Proceedings

AMAPS2011

Algolog Multi-Agent Programming Seminar 2011

2 December 2011

DTU Informatics

Technical University of Denmark

.

Editor

Jørgen Villadsen

Algolog refers to the Algorithms and Logic section at DTU Informatics.

The focus of the seminar is on tools and techniques for programming multi-agent systems.

Key topics are multi-agent programming competitions and genuine multi-agent challenges such as organization, communication and agreement as well as algorithms and logics for multi-agent systems.

Contents

"I wouldn't have thought of it myself"

Emergence and unexpected intelligence in theater performances designed as self-organizing critical systems

by Troels Christian Jakobsen, artistic director at Teater 770° Celsius

Realizing the systemic nature of theater, changed the paradigm for my work. Most theater is designed and governed by the idea of the newtonian machine, attempting to reproduce the exact same procedure and the exact same results at every performance. Cause and effect in one linear chain of events. If you realize that a theater performance in reality is a self-organizing critical (SOC) system, and you begin to design them governed by this idea, a whole new world opens up.

In this text I will seek to cover my road of discovery following a biographical path, the perspectives I see in further exploring the insights from SOC-systems, both artistically, philosophically and politically - and eventually connect it all to one of my new heroes, Alan Turing, who realized the simple, but amazing fact, that mistakes leads to intelligence. A realization which could stand as a credo for Teater 770° Celsius.

And before I dive into that, I want to remind you of a passage Alan Turing wrote, shortly before his untimely death in 1954:

Messages from an unseen world:
Hyperboloids of wondrous Light
Rolling for aye through Space and Time
Harbour those Waves which somehow Might
Play out God's holy pantomime.

In which he shows, that apart from a scientific mind, he was also capable of poetry at an almost Shakespearean level - just as a reminder for everyone, scientist or artist, not to forget the other side of the moon.

In 2003 I was going through a personal crisis; my passion for theater was threatened by the fact, that I got really, really bored by most theater. I began a search to find out, what was wrong; with me or the theater. My first clue was, that I liked the moments in performances, where things go wrong, where the plans fail and the actors are left in the unknown. I analyzed why these moments appealed to me - it was the question "what happens now?", which excited me. That exact moment when nobody in the room will or can know what happens right now or in the moment right after. This made me realize, that most theater makes a huge mistake by designing their performances as newtonian machines, which run like clockworks; where they plan with the ambition of making the exact same thing happen night after night, performance after performance. Their ambition is to eliminate the mistake and the unexpected, understandably enough in the effort to strive for perfection, but they end up eliminating the basic appeal of drama; actions unfolding in time; which is expressed in questions like: "What will happen next?", "Who will succeed, who will fail?". At the most primary level we watch and follow a dramatic performance to see what happens next. In theater, where this happens in a space which contains both audience and performers, we should almost be able to feel this uncertainty in our bodies, if it really exists in the bodies the performers, and in the space between performers and audience. In most performances, it has sadly been eliminated in all, except as pretense and plot.

I had my insights into dramaturgy, directing and acting to help me in my initial exploration of the problem; the basic mechanics of how drama works. My first steps were to redefine those insights. Instead of looking at dramaturgy as models, static form-descriptions of "how an ideal play should look", I began to search for the principles behind those forms. The first biologists described nature as forms - "this is how an xx-mushroom should look" - but as anyone knows who has been out hunting mushrooms in nature, the exact form of the real mushrooms you find often differ a lot from the description in the book. As science developed, instruments became better and curiosity sharpened, science found the DNA, which give rise to those forms. Because I was unsatisfied with the ideal model-descriptions of drama, not the particular models in themselves, as they can have many good point to them, but the whole idea of trying to catch the truth with a static model, as it seemed a part of the striving for perfection, which made theater performances boring in the first place. So I began to look for the DNA of drama; those very basic principles which forever tangle and dance which each other, and in their interplay create the forms, we see and experience as reality.

In the design of drama I have found one primary and four basic principles, which govern the development of well-functioning play, films and tv-series.

1. The principle of conflict is the primary principle. Without conflict there is no drama. Conflict creates both uncertainty of what will happen next and it generates the development of story and plot. The basic conflict of a drama is the defining aspect of a drama as a system.

2. The principle of character is tightly connected to the primary principles, because it is characters with an objective, which will be the agents of conflict. Therefore characters with well-designed objectives will be of more importance than a clever plot - as the clever plot will arise from the characters.

3. The principles of unity dates back to Aristotle, but in his days, it meant more or less that the action, the time and the location should be contained in one single unit – for example a day in a courthouse, where a man is falsely convicted - whereas I understand it as a principle for designing a system, where all elements obviously have to be connected as much as possible. I think of it as feedback-connections - the more all elements of a drama deliver feedback to each other, the more complex and fulfilling it will be. Both in terms of feeling like a real, convincing world and in terms of being.

4. The principle of mystery deals with something very arcane. Drama developed from the mystery plays of ancient Greece which dealt with the great mysteries of life, death and other paradoxical truths. The point of enacting a mystery is that the insight into a mystery is something which cannot be told, explained or delivered to others, except by action and self-experience. By watching a play we can get so close to the mystery as possible without having to go through the real life version. So when designing a drama, you have to be aware of what fundamental mystery it deals with. You'll recognize a mystery on its paradoxical nature; that it will contain two great truths which negates each other.

5. The principle of tonality deals with an often overlooked aspect in the design of drama; how comedy and tragedy acts like the two complementary tonalities, almost exactly like B minor and B major does in music, and as such is fundamental to the construction and design of a piece of drama. This means you have to make the initial decision for a piece, if it basically plays in the tonality of comedy or tragedy. Design-wise will this mean that the first and last tone of your piece has to be sounded in that tonality.

All these principles was stuff I had been working on since around 2000, but more and more dedicated and thoroughly as I began my quest for the answer about boring theater. For a more detailed description of the principles visit:

http://thatdramathing.blogspot.com/search/label/major%20principle

It was only when I read "How Nature Works" by Danish scientist Per Bak and later a well-written popular science book, "Ubiquity" by Mark Buchanan, that I found a theoretical framework for my new understanding of theater. This lead me to develop a new method for the design of theater performances as self-organizing critical systems, where meaning and intelligence emerge unexpectedly.

"Ubiquity" begins with a great question: Why did First World War begin? It tells the story of how Prince Ferdinand was driving through Sarajevo and by chance took a wrong turn, and how by chance a young anarchist was on that wrong road, and how he exactly that day was carrying a gun, so he in the spur of the moment could shoot Prince Ferdinand - which was the incident that set off world war one. But is this then the real explanation of the cause of world war one? No, and there have been countless books by historians, where they each claim to have found the real reason, but they can't all be right? No, because as Per Bak and others discovered, in complex systems where many units interact across the system, the newtonian understanding of cause and effect seems to become irrelevant, when you look at the whole system. Locally cause and effect is still very valid indeed, but systemwide something else is going on.

I could immediately connect the descriptions of Per Bak's sandpile-experiment and Mark Buchanan's real life examples of SOC-systems at work, with my experience of drama and theater. In my world, the theater world, we often talk of the magic moment, the magic night, when everything in a performance happens in an extraordinary way, and gives a new level or new dimension of meaning to the performance - and one of the traits of these moments and nights, is that they are unpredictable and unrepeatable - as is the events in a SOC-system. No matter how hard actors and performers try, they can never repeat the exact same performance. The theater performance is obviously a SOC-system. So my question became, what happens if instead of trying to eliminate the unexpected, we begin to design the performance with the aim of allowing the unexpected? Couldn't we perhaps achieve more 'magic moments'? If instead of focusing on controlling every moment of the performance, we began to create a framework, a system, where within its borders, there would be real feedback, real interaction between the elements of the performance, allowing them to influence each other, allowing them to change course, attitude, color or what aspects they might have, under the influence of the events.

In 2007 I made the first full-scale performance with 5 actors, trying to use SOC-principles. Before that I had experimented in a workshop environment, where I had experienced difficulties, as the actors found it difficult not to know exactly where the performance should end, what the meaning and artistic message was - they felt scared and wanted answers; answers I couldn't or wouldn't give them - because the method was the message: That we live in an unpredictable world, that theater should mirror that reality by being unpredictable itself. But in 2007 everything succeeded. I began the rehearsals full of fear, having spent the previous 3 months avoiding planning too much and feeling anxious because I couldn't plan and have that security.

I began rehearsals only with a basic conflict for the performance, a basic conflict for each character, and the method: which in all its simplicity is to constantly work at the local level, the interaction between characters in specific situations within the framework of the basic conflict. What does each character want, how do they try to achieve that objective at the local level - as repeated improvisations, where we let each and every improvisation be just another experience of how these characters could act and behave, an experience we of course evaluate, but never approve or disapprove of as a final and correct version. In this way experience begins to accumulate; almost as an AI developing by making its own experiences, the actors gradually build an operative system of behavioral and technical knowledge of both their characters personality and of how to play it.

While the actors are focused on the local, contained situations, I begin to build the performance as a whole, by discovering the natural patterns, which arise as we improvise with the situations, and bringing them together in a framework to build a gradual escalation in the basic conflict, often with a kind of gateway situations, where the characters will be forced to develop and take new steps - which so to speak bring them to the next level of basic conflict. So far this is the only element of what in roleplaying world is referred to as 'railroading', meaning; artificially keeping the players on the right track. But the way to build an organic, less artificial gateway, is to allow it to be open-ended.

It should only be a gate-way, leading from one level to anywhere on the next, not an escalator which brings them to an exact point at the next level. Each level is what we call 'an act' in drama. I normally operate with four acts, but a drama could have any number of acts. What are important to understand about the function of the act are only two things; they establish a rhythm throughout the drama and, through a tightening of the basic conflict, they bring the characters to a new level of that conflict. You could think of each act as a sub-system, or as sandboxes within a larger sandbox-system. In each act or sandbox different rules apply to the world of that sandbox and the players are relatively free to explore it, until they meet and enter a new gateway. Then they are forced into some kind of confrontation with the world, where those rules change exactly because the conflict goes to a new level.

After 3 weeks of rehearsals we played our first performance for a test audience, and even though it was far from 'finished', the audience experienced it as very finished, very fulfilling - exactly because more 'magic moments' happened, because the characters seemed alive, because the actors were not pretending, but were in reality acting in the word's fundamental meaning - to act, to do something - because there were no pre-made arrangements, no plans, except to follow the basic conflict of both the performance and the characters. At the same time, the performance seemed to uphold the exact rules the models advocate, even though we were not trying to uphold them. The performance took a satisfying, meaningful form, because we followed the basic principles of drama in the construction of our performance-system. And as a scriptwriter and director, I often found myself thinking that the words that came out the actors' mouths, the way they played the situation, were much better than anything I could have written, or at least it was much more effortless, organically and convincing than if I had written the scenes and directed them in the traditional fashion.

