird is only one unit away from bird bi and 4 units away from bird bj, then it is more important to steer away from bi. A counter is needed if we are too close to multiple birds. We divide the away vectors accumulated in steer with the number of birds that it is too close to. Steer is multiplied by the maxspeed for a bird and then subtracted with the velocity of the bird to make sure it will steer away from the birds (if the velocity would have been greater than steer).
Algorithm 11 Separation

procedure separate(Array theFlock)
    desiredSeparation = 5
    Point3d steer
    counter = 0

    for all Birds b in theFlock do
        distance = thisBird.location - b.location
        if distance > 0 && distance < desiredSeparation then
            Point3d difference = thisBird.location - b.location
            difference /= distance
            steer.add(difference)
            counter++
        end if
    end for

    if counter > 0 then
        steer /= counter
        steer *= maxspeed
        steer.sub(velocity)
    end if

    return steer
end procedure
4.0.12 Alignment

In this procedure we calculate the average velocity of the flock to make all the birds fly at the same speed or catch up to the flock. In this loop we check if the bird(s) \( b \) is close enough for \( \text{thisBird} \) through \( \text{neighbordistance} \) which can be modified if needed. If the birds are close enough we add \( b \)'s velocity to \( \text{sum} \) and increase the \( \text{counter} \). As long as \( \text{thisBird} \) had other birds close enough to itself, we divide \( \text{sum} \) with the \( \text{counter} \), then multiply with the \( \text{maxspeed} \) and lastly subtract \( \text{thisBird} \)'s velocity to make sure it will make use of \( \text{sum} \).

**Algorithm 12 Alignment**

```plaintext
procedure align(Array theFlock)

    neighbordistance = 80
    counter = 0
    Point3d sum
    for all Bird b in theFlock do
        distance = thisBird.location - b.location
        if distance > 0 && distance < neighbordistance then
            sum.add(b.velocity)
            counter++
        end if
    end for
    if counter > 0 then
        sum /= counter
        sum *= maxspeed
        sum.sub(thisBird.velocity) return sum
    end if
    return null
end procedure
```
4.0.13 Cohesion

Calculate the center of the flock and steer towards it. This procedure is almost the same as the one above, except it calculate the center of the flock with help of the locations of the birds. If there were birds close enough so the counter is greater than zero, we will return the result of another procedure previously described (Algorithm 10), called `steer(sum`).

**Algorithm 13 Cohesion**

```plaintext
procedure cohesion(Array theFlock)
    neighbordistance = 80
    counter = 0
    Point3d sum
    for all Bird b in theFlock do
        distance = thisBird.location - b.location
        if distance > 0 && distance < neighbordistance then
            sum.add(b.location)
            counter++
        end if
    end for
    if counter > 0 then
        sum /= counter
        return steer(sum)
    end if
    return null
end procedure
```
5 Description of the Environment

In the game, there is a world that can be modified by the players. Trees can be cut down, houses built and tunnels dug. The birds do not have any collision detection yet, because until today, they have only been seen on great heights and are just billboards. With the implementation of the flocking, the developers can give the birds a collision detection function if they want the birds to fly on lower altitudes. Lower altitudes will cause the birds to fly close to or into/through objects. To make sure they do not fly into a cliff side, tree or other objects, a collision detection function will be needed. On the other hand, collision detection can be time consuming for the birds, because the world is constantly changing, no static map can be used to know where the obstacles are. This implies that each bird will have to ”scan” their environment and adapt their flight path to it. A solution to lower the calculations needed can be achieved by using a pseudo-leader [9] which can have a collision detection function and all the other birds listen to its information.

For now, the birds do not have any enemies and do not need any instance of fear level. This is also another thing that could be implemented in addition to the basic flocking behaviour. I will talk more about this in the conclusion.

6 Experimental Results

The experiment has been done with different range of birds, from 4 up to 55 birds. This was made with a command to spawn the birds and that command looped for a given number of times. There was also a random element which told the individual bird if they should flock or not. This was only used with large number of birds spawning. This made some birds able to ”see” the other flockable birds and group together with them, while the others, non flockable birds flew away.

In this graph below, we can see four birds spawning at the lower left corner and adjusting their flight path to each other. The flight adjustments are a result of the flocking rules and make the birds fly together as a group.
In this next figure, 19 birds have grouped and adjust their flight paths to the rest of the group. They follow each other rather well and when some birds turn, the others follow. We can see in the middle of the graph that some birds start to turn and the others soon follow.
6.1 Bugs and problems

While programming, a lot of problems occurred. Most of them were problems that could have been avoided if the already existing code had been studied more. Others were just plain logical errors like using the wrong variable.

One example is the gameTick() function above. I updated all the birds positions once for each bird in the beginning. So each bird got a gameTick() for each other bird in the array of birds. This caused all the birds to disappear from the screen within a second due to multiple updates each gameTick(). This problem could have been avoided if I had been more cautious.

Another thing that was difficult to implement was if the birds were too old (all birds are removed after a certain set of animations), they should be removed. Since all the birds that were flockable are stored in an array, I had to remove them from it so the array would not point to a null bird. To understand the solution I have to explain how the existing code removed the birds. In the gameTick(), the birds had a life counter as well. When the bird was about to die, the gameTick() returned a false Boolean. So the solution was to check if the gameTick() would return false before it actually returned it and then remove the bird from the array before returning false.