So far I have made three very successful performances following this method, which has now been named the IRL-method, because a lot of the elements in the method is basically about doing as *in real life*, not as much in style, as in the mental approach to the work. Reality is as it is, no matter how we want it to be - the unexpected will always happen, mistakes and errors pop up, and instead of fighting them and reality, we take advantage of them. And when you begin to work like this, you'll experience - as you let go of the fear of not controlling things and you open your mind to the unexpected - that you will get more inspiration, will discover more things and best of all, be constantly surprised by yourself and others, because the things, you couldn't have thought of yourself, they happen. And this is where Alan Turing enters the picture, in more ways than one.

When Alan Turing began to think about the human consciousness, and asked himself the question, how machines could acquire intelligence? Well, actually he used the idea of “thinking machines”, as a gateway to begin understanding what human consciousness really is. Mainly because a friend of his had died, and he was wondering what happened to this friend’s beautiful mind after death, and this let him to the bigger question; what is consciousness? The machines were to be a kind of simple mechanisms, that he could experiment with, as an approach to an understanding of consciousness. And in this work he stumbled on the very basic insight, that the machines should be allowed to make mistakes, in order to become intelligent. If they were only programmed to do the right thing, they could only do that, and nothing else. The difference between a newtonian and an organic machine is the possibility of the mistake, and of learning from the mistake, or integrating the mistake as a purposeful part of the organism. This is how we program our theater performances. It is extremely demanding and often difficult, because we humans tend to dislike mistakes, and we prefer to be perfect, but the more we work with this method, the more we realize, that real intelligence happens as emergence in a complex system in a critical state. I think Alan Turing would have been excited about SOC-systems, and that in it, he would see yet another piece of the puzzle of how consciousness and intelligence work. To me Alan Turing is a great, modern hero. The first, apart from Bohr and his likeminded scientists, who really allowed himself to let go of the newtonian worldview. From my own experience, I know it is difficult and also a bit scary. Maybe this is also, why these insights of how the world works, haven’t become everyman’s knowledge - and even why still to this day, you can still find people, who think we can predict earthquakes, the stockmarket and other SOC-systems.

In a broader perspective, not only the theater worlds, it seems to me that all systems, which behave as SOC-systems, especially the human social systems, could benefit from a re-design based on scientific insights. I believe, and for me as a non-scientist, it can only be a belief, that a dedicated exploration of how to design different systems, could be of immense help for humanity. I know that when I combined it with my area of expertise, it yielded surprisingly strong results - and I look very much forward to emerging one of my next projects, a performance inspired by Alan Turing.

Announcement

Alan Turing Year 2012

The Alan Turing Year 2012 will be a celebration of the life and scientific influence of Alan Turing on the occasion of the centenary of his birth on 23 June 1912.

Turing had an important influence on computing, computer science, artificial intelligence, developmental biology, and the mathematical theory of computability and made important contributions to code-breaking during the Second World War.

Source: http://en.wikipedia.org/wiki/Alan_Turing_Year

See also http://www.turingcentenary.eu/

AMAPS2011 Organization Committee

Google AI Challenge 2011: Ants

Kasper Hjorth Holdum, Christian Kaysø-Rørdam, and Christopher Østergaard de Haas

DTU Informatics

Abstract. This year's Google AI Challenge is a multiplayer game on a 2-dimensional map. All players are given a short time frame each turn to do calculation and issue orders to all owned units. This paper identifies the key problems for a well performing AI system. Solutions to each problem are proposed and discussed.

1 Introduction

Google AI Challenge is an open annual contest, hosted on http://aichallenge.org/. When the annual challenge is announced contestants have a few months to create, modify and perfect a bot. A bot is an AI system to control your ants. During the contest, contestants can upload their bot to the contest servers where they will play against other bots and be given a running rank according to performance. The contest supports any language that can write to the standard I/O. These conditions make a perfect environment for a large varity of people, strategies and skill levels.

We will take a look at which solutions we have found for each of the sub problems.

2 Problem Description

This years Google AI Challenge is called Ants, which is a turn-based AI game. Each turn an ant may move up, down, left or right on the grid-based map. The map is shaped like torus, meaning there are no edges on the map. Maps may include obstacles, which cannot be directly passed. Each player has one or more hills, on which new ants spawn. Additional ants are acquired by collecting food, which is randomly spawned on the map. Ants can attack enemy ants or hills, where only attacking a hill rewards points. The visibility is limited for each player. Each ant provides vision for the player in a constant radius around the ant. There exists several conditions in which the game will end. A few noticable of these are when a certain turn limit is hit, or when all enemy ants are eliminated. The game is won by the player who gathered the most points.

Each turn every bot is given 500ms to do all calculations and issue orders for all its ants. The time limit imposed on all bots greatly alters the problem from finding optimal solutions for each sub problem, which will likely take much longer than the given time limit. Instead we will have to find approximate solutions that are as optimal as possible while staying within the time limit. The most significant problems for the bot are reproduction, exploration, attacking enemy ants/hills and defending hills.

3 Tools

During development of our bot we make use of several tools to assist us in debugging and estimating the quality of the current algorithms. The host of the challenge provides two tools: An application that will run a single match with test bots and/or bots given as input and a visualizer that will display the match turn by turn after it has ended. While both of these are a tremendous help, they do restrict the development a bit. It is not possible to start a bot in our IDE and expect to debug it while a match is playing. Neither is it possible to do any custom visualization. Furthermore, since it is only possible to play against simple test bots and your own bots it is hard to estimate the quality of the gameplay delivered by the bot.

It is possible to upload a bot to the contest page after which it will start to play games against the other uploaded bots. However, due to the large amount of contestants and low amount of server capacity this is a very slow process. At the time of this writing each bot plays about 3-4 games every day. Several people decided to create their own servers and ranking systems where you can connect via TCP. The advantage of these servers are that each bot owner uses the processing power of the computer from which the bot is connected. Since the server no longer has to use it's CPU cores for the bots, it can now host a much larger amount of games. So by utilizing the TCP servers it is possible to play games against real players much faster.

3.1 HPA Visualizer

We implemented the hierarchyal pathfinding algorithm [BMS2004] to improve the running time of path finding. HPA will be covered under in section 4.5. Due to the involved nature of the implementation of HPA, we decided to make a visualizer for debugging. It shows us the map split into the zones and the transition points for each of the zones, and we may ask for a path from one point to another and have it shown in the visualizer as well. Being able to see the actual path generated is a big help, as the path is likely to not be optimal and being able to clearly see what causes this sub optimality is useful when we will design the path smoothing for HPA. Path smoothing will be covered in section 4.5

A screenshot of the visualizer can be seen in the appendix, fig 1.

4 Solution

We have identified several key problems in the challenge:

– Spreading out ants,
– finding and eating food,
– attacking enemy hills,
– and defending own hills.

We will discuss each of these problems and our proposed solution in the following sections. Since each turn is limited in time to 500 ms (subject to change) all algorithms and solutions must be designed with this limit in mind. For more complex solutions we have been forced to implement advanced data structures to minimize the running time. The data structures are briefly mentioned section 4.5.

4.1 Spreading Out Ants

The final objective of the game is to eliminate as many enemy hills as possible. To achieve this goal we need to know the locations of the hills and have a large number of ants to attack them. Both require that we explore the map. Since food randomly spawns on the entire map (taking symmetry into account) we would optimally want to have sight of every single field on the map, and minimize the distance from our closests ant to each field. This is not possible, at least in the early game, due to lack of ants.

We have tried two approaches. One that focused on which part of the map that is not currently visible to us, and the other which ignores visibility and simply uses a simple model of representing ants as magnetic entities that repel each other.

Visibility Spread Out Our first approach is to send every ant to the spot closest to that ant which is not visible. This approach requires information about which fields are visible as well as a method to find the nearest invisible field and a method to find invisible fields in a certain radius of a field. These two operations are provided by the Kd-tree and covered in section 4.5.

The basic algorithm iterates over all the available ants (ants that has not been occupied elsewhere), finds the closest invisible spot, tries to find a path to this spot and then issues an order for the ant to move there.

Algorithm 1 Visibility Spread Out

```
for $i = 1 \rightarrow count(availableAnts)$ do
    $currentAnt \leftarrow availableAnts[i]$
    $closestInvisibleSpot \leftarrow findClosestInvisSpotTo(currentAnt)$
    $pathToGoal \leftarrow findPath(currentAnt, closestInvisibleSpot)$
    if pathToGoal != null then
        $moveAnt(currentAnt, pathToGoal.nextStep)$
    end if
end for
```

The algorithm has been modified afterwards to improve several aspects. First of all, we have added caching so, that if an ant is still heading for the same location, then the whole path should not be recalculated. This issue is trickier than first expected though, since parts of the map might be discovered and invalidate the path. Likewise, ants might get in the way of the calculated path. However, once we finish the HPA data structure, this should (hopefully) no longer be an issue.

The other issue, is the way the closest invisible spot is selected. Instead of just selecting the closest invisble spot, we have had success with instead selecting the closest invisible spot which has not been seen for some amount of turns.

Magnetic Spread Out If an ant has nothing to do, we want it to go explore. But since an ant may only move a single tile per turn, generating and storing a path to some destination becomes slow and ineffective, as many events may occur prior to that ant reaching its destination, making it a bad destination. Generating paths can also be very time consuming, so we wanted a way to spread out our ants without using too much time on it.

The idea is quite simple; we make all our ants repel each other if they are in close proximity and unlike actual magnets they never attract each other. This repulsion should somehow scale with the distance between two ants, to allow some ants to be squeezed closer together. This will cause all the ants to eventually end up in an evenly spaced grid, covering most of the map, giving us as much information about food and enemy locations as possible. When we get new ants from eating food, the grid will automatically adjust and expand to match the new number of ants.

The algorithm for this looks as follows:

Algorithm 2 Magnetic Spread Out

```
$antForce \leftarrow 10$
**for** $i = 1 \rightarrow count(availableAnts)$ **do**
  $currentDirection \leftarrow (0, 0)$
  **for** $j = 1 \rightarrow count(antsInRange)$ **do**
    $scaledForce \leftarrow antForce/distanceTo(ant[i], ant[j])$
    $directionToJ \leftarrow getVector(ant[j], ant[i])$
    $currentDirection \leftarrow currentDirection + directionToJ * scaledForce$
  **end for**
  $currentDirection \leftarrow normalize(currentDirection)$
  $moveAntInDirection(ant[i], currentDirection)$
**end for**
```

For assigning a direction for all our ants the running time of this approach can be as high as $O(N^2)$ (where N is the number of ants), which happens if all the ants were clumped up next to each other.

Once the ants were able to behave in a fashion similar to magnets, we wanted to try and see if it was possible to make them perform other strategies by further extending the magnet concept. Since the repulsion is based on a positive number, we could easily add an auxilary "ant", or attraction point, who attracted other ants. Such attraction points could then be placed on food locations or in narrow choke points to safe guard an area. This does cause a few problems though, as many ants could potentially go for a single food and then end up repelling each other such no ant ever reached the food. For guarding choke points we would face much the same problem as we would simply have an area with high density of ants but this does not ensure that any of the ants are safe from attackers.

4.2 Finding Food

Food spawns randomly on the map at the beginning of every turn. Being able to effeciently collect food and build up an army of ants is one the key aspects of a good bot. Half of the work is done by initially exploring the map and thereby finding the food.

Once the food has been discovered the bot needs to collect it by sending one or more ants to eat it. In our initial and current algorithm for deciding which ants should collect which food we use the stable marriage algorithm [GS1962] though slightly modified. Each pair of ant/food has the same preference for each other, however, we only form the pair where the distance between them is lesser than twice the view radius of the ant. This is a performance optimization. After we have found the ant/food pairs we find the path between every ant and its food and order the ant to start moving along it.

Enemy ants close to the food potentially pose a problem. If they are closer to the food than our closest ant, then we might as well give up, since it will always reach the food before us given that taking the food is the optimal solution and he plays optimally. If we are equally far from the food, then we have the possibility of sending several ants. Both of these considerations have not yet been implemented, but still remain as viable options for further improvement.

Furthermore, when two or more pieces of food are close to each other, it would make sense to only send a single ant to collect all of them.

4.3 Attacking Enemy Hills and Ants

Attacking is a hard and very important problem for the bot. Given the grid-based movement, we must rely on very precise collaboration of two or more ants in order to carry out a successful attack. We must have at least two ants attacking a single enemy ant simultaneously in order to avoid sacrificing an ant. Sacrificing ants is certainly an option, but it is associated with defense, not offence.

Correctly placing ants around an enemy ant, in order to attack it, requires precise timing. Furthermore it involves some prediction of the enemy ant's future movement. As ants can attack each other within a given radius (read circle), we cannot guarantee sacrificing one for one when attacking with just two ants. We can prevent losses in 3 of 4 cases of enemy ant movement. Using three ants in an attack, we can ensure no losses of our own, yet there is still the possibility of the enemy ant fleeing. All of this requires that only one enemy ant is within attack radius. This is mostly fine, as an attack on a single enemy ant is likely due to collecting food, otherwise we will be attacking an enemy hill which obviously cannot move.

Attacks require a number of ants doing basically the same thing for a number of turns. During this time, these ants will not be collecting food, thus reproducing further ants. This makes it hard to determine when an attack is appropriate, and just as important, which ants to use. As the game goes on and the density of ants increase this quickly becomes a complex and time-consuming problem to solve. For the current version of the bot, we rely on only looking at a small subset of closely located ants in order to carry out an attack. We are not yet sure how to prioritize attacks vs. finding food or exploring the map.

4.4 Hill Defence

When an ant moves on top of an enemy hill, the hill is destroyed and can no longer produce ants. Furthermore the owner loses two points.

We work with two zones of danger: Very close and thus dangerous and medium close, possibly dangerous. If an enemy ant is in very close proximity to one our hills, we assign the closest ant to move next to this ant which leads to a one to one trade off. If enemy ants are in medium range we assign an ant to move close to our hill for each enemy ant in this zone.

Algorithm 3 Simple Hill Defence

```
for $i = 1 \rightarrow count(myHills)$ do
    $currentHill \leftarrow myHills[i]$
    $dangerousAnts \leftarrow enemyAnts.FindInRange(currentHill, dangerThresholdRange)$

    for $j = 1 \rightarrow count(dangerousAnts)$ do
        $nearestAvailableAnt \leftarrow myAvailableAnts.FindNearest(currentHill)$
        $goal$
        if $distanceTo(dangerousAnts[j], currentHill) > criticalThreshold$ then
            $goal \leftarrow dangerousAnts$
        else
            $goal \leftarrow myHill$
        end if
        $pathToGoal \leftarrow findPath(nearestAvailableAnt, goal)$
        $moveAntAlongPath(nearestAvailableAnt, pathToGoal)$
    end for
end for
```

4.5 Data Structures

During the development of the bot we have actively used common data structures such as hash arrays and dynamic arrays. Besides these we have implemented three data structures to minimize the running time of our algorithms.

The first, SortList, is a dynamic array which allows us to insert elements and then it keeps them sorted in accordance to a certain logic which is provided when the data structure is created. SortList sorts itself by utilizing a binary search every time a member is inserted. We use this data structure when performing A-star searches to keep unevaluated nodes sorted ascendingly by distance to goal. This data structure did not exist in our language, and is otherwhere known as a priority queue.

Very often our bots has the need for "find nearest neighbour"- and "find nodes in range x"-searches. These are not trivial in 2-dimensional wrapping space. The Kd-Tree data structure provides these operations, and we have implemented it and used it with success so far. However, the Kd-tree has a fundamental flaw: It is not designed to work with spaces that wrap at the borders. We have not yet found a solution to this problem, but we have considered switching to a quad-tree.

During the competition the time limit for each turn was reduced from 2 seconds to 500 ms which caused our bot to time out. The A-star path finding algorithm used almost 85% of the total running time each turn when we had 80-140 ants. The main issue was that we calculated the full path between an ant and its goal every single turn. We tried caching these calculated paths between turns, but since an exploring ant could be disrupted by a spawned piece of food this was not enough when the number of ants grew. To solve this issue we decided to search for data structures/algorithms that would allow us to only compute a partial path and possibly allowing us to cache paths between popular locations. The partial paths found the by the data structure, is the path from the current location, to the nearest transit point - or to goal if it is within the same cluster. Hierachical Path-Finding using A-star solves both of these issues. This solution divides the space into a grid, and then only computes partial paths between these grids. We are still in the process of implementing this data structure.

5 Conclusion

Designing an AI for the challenge has proven to be difficult. Especially the time constraint had great influence on the algorithms. The challenge ends on December 18th 2011 and we still have a lot of ideas for improvements. We are currently ranked 166th of approximately 6000 players.

References

1. A. Botea, M. Müller, and J. Schaeffer. Near Optimal Hierarchical Path-Finding. Journal of Game Development, vol. 1, no. 1, 7-28, 2004.

2. J. Kleinberg and Eva Tardos. Algorithm Design. Pearson International Edition 2006.
3. D. Gale and L. S. Shapley. College Admissions and the Stability of Marriage. American Mathematical Monthly 69, 9-14, 1962.
4. J. L. Bentley. Multidimensional binary search trees used for associative searching. Communications of the ACM, 18(9):509-517, 1975.
5. Chandran, Sharat. Introduction to kd-trees. University of Maryland Department of Computer Science, http://www.cs.umd.edu/class/spring2002/cmsc420-0401/pbasic.pdf, 2002.

Appendix

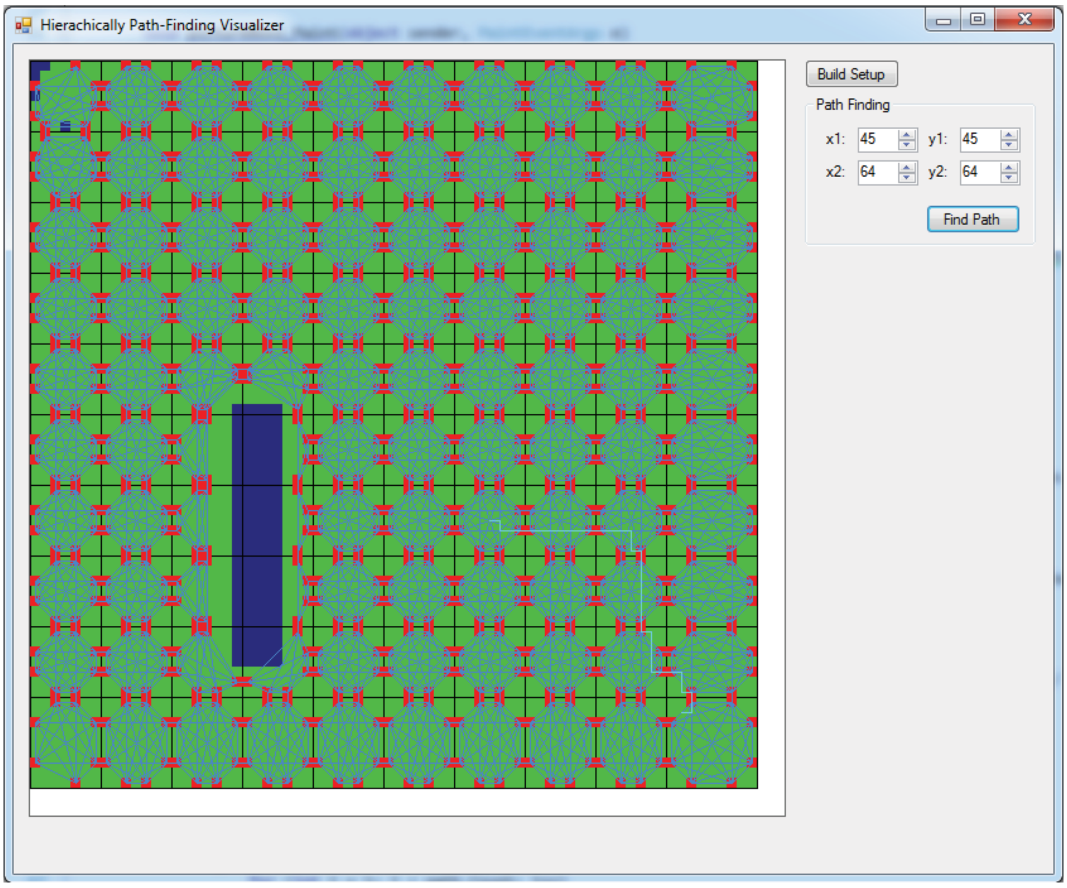


Fig. 1. A screenshot of the HPA visualizer. Green background is walkable terrain, dark blue is water and unwalkable. The clusters are visualized with black rectangles. The red rectangles are the transit points between clusters. The blue line between transit points are the internal cluster connections.

Collaboration and Agreement in Multi-Agent Systems

Mikko Berggren Ettienne

DTU Informatics

Abstract. In this paper we investigate the problems related to agreement, collaboration and coordination in multi-agent systems. We present a distributed auction-based agreement algorithm applicable to networked multi-agent systems in dynamic communication topologies. We also consider how a simple diffusion based approach to pathfinding can be extended, leading to spontaneous emergence of coordinated movement and collaboration in multi-agent systems.