Once the flocking started to work, I found another problem I would like to call the "circular problem". The birds flew in a flock, but some birds flew backwards and in a radius from the center of the flock. They were more or less dragged by the flock even though they tried to fly away from the group.
This was because the rotation was not dependent on the bird's vector, but was rather set to a random value. A function to calculate the bird flight vector was implemented and the result was compared with the current angle. After the implementation the birds rotated towards the group and the flocking behaviour looked nice.

7 Discussion

In this section, things that were not implemented due to time shortage will be discussed.

7.1 Leader/Pseudo-leader

As already mentioned earlier in the report, a pseudo-leader would be more efficient, than the current flocking behaviour, if the group is large. Any agent could be the pseudo-leader and the other agents would listen mainly on the chosen agent and avoid collision with the others [9]. The agents are not informed by all the other agents, just the ones closest to it. Since computations of the average in a large group would be heavy and might cause a MMORPG (Massive Multiplayer Online Roll-Playing Game) to lag due to the heavy computation, the pseudo-leader approach would be the approach most suitable.

In [8] they use a leader which connects to a few birds which in turn connects
to other birds. This causes a chain of information each bird can use to update their own flight path. This lowers the number of calculations needed to achieve flocking.

7.2 Information from closest neighbor

In smaller groups where no leader is needed and we want to lower the calculations for the agents, we can use the information from the closest neighbors. Instead of each agent checking all other agents and adjusting its flight path, it will just check the closest neighbors and adjust to them [7].

7.3 Emotions

As described earlier, a level of fear can be introduced together with collision detection. This way the birds can be scared of the players and split into smaller groups when a player approaches and avoid flying into trees and other obstacles.

The fear or rather use of emotions could be used if predator birds are implemented. The flocking birds should not be as frightened if they are a big group and be more frightened if split into smaller groups. The predators could also use a "scent" to track the prey as described in the article Fear and the Behaviour of Virtual Flocking Animals [10], where emotions trigger pheromones to be emitted and be sensed by other animals. This would cause the animals, or in this case, the birds to be traceable with scents by either predators on land (if the birds land) or by other birds.

A resting state can be implemented together with a tiredness level which will force the bird to land and rest/sleep before continuing.

7.4 Known flightpaths

In the article Wayfinding strategies and behaviours in large virtual worlds [4] they speak of a stimuli for the program, which could be adapted to the birds. If they would get lost or separated from the group, they could fly by the memories from before and hopefully find their group. The known locations in the bird memories would trigger the stimuli and give an indication if it is on the right path. This behaviour would be suitable for migratory birds that each winter flies south to a specific point and then flies back when the spring comes.

Another use of known flight paths is if the birds would know the locations of all cities in the virtual world. In the article Flocking techniques to naturally support navigation in large and open virtual worlds [5] they discuss the use of flocks to guide the player towards towns/cities. This would be a great use of the birds in the game Wurm Online, where players can wander forests and easily get lost. Since the game is set in medieval time, the birds could fly towards the cities and circle around them while "looking for food". Together with the use
of emotions then could land and feed, while being aware of players coming to
close and fly off if needed.

8 Conclusions

In this paper, I have shown you how to create flocking birds by using and
adapting the Boids algorithm to a virtual world. The birds have a basic flock-
ing behaviour and can group up in a natural way and can make decisions on
where to fly depending on the season.

Given more time, more actions and behaviours could have been implemented.
Due to various reasons I could not implement all the functionality I had wanted
to implement. There is still much more things that could be implemented, such
as emotions, leaders, landing and sitting still.

9 Acknowledgement

I would like to thank Code Club AB for giving me this opportunity and helping
me whenever I got stuck at problems. I would also like to thank Alexander
Kleiner for being a helpful and great teacher and examiner.
10 References


På svenska

Detta dokument hålls tillgängligt på Internet – eller dess framtida ersättare – under en längre tid från publiceringsdatum under förutsättning att inga extraordinära omständigheter uppstår.

Tillgång till dokumentet innebär tillstånd för var och en att läsa, ladda ner, skriva ut enstaka kopior för enskilt bruk och att använda det oförändrat för ickekommersiell forskning och för undervisning. Överföring av upphovsrätten vid en senare tidpunkt kan inte upphäva detta tillstånd. All annan användning av dokumentet kräver upphovsmannens medgivande. För att garantera äktheten, säkerheten och tillgängligheten finns det lösningar av teknisk och administrativ art.

Upphovsmannens ideella rätt innefattar rätt att bli nämnd som upphovsman i den omfattning som god sed kräver vid användning av dokumentet på ovan beskrivna sätt samt skydd mot att dokumentet ändras eller presenteras i sådan form eller i sådant sammanhang som är kränkande för upphovsmannens litterära eller konstnärliga anseende eller egenart.

För ytterligare information om Linköping University Electronic Press se förlagets hemsida http://www.ep.liu.se/

In English

The publishers will keep this document online on the Internet - or its possible replacement - for a considerable time from the date of publication barring exceptional circumstances.

The online availability of the document implies a permanent permission for anyone to read, to download, to print out single copies for your own use and to use it unchanged for any non-commercial research and educational purpose. Subsequent transfers of copyright cannot revoke this permission. All other uses of the document are conditional on the consent of the copyright owner. The publisher has taken technical and administrative measures to assure authenticity, security and accessibility.

According to intellectual property law the author has the right to be mentioned when his/her work is accessed as described above and to be protected against infringement.

For additional information about the Linköping University Electronic Press and its procedures for publication and for assurance of document integrity, please refer to its WWW home page: http://www.ep.liu.se/

© Andreas Rydell