1 Introduction

Coordination, collaboration and agreement are very generic and yet very challenging problems of designing and implementing multi-agent systems. In many applications of multi-agent systems, it is necessary to distribute a set of tasks or subgoals between the agents. The optimality of this assignment of tasks will often directly influence the efficiency of the system. We shall take a closer look at a distributed assignment algorithm inspired by [2,3] which was applied by the Python-DTU team in the Multi-Agent Programming Contest 2011 [4] to coordinate and distribute tasks between the agents.

We will dwell on the topic of coordination when considering collaborative diffusion which simultaneously handles pathfinding, coordination and to some extend also tactics in multi-agent systems. The simplicity of the approach makes it very appealing to many applications, especially in the field of competitive multi-agent systems.

The scope of this paper is to briefly present the ideas without going into complex details. However we note that we have actually implemented and tested the presented ideas in multi-agent contests, which somewhat proves, that they are applicable to "real" multi-agent systems.

2 Agreement in multi-agent systems

The obvious agreement problem inherent in most multi-agent systems is how to distribute or negotiate some limited resources between the agents. This problem can be treated under one with the problem of dividing a set of subgoals or tasks between the agents comprising the system, when considering the agents as the limited resource. In their essence, these problems are similar to the assignment problem, which is of great theoretical and practical significance and may often be encountered as a subproblem of more complex problems.

The assignment problem in its most general form, is formally defined as: *Given two sets of equal size, A and T and a weight function $C : A \times T \to \mathcal{R}$. Find a bijection $f : A \to T$, such that the cost function: $\sum_{a \in A} C(a, f(a))$ is minimized.*

The hungarian algorithm is one of many algorithms devised specifically to solve this problem and will do so within a time bounded by a polynomial expression of the number of agents. However the algorithm works in a centralized fashion, namely with a single processor that handles all information. In the following a distributed auction algorithm that will work in situations, where computation and information is distributed between multiple parallel processing agents, in a possibly incomplete communication topology, is presented.

The agreement problem for multi-agent systems can be defined similar to the assignment problem, however, the bijection is replaced by an injective map, such that the number of goals can be greater than the number of agents, while two agents must still not be assigned to the same goal. The problem of agreement on the use of limited resources is easily modeled by letting the limited resources appear as goals and adding imaginary goals to ensure that all agents can be assigned to one.

The algorithm described in the following achieves close to optimal results with a decent complexity, dependent on the communication topology. It resembles an auction where the agents bid against each other on their preferred goal, such that the algorithm will terminate in a finite number of rounds and all agents are assigned to different goals at termination. The algorithm will work in a dynamic communication topology where all pairs of agents are not necessarily connected at all times. However we do assume that the graph of the communication network is connected at all times.

Each goal j will at a given time t have a *price* denoted $p_j(t)$. To follow the intuition of an auction, we consider benefit of goals instead of cost of tasks. The cost function is easily achieved by inverting the benefit. The goal is then to maximize the sum of the benefits. Initially, we let $p_j(0) = 0$ for all goals. The price of a goal will be the highest bid made by an agent on that goal (except when the price is 0). As in a real world auction, agents will now place bids on the goals that give them the greatest *net value*. Here we define the *net value* of goal j for agent i as: $\textit{net value}_{ij} = \textit{benefit}_{ij} - p_j$.

In each bidding round, agents place bids according to their local information about the current prices. If an agent has not currently placed the highest bid on any goal then the agent will place a bid on the goal which maximizes his current *net value* according to his current knowledge of the prices of the goals. The highest bid as well as local information about the current prices and current highest bidders of goals are in each round sent to all neighbor agents, i.e. agents to which a communication channel exists. In addition local beliefs are updated according to the prices received from neighbor agents. If several agents have made the same bid for a goal, the agent with the highest index will win the goal. The updated values are then used by the agents to calculate the *net values* for the next bidding round. Thus, in one bidding round at time step t the algorithm works by letting each agent i do the following:

1. Receive the newest prices and owners of all goals. Update the local belief base if there are higher bids on any of the goals that the agent did not already know about. This includes updating the *net values* of the goals. Also, the agent may have lost a goal it owned in the previous round.
2. If the agent is not currently the owner of a goal, it will place a bid on the goal j with the highest *net value* according to its belief base. It does so by setting itself as owner of j and increasing p_j by $v_i - w_i + \varepsilon$ where v_i is the *net value* of j and w_i is the net value of the goal with the second highest *net value*. Intuitively, this ensures that the agents will not spend unnecessary time, raising each other on a single goal, but will instead go directly to their maximum bid given their current beliefs. This will not influence the final assignment, but will greatly decrease the running time of the algorithm. $\varepsilon > 0$ is a parameter of the algorithm that influences the running time and the quality of the final assignment. Generally speaking, a low value of ε gives high quality assignments but longer running time.

An example run is shown in figure 1 where $\varepsilon = 1$, goal benefits are integers and for simplicity, the communication topology is simulated by a shared data structure.

In general the algorithm terminates when Δ rounds without new bids occurs, in which case all agents have an assigned goal, where $\Delta \leq n-1$ is the maximum network diameter, which is the longest distance between two vertices in the communication network and n is the number of participating agents. Informally this should be clear, as all agents must be assigned to a goal, when no new bids occur and Δ is exactly the maximum number of rounds required, for a new bid to reach all agents. For a formal proof that the algorithm does in fact terminate no matter the choice of $\varepsilon > 0$ and no matter the choice of structure of the connected communication network we refer to [2] where it is also shown that the final assignment obtained by the algorithm is within $n\varepsilon$ from being optimal.

The algorithm terminates in $O(\Delta n^2 \lceil \frac{max_{i,j}\{benefit_{ij}\} - min_{i,j}\{benefit_{ij}\}}{\varepsilon} \rceil)$. To see why, we consider a proof along the lines of [2]: The bound of the running time is obtained by considering the worst case scenario where all agents continuously place minimum bids on every single task. Let δ be a bound on the maximum difference between the highest benefit and the lowest benefit for all agents and every goal, viz.

$$\delta > \max_{1 \leq i \leq n}\{benefit_{i,j}\} - \min_{1 \leq i \leq n}\{benefit_{i,j}\}$$

for all goals $j = 1 \ldots m$ where $m \geq n$. For sufficiently small $\delta > 0$ all agents will bid on the goals in the same order. Now order the goals according to benefit and let M_j be the difference between the minimum benefit for all agents for goal j and the maximum benefit for the less attractive goal $j-1$, viz.

$$M_j = \min_{1 \leq i \leq n}\{benefit_{i,j}\} - \max_{1 \leq i \leq n}\{benefit_{i,(j-1)}\}$$

again for all goals $j = 1 \dots m$, (Let $benefit_{i,0} = 0$ for all i). Now pessimistically assume that the agents will only increase their bids with $\varepsilon > 0$ and let goal n be the most attractive for all agents. The agents will then all bid on this goal until its price is increased by at least M_n which requires $\lceil \frac{M_n}{\varepsilon} \rceil$ bids by each agent and thus $n\lceil \frac{M_n}{\varepsilon} \rceil$ iterations of the algorithm. At this point goal $n-1$ also becomes attractive and will remain so for at least $2\lceil \frac{M_{n-1}}{\varepsilon} \rceil$ iterations for each agent, i.e. each agent will bid on both goal j and goal $j-1$. For all agents this results in a total of $2n\lceil \frac{M_{n-1}}{\varepsilon} \rceil$ iterations. Proceeding in the same fashion and summing up the total number of iterations we arrive at

$$n \sum_{j=1}^{n} j \frac{M_{n-j+1}}{\varepsilon} \leq n^2 \frac{max_{i,j}\{benefit_{ij}\} - min_{i,j}\{benefit_{ij}\}}{\varepsilon} \rceil$$

When taking into account that each bid may take Δ communication rounds to completely propagate in the network the proof is completed.

3 Collaborative diffusion

Diffusion describes the spread of particles through random motion from regions of higher concentration to regions of lower concentration. We will consider the discrete version of diffusion applicable to multi-agent systems, where the environment can be represented as a graph. The formal definition of diffusion in discrete environments as given in [5] is:

$$l_i^{(k+1)} = l_i^{(k)} + c \sum_{j=1}^{m} (l_j^{(k)} - l_i^{(k)}) \tag{1}$$

where $l_i^{(k)}$ is the diffusion value for a vertex i at time k, $m > 0$ is the number of neighbor vertices, l_j is the diffusion value of the neighbor vertex j and $0 < c \leq 1/4$ is the diffusion coefficient which controls the speed of the diffusion.

For simplicity we will in the following consider the special case of a two-dimensional tile based environment such that $n = 4$. Note that in all cases where $c = 1/m$, (1) can clearly be simplified to $l_i^{(k+1)} = c \sum_{j=1}^{m} l_j^{(k)}$, i.e. the new diffusion value for a vertex is the average of its neighbors' values.

To use diffusion as path finding, the idea is to put a constant high diffusion value in the target vertex. The gradual process of diffusion will then cause this value to gradually diffuse through the complete environment. Walls and other obstacles will have a constant diffusion value of zero such that nothing will diffuse through. Pathfinding for an agent is then simply reduced to walking up a hill, i.e. choosing the neighbor vertex with the highest diffusion value and the resulting path is close to optimal. An example of a pathfinding diffusion map is shown in figure 2-1. Whereas trivial pathfinding using stock graph algorithms such as breadth-first search runs in time b^d where b is the branching factor and d is the path length, the cost of computing a complete diffusion map is much worse. As the diffusion only converge one depth level at each iteration, the complexity is

$O((|V| + m) \cdot \log(|V| + m)^d)$, where $|V|$ is the size of the level graph and m is the number of neighbors to consider in the diffusion process.

However, the goal is not to compete with pathfinding algorithms on point to point pathfinding even though the diffusion approach to pathfinding has some benefits. Notice that the time taken to produce the diffusion map is independent in the number of agents searching for the target, thus the diffusion produces multiple paths simultaneously. This of course assumes, that the agents has access to some shared memory, or that a single master-agent computes the diffusion map in which case we may be considering a multi-body system. Figure 2-2 shows how a moving target is is also trackable through diffusion. In this case the constant high diffusion value of the target may move for each iteration dragging a trail to be followed by tracking agents.

Collaborative diffusion is where the true power of the approach becomes evident. Extending the diffusion map with constant diffusion values for agents gives rise to immediate collaborative movement behavior which in other cases can be very hard to achieve. In figure 2-3, a constant low value is assigned to the agents. This ensures that the agents will prefer to not clump up while pursuing the same target. The distance between the agents, i.e. how negatively they influence the diffusion in their proximity, is easily controllable through the choice of the custom diffusion constants given to the agents. Even more of what seems to be intelligent behavior emerges from this idea, when considering environments with multiple or maze like obstacles. As seen in figure 2-4, agents can block the diffusion on a narrow path, leading to other agents choosing different paths, similarly agents can split up at intersections. Using this approach, a set of pursuing agents can thus automatically narrow down the escape routes for a fleeing agent. When considering multiple targets one can choose to model the problem as a single diffusion map with multiple hills, or to create multiple diffusion maps. Note that multiple diffusion maps on the same environment can be updated in the same iteration, thus only influencing the complexity by a constant factor. Targets can then be split amongst agents by comparing the hill steepness of their positions in the different maps.

This approach achieves very complex collaborative and coordinated behavior in an elegant and simple manner, which is generally not the case for other approaches to collaboration and coordination. The collaborative diffusion approach has proven powerful enough to successfully and rationally control eleven players in a simulated football game as shown in [5].

4 Conclusion

In this paper we have considered the problems of agreement, collaboration and coordination in multi-agent systems. A robust distributed auctioning algorithm has been presented, which can handle the assignment of goals to agents, in dynamic networked multi-agent systems. The algorithm has proven to be effective in practice as it was successfully applied in the Multi-Agent Programming Contest 2011.

We have also presented collaborative diffusion as a completely different approach to agent collaboration, mostly concerned with coordination of movement. This is, however, an important and challenging aspect of multi-agent system design and the diffusion approach is especially applicable to AI in games.

We do not claim that the presented ideas, approaches and algorithms are the best solutions or that they will be effective in every case. Nevertheless we hope that we might have caught the readers interest and showed that the there exists many different approaches to the vast amount of complex problems inherent in the design and implementation of multi-agent systems.

References

1. Mikko Berggren Ettienne, Steen Vester, and Jørgen Villadsen. *Implementing a Multi-Agent System in Python with an Auction-Based Agreement Approach*, accepted for publication, 2011.
2. M.M. Zavlanos, L. Spesivtsev, and G.J. Pappas. *A Distributed Auction Algorithm for the Assignment Problem*, Proceedings of the 47th IEEE Conference on Decision and Control, Cancun Mexico, 2008.
3. D.P. Bertsekas, and D.A. Castanon. *Parallel synchronous and asynchronous implementations of the auction algorithm*, Parallel Computing 17, 707–732, 1991.
4. Tristan Behrens, Jürgen Dix, Jomi Hübner, Michael Köster, and Federico Schlesinger. *Multi-Agent Programming Contest — Scenario Description — 2011 Edition*, available online `http://www.multiagentcontest.org/`, 2011.
5. Alexander Repenning. *Collaborative Diffusion: Programming Antiobjects*, ACM SIGPLAN International Conference on Object-Oriented Programming Systems, Languages, and Applications, Portland, Oregon, 2006.

Acknowledgements

Thanks to Steen Vester for his work on the agreement algorithm which we in collaboration implemented for the Multi-Agent Programming Contest 2011 and documented in [1] on which parts of this paper build.

I would also like to thank my supervisor Jørgen Villadsen, who was responsible for igniting my interest in multi-agent systems and who has always been available for both sparring and guidance, for which I am very grateful.

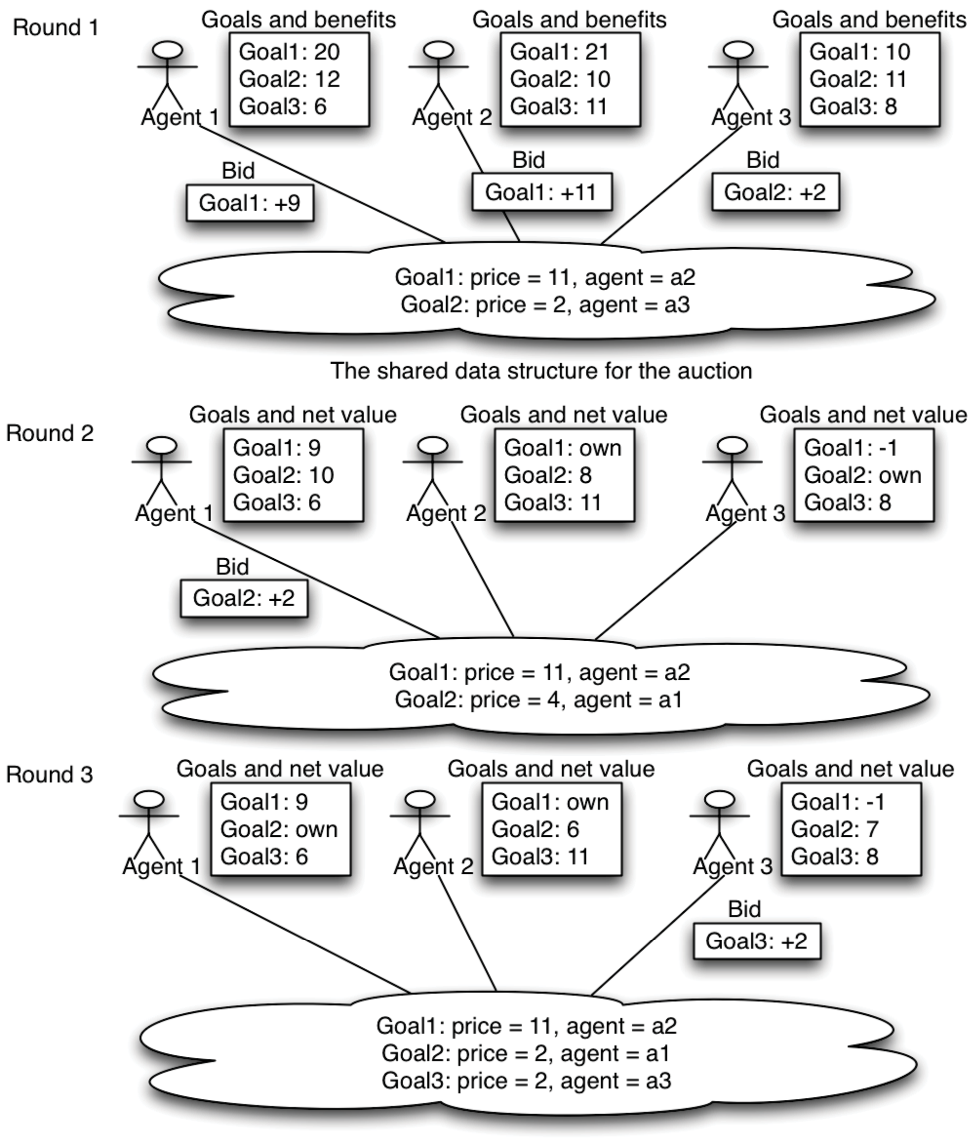


Fig. 1. An example of an auction between three agents. In round 1 the auction data structure is empty before the bidding and thus goal benefits are equal to goal net values. Each agent places a bid on its preferred goal. The bids are calculated as the difference between the two best goals plus ε. The data structure stores the highest bid and the corresponding agent for each goal. Both agent 1 and agent 2 bid on goal 1 and agent 1's bid is discarded as it is lower than agent 2's bid. In round 2 the net values have changed as the new bids in the shared data structure are considered. Agent 2 and 3 have not been over-bidden, so they won't bid in this round and does now consider themselves owners of the goals they bid on in round 1. However agent 1 overbids agent 3 on goal 2 as the shared data structure shows. Now in round 3 agent 1 has become the owner of goal 2 and agent 3 bids on goal 3 as this is the best goal for it considering the latest bids in the data structure. Now all agents are assigned a goal and the auction ends. We see in this case that we do not get the optimal solution (agent 1 = goal 1, agent 2 = goal 3, agent 4 = goal 2, total benefit = 42) but instead a solution very close to the optimal (total benefit = 41).

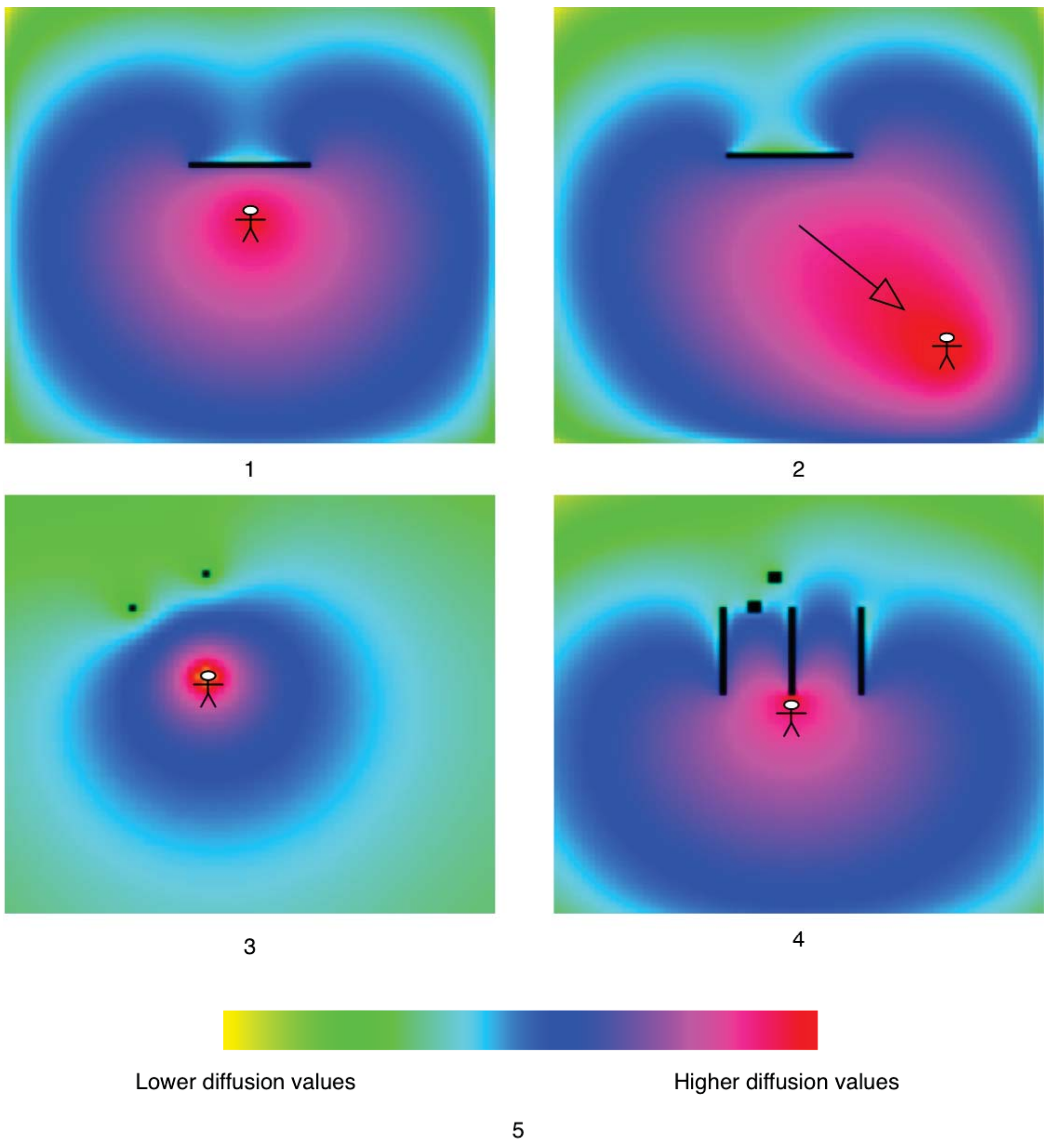


Fig. 2. Different examples of diffusion maps in two-dimensional environments. 5 shows how different colors are interpreted as different diffusion values representing a third diffusion dimension. In 1, the target is in the center and the black line is an obstacle blocking the diffusion. The diffusion values behind and around the obstacle clearly show how agents will be able to navigate around it. 2 shows how the diffusion naturally adapts to a target moving from the center to the lower right corner, dragging a trail of high, but decreasing diffusion values. In 3, the two black dots represent agents moving towards the center target. We see how they both repel the diffusion in their close proximity, ensuring that they will never come to close. Advanced flocking behavior can be achieved in this way. In 4 the furthest agent moving towards the target in the center will choose the right lane instead of following the agent in front of him. This is again achieved by assigning constant low diffusion values to the agents in this case making the closer agent block the diffusion on his path, from the agent behind him.

Inconsistency Handling in Multi-Agent Systems

John Larsen

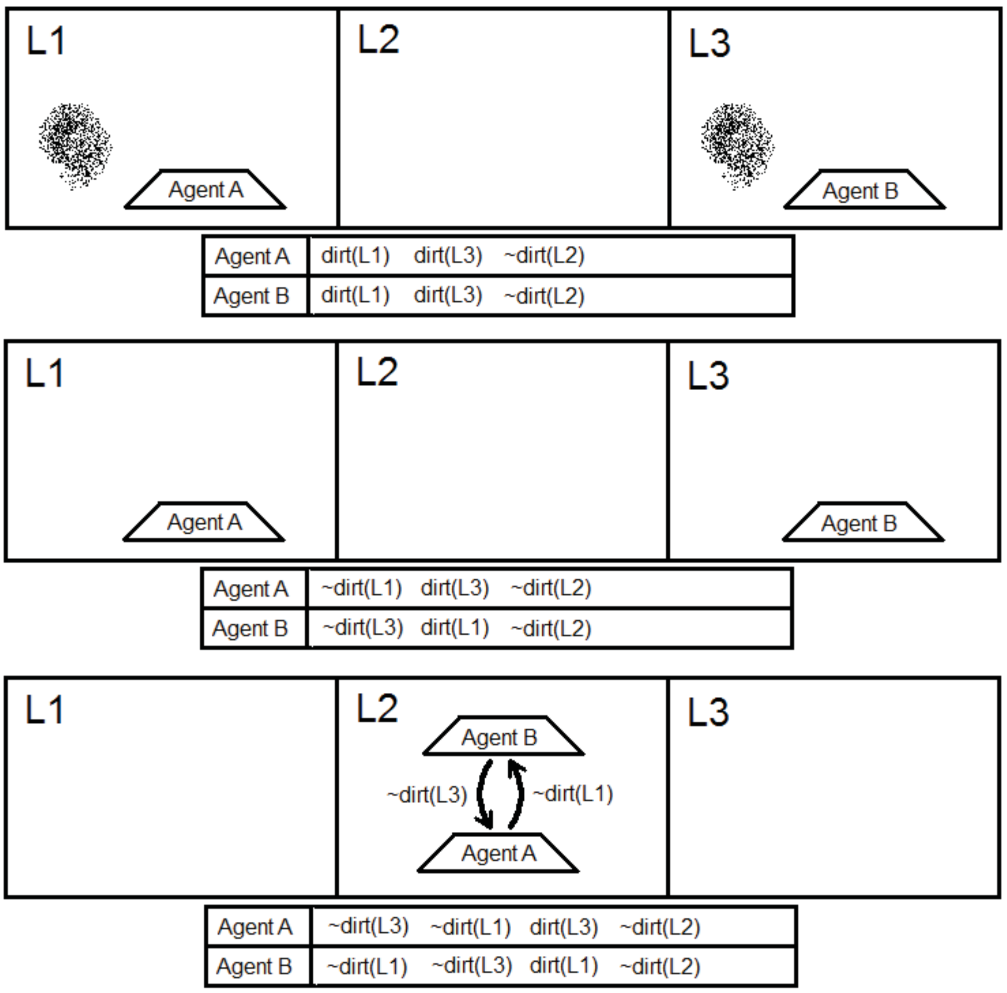
DTU Informatics

Abstract. This extended abstract describes how the problem of inconsistency can be automatically handled in Jason, a general purpose multi-agent system. There are two proposals, one based on belief-revision and one based on paraconsistency such that the agent can reason with an inconsistent belief base.

1 Introduction

Multi-agent systems can be written in any traditional programming language and often a system is optimized for a specific purpose and not anything else. However to use the multi-agent system for another problem, it often requires changing entire data structures of the agents to represent the new environment. One of the purposes of agent-oriented programming languages is to streamline this process by representing the environment internally in each agent as general logical beliefs. Essentially this is a list of logical statements that provides a model of the environment. Jason is a general purpose multi-agent system based on the agent-oriented programming language AgentSpeak. It interprets agents written in a language based on AgentSpeak, where agents reasons and acts from a set of logical beliefs, rules and plans. The necessary details of AgentSpeak will be presented in following sections when needed.

In Jason a very simple vacuum cleaner agent could represent the fact that dirt is at location `l` by the literal `dirt(l)` and the fact that there is no dirt by `~dirt(l)`. In the situation of figure 1, several vacuum cleaners work in the same environment and are expected to clean all locations without wasting energy going to locations cleansed by other agents. In a true multi-agent system they do not have one shared set of beliefs and hence must be able to tell each other "that area is already clean" and update their individual beliefs accordingly (assuming the agents are not lying). It is not difficult to make Jason agents tell each other new beliefs but if the set of beliefs of an agent becomes inconsistent it may behave unexpectedly and try to clean up already clean places anyway. This may be caused by a programming bug (I later consider intentional inconsistency) that can be solved but it requires stopping the agents and figuring out what caused the problem. The solution I consider tries to solve inconsistencies automatically when they appear and acts as a kind of safety mechanism in such cases.



Agent A	dirt(L1) dirt(L3) ~dirt(L2)
Agent B	dirt(L1) dirt(L3) ~dirt(L2)

Agent A	~dirt(L1) dirt(L3) ~dirt(L2)
Agent B	~dirt(L3) dirt(L1) ~dirt(L2)

Agent A	~dirt(L3) ~dirt(L1) dirt(L3) ~dirt(L2)
Agent B	~dirt(L1) ~dirt(L3) dirt(L1) ~dirt(L2)

Fig. 1. Example where two vacuum cleaners have the task of cleaning up dirt from three locations and their communication causes each agent to become inconsistent. Dirt in a location is shown as a dot cloud and each picture shows the state after the agent has performed an action. Below each picture is the current belief base of each agent. Initially both believe there is dirt at L1 and L3 and that L2 has no dirt. This knowledge may be aquired from their owner who told them "The kitchen and bedroom is dirty but the living room is not". The agents follow their goals of cleaning up all dirt and start by cleaning the room they start in. After this action both agents believe that their own room is not dirty. In order for each agent to achieve the goal of cleaning all rooms, it must then move to location 2 where it meets the other agent. When they meet they perform an action of exchanging beliefs of rooms without dirt, but neither of them performs any check of inconsistency after this exchange. Agent A now believes L1 is both dirty and not dirty and Agent B believes L3 is dirty and not dirty and thus both agents are inconsistent. What actually happens from that point depends on how they prioritize these beliefs and if they are ready to give up their previous goal and in general it is hard to tell what will happen. Assuming that the agents trusts each other, the most reasonable thing would be that both agents give up believing that there is dirt in L1 and L3 and consider all goals cleared.

The solution I consider revises the set of beliefs when an inconsistency happens to make sure that the set is consistent after whenever the agent learns something new. This is called *belief revision* and I implement an efficient algorithm for this proposed by the authors of Jason, that they have so far not implemented themselves. I present the key concept of the algorithm here but more details of this algorithm can be found in [1]. I also present the concept of paraconsistence, which is another way of handling inconsistency, and how it can be used in Jason.

2 Belief Revision

The default Jason agent already performs a belief revision, however it only modifies the set of beliefs (the *belief base*) without actually checking for consistency afterwards, so if the belief base already contains `dirt(L1)` and then adds `~dirt(L1)` the belief base afterwards contains both beliefs and is therefore inconsistent. The belief revision algorithm solves this inconsistency by *contracting* one of these beliefs in one of two ways, which is either by *coherence* or by *reason-maintenance.*
Contracting a belief by coherence means to modify the belief base minimally so that it no longer entails this belief. Any beliefs derived from contracted belief are left in the belief base. It is a quite simple revision and if the remaining beliefs later cause inconsistencies they can be contracted in a similar way. Contracting a belief by reason-maintenance both modifies the belief base like before and contracts all beliefs derived from the contracted belief. This can result in big parts of the belief base being removed due to an inconsistency and is useful if the agent should avoid potentially inconsistent beliefs as much as possible. Both styles of contraction are useful in different situations.

2.1 Justificatons

In order to carry out the contraction efficiently the algorithm uses a data structure called *Justification* for linking beliefs together in the way they derived each other. A Justification represents a single argument for a belief and is a tuple of a belief and the list of beliefs used to derive this belief. This list is called the *support list* and it is read as a conjunction, so the Justification can be removed if any of the beliefs in the list are removed.

$$\textit{Justification} : \textit{belief} * \textit{belief}\ \text{list}$$

If a belief is justified in more ways than one, there are multiple Justificatons for this belief but every belief in the belief base should have at least one Justification. Thus to remove a belief, it must lose all Justifications. In Jason an agent can add new beliefs through *plans* and each plan has a *context*, a logical formula that must be true when the plan is choosen to be executed. The context can then be used to form a Justification for this belief.

The Justifications and beliefs can also be seen as nodes of a graph where each belief has an outgoing arrow to each Justification where it occurs in the support list, and an ingoing arrow from each Justification that justifies the belief.

2.2 Contraction Algorithm

The resulting contraction algorithm uses the Justifications to find beliefs that must be removed to avoid entailing the contracted belief. This contraction is defined recursively on the support list in the following way, where the belief b is contracted.

- All outgoing Justifications of b, in other words justifications (b', s) where b occurs in s, are removed.
- For every ingoing Justification $j = (b, s)$ of the contracted belief, if s is empty, then j is simply removed. Otherwise a belief in s is contracted.
- Finally b itself is removed.

When choosing a belief in the support list s to contract, it is decided by a function $w : \textit{belief}\ \text{list} \rightarrow \textit{belief}$ which intuitively defines the least preferred belief in the list of beliefs, which is the belief the agent would be ready to give up first. A possible general definition of this is discussed in [1].
During the execution of this algorithm a belief might lose all Justifications. To do contraction by coherence, these beliefs receives a new Justification with an empty support list and to do contraction by reason-maintenance, these beliefs are contracted as well. In [1] it is argued why both approaches have the same complexity, the focus in this project is how to make an implementation that allows the agent to use both methods and evaluate on the usefulness of them.

3 Implementation

Jason is based on Java and comes with a default Agent class for all agents. This class defines a belief revision function that is used every time the Agent modifies the belief base due to communication or derivations made by the agent itself. This function modifies the belief base but does not check for consistency afterwards. However by extending the Agent class with a new class, this function can be redefined to apply the algorithm described in 2.2 after the belief base has been modified, while keeping everything else not related to the belief revision untouched. This was also suggested in [1]. In Jason the programmer can very easily choose to use such a derived class rather than the default class, and the good thing about this is that it does not affect older Jason systems by default, but it is easy to make them use the new Agent instead. To make it easy accessible I decided to put it in the default Jason library under the name `jason.consistent.BRAgent`.
To actually implement the new Agent however, it is required to know the internal structure of both the belief base, plans and intensions of agents in Jason in order to extract the required data. An overview of the relevant classes can be

found in [2].
The new Agent provides the algorithm for belief revision based on contraction and default definitions for auxillary functions such as $w(s)$ introduced before. Other auxillary functions, such as when and how to display debugging information, are defined in a generally useful way, but $w(s)$ should rather have a domain specific definition. Fortunately any of the functions can be redefined by extending the new Agent class with a domain specific class, such that the algorithm for belief revision is reused but under other conditions. This falls in line with the design principle of Jason, that parts can easily be extended with domain specific components if necessary. All these auxillary functions are discussed in [2].

4 Paraconsistency

An alternative way to handle inconsistency is by using a paraconsistent logic such that the effect of inconsistencies in a sense remains 'local'. A particular paraconsistent logic is the multi-valued logic presented in [3]. In this logic there are truth-values that indiciates uncertainty and the common logical operators are defined in this logic in a way that the truth-values are easy to compute. In [4] it is shown how a knowledge base can be implemented in Prolog to reason with this logic and in [2] it is shown how the paraconsistent logical operators can be defined in Jason as agent code. It remains a question whether a belief base can be build on top of Jason so that the Agent reasons with knowledge from this belief base by using the multi-valued logic.
Another possibility is to simply take advantage of the fact that entailment in Jason already is paraconsistent. A belief base in Jason only entails beliefs that directly occur in the belief base or can be derived by applying logical *rules* in the belief base. As an example assume that the vacuum cleaner is inconsistent and believes `dirt(L1)`, `~dirt(L1)` and `~dirt(L3)`, then it does not believe `dirt(L3)` as no rule entails `dirt(L3)` and `dirt(L3)` does not occur directly in the belief base. This means that it may make weird plans towards cleaning the already clean location `L1` but it does not try to clean `L3`. The problem is of course that the agent never realises by itself that something is wrong while, if it used belief revision, it would try to decide which of `~dirt(L1)` and `dirt(L1)` are true, and, if it used paraconsistency, it would find out that there is a problem with this belief.

5 Conclusion

The implementation of the belief revision by contraction stays true to the algorithm, however this means it also inherits some of the problems in it. The graph of Justifications and beliefs can only be correctly constructed if all plans have a conjunction of beliefs in the context, however Jason allows other types of logical formulae in the context as well. This is a serious limit to any practical use of the algorithm. Another serious limit is that it is undefined how the support list

should look like if the context contains a rule, since the instantiated rule is entailed by the belief base but does not occur directly as a belief. A way to define the support list is then by the beliefs that caused the rule to be true. Another way would be to let the rule itself occur in the support list, rather than what caused it to be true. None of these suggestions have been worked with in this project due to a time limit.
The positive result is that the implementation can solve the trivial problems by itself, which would otherwise require the system to be stopped. It shows some promise in belief revision by contraction.
The experiment with paraconsistency shows that at least a portion of paraconsistent reasoning can be implemented in Jason but it is currently not very useful, as it is only able to compute truth-values with a constant model without variables. It also showed that while an inconsistent agent may be able to do something, it requires more to actually handle the inconsistency on its own.

References

1. Rafael H. Bordini, Jomi Fred Hübner, Mark Jago, Natasha Alechina and Brian Logan. Automating Belief Revision for AgentSpeak. Declarative Agent Languages and Technologies IV, 61-77, SpringerLink, May 8 2006.
2. John B. Larsen Inconsistency Handling in Multi-Agent Systems Technical University of Denmark, DTU Informatics, 2011.
3. Jørgen Villadsen. A Paraconsistent Higher Order Logic Paraconsistent Computational Logic, 33-49 Springer Online First 6 May 2002.
4. Johannes S. Spurkeland Using paraconsistent Logics in Knowledge-Based Systems Technical University of Denmark, DTU Informatics, 2010.

Developing Web Application Clients Using Multi-Agent Programming Patterns

Niklas Christoffer Petersen

DTU Informatics

Abstract. In this paper we justify the reason to consider new approaches when developing complex web application clients, and presents a specific method to incorporate *Multi-Agent Programming* (MAP) patterns, namely the small research project *JaCa-Web*. Together with other MAP technologies, the JaCa-Web project provide a framework for web application clients. We present this framework and provides a small sample as a proof of concept.

1 Introduction

The internet connectivity has continued to increase steadily over recent years cf. [1], while browsers have gained complex features with the adoption of new standards, noteworthy *HTML5* and *CSS Level 3* cf. [2] respectively [3]. The browsers are also executed on more powerful hardware, and utilize it more intensely, e.g. several browsers render using hardware accelerated graphics.

This development has made it more attractive to consider a web application approach when designing new applications or updating existing desktop applications.

There are some very interesting advantages that follows implicitly when choosing a web application over a desktop application. For instance one obtains much higher device independence with minimum efforts. This not only limits to PC devices running different operating systems, but also smart phones, tablets, etc. Likewise it is easy to distribute and update the applications, since all application logic and data is kept central. Client specific resources are downloaded dynamically as they are needed from the application server with configurable caching.

However there are also some strong limitations that can render the web application approach unfit. By default any web application will require recurring access to the application servers, thus they will not function offline. Another concern is the abstraction level, i.e. they have no or very limited access to client hardware, and in newer browsers they often run in a sandboxed environment. A simple task as printing a correctly formatted A4-paper can then become difficult to accomplish.

Taking this into account, there still are many cases where the pros of a web application approach surpasses the cons. Given this it is also worth considering the effort needed to develop such applications.

1.1 Traditional approaches

A deep presentation of how web applications works is out of the scope of this paper, but an essential aspect is, that since web applications relies on standards on the clients, the natively supported programming languages are effectively limited to a single option, namely *JavaScript*.

JavaScript is a multi-paradigm language that follows a mix of prototyped, object-oriented and functional paradigms. It was originally designed as a scripting language as the name hints, and even though its syntax originally was inspired by Java, it shares no other relation to the language. Initially it filled a simple role of validating user input, and for small animations.

The language really gained traction with the introduction of AJAX (*Asynchronous JavaScript and XML*) support, which allows web application clients to asynchronously exchange data with back-end application servers. Prior to this dynamic content was generated on the application server and sent as hole to the client. If any exchange was needed the only way was a complete reload of the web view, called a *server round-trip*. AJAX is a good demonstration of some of the shortcomings of the *JavaScript* language.

Specifically AJAX allows a web view to send requests to an application server without the need of a server round-trip, and can use the response to partially update itself. Without doing this asynchronously, one would break the responsiveness of the web application due to the natural latency of network communication. The way JavaScript implements asynchronicity is by using *callbacks*, thus an asynchronous task as an AJAX request is of the form (1), where () denotes the empty result.

$$\mathbf{request}_{ajax} : \mathit{request} \rightarrow (\mathit{response} \rightarrow ()) \rightarrow () \tag{1}$$

The problem with this structure is that it turns your implementation inside out, where calls to the $\mathbf{request}_{ajax}$ function completes immediately, and the supplied callback function *may* be applied sometime in the future. This poses complications when considering which context the callback function is applied in (e.g. with respect to global state). Furthermore it complicates error handling (e.g. if the request fails). In the latter case it is common to supply a secondary callback function for handling unsuccessful requests as shown in (2).

$$\mathbf{request}_{ajax} : \mathit{request} \rightarrow (\mathit{response}_{suc} \rightarrow ()) \rightarrow (\mathit{response}_{err} \rightarrow ()) \rightarrow () \tag{2}$$

It is argued that such constructs makes it hard to ensure the correctness of the implementation, and thus complicates the development.

1.2 Proposed solution

The proposed solution is to use patterns known from Multi-Agent Programming (MAP). Especially the combination of both reactive and goal-driven agents are useful in web application clients where, as evident from above, several of the operations are handled asynchronously. Likewise the concept of a knowledge base for state representation can simplify the context of which the client react to the completion of asynchronous tasks. Finally using multiple agents allows the separation of responsibility for flow and state.

Continuing with the AJAX example we propose the form (3), where responses are transparently injected, upon their availability, into the knowledge base of agents that observes the request. Due to the reactive behaviour of agents they can choose to perform actions (such as pursue new goals) based on such new knowledge.

$$\mathbf{request}_{ajax} : \mathit{request} \to () \tag{3}$$

General purpose languages such as JavaScript can easily be used to model pure reactive behavior or completely goal-driven systems. However achieving a system that possesses elements from both of these models have shown difficult using these traditional approaches cf. [4, p. 5]. It is worth to emphasize that this in no sense implies that it is not possible to implement a system directly in JavaScript that will allow such behavior. However, from a development perspective, it is not only interesting to consider if a system is possible, but also what effort is needed to implement it.

There are mainly two methods of achieving paradigms that are poorly supported by, or absent from, the standard compliant clients:

- Plugin-based: Many browsers supports plug-ins, such as Java or Flash. This allows the distribution an application in a language that is handled by a plugin. The major downside of this approach is that one really trades off the device independency, since a plugin for each platform is required. This method can create a steeper adoption of the client, since it will require the users to perform additional steps if the plug-in is not present on their systems.
- Synthesize-based: The application is developed in a language that suit the needs (e.g. language-based abstraction, type-safety, etc.) and is then synthesized into standard compliant languages. This is very comparable of how desktop programs are compiled, but instead of compiling into platform specific machine language, the program is "compiled" into the client supported languages. This does not affect the device independency, however developing the toolchain needed to perform such synthesize is clearly not trivial.

Examples of each method are: the Danish common log-in solution *Nem-ID*[1], which runs as a Java-plugin; and respectively *GMail*[2], an email service provided by *Google* synthesized using the *Google Web Toolkit*[3] (GWT).

In this paper we shall consider a plugin-based method to incorporate *Multi-Agent Programming Patterns* for developing web application clients. We consider a small research project named the *JaCa-Web* which relies on the Java-applet-plugin.

[1] `https://www.nemid.nu/`
[2] `https://gmail.google.com/`
[3] `http://code.google.com/webtoolkit/`

2 The JaCa-Web project

The JaCa-Web project builds on top the *Multi-Agent programming* (MAP) technologies *Jason* and *Cartago*. Each of these fills a distinct role in the framework: *Jason* facilitates programming and execution of rational agents, while *Cartago* allows abstraction for creating complex environment artifacts. Lastly the JaCa-Web provide an interface between these technologies and the web compliant languages, specifically JavaScript.

Jason is specifically designed to ease the process of developing systems that are both reactive and goal-driven. It is an implementation of the abstract agent programming language AgentSpeak [5]; and thus based on the *Belief-Desire-Intention* (BDI) model. A complete introduction of the Jason language and framework is out of the scope of this paper, but is comprehensively presented in [4].

The Cartago project introduces the possibility to program and execute virtual environments. The basic concept is to move the interpretation of agent actions away from the environment, and instead bind the actions, called *operations*, to the *artifacts*, i.e. the objects of the environment. More specifically these operations are bound to any artifact that a given action affects. Thus the programmer does not need implement an environment, but instead the artifacts of the environment. Agents can then create defined artifacts, and discover artifacts created by others. This avoids the clutter of having to interpret every possible agent action in the same context, and moves the modeling of the environment up on the abstraction scale.

Specifically Cartago introduces two well-known software concepts on a sufficiently abstract plan, namely *observability*, i.e. agents can subscribe on artifact changes in an efficient and reactive manner; and *concurrency*, i.e. the execution of operations can interleave. For deeper details of Cartago please refer to [6], and for the integration between Jason and Cartago, and their use to [7].

The JaCa-Web project presents the connection between the web application views, and the application logic by supplying a set of Cartago artifacts and a small JavaScript-library cf. [8]. A block diagram of the structure of a complete web application client is shown in figure 1.

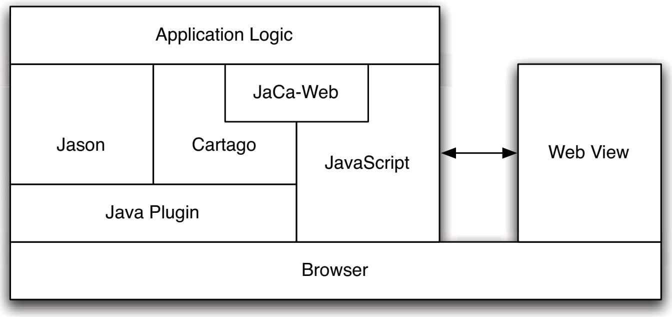


Fig. 1: Block diagram of web application client.

The application logic is mainly implemented by specifying agents in AgentSpeak using Jason, and by implementing needed artifacts and their operations in Java using Cartago. Finally some JavaScript might still be needed to update web views. With multiple agents it is possible do divide logic responsibility between agents. Each agent can have different goals to achieve, that corresponds to its responsibility, while it can react on percepts, for instance user events and artifact changes. Each web view has by default an artifact, namely the *page artifact*, but there is no restriction on creating as many artifacts as needed.

We shall now consider a simple auction web application, which client will show the current price and winner of the auction, and allow its users to place bids, similar to real auction services such as *eBay*[4]. There are two agents involved in this setup: the *updateAgent* will be responsible for keeping the web view updated, and the *bidAgent* will handle bid events from the user. The AgentSpeak code for each agent is shown in figure 2.

```
/* Initial beliefs and rules */
page_artifact_name("MyPageArtifact").

/* Initial goals */
!init.

/* Goal-driven Plans */
+!init
     :    page_artifact_name(PAName)
     <-   lookupArtifact(PAName,PAId);
          focus(PAId);
          !!refresh.

+!refresh
     <-   sendWsRequest;
          .wait(1000);
          !!refresh.

/* Reactive Plans */
+price(P)
     <-   updateView("price", P).

+winner(W)
     <-   updateView("winner", W).
```

(a) updateAgent

```
/* Initial beliefs and rules */
page_artifact_name("MyPageArtifact").

/* Initial goals */
!init.

/* Goal-driven Plans */
+!init
     :    page_artifact_name(PAName)
     <-   lookupArtifact(PAName,PAId);
          focus(PAId).

/* Reactive Plans */
+buttonClicked
     :    price(N)
     <-   bid(N + 1).
```

(b) bidAgent

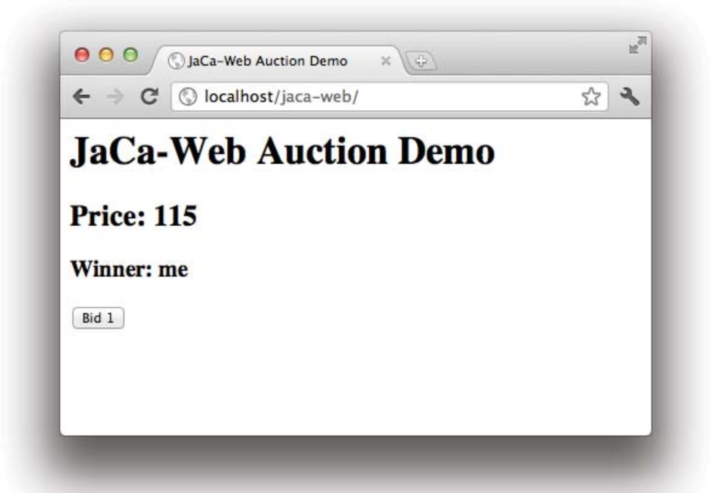
Fig. 2: Sample agents for auction scenario.

The *updateAgent* will continuously have the goal to *refresh* data from the application server. This is done by asynchronous pooling a web service on a regular interval. When responses are available it will change the knowledge base of the agents, that currently are *focussing* on (i.e. observing) the page artifact. Thereby the re-active plans of the *updateAgent* will apply, and the agent will update the web view.

The *bidAgent* will only have an initial goal of focussing on the page artifact, which is completed almost instantly. Thereafter it will only act on the *buttonClicked* knowledge, that are present when the user clicks on the bid button. In this case the *bidAgent* will increase the current known price with one,

[4] http://www.ebay.com/

and place a bid (also a request to the application server). If the bid succeeds (i.e. it is not overbid immediately before or after by somebody else), the *updateAgent* will retrieve this information back upon its next refresh, and show the user as the current winner. A screenshot of the web application is shown in figure 3.



JaCa-Web Auction Demo

Price: 115

Winner: me

Bid 1

Fig. 3: Screen shot of auction web application client.

The presented sample shows that by using MAP patterns it is possibly to divide the logic of a web application client into different responsible agents, that each can have goals and instantly act on newly achieved knowledge (e.g. state-changes and events). It is also possible, though not demonstrated in this sample, to pass messages (i.e. knowledge) between agents, such that for instance one agent can delegate some or all of its responsibilities to another.

As mentioned previously the JaCa-Web project is the outcome of a small research project, and the current version is in a very early alpha development phase. However it has shown a very interesting and different approach for developing web application clients. Besides it was never designed for production purposes, the fact that it depends on Java also makes it somewhat unappealing, since some platforms does not support the plugin-method[5]. However it is possible to imagine the same MAP patterns in a synthesized method. Another obstacle is that the framework still relies on JavaScript to actually update the web views, this is somewhat a stopgap solution. Finally it is also somewhat a downside that all the Java libraries (Jason, Cartago and JaCa-Web) and the application logic are required to be digitally signed by a trusted chain, otherwise the Java Runtime Environment will not execute them.

[5] Noteworthy Apple iOS and Microsoft Windows 8 (ARM version)

3 Conclusion

Justification was presented, showing that traditional approaches raises some problems when web application clients needs to efficiently implement correct handling of asynchronous tasks such as AJAX requests. The proposed solution presents the use of *Multi-Agent Programming Patterns*, and how systems with both reactive and goal-driven behaviour can present a succinct solution model for this.

The JaCa-Web research project presented a specific framework, that builds on known Multi-Agent technologies (Jason and Cartago), for achieving such solutions. The AgentSpeak programming language allow the logic of web application clients to be implemented in a succinct manner, while the Multi-Agent model allow the devision of logic responsibility between agents. The implemented auction sample application demonstrated the capabilities, and performed as a proof of concept. Overall it poses a very interesting approach for solving some of the problems that arises when developing complex web application clients, however it also has some shortcomings that makes it unfit for production.

In order to really compare it against alternative methods such as the GWT it is necessary to consider far more complicated scenarios than the simple auction scenario presented here.

References

1. David Belson, Brad Rinklin, and Tom Leighton. The State of the Internet, *Volume 4, Number 2.* Akamai Technologies Inc., 2011.
2. Ian Hickson et al. *HTML5 - A vocabulary and associated APIs for HTML and XHTML.* World Wide Web Consortium, 2011. `http://w3.org/TR/html5/`
3. Elika J. Etemad et al. *Cascading Style Sheets (CSS) Snapshot 2010.* World Wide Web Consortium, 2011. `http://w3.org/TR/CSS/`
4. Rafael H. Bordini, Michael Wooldridge, and Jomi Fred Hübner. *Programming Multi-Agent Systems in AgentSpeak using Jason (Wiley Series in Agent Technology).* John Wiley & Sons, 2007.
5. Anand S. Rao. *AgentSpeak(L): BDI Agents Speak Out in a Logical Computable Language*, 1996.
6. Alessandro Ricci, Andrea Santi, Michele Piunti, Mirko Viroli, Andrea Omicini, Marco Guidi, and Mattia Minotti. *CArtAgO Project Website.* `http://cartago.sourceforge.net/`.
7. Niklas Christoffer Petersen *Programming Multi-Agent Systems.* Technical University of Denmark, Department of Informatics and Mathematical Modelling, 2011.
8. Mattia Minotti, Andrea Santi, and Alessandro Ricci. *Exploiting Agent-Oriented Programming for Building Advanced Web 2.0 Applications.* Multi-Agent Logics, Languages, and Organisations Federated Workshops, 2010.